AISB Convention 2015

University of Kent, Canterbury

Proceedings of the AISB 2015 Symposium on AI and Games

Edited by Daniela Romano and David Moffat

ISBN: 978-1-908187-47-5

Introduction to the Convention

The AISB Convention 2015—the latest in a series of events that have been happening since 1964—was held at the University of Kent, Canterbury, UK in April 2015. Over 120 delegates attended and enjoyed three days of interesting talks and discussions covering a wide range of topics across artificial intelligence and the simulation of behaviour. This proceedings volume contains the papers from the *Symposium on AI and Games*, one of eight symposia held as part of the conference. Many thanks to the convention organisers, the AISB committee, convention delegates, and the many Kent staff and students whose hard work went into making this event a success.

—Colin Johnson, Convention Chair

Contents

Modelling Cultural, Religious and Political Affiliation in Artificial Intelligence Decision-Making

Mark R Johnson[1]

Abstract. This paper examines cutting-edge work in the generation of individual AI actors who behave according to procedurally-generated social, cultural, political and religious norms. Based on the author's ongoing development of the game *Ultima Ratio Regum* (URR) – built with a hand-made game engine in Python – the paper explores three core aspects of URR's AI actors. Firstly, the generation of a full world population of AI actors and ensuring that they are distributed appropriately and logically for a culturally-varied world; secondly the procedural generation of densely complex religious, political, cultural and socially normative values to assign to these AI actors, and how their decision-making processes are determined by these allegiances; and thirdly and lastly how this game, among other objectives, seeks to forward what I term "qualitative AI" where culture and society, not pathfinding and "optimal" decision-making, are the primary determinants of behaviour. The paper concludes with a summary of both these three points and the future plans for the game's AI systems.

1 INTRODUCTION

This exploratory paper is based on the author's own work, having been for the last three years the sole developer of the roguelike game "*Ultima Ratio Regum"* (URR). Set during the Scientific Revolution, almost everything within the game is procedurally generated – this ranges from the "macro" level of 2000+ years of detailed history, historical figures, empires and nations, religions, wars, dozens of vast cities and a vast world population of procedurally-generated non-player characters (NPCs) unique to each playthrough, to the "micro" level of individual towns and cities, individual NPCs, specific buildings and items, and flora and fauna. Inspired by the works of Umberto Eco and Jorge Luis Borges, the game is an exploration of a number of themes including historiography and the writing of the historical record, metanarrative and political ideology, and the philosophical idealism of George Berkeley. Most crucially the work aims to specifically integrate this "thematic" content with the game's mechanics, rather than leaving such content as "background" or "lore" that the player can take or leave. Much of this will be achieved through the use of innovative AI actors currently being developed at time of writing. Every aspect of the behaviour of these actors – their greetings, their insults, their dress, their farewells, their behaviour in challenging situations, their reaction to those from other nations, and much else – is procedurally generated, and fore-grounded in their decision-making algorithms. It is these actors and the roles they play which this paper focuses upon, and the break they represent from much traditional AI research into decision-making optimization [1] and pathfinding [2].

2 PROCEDURAL GENERATION AND ARTIFICIAL INTELLIGENCE

Firstly, the paper explores the procedural content generation of the game, and how this affects the AI actors. An "average" generated URR world has a population of approximately ten million NPCs. Naturally for such numbers, the management of these NPCs takes place at a number of different "levels" in the game depending on the player's activities – many of the NPCs are "abstracted out" at any given time. The game also contains a system which identifies the most "important" NPCs and ensures that their actions and decisions are always simulated regardless of the player's location (roughly 500-700 NPCs on average are considered "important" by the game's algorithm at any one time, and their actions are carried out constantly, unless one falls below the metric for "importance", at which point that actor is then abstracted out once more). NPCs within the game vary according to a significant range of variables: according to their race, language, cultural background and cultural norms, sex and gender, age, political alignment, religious beliefs (if any), national citizenship, and interests and agendas. A rough calculation currently suggests that there are over 1 trillion possible AI actors that may be procedurally generated within the game world who will, crucially, behave differently according to the social and cultural context within which they are generated. The agendas of these actors (returned to later in this abstract) are largely dependent on their cultural and religious backgrounds, leading to a densely complex world within which the player will uncover information about religious feuds, cultural differences, long-standing war bitterness, language difficulties, and many similar concepts of a sort not normally explored in games. The paper will therefore examine the generation of the AI actors from a creative standpoint; the management of so many AIs from a technical standpoint; and the integration of the two into a culturally and socially variegated and dense world.

[1] Science & Technology Studies Unit, Dept. of Sociology, Univ. of York, YO10 5DD, UK. Email: mrj503@york.ac.uk.

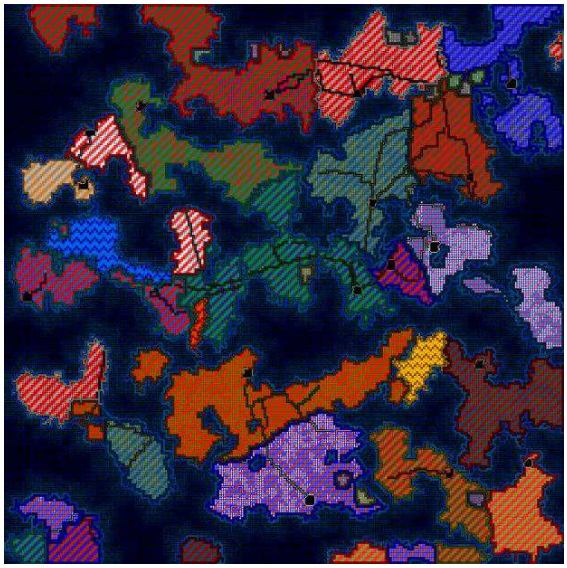


Figure 1. Example of Generated Political Divisions

Secondly, the paper explores how this content generation creates a deeply complex environmental simulation, arguably one of the most detailed and dense generated worlds ever created in a game. As above, the agendas of these AIs are dependent on the procedural generation of their *origins*. A generated URR world contains approximately forty civilizations (Figure 1) designed to emulate the massive variety in real-world civilizations from this historical era. Some may be nomadic desert peoples who travel in lengthy caravan routes across the world, or hunter-gatherer tribes in close to the Arctic Circle who construct their buildings from ice and stone and have limited trading relations with a nearby civilization, or feudal civilizations who range from the imperialist and the expansionist to the protectionist or isolationist, and have widely differing cultural preferences on issues such as aesthetics, slavery, gladiatorial sport, ethics and morality, and so forth. This variety extends into other areas, such as religion, where a complex algorithm can procedurally create over a million detailed religions with information about their beliefs, their god(s), what festivals or special events are on their religious calendar, their relationships with other religions, their presence in civilizations, eschatological and creation beliefs, the appearance of their altars, expectations from worshippers, etc. All of these "cultural actors" inform the creation of the AI actors who exist *within these contexts*. Crucially, therefore, rather than presenting this civilizational/cultural/religious detail as "background" or "lore" as many games do, they are foregrounded in the AI actors, whose motivations, interests and agendas can only be understood via a detailed understanding of the generated cultural backgrounds from which they originate. In turn, this affects their willingness to interact with the player, to assist or communicate with the player, and to potentially oppose the player if the player has aligned themselves with religions or cultures inimical to those of other NPCs. At the same time, it is a game of incomplete information [cf 3] where both the player, and NPCs, must make judgements about the opinions of others based on the data they possess. The actions of AIs are dependent upon the social conditions and expectations into which they are "born", and therefore strongly differentiate between all the procedurally-generated AI actors in a given instance of the game. Equally, the greater the knowledge the player has attained about the world's culture, the more able the player is to make their wishes felt within the game world.

3 TOWARDS QUALITATIVE AI

Thirdly, the paper brings these together to explore the use of this integration of procedural generation and sociological concepts as a method for game-based learning in the fields of philosophy, sociology, and the humanities more generally. AIs respond and behave according to their political, cultural, social and religious affiliations, and this transforms these concepts in the social sciences into gameplay mechanics that affect the behaviour of AI and the world the player explores, rather than simply a method for *constructing* a game world which then has no further impact upon the player's experience. This is in part akin to the world by Mateas on "expressive AI" [4] and Gruenworldt and Katchabaw's "Realistic Reaction System" [5] but develops it into further qualitative and social science domains, and integrates far broader "relationship" structures of religions and cultures into the interpersonal dimension previous focused upon. The paper therefore explores how the game depicts the influence of these many factors on social interaction, and how these influences are represented in the actions, decisions and interests of the game's AI. In turn, this leads to game-based learning where understanding the cultural, political and religious motivations of AI actors is actually essential to success or failure within the game world. Lastly, this also serves to illustrate the potential for the development of 'qualitative' game mechanics in video games more generally, and highlights the potential for the use of complex AI actors in moving away from the ubiquitous stat-based gameplay of levels, items, rewards, and so forth, and towards developing "AI" that can be understood in terms of their as full actors with a range of interest and agendas, rather than as only actors in combat or strategy situations.

4 CONCLUSIONS

The paper explores three central components to the game's AI – the emphasis on procedural content generation and the integration between that and artificial intelligence; the emphasis within this on creating cultures, societies and religions, and having these directly influence AI decisions; and thirdly the potential for this game to develop "qualitative AI" and to create gameplay mechanics based on political, sociological and humanist concepts rarely explored in interactive media. It notes the potential educational and pedagogic value of these, the potential for new forms of gameplay rarely explored in computer games, and the paper lastly notes the planned future developments of the game's in-development system.

REFERENCES

[1] G. Chaslot, S. Bakkes, I. Szita and P. Spronck. Monte-Carlo Tree Search: A New Framework for Game AI. In: *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*. AAAI (2008).

[2] T. Standley. Finding Optimal Solutions to Cooperative Pathfinding Problems. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*. AAAI (2010).
[3] D. Billings, A. Davison, J. Schaeffer, D. Szafron. The Challenge of Poker. In *Artificial Intelligence*, 134:1-2:201-240 (2002).
[4] M. Mateas. Expressive AI: Games and Artificial Intelligence. *edings of Level Up: Digital Games Research Conference* (2003).
[5] L. Gruenwoldt and M. Katchabaw. Creating Reactive Non Player Character Artificial Intelligence in Modern Video Games. In *Proceedings of the 2005 GameOn North America Conference* (2005).

Discerning Human and Procedurally Crafted Content for Video Games

Tommy Thompson[1] and **Rob Watling**[2]

Abstract. We discuss the results of a preliminary study where participants discern between human and computationally crafted content for a video game. Participants were tasked with completing a portion of the game with the knowledge that segments were created either by a procedural generation algorithm or by a game designer. When asked to discern which segments were built by humans and vice versa, overall accuracy of participant guesses is relatively low. However, rationale reached by participants in making these conclusions leads to some interesting discussion about expectations of procedural generation systems and requirements for future studies.

1 Introduction

Procedural Content Generation (PCG) is a popular design paradigm found in video game development. While the origins of this method can be found in the likes of *Elite* [3] to overcome hardware limitations, the emphasis has shifted towards experimentation and challenge. This is typified by the *Borderlands* series [5]: where weapons and tools are presented for the player to discover, adopt or discard based upon personal preference. Meanwhile, *Diablo* [4] and *Spelunky* [21] adopt PCG for map generation in an effort to retain variety, novelty and challenge for even the most seasoned of players

If we consider this transition of the role of PCG systems, what is most interesting is that players perception of in-game content is becoming of greater focus. As problem scope increases, developers place a stronger emphasis on ensuring content is as interesting as it is varied. This has resulted in significant work in Artificial Intelligence (AI) to create intelligent PCG processes [19], with efforts to create 'custom' and more bespoke content [6, 20] and tools to aid the development process [8].

In this paper, we discuss preliminary work in generating content for an 'endless runner' game entitled *Sure Footing*[3]. The game tasks players with navigating a hazardous environment for as long as possible. Players are presented an early build of the game that carries content designed both by the developers and an early build of a PCG system. The task for participants was to identify the human-built and PCG samples and give a rationale for why they reached their conclusion. Our hypothesis was that if we were to base our PCG system on a meta-creative approach; adopting principles from a human designer, that players by-and-large would struggle to identify any key differences.

[1] University of Derby, UK, email: tommy@t2thompson.com
[2] University of Derby, UK, email: therobwatling@gmail.com
[3] A game being developed by Table Flip Games Ltd.: `http://www.tableflipgames.co.uk`

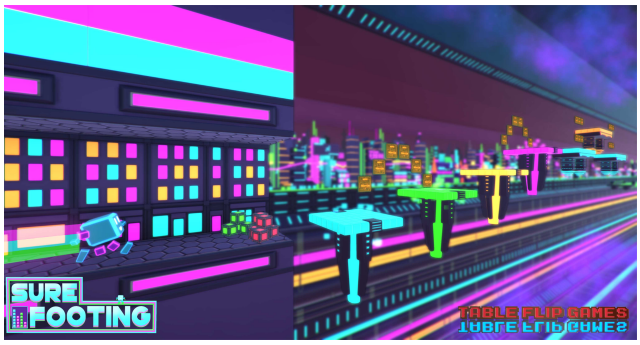


Figure 1. A screenshot of the *Sure Footing* video game, where the player, represented by a blue cube, must navigate a series of platforms and environmental hazards.

2 Sure Footing & Endless Runner Games

Sure Footing, shown in Figure 1 is an 'endless runner', where the player must navigate through a hazardous environment for as long as possible. Player's must traverse a collection of platforms and avoid obstacles placed upon them whilst evading an enemy that is following them throughout. Should the player fail a jump between platforms or be captured by their pursuer, the game will restart from the beginning of the current segment of play.

The endless runner genre is an effective platform for experimenting in PCG given that players are seldom aware of what is ahead of them. This allows for sudden change to the world that the player must adapt to. This is part of the novelty and charm that drove the popularity of seminal endless runner *Canabalt* [13] and subsequently titles such as *Flappy Bird* [10], and *Temple Run* [7].

Endless runners have a difficult balance to attain due to their unpredictable nature: should changes prove too sudden, players may subsequently lose interest. Ultimately, it is crucial that players feel the challenge of the game comes from their own ability to master game mechanics, rather than unfair design of the game. Equally players should be able to understand how to proceed through the game, irrespective of whether particular 'chunks' of level design have previously been seen in play. As discussed in Section 5, we place an emphasis on difficulty and progression in each participant's playthrough.

3 Related Work

Arguably the most established research in PCG for platforming games can be found in the *Mario AI Competition* which ran from 2009 to 2012 and has since been succeeded by the *Platformer AI Competition*[4]. The competition is dependent upon participants adopting a clone of the popular *Super Mario Bros.* [11] series. While originally intended to focus on gameplay, a level generation track was introduced in 2010 [18], with each entrant required to adopt player data from the an initial test level [14]. While the emphasis is to generate an intelligent and customised level generator, the focus of the competition is to find levels that judges deem 'interesting', rather than accurately reflect the designs of the *Super Mario Bros.* series. As such, the competition refrains from having judges compare PCG levels to original *Super Mario* levels built by human designers.

This work, among others in the AI field, focusses on search-based procedural generation. While this is an intelligent process that aims to create customised and unique content, there is seldom any emphasis on modelling the creative processes adopted by human designers in game development [2]. There have been notable exceptions to this, with one of the most prominent examples being the 'Sentient Sketchbook' project. As detailed in [8, 12, 9], this project carries a stronger emphasis on the use of PCG for human-designers as a tool; allowing for intelligent and useful content to be created in line with a designers expectations and habits.

The inspiration for this project is the *Tanagra* project detailed in [17]: a mixed-initiative design tool that aids in the creation of levels for 2D platformer games. The system allows for a designer to establish a timeline of 'beats': setting the pace of gameplay. The first phase of this work detailed in [15] is adopted in this project, where levels are built courtesy of rhythm groups which establish activities that take place.

4 System Design

In this section we give a brief overview of the PCG system adopted for this experiment. As we continue to discuss the design behind this system, we adhere to the taxonomy for PCG techniques defined in [19].

As noted in Section 3, our level generator adopts the rhythm approach discussed in [15]. The generator adopts a generate and test approach: creating and refining the rhythm of play followed by the geometry. The rhythm generator is comprised of a grammar representing player actions. This is encompassed by what is referred to as a *sprint*, a vector of game actions that lasts no longer than 60-90 seconds in-game. Actions are constrained to particular durations, denoted as *short* ($\leq$ 1 second), *normal* ($1-3$ seconds) or *long* ($3-5$ seconds). A full list of all available actions can be found in Table 1.

Once a full sprint vector is established, a critic will briefly evaluate to ensure a sense of flow is retained: the critic may swap pairs of activities, or add segments to give players a brief respite. This vector is passed into the geometry generator to create the level for play. This geometry generator is responsible not only for the selection of geometry but its subsequent placement within the game scene.

Each of the activities identified in Table 1 have one or more prefabricated pieces of geometry, hereby referred to as *prefabs*, that effectively represent the intended behaviour from the player. An example of this can be seen in Figure 2, which is one of the 'hopscotch' prefabs. The geometry generator places these items into the scene, aligning them such that a complete level is constructed. Once a sprint is completed, a 'rest' prefab is placed into the world. Typically this whole procedure is an online process and takes place during play. However, as discussed in Section 5, this process is made offline for the duration of this experiment.

[4] `http://www.platformersai.com/`

Figure 2. One of the prefab geometry pieces adopted by the geometry generator for the 'hopscotch' activity in Table 1.

5 Experiment Design

Our experiment was conducted during the *GameCity* festival in Nottingham, UK[5]. The focus of the experiment was to determine whether users could differentiate between levels crafted by a prototype PCG system, versus levels designed by one of the authors. In an effort to prepare for the festival, we exported six levels from the PCG system and stored them for later use. In addition to the PCG levels, six levels of equivalent length were crafted in the game engine by one of the authors.

While each level that was designed was unique, there are similarities that can be seen throughout. This is in part due to the prefabs discussed in Section 4 which were adopted in all level creation. In addition, given that the PCG system detailed in Section 4 was written by one author, with the other responsible for building the human levels, there is an argument to be made in that design habits of the authors have been injected, albeit rigidly, into the rhythm system. We return to these points in Section 6.1 and note the limitations they present as well as future steps for improvement.

Table 2. A breakdown of the percentage of participants who guessed either human or PCG-crafted level after each stage of completion. Followed by the success rates of those guesses at that particular stage.

Breakdown of Designer Guesses			
Level	Human Level	PCG-Level	Unsure
1	63.15%	23.7%	13.15%
2	50%	23.7%	26.3%
3	28.9%	42.1%	29%
Success Rates			
1	71.43%	25%	N/A
2	92.86%	12.5%	N/A
3	28.57%	37.5%	N/A

Each play-through of *Sure Footing* comprised of three 'levels'. With a minimum of one human and one PCG-crafted level per playthrough. The third and final level was selected at random from the

[5] The festival took place during 25th October to 1st November 2014: `http://www.gamecity.org`

Action	Duration	Description
Run	Short, Normal, Long	A flat section of terrain which the player must run across.
Jump	Short	A gap between platforms which may carry a variation in height, such that can either jump or fall depending upon the context.
Incline	Normal	A series of short platforms closely placed to one another or a ramp that gradually increases in height.
Decline	Normal	A series of short platforms closely placed to one another or a ramp that gradually decrease in height.
Hopscotch	Normal	A series of short platforms with one in the middle that is higher than the others, forcing the player to hop atop or over it.
Fall	Normal, Long	Two platforms with separated by a significant vertical drop. Players are expected to fall or jump down to the lower platform.
Spring	Normal, Long	A long platform with a spring attached to the end that will launch the player to a much higher platform.

Table 1. The collection of actions that can take place in a given 'sprint' of play.

Table 3. A table showing the frequency of reasons left by participants. Including the percentage of responses that left a given reason, followed by a breakdown with respect to whether they guessed a level was human or PCG-crafted.

	Reasons For Decision						
	Difficulty	**Pace**	**Variety**	**Length**	**Item Placement**	**Don't Know**	**Other**
All Responses	35.09%	36.84%	29.82%	14.91%	29.82%	7.89%	8.77%
No Vote	0.88%	2.63%	0.88%	0.88%	1.75%	4.39%	3.51%
Decided Human-Crafted Level							
All Guessed Human	18.42%	23.68%	14.04%	10.53%	19.30%	0.88%	0.88%
Correctly Guessed Human	10.53%	7.89%	7.02%	5.26%	7.89%	0.88%	0%
Decided PCG-Crafted Level							
All Guessed PCG	15.79%	10.53%	14.91%	3.51%	8.77%	2.63%	4.39%
Correctly Guessed PCG	7.89%	4.39%	7.02%	2.63%	3.51%	0.88%	1.75%

PCG and human-designed sets, thus certain users would be exposed to each type of content, with one type more-so than the other.

At the beginning of the play-through, players were briefed that they would play at minimum one of each kind of level and that their task was to discern between the two types. Upon completion, the next level was immediately loaded into the game for the player to complete. In the event that players found these levels too challenging, the option was given to allow for a level to be skipped. Players were given as many tries as was necessary to complete the set of three levels. Upon completion, participants we asked if they could identify PCG and human samples; identifying whether level difficulty, pace, variety of rhythm, length and placement of items informed their decision. In addition, players were also given the option to express in detail additional elements that helped cement their opinion. Only after this questionnaire was completed and the game saved performance data was it revealed to users whether a given level was indeed crafted by a human or PCG system.

6 Results & Discussion

The results from 45 participants can be seen in Table 2, showing the breakdown of guesses at each stage of the process. In addition, we provide a breakdown of the frequency that particular reasons were given and their success in Table 3.

There are a number of interesting results, noting not only gradual trends in guessing patterns, but also the reasons given in certain circumstances. Firstly, we note that players were more likely to correctly denote a level as being crafted by a human than by the PCG system. This is perhaps not surprising, given that players would assume by default that content was man-made if they found it fun or engaging. Another interesting element is that not only is the success rate for voting PCG-levels less accurate, but players are more likely to be left unsure in their decision. Despite the level of accuracy behind human guesses, players became less confident over time in voting for a human-designed level, arguably due to not discovering a significant difference in the content that was being shown during gameplay. We believe this could be a limitation of the current generator, given PCG levels may appear remarkably similar to human-crafted content.

If we look further at the feedback from Table 3, it is interesting to note that that pace and difficulty followed by variety and item placement are deemed the biggest factors for making a given decision. Despite this, in certain circumstances this proved to be an incorrect assertion. For example, less than half of all participants who blamed pace for a human-designed level were proven correct. Overall, there does not appear to be a real consensus from this study for understanding whether a level was human or PCG-crafted.

In addition to the provided reasons, there was written feedback that was provided through the 'Other' column of the questionnaire. This yield some equally interesting yet contradictory reasons for participants decisions. Specific written feedback from participants noted that levels were "very good" or "intriguing", with several participants noting "flow" as one of the reasons for human-crafted samples, only to be proven wrong. One participant went so far as to criticise the design of one level, noting that "no human would place" a particular

segment of prefabs together and was correct in that assertion.

We note that the average success rate was 25%, with 29% of participants failing to recognise *any* level successfully. Meanwhile 13% were capable of scoring 100% accuracy, identifying all PCG and human-crafted levels. It is arguably their written feedback or experience that proved most valuable. One participant was an independent game developer who could 'see' the patterns at play. Meanwhile another noted that item placement in particular showed an emphasis on human design. Given blocks and power-ups would be dropped in what they deemed "easier" segments of play. One fact that is not made visible in Table 2 is that in two cases, participants completely ignored the briefing given to them and stated that all levels were man-made. We would argue that part of this challenge in the eyes of players originates in the problem domain. As discussed in Section 2, the endless runners constrain the amount of change available to the designer. In addition, there are still numerous limitations in our system which we will now discuss.

6.1 Study Limitations

While this study does yield some interesting results, there are some notable limitations both with the study as well as the current generation system that we aim to address in future studies.

Firstly, the *Sure Footing* generator is a weak computationally creative system [1]: given it is largely reliant upon the pre-conceived notions of the human authors. Art assets are stored in pre-built chunks the system is reliant upon and the generator is not overly flexible. As such, any level built will carry heavy influences from human designers. More importantly, this generator was not particularly expressive, with only differing configurations of one base level 'template' that could be achieved. While the range of expression permitted to the generator must be improved, relating back to our previous point, future studies must also focus on measuring the full expressivity of the system. This notion, as discussed in [16], can help us identify the range of content the generator can establish and subsequently what impact this has on player perceptions. In addition, this would allow for assessment of whether current generators can build the same range of content as a human designer.

Furthermore, future studies would benefit from multiple generators for players to consider: ranging from humans, to intelligent procedural generations systems, with a variety of purely random generators in between. Lastly, future studies would benefit from testers being able to identify particular areas of gameplay where their suspicions of PCG or human-driven design are raised.

7 Conclusion

In this paper we highlighted a short study assessing players perceptions of procedurally generated versus human-crafted content for an endless-runner game. Players proved more successful in identifying human-crafted content than one by a PCG system, which in some respects is a positive step for the level generator; given that the majority of players could not find any patterns or trends that identified a given sample as procedurally generated. Given that this generator is influenced by a human creative process, it is perhaps to be expected that players find it harder to identify PCG-crafted levels. However, when we consider that the PCG system is rather rigid in this current version, it is surprising that the majority of users do not identify any real differences.

The feedback from this process has been adopted by the *Sure Footing* team who aim to build an improved level generator. Future work is focussed on building a more intelligent solution, in addition to addressing the issues raised in Section 6.1, such that a second study may be conducted over a longer period. This would allow for richer discussion of players perceptions of procedurally generated content as the generator becomes more expressive and their restrictions lifted.

ACKNOWLEDGEMENTS

The authors would like to thank the *Sure Footing* development team: Jonathan Boorman, Neall Dewsbury, Charlotte Sutherland, Matthew Syrett and James Tatum, for their assistance with this study.

REFERENCES

[1] Mohammad Majid al Rifaie and Mark Bishop, 'Weak and strong computational creativity', in *Computational Creativity Research: Towards Creative Machines*, 37–49, Springer, (2015).
[2] Nuno Barreto, Amílcar Cardoso, and Licínio Roque, 'Computational creativity in procedural content generation: A state of the art survey', in *Proceedings of the 2014 Conference of Science and Art of Video Games)*, (2014).
[3] Bell, I. and Braben, D. Elite. Acornsoft, 1984.
[4] Blizzard North. Diablo. Blizzard Entertainment, 2009.
[5] Gearbox Software. Borderlands. 2K Games, 2009.
[6] Erin J Hastings, Ratan K Guha, and Kenneth O Stanley, 'Evolving content in the galactic arms race video game', in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pp. 241–248. IEEE, (2009).
[7] Imangi Studios. Temple Run, 2011.
[8] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius, 'Sentient sketchbook: Computer-aided game level authoring.', in *Proceedings of the 8th International Conference on the Foundations of Digital Games*, pp. 213–220, (2013).
[9] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius, 'Designer modeling for sentient sketchbook', in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pp. 1–8. IEEE, (2014).
[10] Nguyen, D. Flappy Bird. .GEARS Studios, 2013.
[11] Nintendo EAD. Super Mario Bros. Nintendo, 1985.
[12] Mike Preuss, Antonios Liapis, and Julian Togelius, 'Searching for good and diverse game levels', in *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pp. 1–8. IEEE, (2014).
[13] Saltsman, A. Canabalt. Semi-Secret Software, 2009.
[14] Noor Shaker, Julian Togelius, Georgios N Yannakakis, Ben Weber, Tomoyuki Shimizu, Tomonori Hashiyama, Nathan Sorenson, Philippe Pasquier, Peter Mawhorter, Glen Takahashi, et al., 'The 2010 mario ai championship: Level generation track', *Computational Intelligence and AI in Games, IEEE Transactions on*, **3**(4), 332–347, (2011).
[15] Gillian Smith, Mike Treanor, Jim Whitehead, and Michael Mateas, 'Rhythm-based level generation for 2d platformers', in *Proceedings of the 4th International Conference on Foundations of Digital Games*, pp. 175–182. ACM, (2009).
[16] Gillian Smith and Jim Whitehead, 'Analyzing the expressive range of a level generator', in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, p. 4. ACM, (2010).
[17] Gillian Smith, Jim Whitehead, and Michael Mateas, 'Tanagra: A mixed-initiative level design tool', in *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pp. 209–216. ACM, (2010).
[18] Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N Yannakakis, 'The mario ai championship 2009-2012.', *AI Magazine*, **34**(3), 89–92, (2013).
[19] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne, 'Search-based procedural content generation: A taxonomy and survey', *Computational Intelligence and AI in Games, IEEE Transactions on*, **3**(3), 172–186, (2011).
[20] Geogios N Yannakakis, 'Game ai revisited', in *Proceedings of the 9th conference on Computing Frontiers*, pp. 285–292. ACM, (2012).
[21] Yu, Derek. Spelunky. Mossmouth, 2009.

Hybrid Procedural Content Generation: A Proposal

Michael Cook and Simon Colton[1]

Abstract. Procedural content generation in games tends to target content that is abstract, dry and devoid of connection with the game's meaning. This paper proposes merging user-driven content generation approaches with procedural content generation to create a new paradigm which we call *Hybrid Procedural Content Generation*. By replacing aspects of existing procedural generation techniques with humans, we can give rise to new kinds of game experiences.

1 Introduction

Procedural content generation and user-generated content (PCG and UGC respectively) are two concepts which are familiar to anyone who has played or made games in the past decade. The idea that content for a game can be created after it has shipped enables many new kinds of game experience, as well as engaging players in new kinds of activities, including creative involvement in the game. They also provide interesting research platforms to ask new questions and build intelligent systems to help shape these new ideas about games.

In this paper we introduce the concept of *Hybrid* Procedural Content Generation (HPCG), a fusion of user-generated and procedurally-generated content that similarly offers new kinds of game design and also new opportunities for artificial intelligence in games. By incorporating players into procedural content generation systems we can produce hybrid systems that are much stronger than standard procedural or user-driven generative approaches.

We illustrate the concept of HPCG by giving three examples of prototype games which incorporate some kind of HPCG system into their game design. *Murder* is an assassination game set in a Cluedo-esque mansion at a dinner party, in which the player must perform several narrative actions and then kill another character at the party. *Mystery* is a Poirot-style detective game in which the player must solve a murder using deduction and exploration. *The Book Of A Thousand Tales* is a roleplaying game in which the player leads a band of heroes through a branching narrative.

The remainder of the paper is organised as follows: in *Background* we discuss both PCG and UGC and their relative weaknesses. In *Hybrid PCG* we briefly introduce the concept of HPCG, its motivating factors and how we see it being used within games. We then describe two simple game designs that comprise a HPCG system. Finally in *Opportunities for Computational Intelligence* we talk about the longer-term impact of such approaches and the potential for new research directions HPCG could give rise to. We then sum up our proposal in *Conclusions*

2 Background

According to [6], most PCG systems can be categorised as either *constructive* or *generate-and-test* systems. In the former, content is gradually built up out of successive passes at generation, and each layer of generation is "*guaranteed to never produce broken content*" [6]. Spelunky [2] is a good example of this style of generation, where dungeon levels are built out of several different layers of content which are hand-crafted to some extent to guard against failure [7]. Generate-and-test approaches employ a generative step that produces content, and then an evaluative step which assess what was generated and either triggers further generation/alteration (such as an evolutionary system which will run many times to evolve a result, as in [2]) or simply reject the generated content and begin again from scratch. *Dwarf Fortress* [3] employs a generate-and-test approach during its world generation.

PCG has been applied very effectively to many kinds of content generation, particularly level design [3] and general game content such as item generation in roleplaying games. However, many types of content are hard to generate using either of the above approaches. In particular, content which requires an understanding of context of the real world is hard to generate, such as game narratives or replicating human-like qualities in NPC actions such as deception or fallibility. These dynamic kinds of content rely on an understanding of the real-world, from cultural knowledge (like understanding symbolism when constructing a narrative) to common-sense reasoning (when deciding how a character should react to a particular situation, for instance). As a result, most content generation focuses on abstract data that is detached from the game's setting and theme (the levels in Spelunky are simply arrays of numbers, for instance – the system does not need to understand what a cave looks like or what an explorer does).

User-generated content (UGC) is also a common feature in many modern games. Allowing the player to create content for a game both increases the amount of content available at no extra cost to the developer, and gives players a sense of engagement and investment in the game world by allowing them to contribute to it. *Spore* [4] is a prominent example of user-generated content – players designed animal species for inclusion in their games using an assortment of body parts and customisations. These animal species propagated not only throughout the player's world but also to their friends' worlds via cloud sharing online.

UGC is one of the biggest recent trends in the mainstream industry thanks to the enormous success of *Minecraft* [5], which merged user-generated content with the core mechanics of the game. In Minecraft, generating content is how one plays the game: building structures, artworks and shaping the world as the player sees fit. UGC has drawbacks, however. In the case of generators like Spore's, which present themselves as tasks outside of gameplay, the user is consciously

[1] Computational Creativity Group, Goldsmiths, University of London

[2] Mossmouth Games, 2009
[3] 2006, Bay Twelve Games
[4] Maxis, 2008
[5] Mojang, 2011

aware that they are generating content. As a result they are thinking about how the content will be perceived by others, which has an impact on how and what they create. This can be seen somewhat in the comedic nature of many of Spore's creatures – players know they are creating things which will amuse or confuse other people. While this may be seen as a positive for some tasks (in Spore's case the objective is specific content generation) because the player is consciously considering the design of their content, for other tasks it may be less good – particularly those that take place in a fictional context. For example, in Minecraft it is possible to construct floating houses, which may break the suspension of disbelief for other players. It is preferable here that all players construct buildings in a similar way, so that they can maintain the narrative fiction for everyone equally.

The second drawback is that players tend not to be designers, and UGC systems rarely have any kind of feedback mechanism or assistive aspect to them. Content is either used wholesale or not used at all, and frequently even this decision is made by players rather than an intelligent software system. Creatures in Spore are uploaded and shared online, structures in a Minecraft world exist for all players in that world and can't be edited or changed by the game. UGC is all-or-nothing and thus lives or dies on the skill and appreciation of the players using these systems. In some cases this can be worked around – ratings systems in games such as LittleBigPlanet [6] simply filter the best creations and downplay the rest. In this case, however, UGC simply becomes a means by which to discover talented people and get them to produce content, rather than allowing everyone to contribute equally.

3 Hybrid PCG

We propose that PCG and UGC approaches can be combined in a single approach that solves some of the problems mentioned in the previous section while opening up new challenges and research questions for computational intelligence research to tackle. We call this combined approach *Hybrid PCG* because it synthesises software-driven content generation with player activity. The underlying premise is to replace generative systems or parts of systems with playable games, resulting in new ways of generating, evaluating and filtering content, not just for single games by potentially for many different games at once.

To illustrate this approach, we will describe in this section two in-development game prototypes, *Murder* and *Mystery*, which utilise a HPCG approach to generate a large corpus of content and filter it. These games not only supply content to one another: by generating content that is transferred between games, they also produce a corpus that can be used by other games or intelligent systems. After describing the games we will discuss the new affordances such a setup offers and then lead into a discussion of the opportunities for computational intelligence they represent.

3.1 Illustrative Example - Murder/Mystery

In *Murder* the player takes on the role of a character attending a dinner party at a mansion, as either a guest, a family member, or an employee of the host. Like most of the people present they have a motive to kill the host, and must do so at some point during the evening. In addition, they must also complete one or more objectives relating to their motive (such as confronting the host in an argument, or breaking into a room and stealing something). The game operates in a 'sandbox' style, where the player can explore the house freely and approach their objectives in many different ways. However, the game simulates player action carefully and records things like fingerprints left on surfaces, sightings by other people in the house, and so on.

At the end of the game, once their tasks are completed, the player can choose to 'discover' the body themselves or wait for it to be discovered by someone else. They are then asked to provide an account of their whereabouts for the evening by being shown their actual movements and then editing them to change their version of events – for example, by claiming they were never in a particular room at a certain time, and so on. The game then assesses how quietly and quickly they completed the game, as well as how well their alibi compares to the evidence they left behind, and gives them a rating.

In *Mystery* the player takes on the role of a detective tasked with solving a murder at a dinner party. They play a point-and-click adventure in which they can examine the alibis and backgrounds of the characters present, ask for accounts of events, and walk around the house looking for clues or analysing parts of the crime scene. The case files are built from case descriptions produced by *Murder*, potentially converted using an automated system that can filter the case to make it harder or easier (by making certain evidence more or less conclusive or adjusting the memories of other characters, for example) or simply presented to players unaltered – we discuss this further in section 4.

There is a time and resource limit on solving a case - if the player takes too long or uses up all of their investigative resources (such as sending objects for fingerprinting) the case remains unsolved. Whatever the result, the case file data gets sent back to a central server which both affects the value of a case (repeatedly unsolved cases rise in value to detectives) and the reputation of the player who created the case file in *Murder*.

3.2 HPCG in Murder/Mystery

Both *Murder* and *Mystery* are standalone games that are effectively separate from one another. If the data format for case files is open, anyone could design a game which retrieved case files produced by *Murder* players and use them in their game. Similarly, several games might produce case files with the right format that could be used by *Mystery* as game content for the player to investigate and solve. The games are not intrinsically linked except through the exchange of information about the case files and whether or not they are solvable by players.

In the language of PCG, players of *Murder* are acting as a generator of case files, in the first step of a *generate-and-test* system. There are two important consequences of this. Firstly, unlike UGC approaches, the players are *engaged in a game* while generating content, pursuing objectives in whatever way they see fit. We argue that this leads to more natural behaviour by players and therefore a more human-like kind of content generated than if players had been asked to manually design case files as authors. Secondly, the content being generated is complex - it involves creative problem-solving and asks the player to respond to social situations (such as confronting someone about a personal relationship, or making small-talk at a dinner party). Such content is difficult to generate automatically without a lot of involvement from a designer, and even with such involvement the content is likely to be lacking in variety over a long period of play. By using players to generate it, we make this difficult generative task easier.

To continue the PCG metaphor, players of *Mystery* act as evaluators of the content generated by *Murder* players. Let us assume that

[6] Media Molecule, 2008

Mystery either does not edit the case files at all, or at most edits them in order to ensure that they can be solved by some process of deduction (by ensuring that at least one piece of incriminating evidence exists, for instance). Players solving, or attempting to solve, cases are providing data about how easy a case is to solve. The routes players took, the order in which they examined evidence or questioned people, and their ultimate success at solving the murder can all be recorded as additional metadata attached to the original case file. In the same way that people can be used to generate content that requires complex understanding of the real world, people can also be used to provide evaluation metrics that would be difficult to encode into a system by hand (and too subjective to source from a single designer).

3.3 Desirable Properties of HPCG Scenarios

While this remains a preliminary proposal for HPCG, and the idea still needs much exploration, we posit that certain game designs or scenarios are better suited for the application of HPCG. We discuss them briefly here, and hope to clarify this in future work after more experimentation and prototype development.

3.3.1 Asynchronous Activity

The most important property for employing HPCG is that the games involved deal with asynchronous activity. Murder/Mystery work well because the two game phases are chronologically non-overlapping: one player commits a crime, then after they are finished the second player can arrive and solve it. This means that no player is left waiting for action to be completed in real-time, which could affect the experience of either player and slow down gameplay, and it also means that any PCG systems have complete information from the other game or games when they begin generating content.

3.3.2 Well-Defined And Decoupled Interfaces

Keeping the interfaces between games as simple as possible is a good feature if the designer intends for other systems to feed data into the HPCG besides their own. For Murder/Mystery we noted that in theory it is possible for other games to generate crimes for Mystery to solve, or to design other games which use Murder case files as input content. In order to enable this, it's important that the interfaces between the games are very well-defined and public so that other developers can take advantage of them. Making sure the games can export data as well (such as putting Murder's case files in external text documents) also makes this easier.

3.3.3 Guided Player Activity

Depending on the kind of content being generated or the roles the players are taking on in the larger HPCG system, it may be desirable for the gameplay to be very directed or guided. The reason for this is that the HPCG system is making assumptions that the data they collect represents a certain kind of behaviour from the player - for example, committing a crime, not wanting to leave evidence behind, acting in order to blend in. It's important to be able to encourage and motivate the player to work towards certain objectives so that these assumptions carry through into the data they generate, and can then be relied upon to generate good quality content in other areas of the HPCG system. If a player begins acting differently, or isn't sufficiently motivated to play properly, the HPCG system will still proceed with the data and this can generate undesirable outcomes in other games.

4 Opportunities for Computational Intelligence

On the surface, HPCG appears to replace software-driven PCG systems with players that perform the same tasks, therefore resulting in systems that involve *less* computational intelligence, rather than more. However, HPCG systems open up new research questions that demand answers, and also create opportunities to build even more complex generative software. In this section we discuss several possibilities in brief.

4.1 Learning From Human Generators

One possible outcome from HPCG systems is that they eventually transition back into being PCG systems which use a player's in-game activity as a source of training data. In [4] Orkin and Roy describe *The Restaurant Game* (TRG), an experiment in which participants played through an interactive scenario in pairs and their behaviour was then recorded and later analysed using machine learning to build behaviour models of characters in those situations. TRG suffers from some of the same problems that we mentioned in the context of UGC earlier in the sense that players are aware they are generating content as they play. Nevertheless, the authors' argument is that automatic content generation (in this case speech and behaviour patterns) can be mined from large-scale data corpora [5].

By employing HPCG to tackle complex generative tasks, like the generation of creative behaviour in *Murder*, such systems produce special cases of the kinds of corpora Orkin and Roy present with The Restaurant Game. They are special cases in the sense that they are obtained through observing players at a time when their primary concern is *completing* a game rather than performing for another observer (whether that observer is a human or a data-mining program). The player is not participating in an experiment, nor is their ultimate goal to provide good data. Instead, they are focused on achieving objectives and are immersed in a ludic task. As a result, we argue that their behaviour is more natural and as a result more valuable, resulting in useful corpora of data that can be mined, as with TRG, to obtain behaviour. In the case of games such as *Murder*, the available information is particularly valuable because the player is providing information that an ordinary PCG system would not have access to - such as solving problems in creative or innovative ways, as well as failing at tasks in a natural, humanlike way.

4.2 The Computer As Curator

The game *Murder* can be seen as a generator of content for the game *Mystery*, but raw generated case files from the game may not be interesting, fun to solve or, indeed, solvable at all. Building *Murder* as a HPCG system provides us with a wealth of generated murder cases for players to solve, but it doesn't guarantee their quality or difficulty level. If a player plays a perfect game, it will be fairly unsatisfying for players of *Mystery* to repeatedly fail to solve. Similarly, the player may make an obvious mistake that renders a case trivial. This poses an interesting problem: how can software curate, tweak and improve raw HPCG output to ensure consistently entertaining content for another player?

There are many factors to tweak in a case file produced by *Murder* – both the actions of the players and the other characters, the evidence left behind, the ordering of events. Altering this information requires

an understanding of how people's behaviour is interpreted by others, to assess whether a change will make a case easier or harder to solve for a player detective. HPCG systems leverage human players to solve creative, complex problems that are hard to solve using generative software alone. It follows, therefore, that curating and improving the results of a HPCG problem requires an understanding of how these players reason about problems and act in certain situations. The task of curating complex creative content sourced from humans may have parallels with the problem of curating and evaluating in Computational Creativity [1].

Recall that in section 2 we discussed the problems with existing UGC and PCG paradigms. One problem with UGC approaches is that players are not designers, and expecting them to be able to produce quality game content, either knowingly or not, is unreasonable and often results in a large volume of low-quality content that no-one wants to use. HPCG offers an opportunity to leverage the output of users and improve it using computational intelligence, obtaining content that has its foundations in the creativity of real players, but has been curated and refined by software to be of higher quality.

5 Acknowledgements

The authors would like to thank the reviewers who provided helpful feedback which improved this paper. This work was sponsored in part by EPSRC grant EP/L00206X.

6 Conclusions

In this paper we briefly outlined a proposal for *Hybrid Procedural Content Generation* or HPCG, a synthesis of user-generated content and procedural content generation where subsystems in a content generation pipeline are replaced with players playing games achieving similar tasks. We illustrated the idea with two connected games – *Murder* and *Mystery* – in which players of the former acted as a generator of content which was then filtered and evaluated by players of the latter. We discussed what new avenues of research such an approach might offer and how it solves some of the problems that procedural content generation and user-generated content can have.

This paper is an early proposal for such games and systems to be designed, but we hope that it will spark discussion and potentially lead to interesting new kinds of games and intelligent software. We believe that working with game developers may be of essence here, to leverage good game design alongside new kinds of computational intelligence. Collaboration is difficult, but we believe this is a promising avenue to explore.

REFERENCES

[1] Simon Colton, Michael Cook, Rose Hepworth, and Alison Pease, 'On acid drops and teardrops: Observer issues in computational creativity', in *Proceedings of the 7th AISB Symposium on Computing and Philosophy*, (2014).

[2] Erin J Hastings, Ratan K Guha, and Kenneth O Stanley, 'Evolving content in the galactic arms race video game', in *IEEE Symposium on Computational Intelligence and Games*, (2009).

[3] Britton Horn, Steve Dahlskog, Noor Shaker, Gillian Smith, and Julian Togelius, 'A comparative evaluation of procedural level generators in the mario ai framework', (2014).

[4] Jeff Orkin and Deb Roy, 'The restaurant game: Learning social behavior and language from thousands of players online', *Journal of Game Development*, (2007).

[5] Jeff Orkin and Deb K. Roy, 'Understanding speech in interactive narratives with crowdsourced data.', in *AIIDE*. The AAAI Press, (2012).

[6] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne, 'Search-based procedural content generation: A taxonomy and survey', *IEEE Transactions on Computational Intelligence and AI in Games*, (2011).

[7] Derek Yu. The full spelunky on spelunky, 2012.

Revealing Social Identity Phenomena in Videogames with Archetypal Analysis

Chong-U Lim[1] and **D. Fox Harrell**[2]

Abstract. In this paper, we present a novel approach toward revealing social identity phenomena in videogames using archetypal analysis (AA). Conventionally used as a dimensionality reduction technique for multivariate data, we demonstrate how AA can reveal social phenomena and inequity such as gender/race-related steretoyping and marginalization in videogame designs. We analyze characters and default attribute distributions of two critically acclaimed and commercially successful videogames (*The Elder Scrolls IV: Oblivion* and *Ultima IV*) together with 190 characters created by players in a user-study using a third system of our own design. We show that AA can computationally 1) reveal implicit categorization of characters in videogames (e.g., base player roles and hybrid roles), 2) model real world racial stereotypes and stigma using character attributes (e.g., physically dominant attributes for *Oblivion*'s ostensibly African-American "Redguard" race) and 3) model gender marginalization and bias (e.g., males characterized as more archetypal representations of each race than females across attributes.) We highlight how AA is an effective approach for computationally modeling identity representations and how it provides a systematic way for the critical assessment of social identity phenomena in videogames.

1 INTRODUCTION

Videogames often construct virtual environments and worlds that are populated with virtual characters. Both these worlds and characters may be represented in a multitude of ways. Graphical 2-dimensional (2D) or 3-dimensional (3D) assets grant visual appearances, textual descriptions provide intriguing narrative, backstories, and characteristics, while numerical statistical attributes provide quantifiable measurements defining character skills and capabilities for a variety of interactions, from dealing damage against a mighty adversary, to charming a non-playable character into handing over an elusive item.

Though often considered to be purely virtual, these representations are in fact blended real/virtual identities that are both affected by, and capable of influencing, aspects of real world identities. Even in the case of a fairly rudimentary character such as Pac-Man, in action we have a blend of a real users control with a 2D animated sprite. Recent studies have shown how representations of race and gender within videogames have deep social implications [8]. In the commercially successful and critically acclaimed role-playing game (RPG) *The Elder Scrolls IV: Oblivion*, some character designs "implement and amplify many disempowering social identity constructions" [9]. "Females of some races are more intelligent than their male counterparts and individuals of the ostensibly French 'race' (Bretons) are twenty points more intelligent than their ostensibly Norwegian (Nords) counterparts, regardless of gender" [9]. It highlights the *importance of the underlying implementations and data structures used to construct these representations*. If developed without due consideration, undesirable social implications related to identity such as marginalization and stereotyping may be further perpetuated. Research has shown that peoples' performances are impacted by stereotypes [20] and behaviors in the physical worlds are altered by their avatar use [22].

However, it is important to recognize that these issues are not simply technical in nature. We adopt a *critical computing* [9] approach, using algorithmic processing and data structuring for critically assessing and providing commentary about the real world and related social phenomena. In this paper, we demonstrate an how archetypal analysis can be used as an Artificial Intelligence (AI) tool for such critical assessments of computational identity-related social phenomena in two commercial videogames, as well as a character creation system of our own design. The upshot is that we found that AA is a robust method for computationally modeling underlying social identity phenomena grounded in cognitive science. We use AA to model social phenomena within games, such as male characters being favored over female characters based on statistical attribute distributions or in-game races having real world stereotypes imparted upon them (e.g., ostensibly African-American "Redguard" characters in *Oblivion* given better physical but lower mental stats.) To the best of our knowledge, this application of cognitive science (apart from some notable exceptions like Santa Ana's work on discrimination and racism [17] and Lakoff's work on political affiliations [12]) and AI has not often been applied to analyze nuances of identity. Most computational systems, like videogames, are built with classical models and categories explicitly built into software. Hence, they provide a good venue to critically assess such cognitively grounded AI approaches to studying digital identity.

2 BACKGROUND

In this section, we present the theoretical framework for our work and provide an overview of the videogames used for our analysis.

2.1 Cognitive Categorization and the Sociology of Classification

Our view of categorization is based upon cognitive scientist George Lakoff's work in cognitive categorization [11] termed *category gradience* and psychologist Eleanor Rosch's *protoypes* [16]. As opposed to outmoded classical or "folk" approaches, which character-

[1] Computer Science & Artificial Intelligence Laboratory, Massachusetts Institute of Technology, USA, email: culim@mit.edu

[2] Computer Science & Artificial Intelligence Laboratory, Comparative Media Studies Program, Massachusetts Institute of Technology, USA, email: fox.harrell@mit.edu

ize category membership to be defined by a fixed set of characteristics, centrality gradience recognizes that some members are typically deemed "better examples" of a category than others. Extending upon this, we use the following concepts from the sociology of classification by Geoffrey Bowker and Susan Leigh Star [3] for describing categorization-related social phenomena. **Membership** is the experience of encountering and interacting with objects within certain social groups, and increasingly engaging in naturalized relationships with them. **Naturalization** is the deepening familiarity of such interactions within a given social group. **Marginalization** is a result of enforced naturalization occurring where members of a marginal category exist outside of social groups, or are less prototypical members of communities. It is also characterized by exclusion from a social group or an individual having *multiple memberships* and often refers to *exclusion or difference from normative behaviors* (Stigma) [7, 9]. **Markedness** indicates that, unlike normative categories, marginal categories are demarcated visually and linguistically.

To reconcile these concepts with the systems in this paper, we use a cognitively-grounded model for critically assessing computing systems for social analysis [9]. It suggests that category gradience enables semantic relations to be structured or ranked according to how constitutive they are of the category. Naturalization may be assessed by *user actions and attributes* that reinforce category semantics, resulting in a higher degree of membership. Marginalization may be implemented through enabling *degrees of membership* and represented as being *further away from the prototypes*. Normative groups that are often unnamed and unmarked may possess *implicitly assumed* normative privileges that may be identified and modeled. This theoretical framework forms the basis for using archetypal analysis as an approach for social analysis and empowerment through critically assessing the statistical attributes of characters within videogames for revealing implicitly-derived social phenomena such as gender-related marginalization and stereotyping.

2.2 Archetypal Analysis

Archetypal Analysis (AA), introduced by Cutler and Breiman [5], is a method for reducing the dimensionality of multivariate data [1]. Given a set of multivariate data points, the aim of AA is to be able to represent each data point as a *convex combination* of a set of key data points called **archetypes**. For example, applying AA on a dataset of basketball players and their statistics [6] computationally revealed and represented the following four archetypes – "benchwarmer," "rebounder," "three-point shooter," and "offensive." Every individual player in the entire data set could then be represented as a hybrid mixture of these archetypes [18]. Formally, given a data set of points $\{x_1, x_2, ..., x_n\}$, AA seeks to find a set of archetypes $\{z_1, z_2, ..., z_k\}$, where $z_j = \sum_{i=1}^{n} \beta_{ij} x_i$, and enables each data point x_i to be represented in terms of the k archetypes as $x_i = \sum_{j=1}^{k} \alpha_{ji} z_j$. The objective function minimizes the residual sum of squares $RSS = ||x_i - \sum_{j=1}^{k} \beta_{ij} z_j||^2$ under the constraints that the weights $\sum \beta_{ij} = 1$ $\beta_{ij} \geq 0$ and coefficients $\sum \alpha ji = 1$ $\alpha_{ji} \geq 0$. These ensure the archetypes *meaningfully resemble* and are *convex mixtures* of the data. These archetypes are located on the data convex hull [5] and are represented are combinations of individual points, making them more easily interpretable [1], unlike other dimensionality reduction techniques like Principal Component Analysis [10] and Non-negative Matrix Factorization [13]. AA has been shown to be effective compared to other techniques for various AI-related problems. Compared to other recommender models (nearest neighbor, two popularity, random baseline) AA provided the highest recall rates for archetypal recommender systems [19] in games, demonstrating robustness for finding relevant recommendations. Here, AA is an appropriate approach given our aim to computationally model individuals that are more "prototypical" than others (archetypes) and being able to measure the "centrality gradience" of each individual with respect to these archetypes. As described in Section 2.1, we believe that such models would enable us to begin critically assess social phenomena such as marginalization and stereotyping computationally.

2.3 Overview of Videogames

We provide an overview of the two commercially successful videogames used in this paper. Both are important open-world single-player RPGs with strong customization. *Ultima IV: Quest of the Avatar* is arguably the most influential game on the open world RPG genre and *The Elder Scrolls IV: Oblivion* is a stunning recent success with a strong customization system and diversity. Even in excellent games, there is the potential for implicit stereotypes and inequity. Our observations are meant to be useful for improvement in this regard.

The Elder Scrolls IV: Oblivion is the fourth installment of the popular *Elder Scrolls* computer role-playing game series, developed by *Bethesda*. In the lore of the game designed by game designers there are several races, each with their own fictional background stories and histories. Three basic player roles exist in the game – "Fighter", "Mage" and "Thief" [15], which are derived from common roles across most RPGs stemming from old table-top RPGs like Dungeons and Dragons. Each race is associated with the three basic roles in varying degrees (hybrid roles), which compliment the game's lore about its people and races. Players choose to play as one of the ten different races available, customizing characters over 7 basic attributes (strength, intelligence, willpower, agility, speed, endurance, and personality,) together with their height and weight.

Ultima IV: Quest of the Avatar is the fourth installment of the *Ultima* series of role-playing games, and the first in the "Age of Enlightenment" trilogy, *Ultima IV* was first released in 1985 by *Origin Systems*. The player is assigned one of eight classes to play and does not directly choose or assign values to attributes. Instead, the user is posed several questions embedded within the games narrative at the beginning, resulting in the players ranking of eight **virtues** in the game based on the game's three **principles** of Truth, Love, and Courage. There are seven companions that the player may choose to form a party with. Each character has a particular class, each associated with a virtue, and possesses seven numerical attributes (strength, dexterity, intelligence, hit points (HP), magic points (MP), level, and experience,) an armor type, a weapon type, and their gender.

3 APPROACH

1. Analyzing existing systems for designer-centered phenomena. In order to assess the kinds of categorization and social identity phenomena that arise as a result of designer choices (top-down), we applied archetypal analysis to the statistical attribute allocation for new characters in both *Oblivion* and *Ultima IV*. For *Oblivion*, the variables included the races, gender, and eight attributes. For *Ultima IV*, the variables included the character classes and seven attributes.

2. Analyzing emergent phenomena with a system of our own creation. For the purpose of assessing the kinds of categorization and social phenomena that may be *implicitly-derived* from players (bottom-up), we conducted a user-study with 190 players where they constructed avatars in an avatar constructor of our own creation. Players customized both their character's visual appearance and statistical attributes values of six commonly used videogame attributes (strength, endurance, dexterity, intelligence, charisma, and wisdom) on a 7-point Likert scale with a total of 27 allocatable points. The avatar constructor used our avatar game data-mining system called *AIRvatar* [14], that stores each created avatar, the statistical attribute allocations, and textual descriptions made by the players.

3. Determining the number of archetypes During AA, we varied the number of archetypes k in the range $1 \leq k \leq 10$. We adopt the convention of the Cattell scree test [4] for using the residual sum-of-squares (RSS) to determine the optimal number of archetypes by picking the value of k matching the first point of the "elbow" of a screeplot with corresponding to the biggest change in RSS. This balances the trade off between minimizing RSS and overfitting.

4 RESULTS

We present results describing the archetypes obtained from analyzing the statistical attributes of each system using archetypal analysis.

4.1 Oblivion

In *Oblivion* we found $k = 3$ to be optimal. Both Archetypes 2 and 3 were pure archetypes ($\alpha_j = 1$). The ternary plot in Figure 2(a) of the Appendix shows a visualization of the α coefficients of these archetypes. We also observed the following characteristics:

- Archetype 1 had the highest "Strength" and "Endurance", but lowest "Intelligence". Archetype 1 had the biggest "Size".
- Archetype 2 was relatively balanced across the attributes, with highest "Willpower" and "Personality".
- Archetype 3 had highest "Intelligence", "Agility" and "Speed", but lowest "Willpower". Archetype 3 had a relatively small "Size".

4.2 Ultima IV

In *Ultima IV*, we found $k = 3$ to be optimal. All three were pure archetypes. The ternary plot in Figure 2(b) of the Appendix visualizes the α coefficients of these archetypes. We also observed that :

- Archetype 1 had the lowest values across all attributes.
- Archetype 2 had the highest values across all attributes, except for "Intelligence" and "Magic Points".
- Archetype 3 had the highest "Intelligence" and "Magic Points".

4.3 AIRvatar

For characters created using *AIRvatar*, we found $k = 3$ to be optimal. The bar plot in Figure 1 shows the three archetypes obtained, represented with the same six RPG attributes. We observed the following:

- Archetype 1 had highest "Intelligence" and "Wisdom" attributes, but lowest "Strength" and "Endurance".
- Archetype 2 had the highest "Strength", "Endurance", and "Dexterity" attributes, but the lowest "Wisdom".
- Archetype 3 had the highest "Charm" but lowest "Dexterity".

5 FINDINGS

5.1 Classes, Roles, and Category Gradience

In *Oblivion*, we found that each **archetype corresponded with the primary roles of the game**, namely "Fighter" (Archetype 1), "Mage" (Archetype 2), and "Thief" (Archetype 3). We used descriptions in the *Unofficial Elder Scrolls Pages* [15], to help identify these roles from obtained archetypes. "Fighters" '*rely heavily upon melee combat to attack enemies, expect to receive a lot of damage rely upon high health...*', "Mages" '*avoid combat, use decoys, and rely upon magical attacks.*' *Magicka*, used for spells and magic, is affected by both "Intelligence" (Capacity) and "Willpower" (Regeneration). A "Thief" '*relies upon sneak attacks and avoids face-to-face combat, uses a poisoned bow as a primary means of attack,*' corresponding to the high "Speed" and "Dexterity" (Bow Accuracy) attributes.

Likewise, in *Ultima IV*, we observed from our results that each **archetype corresponded with characters of primary roles in the game**. Katrina the Shephard is Archetype 1 as her description in the *Unofficial Ultima IV Strategy Wiki* [21] states "...she has the lowest attributes, no magic power and a limited selection of equipment; start the game with her if you're looking for a challenge". Archetype 2 corresponds to "Iolo the Bard", who has the highest "Dexterity" described as "probably the most important attribute because it rules the probability of hitting enemies, avoiding traps and dodging enemies."Archetype 3 corresponds to "Mariah the Mage", with highest "Intelligence" (determines maximum "Magic Points".)

For characters created by players in *AIRvatar*, we observed from our results that the archetypes corresponded with **traditional RPG roles used in games**, which we term "Intelligent/Wise-Cleric" (Archetype 1), "Physical-Fighter," (Archetype 2) and "Charming-Thief" (Archetype 3). We the descriptions of traditional *Dungeons and Dragons* classes to match against the highest-scoring attributes of each archetype to identify these roles. Magic using "Mages/Clerics" focus on magic, and generally have lower strength. "Fighters" are usually strong in attack and defense, but usually have little to no magic capabilities, while "Thieves" often are in-between, but have high capabilities in social skills, cunning and stealth.

We validate this based on the free-text responses that players provided for their avatars, in addition to customizing their characters. We provide selected responses from the highest scoring players for each archetype to highlight this behavior:

1. **Archetype 1 (Intelligent/Wise-Cleric):** "*Stephanie is a wandering wolf **mage**. She was born to a poor family, but her parents did their best to support her academic ventures. She studied hard and was eventually admitted to the nation's most prestigious **arcane** academy.*"
2. (**Archetype 2 (Physical-Fighter)**: "*Gerald ... is a veteran of many wars in Elibca, serving as a **knight** and later as a **general** for the kingdom of Calmenia ... living the remainder of his life in modesty as he nurses **old scars**.*" & *"Saya is an independent **Mercenary** selling her contract not to the highest bidder, but to those she deems in the most need of her services. Secretly, she dreams of becoming a Paladin some day but believes that she has **far too candor in her speech** and methodology to fit in ... "*
3. **Archetype 3 (Charming-Thief):** "*She is **friendly** and ready to reach out to the other villages. She **prefers talking to fighting**, but is tough enough to fight if she needs to.* "

These results shows that AA can effectively model implicit categories, such as intended player roles and relationships between attributes from analyzing raw statistical attribute data. For example, in

both *Oblivion* and *AIRvatar*, "Strength" and "Intelligence" attributes are always maximized on different archetypes, while "Strength" and "Endurance" were be maximized on archetypes together. Additionally, with archetypes corresponding to prototypical player roles, we observed that **each individuals could meaningful represented as mixtures of these archetypes**, corresponding to hybrid roles intended by most designers.

5.2 Revealing Stereotypes, Marginalization, and Inequity

5.2.1 Race-related Stereotyping

From the archetypal analysis results on characters in *Oblivion*, we were able to observe that **some of the in-game races were deemed more "prototypical" with respect to player roles and that we could observe how these in-game races reflected real world stereotypes**. To visualize this, we make use of the ternary plot of results shown in Figure 2(a) of the Appendix. This is best visualized using a ternary plot, as shown in Figure 2(a) of the Appendix. We observe that the ostensibly Norwegian "Nords" are viewed as archetypal Fighters, the ostensibly French "Bretons" as archetypal Mages, and ostensibly South American "Bosmers" as archetypal Thieves. Additionally, the obstensibly African-American "Redguards" stereotypically close to the physical-fighter archetype with no characteristics of the intelligence-mage archetype, though exhibiting some stealth-thief archetype characteristics. This corresponds with findings by Harrell in his assessment of racial stereotypes in *Oblivion* [9].

5.2.2 Gender-related Inequity & Marginalized Characters

In *Oblivion* we also note that **for each race, male characters are consistently deemed more prototypical than their female counterparts than their female counterparts.** This is illustrated in Figure 2(a), where for each archetype, the male characters are always at least as close, or closer to the archetypes, than their counterpart female characters. Insight into the significance of characters being closer to the centers (i.e., further away from archetypes) is highlighted in the design choices made in *Ultima IV*, wherein the NES version of *Ultima IV*, "Julia" was replaced by a male character "Julius", with no modification to the stats. From the ternary plot in Figure 2(b) of the Appendix, it can be seen that "Julia" is the character with negligible "Intelligence" and "MP" attributes and located between the overall lowest and highest-performing archetypes, possessing multiple memberships. This computational modeling of a **less prototypical individual would, by Lakoff's definitions [11], represent the marginalization of that individual**. We hypothesize that the implications of this made it seem "low-stakes" to swap her gender within the game and that it might have been more difficult to swap the genders of an archetype instead (i.e., making a Katrina a male to have the lowest stats or Iolo a female while having the highest stats.) To validate the effects of marginalization (being further away from archetypes), we sampled characters created with *AIRvatar* that had coefficient values $.3 \leq \alpha_k \leq 0.6$ for *all three archetypes*. These reflected characters that players created to be less prototypical.

- **Character #41:** "*Pinkie is a girl with a **unique gift** for magic, ... works **best in a team but can hold her own** when needed.*"
- **Character #102:** "*A spellcaster ... a love for **forbidden magics**. **Chaotic good**, generally **tries to do the right thing but isn't afraid** to crack a few eggs to make an omlette.*"

5.2.3 Gender-related Stereotyping

In the results of characters created using *AIRvatar*, we observed that **players constructed characters with more homogeneous gender distributions between archetypes and also when close to the archetypes**. We define close as individuals with coefficient values $\alpha_k >= 0.80$. In Table 1 of the Appendix, we observe that all three archetypes had a mixture of male and female avatars close to each of them. Both Archetype 1 ("Intelligent/Wise-Cleric") and Archetype 3 ("Charming-Thief") had more female avatars closer to the the archetypes than male avatars, while Archetype 2 ("Physical-Fighter") had more male avatars closer to it. These results share similarities with those of *Oblivion*, *Ultima IV*, as well as our precious analyses in [14] where males avatars were associated with more physical roles, and female avatars with magic-related roles. For the "Charming-Thief" role, neither females nor males were closely associated with it – showing that "Thief"-like roles have less gender stereotyping associated with them. These results appears to suggest that taken collectively, players seek to reduce the degree of marginalization or privilege of either gender relative to what designers commonly portray. We hypothesize that perhaps, in the absence of a well-known game series, people relied more on real-world gender stereotypes. Thus, these results may reveal what people do without being restricted to canonical classes and roles – an observation perhaps useful for developers incorporating race and gender into their designs.

6 LIMITATIONS & FUTURE WORK

Here we discuss several limitations of our approach and describe potential avenues for overcoming them with future work and directions.
1. Determining the number of archetypes The approach we outlined in Section 3 adopts Occam's Razor [2] in that we pick the lowest number of archetypes k from the minimization of the residual sum-of-squares (RSS). However, this may not always be effective, with the result possibly being that the archetypes discovered are not sufficient to *adequately represent* the rest of the data points. For example, with $k = 3$ archetypes applied to results from *AIRvatar*, we discovered that no close individuals ($\alpha_j >= .9$) for one of the archetypes. It is possible that other metrics for determining k could be employed (e.g., choosing higher values of a scree plot's elbow.)
2. Normalizing Statistical Attributes While there are similarities between the statistical attributes used for defining characters in various videogames, there are issues with standardizing the number, the descriptions, and the effects that each attribute has. Also, there is a tension between the gaming use of these terms like "Intelligence" or "Wisdom." and their real meanings. Additionally, different games use different numerical scales (e.g., upon-100 in *Oblivion* but upon-7 in *AIRvatar*) for these attributes. It is difficult to translate the significance of each point due to different granularities. A standardized list and scale would be useful for such cross-platform comparisons.
3. Representation Beyond Statistical Attributes Representation in computing systems spans across several other technical components of the system, including graphical assets and textual descriptions [8]. Our next step is to analyze additional data collected using *AIRvatar*, which include the images of the constructed avatars, textual descriptions made by players, and other behavioral data obtained using the analytical capabilities of *AIRvatar*. We believe that these additional sources of information will enable further insight into the types of social phenomena that players experience and encounter through virtual representations in videogames and other computing systems.

7 CONCLUSION

We have demonstrated a novel approach to computationally model cognitively grounded social identity phenomena in videogames using archetypal analysis (AA). Previous work in this area has relied on qualitative methods (e.g., self-reported surveys) to identity and assess the presence of social identity-related issues such as marginalization, stereotyping, and discrimination. We demonstrated AA's effectiveness for modeling gender-related marginalization and biases like males being represented as closer archetypes than females and race-related stereotypes like in-game races possessing attributes that reflect characteristics of real-world stereotypes. AA was also able to reveal implicit categories like prototypical RPG roles used in videogames, which had implications to such race and gender-related phenomena. Being able to reveal such emergent phenomena through analyzing the data structures and designs of systems mean that computing systems can be analyzed in a systematic way, enabling quantifiable insight to be gained while minimizing the common effects of subjective evaluations such as survey bias. We believe that these findings contribute towards substantiating the use of AI to better understand the effects of virtual characters on players behaviors.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1064495.

REFERENCES

[1] Christian Bauckhage and Christian Thurau, 'Making archetypal analysis practical', in *Pattern Recognition*, 272–281, Springer, (2009).
[2] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth, 'Occam's razor', *Information processing letters*, **24**(6), 377–380, (1987).
[3] Geoffrey C. Bowker and Susan Leigh Star, *Sorting Things Out: Classification and its Consequences*, MIT Press, 1999.
[4] Raymond B Cattell, 'The scree test for the number of factors', *Multivariate behavioral research*, **1**(2), 245–276, (1966).
[5] Adele Cutler and Leo Breiman, 'Archetypal analysis', *Technometrics*, **36**(4), 338–347, (1994).
[6] Manuel JA Eugster, 'Archetypal athletes', *arXiv preprint arXiv:1110.1972*, (2011).
[7] Erving Goffman, *Stigma: Notes on the Management of Spoiled Identity*, 1963.
[8] D Fox Harrell, 'Computational and cognitive infrastructures of stigma: Empowering identity in social computing and gaming', *Proceedings of the 7th ACM Conference on Cognition and Creativity*, 49–58, (2009).
[9] D. Fox Harrell. Toward a theory of critical computing. CTheory, ctheory.net/articles.aspx?id=641, 2010.
[10] Ian Jolliffe, *Principal component analysis*, Wiley Online Library, 2005.
[11] George Lakoff, *Women, Fire, and Dangerous Things: What categories reveal about the mind*, 1990.
[12] George Lakoff, *Moral politics: How liberals and conservatives think*, University of Chicago Press, 2010.
[13] Daniel D Lee and H Sebastian Seung, 'Learning the parts of objects by non-negative matrix factorization', *Nature*, **401**(6755), 788–791, (1999).
[14] Chong-U Lim and D Fox Harrell, 'Toward telemetry-driven analytics for understanding players and their avatars in videogames', in *In CHI'15 Extended Abstracts on Human Factors in Computing Systems*, (2015).
[15] Oblivion: Character Creation. The Unofficial Elder Scrolls Pages, 1995.
[16] Eleanor Rosch, 'Principles of categorization', *Concepts: Core readings*, 189–206, (1999).
[17] Otto Santa Ana, *Brown tide rising: Metaphors of Latinos in contemporary American public discourse*, University of Texas Press, 2002.
[18] Sohan Seth and Manuel J. A. Eugster, 'Probabilistic archetypal analysis', Technical report, arXiv.org, (2014).
[19] Rafet Sifa, Christian Bauckhage, and Anders Drachen, 'Archetypal game recommender systems', *Proc. KDML-LWA*, (2014).
[20] Claude M Steele and Joshua Aronson, 'Stereotype threat and the intellectual test performance of african americans.', *Journal of personality and social psychology*, **69**(5), 797, (1995).
[21] Ultima IV: Quest of the Avatar - Companions. StrategyWiki, 2013.
[22] Nick Yee and Jeremy Bailenson, 'The proteus effect: The effect of transformed self-representation on behavior', *Human communication research*, **33**(3), 271–290, (2007).

A COEFFICIENT TABLES

Description	α_1	α_2	α_3	Player Gender	Avatar Gender
Archetype 1					
("Intelligent/Wise-Cleric")	***1.00**	0.00	0.00	Female	Female
	0.90	0.00	0.10	Female	Female
	0.82	0.17	0.01	Male	Male
Archetype 2	0.00	***1.00**	0.00	Male	Male
("Physical-Fighter")	0.00	***1.00**	0.00	Female	Female
	0.00	***1.00**	0.00	Male	Male
	0.00	***1.00**	0.00	Male	Male
	0.00	***1.00**	0.00	Male	Male
	0.00	0.88	0.12	Male	Female
	0.00	0.87	0.13	Male	Male
	0.14	0.86	0.00	Male	Male
	0.15	0.85	0.00	Male	Male
	0.14	0.85	0.02	Female	Female
	0.00	0.83	0.17	Female	Male
	0.00	0.80	0.20	Male	Male
Archetype 3	0.10	0.00	***0.90**	Male	Male
("Charming-Thief")	0.00	0.11	0.89	Female	Female
	0.15	0.00	0.85	Female	Female

Table 1. Table of characters created with *AIRvatar* with high α coefficients to each archetype. Values ≥ 0.90 are bolded. * marks the closest individual(s) of each archetype.

B BAR PLOTS

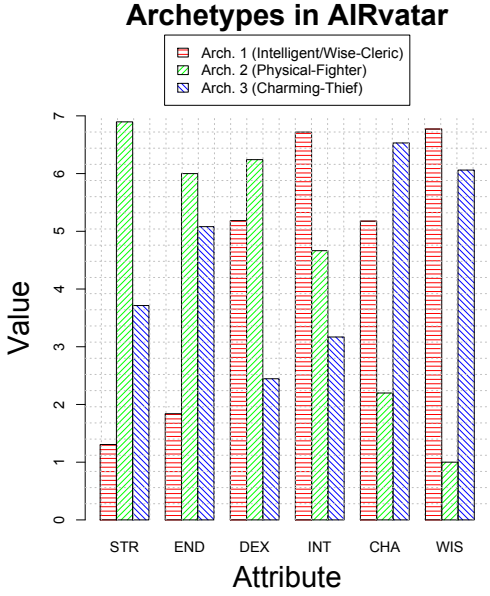
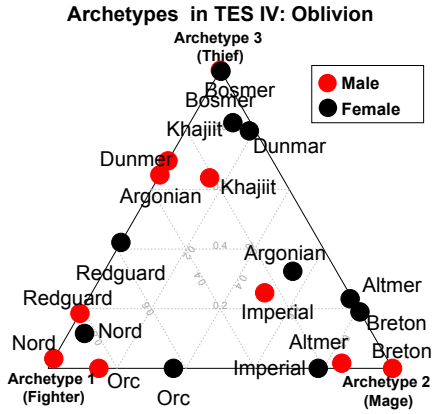
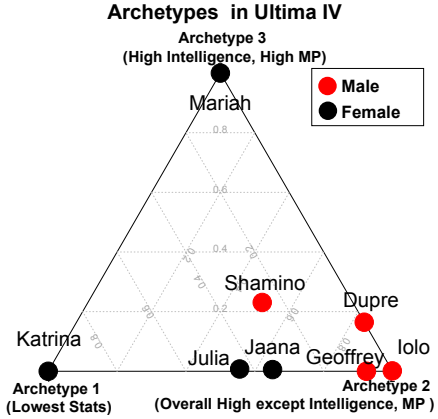


Figure 1. The plot above shows the $k = 3$ archetypes obtained from archetypal analysis on the data set of players and their statistical attribute allocations to each of their avatars. Due to convexity constraints, archetypes can be meaningfully represented with the same features of the original data.

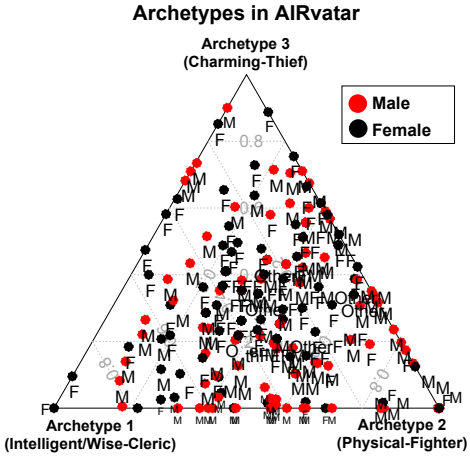
C TERNARY PLOTS



(a) Oblivion



(b) Ultima IV



(c) AIRvatar

Figure 2. Ternary plots representing characters as mixtures of archetypal archetypes in *The Elder Scrolls IV: Oblivion*, *Ultima IV*, and from our *AIRvatar* system. Labels for (a) denote races in *Oblivion*, (b) denote names in *Ultima IV*, and (c) player gender in *AIRvatar*.

PALAIS: A 3D Simulation Environment for Artificial Intelligence in Games

Patrick Schwab and **Helmut Hlavacs**[1]

Abstract. In this paper we present PALAIS — a virtual simulation environment for Artificial Intelligence (AI) in games. The environment provides functionality for prototyping, testing, visualisation and evaluation of game AI. It allows definition and execution of arbitrary, three-dimensional game scenes and behaviors. Additionally, PALAIS incorporates a plugin system that supports swift integration of custom AI algorithms. As a result, PALAIS effectively reduces the effort necessary to research, develop, prototype and showcase behaviors used for non-player characters in games. Finally, we demonstrate the power of the provided plugin system by exemplarily extending the functionality of PALAIS with an external module. PALAIS is available at http://www.palais.io.

1 INTRODUCTION

The development of game AI typically requires a testbed environment to validate and visualise results in a virtual-world scenario. Game developers and researchers frequently employ either game engines or custom-coded game scenes as their testbed environments. Using these environments for simulation has several disadvantages: suboptimal code reuse, significant barriers to entry and increased development time over using a more domain-specific environment. PALAIS attempts to solve these issues by providing commonly required functionality, such as a graphical user interface (GUI), loading required assets, data visualisation, scripting, entity management and rendering, in an existing, accessible framework. Having this framework in place enables the user to focus her efforts on AI-related code.

Moreover, custom-built solutions are often not easily distributed. We propose a container format that stores all scene-related assets in standardised formats. In PALAIS these scene containers are called *scenarios*. Any instance of PALAIS can execute these scenarios. The scenario structure, which is further described in section 3, and its distribution process is depicted in figure 1. The scenario structure allows users to share their scene definitions, graphical assets and game AI. This simplified distribution process gives others the opportunity to learn from, and build on, existing work. Consequently, our tool is also suitable for use in game AI education. Teachers can utilise the provided environment to supply students with interactive demonstrations of game AI techniques. We believe this form of hands-on education, where students can monitor and adapt execution parameters in actual game scenarios, can significantly increase the accessibility of game AI. Similarly, the simulation environment can serve as a demonstration platform for researchers to showcase their algorithms and techniques.

[1] University of Vienna, Faculty of Computer Science, Research Group Entertainment Computing, Austria, email: a0927193@unet.univie.ac.at and helmut.hlavacs@univie.ac.at

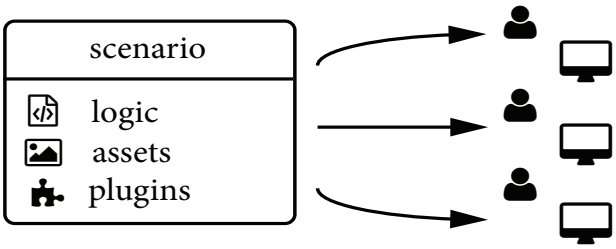


Figure 1. A schematic overview of the scenario structure and its distribution.

2 RELATED WORK

As mentioned, game developers and researchers commonly turn to commercial [15][7], open-source [14] or in-house engines for AI simulation. These general game engines overlap in functionality with PALAIS, particularly in the 3D rendering domain. PALAIS is more suitable for the simulation of game AI, because it provides the domain-specific functionality required for game AI development. Other toolkits, such as MASON [11], BREVE [10] and NetLogo [17], also provide full simulation environments. A significant drawback of some of the listed alternative simulation toolkits is the lack of extensibility via native code. Game developers strive to reach the maximum performance possible with the available computational resources. Thus, time-critical AI code for games is frequently written in native code. Our proposed simulation environment pays tribute to this by offering a plugin system [6] that allows extension through native, dynamically loaded libraries. The plugin system enables developers to test, prototype and evaluate the same native code that they use in their game engine. Ultimately, the ability to interface with native plugins also leads to more independent AI code compared to alternative simulation environments, because only the minimal necessary application programming interface (API) is exposed to plugins. Although the level of abstraction is not as high as it is with realisation-independent approaches. For example, [16] present such an realisation-independent approach.

Additionally, PALAIS provides a scripting API to increase its general accessibility and suitability for rapid prototyping. The scripting API is accessed via ECMAScript [5]. ECMAScript is one of the most widely-understood programming languages. Its most notable implementation is JavaScript, which is used to perform client-side scripting in Internet browsers. As a result of its prevalence, ECMAScript is a natural choice to provide scripting functionality in PALAIS.

To summarise, compared with the mentioned, existing works, the key distinguishing features of PALAIS are domain-specific functionality, interactivity, accessibility and extensibility.

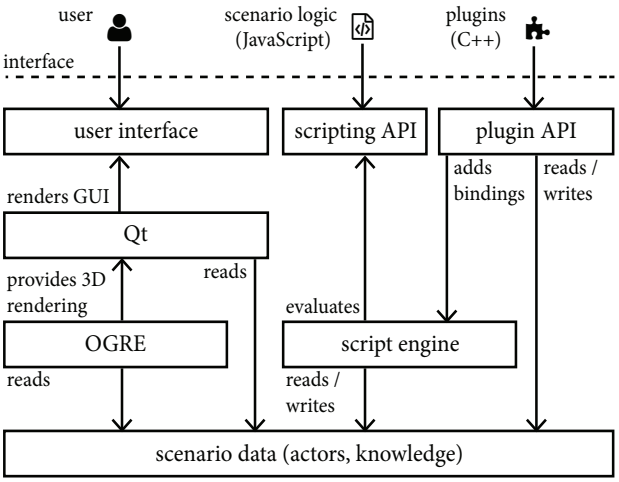


Figure 2. A schematic overview of the most significant interactions between the internal components of the simulation environment and its external accessors.

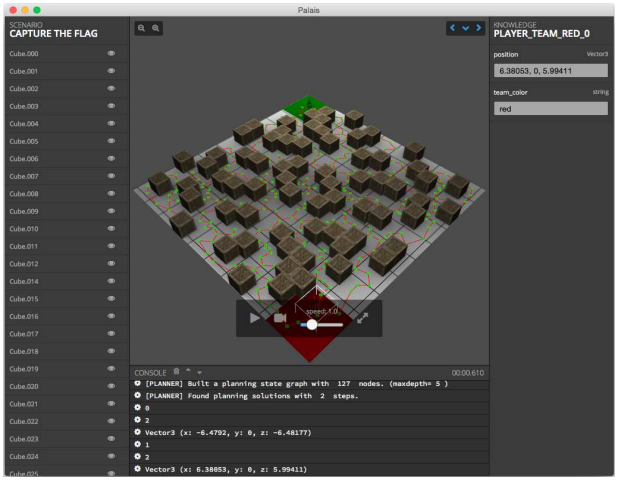


Figure 3. The GUI of PALAIS after loading a scenario. The left panel lists all active actors in the game scene. The right panel shows the knowledge inspector. The center panel displays a rendering of the scene itself.

3 SCENARIO STRUCTURE

Scenarios are the entity corresponding to a given game scene in PALAIS. They encapsulate specific game situations defined by users. The common use case is to define scenarios that provide a minimal environment for evaluation of AI behaviors and algorithms. Essentially, these scenarios are self-contained packages that include the assets, logic scripts and plugins necessary to execute a game scene. The following sections describe the components of a scenario.

3.1 Assets

The term 'assets' in the context of scenarios refers to all scene-related data files that don't contain, native or interpretable, code. Typically, assets mainly consist of the files needed for rendering the scene, such as 3D mesh data, textures and materials. PALAIS can load scene files created with external 3D modelling tools like [1]. However, PALAIS currently only supports the scene and mesh formats native to OGRE.

3.2 Logic Scripts

Logic scripts are the files containing ECMAScript code. PALAIS interprets these files at runtime. Since no compilation is required, the user can simply reload scripts after changes. The ability to reload scripts allows for frictionless development of behaviors, as the results of code changes can be evaluated quickly.

3.3 Plugins

Plugins are the other group of code attached to a game scene. Plugins, unlike logic scripts, contain compiled code. Plugins are standard shared libraries. Their specific file format depends on the operating system (OS) and the processor architecture for which the code was compiled. Relying on platform-specific formats impedes the portability of scenarios across platforms. However, we accept this price to support the integration of precompiled code. In practice, this means that a scenario must contain plugins compiled for every required target platform.

4 PROGRAMMING MODEL

We call programmable entities within a scenario in PALAIS *actors*. A generic key-value store, labeled *blackboard*, represents the individual knowledge of every actor. As the naming suggests, blackboard systems [3] inspired this form of knowledge representation. We chose a blackboard architecture because it offers flexibility and is conceptually easy to grasp and use for developers. To represent global knowledge, the game scene itself incorporates a blackboard as well. For visualisation, all actors must be connected to a rendered object in the 3D game scene. PALAIS implicitly makes all rendered objects within a game scene available as actors. Additionally, native or interpreted code can instantiate new actors at runtime.

4.1 Time Simulation

All code instances, native and interpreted alike, receive notifications of time advances. These tick events are independent of the frame rate of the simulation and represent fixed, simulated time steps. PALAIS adjusts the simulation speed by adapting the rate at which it emits these tick events relative to the passed time. This ensures the simulation results are the same, regardless of simulation speed.

5 INTERFACES

Figure 2 depicts a general overview of the interfaces of PALAIS. PALAIS exposes several external interfaces to fulfil the previously mentioned requirements.

5.1 Graphical User Interface

For users, the main external interface is the graphical user interface (GUI) provided by the runtime of PALAIS. Its main purpose is to display the data related to the currently active scenario. Most importantly, it displays the current state of the scenario in a 3D game scene. We integrated the open-source rendering engine OGRE [14] with the Qt framework [4] to provide a cross-platform GUI and 3D view. The GUI (figure 3) allows the user to configure certain rendering parameters, such as the camera's 3D orientation, zoom level and viewing

direction. The user can also view blackboards of the scenario and actors in the knowledge inspector panel of the GUI.

5.2 Scripting API

The scripting API is another external interface of PALAIS. The scripting layer is primarily meant to enable definition of arbitrary scenario logic as well as to facilitate rapid prototyping of algorithms and behaviors. PALAIS integrates a scripting engine to interpret ECMAScript code. The scripting API provides access to the currently loaded scenario and its actors. Scripts are able to read and write knowledge to the blackboards of the scenario and the actors. Lastly, scripts can consume core functionality provided by the runtime environment, e.g. dynamic actor instantiation, destruction and ray casting.

5.3 Plugin API

The last external interface to access PALAIS is the plugin API. The plugin system allows dynamic loading of third-party code. This core feature makes PALAIS suitable for integration of existing, custom AI code. The plugin API offers the same functionality as the scripting API, plus some more advanced features. Also, plugins are able to expose their functionality to the scripting layer by installing custom bindings. Custom bindings allow the use of arbitrary interaction patterns between native code in plugins and interpreted code in scripts.

5.4 Using Interpreted or Native Code in PALAIS

In essence, either scripting or plugins can be used to implement the same resulting scene logic. In fact, internally, the scripting interface is simply another layer on top of the same functionality. There is a performance overhead associated with the use of the the scripting layer, due to the additional code interpretation. Practically, that overhead means that computationally intensive tasks and tasks that run multiple times per time tick are more suited for implementation as plugins. Thus, the suggested workflow is to make all computationally intensive tasks available to the scripting layer via bindings. The extended scripting API can then be used to orchestrate the scene-specific logic.

6 INTEGRATING AN EXTERNAL MODULE

To demonstrate the power of its extension system we extended PALAIS with an external pathfinding module. The module is based on the A* search algorithm [9]. Our implementation of the pathfinding system follows the one described in [12]. A* pathfinding is a technique for determining shortest paths. It allows non-player characters (NPCs) to navigate game worlds. In this role, A* pathfinding is part of the standard repertoire of AI in games. Therefore, it is well-suited to serve as an example for exhibiting the potential of PALAIS. In particular, adding the functionality of the pathfinding module to PALAIS shows how easily existing AI code can be integrated with its environment.

6.1 Pathfinding Module

The pathfinding module provides methods for constructing and searching shortest paths on navigation graphs. As is typical for game middleware, the module is implemented in C++. The compiled, executable code is in binary form. It contains native code that depends on the processor architecture. Consequently, to integrate the module, we must exploit the ability of PALAIS to load native code as plugins.

6.2 Plugin Integration Workflow

A shared library must conform to a simple, well-defined interface to be loadable in the plugin system of PALAIS. In the current version of PALAIS, said interface consists of just 5 methods. Specifically, it consists of two methods corresponding to the loading and tear-down of the plugin, two methods corresponding to the loading and tear-down of a scenario and one method realising the time tick notification. The methods for the loading and tear-down of plugins give plugins an opportunity to initialise and destroy any general setup structures they require. Similarly, the methods for the loading and tear-down of scenarios can be used to initialise and destroy per-scenario bookkeeping information and to install script bindings with the script engine of the scenario. Finally, the time tick event initiates all time-dependent or regularly scheduled functionality. As a complementary measure, the user can register script bindings to define additional entry points.

6.2.1 Example

As is the case with most custom AI code, our pathfinding module does not conform to the plugin interface. Adapting existing code to the defined interface is the integration effort required to make the functionality of a plugin available to PALAIS. We employ the adaptor design pattern [8] to adapt the interface of our pathfinding module to the interface required by the plugin system of PALAIS. The following steps are necessary to integrate the pathfinding module:

1. First, we use the method corresponding to the initialisation of a scenario to load the navigation mesh of the currently active scenario. A navigation mesh [12] is a continuous representations of the walkable area in a game scene. After loading, the pathfinding module constructs a navigation graph from this navigation mesh. The resulting navigation graph can be searched in response to navigation requests. Furthermore, we install a script binding to make the pathfinding functionality available to scripts. These are the per-scenario steps necessary to provide a pathfinding service.
2. Next, we implement the process of searching a path. The first step in this process is initiated by script code calling the plugin via the binding registered previously. In response, the pathfinding system writes the shortest path to the blackboard of the actor that requested the shortest path.
3. Lastly, we add the actual actor movement according to the plans stored in their blackboards. For this, we use the time tick event: We sequentially check the blackboard of every actor for remaining paths to determine which actors in the current scenario must be moved. Finally, we remove a path node from the blackboard, once the actor that it belongs to reaches it.

This example demonstrates the potency of the blackboard architecture used in PALAIS. Due to the blackboard architecture the plugin system requires only a minimalist plugin interface. As a result, the blackboard architecture effectively decreases the effort required to integrate existing AI code with PALAIS.

6.3 Data Visualisation

Procedures for the in-scene visualisation of data are part of the core functionality of PALAIS. In addition to providing rendering

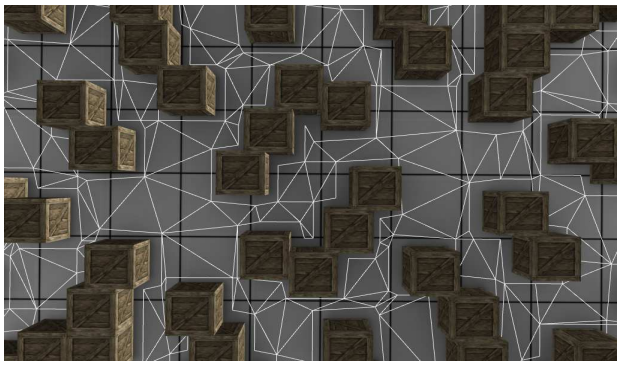


Figure 4. A rendering in PALAIS showing the navigation mesh used by the pathfinding module.

Figure 5. A rendering in PALAIS showing the navigation graph constructed from the navigation mesh in figure 4.

of arbitrary textured meshes, PALAIS provides means for rendering coloured primitives, such as lines, circles, quads, cuboids and spheres. As an example, the pathfinding module renders the navigation graph using the visualisation primitives of PALAIS. Figure 4 and figure 5 depict renderings of the navigation mesh and the navigation graph in PALAIS.

6.4 Accessing the Pathfinding Module

The plugin installs its script bindings when a scene is loaded. In our pathfinding example, all scripts in a scenario, that includes the pathfinding plugin, can invoke the process to navigate an actor to a goal along a shortest path. The script delegates the computation and handling of the movement to the plugin. This abstraction provided by plugins also allows the reuse of plugins in different scenarios.

7 CONCLUSION

PALAIS is a powerful environment for the simulation of AI in games. It caters specifically to the needs of game developers by granting access to its programming interface via interpreted and native code. Our exemplary integration of an external pathfinding module demonstrates that PALAIS is an apt choice for the simulation of scenes that depend on third-party AI libraries. Additionally, the ability to extend PALAIS with plugins lowers the barrier to entry for the usage of the simulation environment, since the same native code, that is used for the simulation in PALAIS, can easily be shared with game engines.

8 FUTURE WORK

The work on the simulation environment PALAIS is part of a larger, ongoing project to build a unified framework for game AI development. The framework includes functionality for each of the layers of the game AI model proposed in [12]. Particularly, it encompasses algorithms that facilitate the implementation of movement, decision making and strategy for non-player characters in games. Pathfinding, Behavior Trees [2] and Goal-Oriented Action Planning (GOAP) [13] are among the standard techniques the framework implements. These techniques will be integrated with PALAIS in the form of plugins to provide users with a solid foundation that allows the rapid development of AI behaviors. On the feature side, future work on PALAIS could involve refinement by adding support for physics-based dynamics and statistical evaluation of behaviors.

REFERENCES

[1] Blender Online Community. Blender - a 3D modelling and rendering package. Retrieved from http://www.blender.org.

[2] Alex Champandard, 'Behavior trees for next-gen game AI', in *Game Developers Conference, Audio Lecture*, (2007).

[3] Daniel D Corkill, 'Blackboard systems', *AI expert*, **6**(9), 40–47, (1991).

[4] Digia Plc. Qt: cross-platform application and UI framework, 2012.

[5] ECMA International, *Standard ECMA-262 - ECMAScript Language Specification*, 5.1 edn., June 2011.

[6] Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[7] Epic Games. Unity engine documentation. Retrieved from https://www.unrealengine.com/.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design patterns: elements of reusable object-oriented software*, Pearson Education, 1994.

[9] Peter E Hart, Nils J Nilsson, and Bertram Raphael, 'A formal basis for the heuristic determination of minimum cost paths', *Systems Science and Cybernetics, IEEE Transactions on*, **4**(2), 100–107, (1968).

[10] Jon Klein, 'Breve: a 3d environment for the simulation of decentralized systems and artificial life', in *Proceedings of the eighth international conference on Artificial life*, pp. 329–334, (2003).

[11] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan, 'Mason: A multiagent simulation environment', *Simulation*, **81**(7), 517–527, (2005).

[12] Ian Millington and John Funge, *Artificial intelligence for games*, CRC Press, 2009.

[13] Jeff Orkin, 'Applying goal-oriented action planning to games', *AI Game Programming Wisdom*, **2**(2004), 217–227, (2004).

[14] Torus Knot Software. Object-oriented graphics rendering engine (OGRE) Engine documentation. Retrieved from http://www.ogre3d.org/.

[15] Unity Technologies. Unity documentation. Retrieved from http://unity3d.com/.

[16] Marco Vala, Guilherme Raimundo, Pedro Sequeira, Pedro Cuba, Rui Prada, Carlos Martinho, and Ana Paiva, 'ION framework–a simulation environment for worlds with virtual agents', in *Intelligent virtual agents*, pp. 418–424. Springer, (2009).

[17] Uri Wilensky, 'Netlogo', http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, (1999).

Simulating Autonomous Non-Player Characters in a Capture the Flag Scenario Using PALAIS

Patrick Schwab and **Helmut Hlavacs**[1]

Abstract. PALAIS is a 3D simulation environment for artificial intelligence (AI) in games. It has built-in support for much of the standard functionality required when simulating AI behaviors. Most importantly, PALAIS allows users to define their own arbitrary game scenes with custom game rules. This paper presents the workflow of authoring game scenes in PALAIS by the example of a Capture the Flag scene. In particular, we demonstrate how users can take advantage of the provided scripting layer to rapidly define their simulation logic. This paper also serves as a description of the content of the accompanying demonstration given at the conference.

1 Simulation Environment

Game scenes in PALAIS are defined in packages called *scenarios*. These scenarios contain all code and graphical assets required for the simulation of the game scene. Users define the visual appearance of scenarios in an external 3D modelling tool. At runtime, users can access the functionality of PALAIS via a scripting or a native programming interface. The scripting interface can be accessed from the ECMAScript [2] programming language. Additionally, users can extend the functionality available to scripts by utilising the plugin system [3] incorporated in PALAIS. The combination of plugins and scripts allows for the definition of rich interaction patterns.

PALAIS automatically creates a blackboard [1] for each actor in a scenario. This form of knowledge representation provides a very flexible means of managing the data flow between the different components of a scenario. The contents of the blackboards of each actor can be examined during the simulation of a scenario. Figure 1 shows the knowledge inspector in action.

2 Capture the Flag Scenario

We chose a Capture the Flag Scenario as our exemplary game scenario. The Capture the Flag scenario involves two opposing teams. Each team has to capture the flag of the opposing team to score points. Characters can capture a flag by taking it from the initial spawning point of the opposing team to the initial spawning point of their own team. Implementing AI for non-player characters in a Capture the Flag scenario is a standard problem in game AI. Thus, it is well-suited to showcase the abilities of PALAIS. The arena of the implemented Capture the Flag scenario is shown in figure 2.

[1] University of Vienna, Faculty of Computer Science, Research Group Entertainment Computing, Austria, email: a0927193@unet.univie.ac.at and helmut.hlavacs@univie.ac.at

3 Authoring Workflow

To implement the Capture the Flag scenario we employ plugins that provide standard algorithms of game AI. These plugins allow us to delegate computationally intensive tasks, such as pathfinding, to native code. We use the scripting interface of PALAIS to orchestrate the actors of the scenario and to define the possible actions they can take.

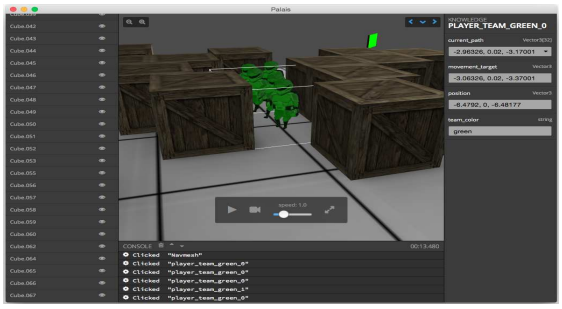


Figure 1. A demonstration of the live inspection of blackboards available in PALAIS. The panel on the right shows the contents of the blackboard of the frontmost actor of the green team.

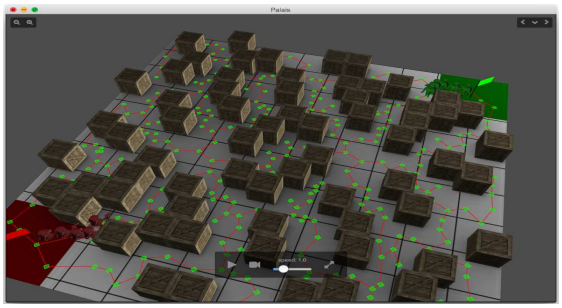


Figure 2. A rendering in PALAIS that shows the arena of the Capture the Flag scenario.

REFERENCES

[1] Daniel D Corkill, 'Blackboard systems', *AI expert*, **6**(9), 40–47, (1991).
[2] ECMA International, *Standard ECMA-262 - ECMAScript Language Specification*, 5.1 edn., June 2011.
[3] Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

EmohawkVille: Virtual City for Everyone

David Holaň and **Jakub Gemrot** and **Martin Černý** and **Cyril Brom** [1]

Abstract. Despite recent progress, behavior of non-player characters (NPCs) in contemporary games is still kept rather simple. This is an opportunity for the academia to develop novel techniques and tools that would allow for easier creation of complex behaviors that are resilient to the dynamicity implied by the presence of the player. There already exist languages within multiagent community that are thought to be suitable for NPC behaviors creation, but they are usually tested in simplistic environments and our experience indicates that applying them to complex 3D worlds introduces significant obstacles. This is part of the reason why simple reactive techniques are prevalent in game industry practice. Moreover there is no publicly available research-friendly 3D virtual world with sufficient complexity that would allow developers to evaluate their languages and tools in a more realistic setting and improve them toward practical applicability. In this demo we present EmohawkVille: an open-source first-person 3D virtual world that is a candidate for such an environment.

1 Introduction

Many contemporary computer games take a great effort to achieve a high level of believability of their virtual worlds. This is especially true for games with large open worlds, where the user is free to discover the environment on his own and is relatively unconstrained by the game. One of the challenges that arise in this scenario is the problem of choosing the right higher-level action for the NPCs (e.g., move to a point, pick up an item, use an item, ...). Since the game industry relies almost exclusively on simple reactive techniques which make creation of complex behaviors rather time-consuming and costly, non-player characters (NPCs) display complex behaviors only during crucial game events. In between, the NPC behaviors are schematic at best.

The main issue is that going beyond simple behavior and still maintaining the suspension of disbelief introduces significant difficulties to the NPC behavior authoring. There are many possible obstacles to NPC goals and if they are not taken into account, the NPCs are easy for the player to "break" and may provide even worse illusion of a real world than rather static NPCs.

For a truly alive open world, dozens of different and often complex scenarios are needed, which implies that the world needs to be equipped with a rich ontology of items and actions NPCs (as well as the player) can perform.

As the world ontology grows, the number of meaningful NPC action sequences increases and the behavior complexity rises. Not only the means-ends analysis becomes more demanding, new problems emerge such as transitional behaviors, joint behaviors, behaviors ordering or behaviors interleaving [6]. At the same time, game studios usually cannot afford to let an expert AI programmer design such day-to-day behaviors, because that would be cost-prohibitive. Most of the NPC design is thus usually carried out with the aid of some visual tool by scripters with little programming experience.

At this place, academia could provide action selection mechanisms (ASM) and accompanying tools that would help inexpert scripters to create complex behaviors that are *interactively believable*, that is, behaviors that sustain their believability under nondeterminism brought by the player. However, most of the academic research is carried out in environments that either have simple ontologies or are static or discrete. Games on the other hand are dynamic, multi-agent environments that can be for all practical purposes considered continuous in both time and space. There are languages and techniques that can be applied to such worlds: either from the multiagent community or the field of robotics or automated planning. However, to our knowledge, there is currently no 3D virtual world publicly available that would provide rich ontology for NPCs out of the box. This means that in this particular problem area, academia is one step behind the industry — we do not even have an environment to work with.

Note that raw frameworks such as Unity [7] are not sufficient as creating a rich world in a raw framework is a substantial amount of work. An important part of the environment is also the possibility to develop the NPC behavior with a high-level language such as Java since nearly all agent languages of interest can be invoked from Java code. We are not aware of any complex 3D environment that would meet all those requirements. See our paper [4] for a thorough comparison of possible candidates.

Previous research has shown that applying agent languages to 3D environments is neither straightforward nor guaranteed to yield better results than using a general programming language [2, 5]. Common issues with agent languages are incomplete debugging and tool support, some of the architectures are also hard to debug in principle (e.g., because of inherent paralellism). Many agent languages are also declarative in nature, while game worlds feature lots of mechanics that are hard to express declaratively (e.g., determining which object is hit by an arrow). Proper evaluation of agent languages is thus critical.

In this demo, we present an extension of the Pogamut 3 platform [3] called EmohawkVille, the first step towards an open-sourced complex simulation of NPC everyday life in 3D virtual world. We believe that creating a fully working, accessible and polished environment fosters academic progress. The large amount of research work evaluated on Pogamut for Unreal Tournament 2004 supports this view. We have also exerted great effort to make EmohawkVille a mature tool. In practice, there is a long chain of components that are needed to fully connect high-level AI with an NPC: sensors and actuators interface, navigation and pathfinding, character animation support are among the most important, but the list is far from exhaus-

[1] Charles University in Prague, Czech Republic email: {paladin.invictus,jakub.gemrot,cerny.m}@gmail.com, brom@ksvi.mff.cuni.cz

tive. In EmohawkVille, we have resolved large part of those issues on behalf of the researcher. The quality of the EmohawkVille environment was evaluated in a small-scale user study and by use of the environment in our teaching curriculum.

2 General Description

EmohawkVille is a first-person virtual world with detailed interactive elements of day-to-day life. There is a general framework that supports interaction with items, continual actions and processes and inter-agent communication including trade. There is a set of ready-made assets for a cooking scenario. For example, an agent or a human player can pick up a piece of meat, put it on a chopping board and slice it and then fry the slices on a pan (charring the food if he does not add oil or forgets to flip the meat). The cooking scenario was chosen as our first because it features plethora of complex procedures yet it is easy to grasp by programmers and non-programmers alike and is gender-neutral.

EmohawkVille is based on Unreal Development Kit (UDK) [1] and thus is capable of displaying the world in state-of-the-art graphics. UDK is free for educational and non-commercial use and EmohawkVille itself is available under GPLv3[2].

In EmohawkVille the world mechanics are implemented in UnrealScript - a proprietary language deployed with the UDK toolkit. The Pogamut platform provides a high-level Java interface to the UDK for writing the actual AI and takes care of many common tasks (pathfinding with A* and smooth path following, caching sensory data to a blackboard, etc.). Both the UDK and the Java part have been designed with possible further extensions in mind and the basic NPC support is separated from the model of the general EmohawkVille ontology, which is in turn separated from the implementation of the specific mechanics for our cooking scenario. The UDK part also fully supports interaction with a human user through the UDK visual client.

At this moment, EmohawkVille features 20 item types (food, cutlery, cooking tools, ...) and a cooking stove (part of the environment). Interaction is provided by 14 actions, of which nine are instant and four initiate a longer-lasting process, e.g., chopping a vegetable or stirring a broth. An overview of the available items is visible in Figure 1.

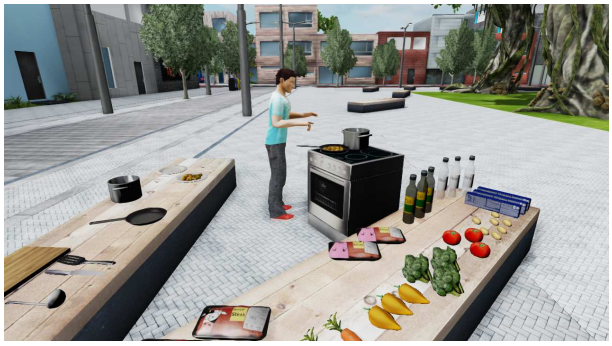


Figure 1. A screenshot of the environment.

The central complexity of the NPC behavior stems from the simulation of cooking. Some ingredients can be boiled, some fried. The speed of cooking is determined by the temperature of respective stoves. Water evaporates from pots and ingredients may burn or char if not stirred or flipped in the pot or the pan. The cooking theme provides important challenges to the NPC behavior creation: cooking a meal may require a long sequence of actions (more than 20), effectivity is increased by performing processes in parallel possibly requiring cooperation of multiple chefs, the player may both support and sabotage the cooking NPC.

Every aspect of the environment and the agents is programmable. EmohawkVille is ready for a researcher to plugin any high-level decision making mechanism (planning, machine learning, ...) without the need to handle low-level details. More detail of the environment is given in our paper [4].

3 Demo Presentation

In our demo presentation we would like to show the environment and its richness, let the spectators interact with the environment themselves, helping a preprogrammed agent to cook a complex meal or sabotaging his effort. We would also like to show that programming the behaviors is easy and EmohawkVille thus lets the researcher focus on the action selection exclusively. This will be demonstrated by a live creation of a cooking NPC and we would enable hands-on programming experience to the spectators.

A video presentation of the environment may be found at `http://www.youtube.com/watch?v=G7lKXkR2Xgg`

ACKNOWLEDGEMENTS

This research was supported by SVV project number 260 224.

REFERENCES

[1] Epic Games Inc. Unreal development kit documentation. `http://www.unrealengine.com/udk/documentation/`, 2009. Last checked: 2015-03-30.

[2] Jakub Gemrot, Zdeněk Hlávka, and Cyril Brom, 'Does high-level behavior specification tool make production of virtual agent behaviors better?', in *Proceedings of CAVE'12*, pp. 167–183, Berlin, Heidelberg, (2013). Springer-Verlag.

[3] Jakub Gemrot, Rudolf Kadlec, Michal Bída, Ondřej Burkert, Radek Píbil, Jan Havlíček, Lukáš Zemčák, Juraj Šimlovič, Radim Vansa, Michal Štolba, Tomáš Plch, and Cyril Brom, 'Pogamut 3 can assist developers in building ai (not only) for their videogame agents', in *Agents for Games and Simulations*, eds., Frank Dignum, Jeff Bradshaw, Barry Silverman, and Willem Doesburg, LNCS 5920, 1–15, Springer-Verlag, (2009).

[4] Gemrot J. Černý M. Holaň, D., 'Emohawkville: Towards complex dynamic virtual worlds', in *Proceedings of GAMEON'2013*, pp. 52–58, (2013).

[5] Radek Píbil, Peter Novák, Cyril Brom, and Jakub Gemrot, 'Notes on pragmatic agent-programming with Jason', in *Programming Multi-Agent Systems*, LNCS 7217, 58–73, Springer, (2012).

[6] Tom Plch, *Action selection for an animat*, Master's thesis, Charles University in Prague, 2009.

[7] Unity Technologies. Unity documentation. `http://docs.unity3d.com/Documentation/Manual/`, 2005. Last checked: 2015-03-30.

[2] EmohawkVille may be downloaded from `http://pogamut.cuni.cz/main/tiki-index.php?page=EmohawkVille`

An interactive, generative Punch and Judy show using institutions, ASP and emotional agents

Matt Thompson [1] and **Julian Padget** and **Steve Battle**

Abstract. Using Punch and Judy as a story domain, we describe an interactive puppet show, where the flow and content of the story can be influenced by the actions of the audience. As the puppet show is acted out, the audience reacts to events by cheering or booing the characters. This changes the emotional state of each agent, potentially causing them to change their actions, altering the course of the narrative. An institutional model is used to ensure that the narrative is constrained to remain consistent with the Punch and Judy canon.

1 Introduction

Agent-based approaches for interactive narrative generation use intelligent agents to model the characters in a story. The agents respond to the interactions of a player with dialogue or actions fitting the shape of a story. However, these agents have little autonomy in their actions, bound as they are to the strict requirements of their role in the narrative.

An institutional model can be used as normative framework for governing the actions of agents in a story. By describing the rules of a narrative in terms of social expectations, the agents are encouraged to perform certain types of actions while still remaining free to break free of these expectations. As in society in the real world, breaking agreed norms comes with consequences, and only generally happens in exceptional circumstances.

One situation where this is desirable is with the use of emotional agents. An agent experiencing an extreme emotion in an emotional model (such as rage or depression) may be allowed to act unusually or uncharacteristically. Allowing characters to break from narrative norms enables them to be 'pushed too far' by circumstances, with results that add an extra dimension of richness to a story.

Through this implementation, we introduce two novel approaches: (i) the use of an institutional model to describe a narrative 'world' or domain, and (ii) how emotional models can give intelligent agents some degree of autonomy to both act in idiosyncratic ways and to react emotionally to input from the audience.

The puppets in the show are each belief-desire-intention (BDI) agents with a valence, arousal, dominance (VAD) emotional model described in section 5. The story is modelled by a set of institutional norms (section 6.1) that describe the Punch and Judy story domain in terms of Propp's 'story moves' [8] (section 3). The agents communicate with their environment using the Bath Sensor Framework, described in section 6.3 [6]. In the final sections, we describe the animation system that functions as the agents' environment (section 6.4), and how the audience interacts with the system (section 7).

[1] University of Bath, United Kingdom, email: m.r.thompson@bath.ac.uk

2 Propp moves and roles

To express story events as an institution, we must look to narrative theory for inspiration. Instead of describing parts of the Punch and Judy story explicitly (such as 'Punch is expected to hit the policeman in this scene'), it is more desirable to describe scenes in a more abstract way ('The villain fights the victim in this scene'). The use of more general story components allows us to reuse them in multiple scenes, or even in other stories.

Narratology, and structuralism in particular, supply such generalised building blocks for stories. Russian formalism is an early movement in narrative theory to formalise the elements of narrative, of which Vladimir Propp is a prominent figure.

In order to direct the course of the narrative, we use a model built upon Propp's 1928 formalism of Russian folktales, *The Morphology of the Folktale* [8]. In this formalism, Propp identifies recurring characters and motifs in Russian folklore, distilling them down to a concise syntax with which to describe stories.

In this formalism, characters have *roles*, such as *hero*, *villain*, *dispatcher*, *false hero*, and more. Characters performing a certain role are able to perform a subset of *story moves*, which are actions that make the narrative progress. For example, the *dispatcher* might send the *hero* on a quest, or the *victim* may issue an *interdiction* to the *villain*, which is then *violated*.

Propp defines a total of 31 distinct story functions, some of which can have subtle variations from story to story. Each function is given a number and symbol in order to create a succinct way of describing entire stories. Examples of such functions are:

- One of the members of a family absents himself from home: *absentation*.
- An interdiction is addressed to the hero: *interdiction*.
- The victim submits to deception and thereby unwittingly helps his enemy: *complicity*.
- The villain causes harm or injury to a member of the family: *villainy*.

Each of these functions can vary to a great degree. For example, the *villainy* function can be realised as one of 19 distinct forms of villainous deed, including *the villain abducts a person*, *the villain seizes the daylight*, and *the villain makes a threat of cannibalism*.

These functions are enacted by characters following certain roles. Each role (or *dramatis personae* in Propp's definition) has a *sphere of action* consisting of the functions that they are able to perform at any point in the story. Propp defines seven roles that have distict spheres of action: *villain*, *donor*, *helper*, *princess*, *dispatcher*, *hero*, and *false hero*.

In a typical story, one story function will follow another as the tale progresses in a sequential series of cause and effect. However, Propp's

formalism also allows for simultaneous story functions to occur at once.

2.1 Propp example: sausages and crocodile scene

The common elements of Punch and Judy are easily described in terms of Propp's story functions. Here we pick one scene from the Punch and Judy show to use as an example: the scene where Punch battles a crocodile in order to safeguard some sausages.

In this scene, Joey the clown (our narrator) asks Punch to guard the sausages. Once Joey has left the stage, a crocodile appears and eats the sausages. Punch fights with the crocodile, but it escapes. Joey then returns to find that his sausages are gone.

The appropriate story functions are:

1. Joey tells Punch to look after the sausages (*interdiction*).
2. Joey has some reservations, but decides to trust Punch (*complicity*).
3. Joey gives the sausages to Punch (*provision or receipt of a magical agent*).
4. Joey leaves the stage (*absentation*).
5. A crocodile enters the stage and eats the sausages (*violation*).
6. Punch fights with the crocodile (*struggle*).
7. Joey returns to find that the sausages are gone (*return*).

3 Institutional model

An institution describes a set of 'social' norms describing the permitted and obligated behaviour of interacting agents. Noriega's 'Fish Market' thesis [7] describes how an institutional model can be used to regiment the actions of agents in a fish market auction. Cliffe [3], Baines and Lee [6] extend this idea to build systems where institutions actively regulate the actions of agents, while still allowing them to decide what to do. Adapting this idea to the world of narrative, we use an institutional model to describe the story world of Punch and Judy in terms of Propp moves and character roles.

Institutional models use deontic logic to describe obligations and permissions that act on interacting agents in an environment. By combining this approach with Propp's concepts of *roles* and *story moves*, we describe a Propp-style formalism of Punch and Judy in terms of what agents are *obligated* and *permitted* to do at certain points in the story.

For example, in one Punch and Judy scene a policeman enters the stage and attempts to apprehend Punch. According to the rules of the Punch and Judy world, Punch has an obligation to kill the policeman by the end of the scene (as this is what the audience expects to happen, having seen other Punch and Judy shows). The policeman has an obligation to try his best to catch Punch. Both agents have permission to be on the stage during the scene. The policeman only has permission to chase Punch if he can see him (Punch is obligated to hide from him at the start of the scene).

The permissions an agent has constrain the choices of actions available to them at any given moment. Obligations affect the goals of an agent. Whether or not an agent actively tries to fulfil an obligation depends on their emotional state.

3.1 Institution example

Here we continue the 'sausages and crocodile' scene example from section 3.1, taking the Propp story functions and describing them as an institutional model.

We define our institution in terms of *fluents*, *events*, *powers*, *permissions* and *obligations*.

3.1.1 Fluents

Fluents are properties that may or may not hold true at some instant in time. *Institutional events* are able to *initiate* or *terminate* fluents at points in time. A fluent could describe whether a character is currently on stage, the current scene of a story, or whether or not the character is happy at that moment in time.

Domain fluents ($\mathcal{D}$) describe domain-specific properties that can hold at a certain point in time. In the Punch and Judy domain, these can be whether or not an agent is on stage, or their role in the narrative (equation 1).

$$\mathcal{D} = \{\text{onstage, hero, villain, victim, donor, item}\} \quad (1)$$

Institutional fluents consist of *institutional powers*, *permissions* and *obligations*.

An **institutional power** ($\mathcal{W}$) describes whether or not an external event has the authority to meaningfully generate an institutional event. Using Propp as an example, an *absentation* event can only be generated by an external event coming from a *donor* character (such as their leaving the stage). Therefore, any characters other than the donor character would not have the institutional power to generate an *absentation* institutional event when they leave the stage.

Equation 2 shows a list of possible empowerments, essentially a list of institutional events.

$$\mathcal{W} = \{\text{pow(introduction, interdiction, give, absentation, violation, return)}\} \quad (2)$$

Permissions ($\mathcal{M}$) are external actions that agents are permitted to do at a certain instant in time. These can be thought of as the set of *socially permitted* actions available to an agent. While it is possible for an agent to perform other actions, societal norms usually prevent them from doing so.

For example, it would make sense in the world of Punch and Judy if Punch were to give the sausages to the Policeman. It is always Joey who gives the sausages to Punch. Also, it would be strange if Joey were to do this in the middle of a scene where Punch and Judy are arguing. We make sure agents' actions are governed so as to allow them only a certain subset of permitted actions at any one time. Equation 3 shows a list of permission fluents.

$$\mathcal{M} = \{\text{perm(leavestage, enterstage, die, kill, hit, give, fight)}\} \quad (3)$$

Obligations ($\mathcal{O}$) are actions that agents *should* do before a certain deadline. If the action is not performed in time, a *violation event* is triggered, which may result in a penalty being incurred. While an agent may be obliged to perform an action, it is entirely their choice whether or not they actually do so. They must weigh up whether or not pursuing other courses of action is worth suffering the penalty that an unfulfilled obligation brings.

Anybody who has seen a Punch and Judy show knows that at some point Joey tells Punch to guard some sausages, before disappearing offstage. Joey's departure is modelled in the institution as the *absentation* event. It could be said that Joey has an obligation to leave the stage as part of the *absentation* event, otherwise the story function is violated. Equation 4 shows how this would be described in the institution.

$$\mathcal{O} = \{\text{obl(leavestage, absentation, viol(absentation))}\} \quad (4)$$

3.1.2 Events

Cliffe's model specifies three types of **event**: *external events* (or 'observed events', $\mathcal{E}_{obs}$), *institutional events* ($\mathcal{E}_{instact}$) and *violation events* ($\mathcal{E}_{viol}$).

External events are observed to have happened in the agents' environment, which can *generate institutional events* which act only within the institional model, *initiating* or *terminating* fluents, permissions, obligations or institutional powers. An external event could be an agent leaving the stage, an agent hitting another, or an agent dying. Internal events include narrative events such as scene changes, or the triggering of Propp story functions such as *absentation* or *interdiction* (described in section 3). Violation events occur when an agent has failed to fulfil an obligation before the specified deadline. These can be implemented in the form of a penalty, by decreasing an agent's health, for example.

$$\mathcal{E}_{obs} = \{\text{startshow, leavestage, enterstage, die, give, harmed, hit, fight, kill, escape}\} \quad (5)$$

$$\mathcal{E}_{instact} = \{\text{introduction, interdiction, give, absentation, violation, return, struggle, defeat, complicity, victory, escape}\} \quad (6)$$

$$\mathcal{E}_{viol} = \{\text{viol(introduction), viol(interdiction), viol(give), viol(absentation), viol(violation), viol(return), viol(struggle), viol(defeat), viol(complicity) viol(victory), viol(escape)}\} \quad (7)$$

3.1.3 Event Generation and Consequences

An **event generation** function, $\mathcal{G}$, describes how events (usually external) can generate other (usually institutional) events. For example, if an agent leaves the stage while the *interdiction* event holds, they trigger the *leavestage* event. This combination generates the *absentation* institutional event (equation 11).

Event generation functions follow a $\langle$preconditions$\rangle \rightarrow \{$postconditions$\}$ format: $\langle \mathcal{G}(\mathcal{X}, \mathcal{E})\rangle \rightarrow \{\mathcal{E}_{out}\}$, where $\mathcal{X}$ is a set of fluents that hold at that time, $\mathcal{E}$ is an event that has occurred, and $\mathcal{E}_{out}$ are the events that are generated. They are generally used to generate internal, institutional events from external events.

Consider the Punch and Judy scenario described in section 3.1. There are seven institutional events (story functions) that occur during this scene: *interdiction*, *complicity*, *receipt* (from Propp's *receipt of a magical agent*) *absentation*, *violation*, *struggle*, *return*. These institutional events are all generated by external events. The *interdiction* is generated when Joey tells Punch to protect the sausages. Punch agreeing amounts to *complicity*. Joey *gives* punch the sausages (*receipt*), then leaves the stage (*absentation*). The crocodile eating the sausages is a *violation* of Punch's oath, the agents fight (*struggle*), then Joey enters the stage again (*return*).

It is desirable that these story function occur in this sequence in order for a satisfying narrative to emerge. Agents may decide to perform actions that diverge from this set of events, but the institution is guiding them towards the most fitting outcome for a *Punch and Judy* world. For this reason, a currently active story function can be the precondition for event generation. For example, the *receipt* event may only be triggered if an agent externally performs a *give* action **and** if the *complicity* event currently holds (equation 10).

Examples of event generation function for this scenario, complete with preconditions, are listed in equations 8 to 14.

$$\mathcal{G}(\mathcal{X}, \mathcal{E}) : \langle \emptyset, \mathit{tellprotect}(\text{donor, villain, item})\rangle \rightarrow \{\mathit{interdiction}\} \quad (8)$$

$$\langle \{\mathit{interdiction}\}, \mathit{agree}(\text{villain}))\rangle \rightarrow \{\mathit{complicity}\} \quad (9)$$

$$\langle \emptyset, \mathit{give}(\text{donor, villain, item}))\rangle \rightarrow \{\mathit{receipt}\} \quad (10)$$

$$\langle \{\mathit{interdiction}\}, \mathit{leavestage}(\text{donor})\rangle \rightarrow \{\mathit{absentation}\} \quad (11)$$

$$\langle \{\mathit{interdiction}\}, \mathit{harmed}(\text{item})\rangle \rightarrow \{\mathit{violation}\} \quad (12)$$

$$\langle \{\mathit{interdiction}, \mathit{absentation}\}, \mathit{enterstage}(\text{donor}),\ \text{onstage(villain)}\rangle \rightarrow \{\mathit{return}\} \quad (13)$$

$$\langle \emptyset, \mathit{hit}(\text{donor, villain})\rangle \rightarrow \{\mathit{struggle}\} \quad (14)$$

Consequences consist of fluents, permissions and obligations that are *initiated* ($\mathcal{C}^{\uparrow}$) or *terminated* ($\mathcal{C}^{\downarrow}$) by institutional events. For example, the institutional event *give* could initiate the donor agent's permission to leave the stage, triggering the *absentation* event (equation 16). When the *interdiction* event is currently active and a *violation* event occurs, the interdiction event is terminated (21). Equations 15 to 22 describe the initiation and termination of fluents in the Punch and Judy sausages scenario detailed in section 3.1.

$$\mathcal{C}^{\uparrow}(\mathcal{X}, \mathcal{E}) : \langle \emptyset, \text{interdiction}\rangle \rightarrow \{\text{perm(give(donor, villain, item))}\} \quad (15)$$

$$\langle \emptyset, \text{receipt}\rangle \rightarrow \{\text{perm(leavestage(donor))}\} \quad (16)$$

$$\{\mathit{active}(\mathit{interdiction})\}, \text{violation}\rangle \rightarrow \{\text{perm(enterstage(dispatcher))}\} \quad (17)$$

$$\{\mathit{active}(\mathit{absentation}), \mathit{active}(\mathit{violation})\}, \text{return}\rangle \rightarrow \{\text{perm(hit(donor, villain))}\} \quad (18)$$

$$\mathcal{C}^{\downarrow}(\mathcal{X}, \mathcal{E}) : \langle \emptyset, \text{interdiction}\rangle \rightarrow \{\text{perm(give(donor, villain, item))}\} \quad (19)$$

$$\langle \{\mathit{active}(\mathit{interdiction})\}, \text{absentation}\rangle \rightarrow \{\text{perm(leavestage(donor))}\} \quad (20)$$

$$\langle \{\mathit{active}(\mathit{interdiction})\}, \text{violation}\rangle \rightarrow \{\mathit{active}(\mathit{interdiction})\} \quad (21)$$

$$\langle \{\mathit{active}(\mathit{absentation}), \mathit{active}(\mathit{violation})\}, \text{return}\rangle \rightarrow \{\mathit{active}(\mathit{absentation})\} \quad (22)$$

4 VAD emotional model

In order to make the agents acting out the Punch and Judy show more believable, we apply an emotional model to affect their actions and decisions. For this, we use the valence-arousal (circumplex) model first described by Russell [10].

In order to give each character its own distinct personality, we extend this model with an extra dimension: dominance, as used by

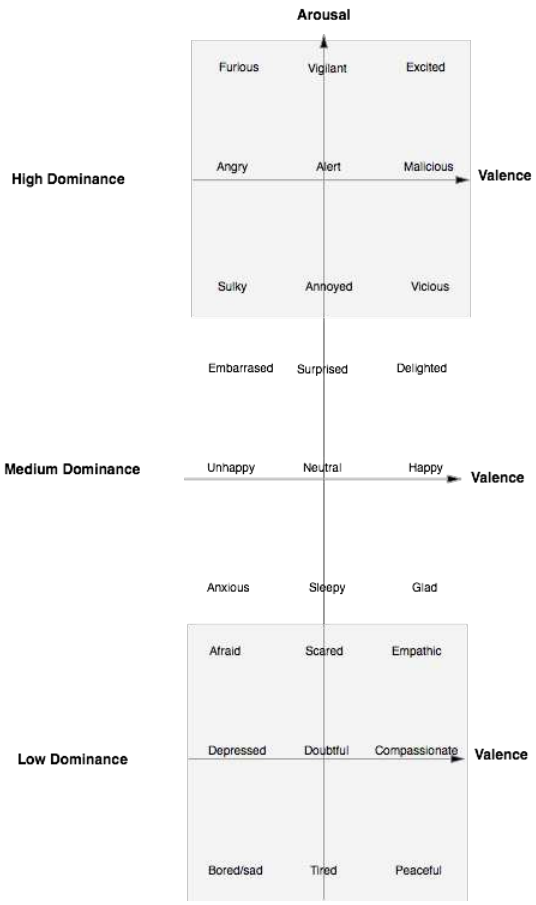


Figure 1. VAD emotional values, adapted from Ahn et al [1]

Ahn et al in their model for conversational virtual humans [1]. This dominance level is affected by the reactions of the audience to the agents' actions. For example, Judy may become more dominant as her suggestions to hit Punch with a stick are cheered on by the audience, emboldening her into acting out her impulses.

Figure 1 shows how valence, arousal and dominance values map to identifiable emotions. Valence, arousal and dominance can each have a value of low, medium or high. This allows the agents to have a total of 27 distinct emotional states.

Valence and arousal levels of each agent are affected by the actions of other agents. For example, a character being chased around the stage by Punch will see their valence level drop while their arousal increases. According to Russell's circumplex model of emotion [10], this would result in them becoming *afraid* (if their dominance level is low).

An agent's emotional state affects its ability to fulfil its institutional obligations. An agent that is *furious* would have no problem carrying out an obligation that requires them to kill another agent. If that same agent is *happy* or *depressed*, however, they might not have the appropriate motivation to perform such a violent action.

5 Architecture

5.1 Multi-Agent System

We use the JASON framework for belief-desire-intention (BDI) agents [2], programming our agents in the AgentSpeak language.

The VAD emotional model is represented inside each agent as a set of beliefs. Each agent has beliefs for its *valence*, *arousal* and *dominance* levels, each of which can take the value of low, medium or high. This combination of VAD values creates one of the 27 emotional states shown in figure 1, affecting whether or not an agent breaks from its permitted or obliged behaviour.

5.2 Institutional Framework

To describe our institutional model, we use instAL [3], a DSL for describing institutions that compiles to AnsProlog, a declarative programming language for Answer Set Programming (ASP). instAL's semantics are based upon the Situation Calculus [9] and the Event Calculus [5]. It is used to describe how external events generate institutional events, which then can initiate or terminate fluents that hold at certain instances in time. These fluents can include the permissions and obligations that describe what an agent is permitted or obligated to do at specific points in time.

For example, if an agent with the role of *dispatcher* leaves the stage, it generates the *absentation* Propp move in the institution:

```
1 leaveStage(X) generates intAbsentation(X) if
      role(X, dispatcher), activeFunction(
      interdiction);
```

The *absentation* institutional event gives the crocodile permission to enter the stage if there are any sausages on the stage. It also terminates the permission of the absented agent to leave the stage, as they have already done so:

```
1 intAbsentation(X) initiates perm(enterStage(
      croc)) if objStage(sausages);
2 intAbsentation(X) terminates onStage(X), perm(
      leaveStage(X));
```

instAL rules like those shown above are compiled into AnsProlog ASP rules. Once the instAL model is compiled to AnsProlog, we use the *clingo* answer set solver [4] to ground the logical variables, and 'solve' queries by finding all permissions and obligations that apply to any agents, given a sequence of events as the query input. The agents' percepts are then updated with their permitted and obliged actions from that moment in time onwards.

5.3 Bath Sensor Framework

The Bath Sensor Framework (BSF) [6] is a framework supporting publish/subscribe-style communication between distributed software components, in this case connecting intelligent agents with their virtual environments. It uses the XMPP publish/subscribe protocol to allow the communication between agents and their environments. Each agent subscribes to receive notifications of environment changes via XMPP server, which relays messages between publishers and subscribers. If any environment change occurs, all subscribed agents are informed of the changes.

This allows agents' environments to be created using entirely different technologies and programming languages from the agents themselves. In our case, BSF is especially useful as the animation engine that acts as the agents' environment is written in Javascript and runs in the browser. This means that the clingo solver and JASON agent

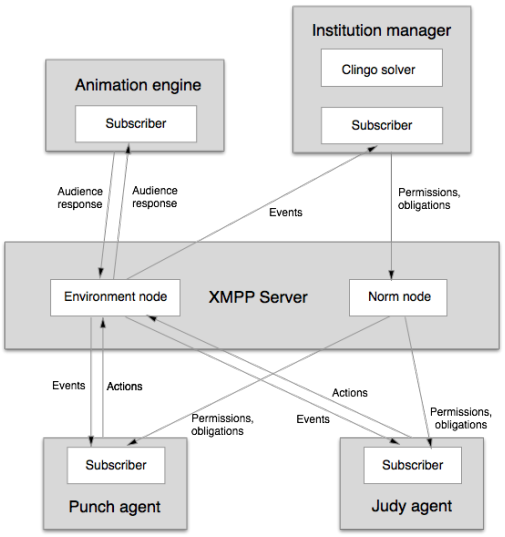


Figure 2. System architecture

framework can run on a central web server and communicate to any connected clients using BSF and XMPP.

Figure 2 shows how BSF is used to coordinate the components of the system. An XMPP server runs two publish/subscribe nodes. One node is for events related to changes in the environment (the *environment* node), the other is for changes in agents' permissions and obligations (the *norm* node).

All agents (in this case, Punch, Judy, the Policeman, etc) are subscribed to both the environment and norm nodes. They can also publish events to the environment node, but not the norm node. Only the institution manager (connected to the *clingo* solver) can publish permissions and obligations to the norm node. This manager (labelled in figure 2 as *institution manager*) is subscribed to the environment node of the XMPP server, watching it for events. These events then get passed to the *clingo* solver with the institutional model, which outputs the new permissions and obligations, publishing them to the norm node.

The animation engine is subscribed to the environment node, watching it for any events that need animating for the puppet show. In addition, it can publish input from the audience ('cheers' or 'boos') as events to the same node.

5.4 Animation

The animation engine that shows the visual output of the agents actions is written in Javascript and the Phaser game framework. It runs entirely in a browser, and communicates with BSF using the Strophe XMPP library.

If the user allows the program access to their microphone, they can cheer or boo the actions of the agents by shouting into the microphone. Otherwise, they can simulate these actions by clicking on 'cheer' or 'boo' buttons at the bottom of the screen.

6 Audience Interaction

The puppet show is designed to be run in front of either a single user's computer, or on a large display in front of an audience. The

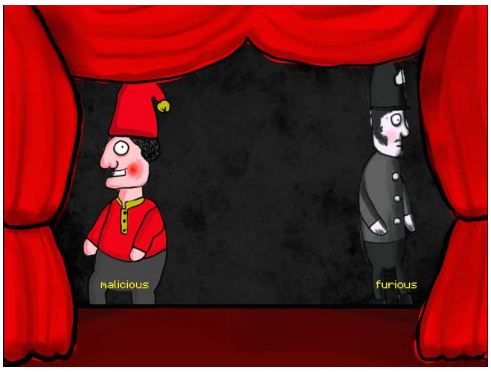


Figure 3. A screenshot of the Punch and Judy show

user/audience is instructed to cheer or boo the actions of the characters of the show, which will be picked up by a microphone and 'heard' by the agents. This will then affect the emotional state of the agents and change the actions they make in the show. Their actions are constrained by the set of 'Punch and Judy' world norms as described in the institutional model.

There are many different ways in which the audience's responses can affect the outcomes of the show. If the audience craves a more 'traditional' Punch and Judy experience, then they can cheer Punch into beating and killing all of his adversaries (including his wife, Judy). Alternatively, a more mischievous audience could goad Judy into killing Punch and then taking over his role as sadist and killer for the rest of the show. The narrative outcomes are dependent on how the audience responds to the action, yet still conform to the rules of the Punch and Judy story world.

7 Conclusion

With our approach to interactive narrative generation, we regulate the rules of the story domain using an institutional model. This model describes what each agent is permitted and obligated to do at any point in the story. This approach alone would be too rigid, however. Though the audience's interactions (cheering or booing) may alter the course of the narrative, the agents would still have to blindly follow a pre-determined set of paths. By giving our agents emotional models that change their willingness to follow the narrative, a degree of unpredictability is added to each run-through of the show, giving the impression that the agents are indeed characters capable of free will.

REFERENCES

[1] Junghyun Ahn, Stéphane Gobron, David Garcia, Quentin Silvestre, Daniel Thalmann, and Ronan Boulic, 'An NVC emotional model for conversational virtual humans in a 3d chatting environment', in *Articulated Motion and Deformable Objects*, 47–57, Springer, (2012).

[2] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*, volume 8, John Wiley & Sons, 2007.

[3] Owen Cliffe, Marina De Vos, and Julian Padget, 'Specifying and reasoning about multiple institutions', in *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, 67–85, Springer, (2007).

[4] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider, 'Potassco: The potsdam answer set solving collection', *Ai Communications*, **24**(2), 107–124, (2011).

[5] Robert Kowalski and Marek Sergot, 'A logic-based calculus of events', in *Foundations of knowledge base management*, 23–55, Springer, (1989).

[6] Jeehang Lee, Vincent Baines, and Julian Padget, 'Decoupling cognitive agents and virtual environments', in *Cognitive Agents for Virtual Environments*, eds., Frank Dignum, Cyril Brom, Koen Hindriks, Martin Beer, and Deborah Richards, volume 7764 of *Lecture Notes in Computer Science*, 17–36, Springer Berlin Heidelberg, (2013).
[7] Pablo Noriega, *Agent mediated auctions: the fishmarket metaphor*, Citeseer, 1999.
[8] Vladimir Propp, 'Morphology of the folktale. 1928', *Trans. Svatava Pirkova-Jakobson. 2nd ed. Austin: U of Texas P*, (1968).
[9] Raymond Reiter, 'The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression', *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, **27**, 359–380, (1991).
[10] James A Russell, 'A circumplex model of affect.', *Journal of personality and social psychology*, **39**(6), 1161, (1980).

Search and Recall for RTS Tactical Scenarios

Jason Traish, James Tulip and Wayne Moore [1]

Abstract. The success of a Real-Time Strategy agent is heavily dependent on its ability to respond well to a large number of diverse tactical situations. We present a novel method of tactical decision making called Search and Recall (S&R) which is a hybrid of Search and Case Based Reasoning (CBR) methods. S&R allows an agent to learn and retain strategies discovered over the agent's history of play, and to adapt quickly in novel circumstances.

The sense of memory that S&R provides an RTS AI agent allows it to improve its performance over time as better responses are discovered. S&R demonstrates an minimum win rate of 92% in standard scenarios evaluated in this paper.

S&R decouples search from the main game loop which allows arbitrary computational complexity and execution time for search simulations. Meanwhile in-game decision making is based on CBR and remains fast and simple.

This paper presents an S&R model which extends the ability of an RTS AI agent to deal with complex tactical situations. These situations include special unit abilities, fog of war, path finding, collision detection and terrain analysis.

1 Introduction

Real-time strategy (RTS) games are a popular genre of commercial games that require substantial practice, skill and experience to master. In order to conquer an opponent, a player must manage a number of in-game systems with precision, using a large number of possible commands. In-game systems include research, economics, exploration, managing an army, and executing a strategy with the potential of defeating the opponent's strategy. On top of all this complexity a player is expected to complete all these tasks in a real-time environment of uncertainty.

The number of in-game systems and possible commands illustrate the complexity of the RTS genre and form the basis of it's appeal to players and researchers alike. Developing an RTS agent poses many challenges that are not present in traditional strategy board game environments such as GO and Chess. In particular, the large number of units and possible commands, the uncertain environment, the effect of terrain, and the real-time execution constraints are unique to the computer based RTS genre.

It is very difficult to write scripted agents that vary their responses in different situations. This results in easily exploited AI agents which fail to give experienced players an enjoyable challenge.

Case Based Reasoning is one approach that has been used to create adaptive RTS AI agents [1, 2, 8, 9]. Search based methods have also became a point of interest to the RTS research community [3, 4, 12]. However, both approaches have intrinsic limitations.

[1] Charles Sturt University, Mining Lab, Australia, email: {jtraish & jtulip} @csu.edu.au & wmoore@lisp.com.au

1.1 Case Based Reasoning and Search in RTS AI

Case based reasoning (CBR) methods have been used successfully to create adaptive RTS agents. In general, such methods store plans with an associated game state and use this data to reason about future encounters.

Aha et al. [1] demonstrated a CBR agent capable of identifying and adapting to a randomly selected opponent which demonstrated good results. Their agent relied on the availability of a set of pre-generated responses, each capable of winning against an opponent from a given position.

McGinty et al. [8] improved CBR approaches by changing the structuring and case retrieval approach, leading to significantly better results. Their agent demonstrated a high win rate in experiments with imperfect information. Other CBR methods have focused on the use of recorded human player interactions to make decisions [2, 9].

However, while CBR has been successful in creating adaptive RTS agents, they face a number of challenges. Responses derived from human players can be of inconsistent quality due to the diversity of human player skills and the nature of human play. Standard CBR approaches are also ill equipped to make decisions if there is no similar recorded context.

As a result, search based methods, and in particular Monte Carlo simulations have gained the interest of the RTS AI research community [3, 4, 5, 6, 12]. Search based methods enable an agent to adapt in real-time to whatever circumstances it is currently facing, assuming the simulator can correctly predict the outcome of a given response action. Significant research on adaptive agents using search based techniques has been performed in the context Chess and GO [7] and the application of such techniques to RTS games is an attractive prospect.

However, complexities such as path finding and collision detection are required for an agent to appropriately handle commercial game type tactical situations. Such situations include moving units in a environment affected by terrain, or engaging armies of many varied unit types, some with special abilities.

The complexity inherent in commercial RTS games places huge computational demands on the simulations required to perform a search for a tactical solution. For this reason most of the published search simulation approaches are very simple relative to the demands of fully realised commercial game agents and ignore issues such as terrain, path finding, and collisions between units.

The problem is that simulations conducted within the game loop are heavily constrained to execute in an extremely limited amount of time, due to the demands of other aspects of the game loop such as animation and rendering.

In the rest of this paper we present a hybrid search/CBR approach called Search and Recall (S&R) which enables simulations capable of dealing with commercial grade RTS game complexity, while offering CBR level in-game performance. We demonstrate these capabil-

ities in the context of Starcraft Broodwar; a commercial RTS which has become a popular RTS AI research platform.

The main contribution of this work is to demonstrate the utility of responses generated using Search simulations as recorded responses in a CBR-like database. The technique was inspired by case base reasoning literature that focused on constructing databases using player responses [10]. We also demonstrate an approach for making computationally intensive search simulations feasible in the context of a real-time game.

2 Search and Recall - Overview

Search and Recall (S&R) is a novel method of tactical decision making which is a hybrid of Search and Case Based Reasoning (CBR) methods. It allows an agent to learn and retain strategies discovered over the agent's history of play, and to adapt quickly in novel circumstances.

Similarly to CBR methods, S&R uses a database of previously discovered successful responses associated with a collection of identified game states. S&R agents use these responses to quickly identify a solution without extensive simulation within the game loop. However, unlike other CBR methods, S&R does not populate it's response database with a static set of game states identified from previously played games. Rather, it populates the database dynamically with the results of search simulations conducted in response to actual game states encountered during play.

By combining the adaptive learning of MCS with the memory of CBR, S&R allows an agent to improve the quality of its responses over the course of multiple games.

In essence, we decouple the search tasks from the game loop by allowing them to execute asynchronously and in parallel with the game loop. Searches are pushed into concurrent threads, allowing them to take as long as necessary without delaying game rendering. The agent makes its decisions based on its current database of solutions, and the search tasks update that database asynchronously with the results of new simulations based on possible responses to the current game state. As many searches can be carried out as are appropriate to the CPU resources available to the game.

Search time is limited only by the length of a game or an arbitrary stopping condition, and is substantially longer than the 5ms generally allocated for an agent's decision making process within the standard game loop. The downside is that the longer it takes to evaluate potential decisions the more likely it is that the response will come too late to be useful in the current situation. However, the next time a similar situation is encountered, the simulation results will be available in the CBR database (response library) ready for near instant access.

Search results are used to update a CBR like database as they become available, and the AI task within the game loop is reduced to selecting the appropriate response as in a conventional CBR system.

We apply this architecture in the context of the commercial game Starcraft Broodwar. Starcraft is an immensely popular and sophisticated RTS game, famous for its balanced asymmetric game play and status as a professional spectator sport in Korea. Starcraft Broodwar is a version of Starcraft for which an external programming interface has been developed called the Brood War API (BWAPI). The availability of BWAPI has made Broodwar an attractive platform for RTS AI research.

3 Search and Recall - Agent Components

The S&R agent is composed of a recall-playback component (RPC), a search component (SC), and a response library (RL). This basic architecture is illustrated in Figure 1. The RPC component acts as coordinator for the agent and interacts with the BWAPI interface. As soon as a Broodwar game begins the S&R agent starts the recall-playback component and initialises the search component with a number of threads.

3.1 Recall/Playback Component (RPC)

The RPC matches the current game state against the game states currently recorded in the response library. Game states in the database are identified by a simplified descriptor containing only the number and types of unit present.

The RPC then retrieves the response associated with the current game state from the response library. The response associated with a game state is always the most favourable response generated by the search simulations carried out in the search component. If no matching game state is found, the RPC assigns random behaviours to the agent's units. If a response was loaded earlier from a previous game state then those previous behaviours are not changed.

The RPC has a simulator similar to those being used for searching. It uses this to simulate a single time step using the unit actions specified in the response. This step is carried out in order to map from the actions specified in the response to a set of Broodwar commands that must be issued through the BWAPI interface. The raw actions that the units must perform are recorded (e.g. move[x,y], attack[unitId]) and forwarded to the BWAPI.

Although games states are identified in the RL only by the number and type of units present, actual game state is defined with considerably more information on unit positions, current unit states, what projectiles have been created, which units are damaged, and which weapons have entered their cool down periods. All of this information is captured from the BWAPI and sent through to the search component (SC) in addition to the number and type of units present in the scenario. The RPC buffers these changes in actual game state for the SC, updating the information used by that component as a basis for simulation only after 200 simulations have completed. This allows a sufficient number of searches associated with a particular game state to complete to be useful in subsequent games.

The execution of the RPC is constrained to take less than 5ms per frame since it executes as a part of the main game loop. This constraint is easily achieved since the simulator used to calculate the BWAPI commands simulates only a single time step.

3.2 Search Component (SC)

The search component is represented in Figure 1 as the Concurrent Search Simulators (CSS). It consists of a number of search threads which repeatedly run simulations for the combat scenario utilizing the current actual game state, a simulator engine, and a set of actions assigned to each unit in the scenario.

At the beginning of a simulation, each search thread is given the current identifying game state (unit numbers and types) as well as information describing the actual game state (terrain, unit positions, unit health, current unit action states, etc).

We randomly assign behaviours to each unit for each simulation so we can evaluate the effect of utilising different tactics on the outcome of a battle . If simulations complete quickly, many different

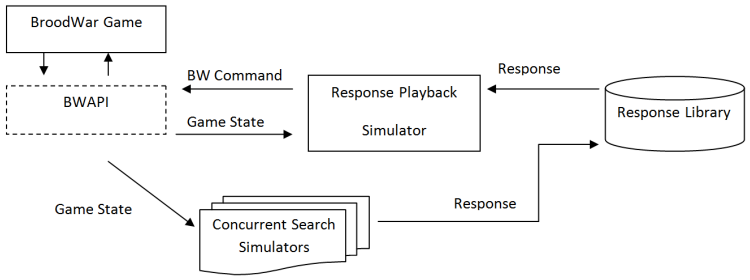


Figure 1. Search and recall agent process

possible outcomes can be calculated and used to update the solution available to the RPC before the time allotted to its execution within the game loop (5ms) expires. However, if it takes longer to simulate an outcome than the time Broodwar allows, then the result will not be available to the RPC during the current game loop. This results in the game agent taking longer to respond to a game state in real time, although simulation results do become available to the RPC over the next few game loop cycles as simulations complete.

When a simulation completes, the quality of the response is calculated as the total health percentage of the remaining allied units at the end of simulation. A quality of 0 is given for prediction in which all allied units are killed. This formula favours victories with lower casualties, and ranks all losses equally.

As in [6], the simulator is a mathematical model of the combat mechanics implemented in Broodwar, that allows simulations to be run without any frame rate derived speed limitations. As such, it is not an exact model of the combat mechanics implemented in Broodwar.

3.2.1 Simulators

Asynchronous execution of search allows the complexity and execution cost of the simulation engine used to be increased arbitrarily. In this work we explore the effect of increasing the complexity of the simulation engine used by evaluating the performance of two different simulators. These are:

1) Basic Simulator: This simulator handles unit health, shields, healing, attacking, and movement without collision or path finding. It can complete up to 2000 combat simulations per second per thread.
2) Complex Simulator: This simulator handles unit health, shields, healing, and attacking. However, the movement function detects collisions and finds paths around obstacles such as terrain and other units. Influence maps from [11] have also been integrated to support a 'kiting' behavior which has been added to the list of available behaviours. This simulator can complete only up to 200 combat simulations per thread per second.
   Kiting is a highly successful behaviour that fast moving ranged units can use against slower units. Kiting is the act of attacking an enemy unit and then moving away while reloading.

3.2.2 Response Divergence

A response grows stale the longer it is in effect. This is due to differences in mechanics between the Broodwar game and the simulator that even a very sophisticated model will find challenging to eliminate, in particular because there are random elements built into the Broodwar game engine. We call the differences between the simulated outcome and what actually happens in Broodwar as divergence. Divergence represents the cumulative error between the game states of the simulation and Broodwar as time passes.

Different game systems suffer differing amounts of divergence. While systems like health regeneration and attack damage are straight forward, other components such as attack cool downs are randomisied slightly, introducing small changes in combat outcomes. The precise mechanics of other systems such as path finding are unknown and this also increases the divergence of simulations from actual game encounters. Furthermore, an opponent model is not necessarily a precise model of the Broodwar AI, and this also leads to a large amount of divergence. Finally, the actual precise game state used to drive the search simulation that generated the response recorded in the database may differ from the precise current game state. If differences in precise unit location and health affect the outcome of the battle, divergence will occur.

3.2.3 Opponent Models

In order to combat the effects of divergence, solutions that generalise well are sought. The simulation outcome is heavily dependent on the strategy used by the opponent, so we attempt to find generalized solutions by taking the minimum of the solution quality score over a small set of opponent models. This favours the selection of robust strategies that are successful against a variety of opponent models for the response library (RL). In the current work this set of opponent models contains only 2 strategies; one using an 'Attack Weakest strategy, and the other using an 'Attack Closest' strategy.

3.2.4 Unit Behaviours and Grouping

A behaviour describes what action the unit should take in any given circumstance. A behaviour consists of a series of actions which a unit executes in sequence, moving on to the next action when the previous action is complete or appropriate conditions are met. For each behaviour we identify a primary action, and a secondary action which is applied if multiple targets are identified for the primary action. For example, if 'Attack Weakest' is the primary action, and all enemy units have the same health, then the secondary action 'Attack Closest', is applied. Behaviours are described in Table 1.

In order to allow the S&R agent some flexibility in terms of choosing and targeting particular units or types of enemy units, we provide the agent with the ability to separate the enemy into groups.

When setting up a simulation, not only are a random set of behaviours assigned to the agent's units, but the enemy is divided into 4 random groups. Actions are then made specific to groups. For example, the generic "Attack Closest" behaviour becomes "Attack Closest in Group 1". Grouping allows the agent to create plans that can focus fire individual or groups of units. This greatly increases the degree of freedom with which the agent can respond to situations.

3.3 Response Library Component (RLC)

The S&R agent receives its recall ability from the use of the response library. The response library is responsible for the storage and communication of the best recorded responses from the search simulations. The database is updated asynchronously by the SC, and queried from within the game loop by the RPC. It acts as a constantly growing and improving database of best seen responses to recorded tactical situations.

3.3.1 Game State and Response Descriptors

Preliminary testing identified that actual game state needed to be generalized for successful game state matching to occur. Furthermore, only a small number of game state attributes were required for the agent to adapt competently. Hence, the attributes used to identify game state within the RLC include only the number and type of each unit involved in the current scenario. Adding more detailed game state descriptors such as those describing unit health or position causes an explosion of possible states, this drastically shortens the time that a game remains in a particular state, and makes it difficult to match the current game state with a state recorded in the response library.

Describing game state by only the number and type of units involved results in relatively stable states that recur sufficiently frequently to make matching effective, and balances the frequency of response adaption. This approach effectively forces the chosen response to change only to when units are removed from or added to the game.

In addition to the game state information that is used as a key in the response library, each entry in the response database records the behaviour assigned to each unit, and the groupings assigned to the enemy units.

Response behaviours do not correspond with BWAPI commands: they need to be mapped into BWAPI commands by the simulator associated with the RPC.

4 Experimental Setup

The following experiments contain four tactical scenarios that an agent cannot resolve with a singular response. These are illustrated in Figure 2 and listed below:

A) 3 Zealots vs 3 Vultures (Attack Closest agent): This scenario pits 3 fast ranged units (Vultures) controlled by the agent against 3 slow close attack units (Zealots). This scenario favours the kiting strategy as it is extremely difficult to solve without it.
B) 6 Fast Zerglings vs 2 Dragoons (Attack Closest agent): This scenario pits 2 strong ranged units (Dragoons) controlled by the agent against 6 fast close attack units (Fast Zerglings). Once again a kiting solution is favoured, but far more precision is required to make this work.
C) 3 Zealots and 3 Dragoons vs 3 Zealots and 3 Dragoons (Default AI): This is a symmetrical scenario pitting ranged (Dragoons) and close attack (Zealots) units against each other. Precise control over unit attacks which enemy unit as well as unit placement is required to be successful.
D) 8 Dragoons vs 8 Dragoons (Default AI): Once again this is a symmetrical scenario that pits equal numbers of ranged units against each other. Control of attack strategy is important in this scenario, but unit placement is less important than in Scenario C.

The experimental setup is based on work by [5] although the experimental setups for scenarios A and B differ from Churchill's implementation. Due to problems encountered with the BroodWar AI's default behaviour it was replaced with a scripted agent designed to constantly attack the closest unit.

Each scenario is run against a particular configuration of the S&R agent for a total of 200 games at an acceleration of 5ms per frame. This is necessary since due to stochastic variation between games, the outcome of an actual game is not completely deterministic. The scores recorded in Table 2 are defined by the following function to the nearest percentage.

$$Score = (wins + draws/2)/200$$

Our experiment compares several different configurations of the SR agent. The performance of the basic and the complex simulator engine are compared in two modes: in pure search mode (ie without access to any stored responses), and in combined search and recall mode (with access to stored responses). This tests whether there is any advantage in retaining results from earlier simulations. For comparison purposes, the performance of two scripted agents was also evaluated: one based on an 'Attack Closest' strategy, and another which favours Kiting. Each configuration or agent is tested on the four scenarios listed above.

For the S&R agents, each configuration is initialised with a new empty response library at the beginning of the evaluations for all scenarios. All recorded responses are generated by simulations run during the actual games.

All S&R experiments utilise 4 threads within the SC for running simulations. Each search was limited to 2000 time steps although this number of steps was never reached. The results of the experiments are shown in Table 2.

5 Results and Discussion

The results of the experiments for the scripted agents show clearly that to do well in all four scenarios requires adaptive agent behaviour. The 'Attack Closest' scripted agent performs poorly in scenarios A and B, but is successful in scenarios C and D while the reverse is the case for the 'Kiting' scripted agent.

Results for the simple simulator, which does not have a kiting behaviour available are similar to the 'Attack Closest' scripted agent. This illustrates the importance of the simulator model containing a set of behaviours sufficient to cover what is required in a scenario.

On the other hand, results in scenarios A and B for the complex simulator show that agent clearly discovered and utilized the appropriate kiting behaviour. Results in Scenario A are stronger than in Scenario B, likely because the large speed difference between Vultures and Zerglings makes a wide range of successful kiting solutions relatively easy to find. In Scenario B, if the Dragoons performed a suboptimal action for even a small period they would lose to the larger numbers of Zerglings.

Table 1. Behaviour Descriptions

Behaviour	Primary Function	Secondary Function	Condition
G1, G2, G3 and G4	Attack unit of least health in group X	Attack closest unit in group X	No units in group X
Attack Closest	Attack closest unit	Attack unit of least health	N/A
Attack Wounded	Attack unit of least health	Attack closest unit	N/A
Kite	Attack unit of least health in range when ready to fire	Move away from all enemies and terrain	N/A

Table 2. Experiment 1 Results. S&R: Search and Recall. IM: Influence Map.

Setup	Churchill Search	Search	S&R	Search (IM)	S&R (IM)	Attack Closest	Kiter
A	0.81	0	0	0.96	1.00	0	1.00
B	0.65	0	0	0.65	0.92	0	1.00
C	0.95	0.95	0.80	0.76	0.94	0.77	0.26
D	0.96	1.00	1.00	1.00	1.00	0.97	0.14

The results for the complex simulator with recall enabled are better than for search alone, indicating that the recall capability provides a considerable advantage. The advantage conferred by the recall ability is much greater in Scenario B than in Scenario A. This suggests that the advantage of accumulating knowledge in the response database is greatest when solutions are relatively exact, and the exploration of the solution space is relatively slow.

Results for the simple simulator are equivalent or better than the complex simulator for scenarios C and D. This indicates that the range of behaviours available to the simple simulator are sufficient in these scenarios, and that the complexities introduced for the complex simulator have little impact in these scenarios. This result is not terribly surprising since the influence map affects only the kiting behaviour which is not necessary in these scenarios, and the close ranged combat and lack of terrain features in these scenarios reduces the impact of the path finding capability of the complex simulator. Given these considerations, it may be that the much greater number of simulations that the simple simulator can perform (2000 vs 200 per second) allows it to find better solutions than the complex simulator.

Results for scenario C yield are the most varied. The winning solutions for this scenario required more complex behaviours than in the other scenarios. Scenario C is similar in some respects to Scenario B with its rigorous success requirements.

Results for the complex simulator in Scenario C show a large difference between search only and combined search and recall. Once again it appears that that the recall capability becomes a significant advantage when solutions are hard to find and the exploration of solution space is slow.

Results degrade when recall is enabled for the simple simulator. It is likely that this is an example of the effects of divergence. The simulator has discovered an action set that is effective in simulation, but that does not translate well into the actual game. This indicates the importance of the simulator's combat model being a close match to the actual game's.

Results for Scenario D are both extremely strong and uniform across both the simple and complex simulators, both with and without recall enabled. This is probably a result of the scenario being relatively easy to solve, as indicated by the strong result also generated by the 'Attack Closest' scripted agent.

Over all scenarios, the strongest performance is shown by the complex simulator with recall enabled. This configuration of the S&R agent adapts strongly to all scenarios, even though its performance without recall enabled is relatively weak. The result is important, since it indicates that the build up of experience over many game cycles becomes greatly beneficial when solutions are hard to find, and simulation rates are slow. This is exactly the situation faced when attempting to apply accurate simulation models to complex commercial grade RTS AI problems.

Note that for all the search based configurations, results between zero and one are in some ways a measure of divergence, since the simulations return what they estimate as a winning solution or a loss. Solutions that win sometimes reflect differences between what the simulators calculate and what actually happens in Broodwar. This tends to impact weaker solutions to a greater extent, resulting in lower scores where search is less effective. Given this interpretation of each scenario score, it is an important result that the scores for the complex simulator with recall enabled are consistently high across all scenarios. This reflects relatively little divergence between what the complex simulator predicts and what happens in Broodwar, given a sufficient accumulation of simulations, and the capacity to retain the results.

Another important result is that the benefits of recall are delivered to the agent relatively quickly. There is a marked improvement for the complex simulator with recall enabled in the difficult scenarios even though the scenario is evolving in real time. This indicates that the advantage of receiving high quality solutions outweighs the disadvantage of them taking more than a game cycle to calculate.

In comparison with Churchill's results, the complex simulator with recall enabled dominates by a large margin in all but Scenario C, where it is only marginally weaker. Given the divergence interpretation of the evaluation scores, the results suggest that the complex simulator is a much closer approximation of the Broodwar combat mechanics, and that the predictions made by the complex simulator are much more accurate. The 'complex simulator with recall' approach is an approach worth pursuing.

6 Conclusions and Future Work

Overall the results of this preliminary study can be summed up as: high quality responses are worth remembering, when solutions are hard to find, the exploration rate of the solution space is low, and when the fidelity of the simulations is high.

The results strongly indicate that retention of results from search simulations is worthwhile, and that Search and Recall is a useful approach. This eliminates the need for a huge and uneven quality

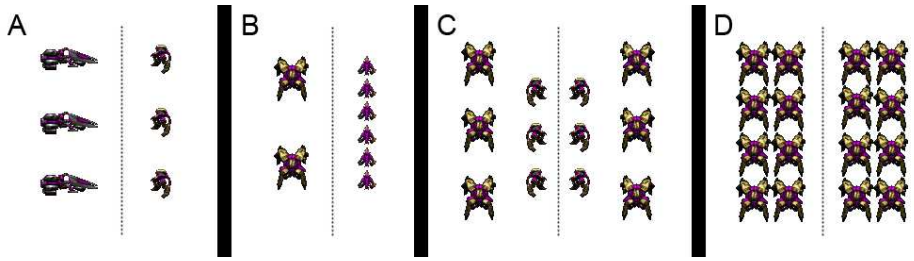


Figure 2. Experimental Setup

database of pre-played games on which to base CBR, and allows the situations a game AI can respond intelligently to grow over time. At the same time it guarantees fast decision making within the game loop.

An important implication of the proposed architecture is that because simulations are decoupled from the game loop, they become amenable to parallel, distributed, or offline processing. The exact actual game states sent to the SC, could instead be sent out over the network, or logged for later processing. Regardless of whether results arrive in time to advantage the S&R agent in the current game, the fact that the results are generated improves the response database over time, even when the game is not being played. Another implication is that simulation results from many separate instances of a game can be shared between games, allowing games to cooperate in improving the AI for all games.

A final implication is that simulations are not restricted to the CPU capacity of an ordinary gaming PC. Simulations could be conducted on server farms or supercomputers in the cloud, and the results used to update a global database available to all instances of a game.

Because the constraints on execution times and hence simulations complexity have been eased, future work could extend simulation models to scenarios of greater complexity such as working with terrain and larger unit encounters. It would also be interesting to explore the feasibility and utility of more detailed game state descriptors, and the associated much larger response databases required.

Once response databases become larger and more populated, game progression paths through state space and discovering general patterns of game progression could prove interesting. The sensitivity of results to the range of available behaviours also indicates that further work into more complex behaviour sets is also warranted.

S&R removes computational execution time restrictions on search but retains the ability of search based agents to adapt to new situations. The S&R agent model allows simulators used in searches to use much more complex models to deal with complex tactical situations. Simulators can include path finding, unit and terrain collision avoidance, and specialized behaviours. These complex simulators greatly improve the fidelity of the results produced, which reduces the divergence between predicted outcomes and those produced by the game. This makes the S&R method potentially useful in applying search techniques to commercial grade levels of combat scenario complexity.

REFERENCES

[1] David W. Aha, Matthew Molineaux, and Marc Ponsen, 'Learning to win: Case-based plan selection in a real-time strategy game', in *Case-Based Reasoning Research and Development*, volume 3620 of *Lecture Notes in Computer Science*, 5–20, Springer Berlin / Heidelberg, (2005).

[2] Klaus-Dieter Althoff, Ralph Bergmann, Mirjam Minor, Alexandre Hanft, Neha Sugandh, Santiago Ontan, and Ashwin Ram, 'Real-time plan adaptation for case-based planning in real-time strategy games', in *Advances in Case-Based Reasoning*, volume 5239 of *Lecture Notes in Computer Science*, 533–547, Springer Berlin / Heidelberg, (2008).

[3] Radha-Krishna Balla and Alan Fern, 'Uct for tactical assault planning in real-time strategy games', pp. 40–45. Morgan Kaufmann Publishers Inc., (2009).

[4] Michael Chung, Michael Buro, and Jonathan Schaeffer, 'Monte carlo planning in rts games', in *IEEE Symposium on Computational Intelligence And Games (Cig)*, (2005).

[5] David Churchill and Michael Buro, 'Incorporating search algorithms into rts game agents', in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, (2012).

[6] David Churchill, Abdallah Saffidine, and Michael Buro, 'Fast heuristic search for rts game combat scenarios', *AIIDE*, (2012).

[7] Leandro Soriano Marcolino and Hitoshi Matsubara, 'Multi-agent monte carlo go', pp. 21–28. International Foundation for Autonomous Agents and Multiagent Systems, (2011).

[8] Lorraine McGinty, David Wilson, Ben Weber, and Michael Mateas, 'Conceptual neighborhoods for retrieval in case-based reasoning', in *Case-Based Reasoning Research and Development*, volume 5650 of *Lecture Notes in Computer Science*, 343–357, Springer Berlin / Heidelberg, (2009).

[9] Manish Mehta and Ashwin Ram, 'Runtime behavior adaptation for real-time interactive games', *IEEE Transactions on Computational Intelligence and Ai in Games*, **1**(3), 187–199, (2009).

[10] Santiago Ontan, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Case-based planning and execution for real-time strategy games, 2007.

[11] Alberto Uriarte and Santiago Ontan, 'Kiting in rts games using influence maps'. AIIDE, (2012).

[12] Wang Zhe, Kien Quang Nguyen, Ruck Thawonmas, and Frank Rinaldo, 'Using monte-carlo planning for micro-management in starcraft', in *GAMEON Asia*, pp. 33–35, Japan, (2012).

Follow-up on Automatic Story Clustering for Interactive Narrative Authoring

Michal Bída, Martin Černý and Cyril Brom[1]

Abstract. One of the challenges in designing storytelling systems is the evaluation of resulting narratives. As the story space is usually extremely large even for very short stories, it is often unfeasible to evaluate every story generated in the system by hand. To help the system designers to maintain control over the generated stories a general method for semi-automatic evaluation of narrative systems based on clustering of similar stories has been proposed. In this paper we report on further progress in this endeavor. We added new distance metrics and evaluated them on the same domain with additional data. We have also successfully applied the method to a very different domain. Further, we made first steps towards automatic story space exploration with a random user.

1 INTRODUCTION

Developing interactive storytelling (IS) systems is a challenging task involving multi-disciplinary knowledge, yet a number of IS systems was developed in the past, such as Façade [1], ORIENT [2] or FearNot! [3]. Bída et al. [4] notes that the evaluation of complex IS systems is a demanding process often requiring extensive effort. To mitigate this, the authors propose a computer assisted method of story evaluation based on clustering the stories into clusters according to their similarity. The general idea is that by meaningful clustering of the stories into groups the human designer will not be required to evaluate all the stories, but only few from each cluster and thus save development time. Authors also reported on the performance of the method on two domains - SimDate3D (SD) Level One and SD Level Two [5]. The first results indicated that the main metric could scale better than the other metrics on the complex domain of SD Level Two.

In this paper we report on further progress in a similar endeavor. Firstly, we have added two new features for the clustering algorithm in the SD domain - a) automatic extraction of sub-scenes from the recorded story and b) condensed tension difference curve based on the sub-scenes. We have managed to reproduce previous results on an extended domain of SD Level Two getting good performance using some of the new features. Secondly, we have implemented a random user that tries to explore the story space of SD Level Two by playing differently than an input set of previous stories hence exploring parts of story space not seen in the input set of stories. We show the performance of the metrics in distinguishing between stories generated by the random user and the original set of stories. Thirdly, we have applied the method on stories generated by the MOSS system [6] in order to investigate the performance of the method on a different domain.

Aside from the work mentioned, little has been done on story clustering. Weyhrauch [7] implemented several evaluation functions specific for his emergent narrative system. Ontañón and Zhu [8] proposed an analogy-based story generation system, where they evaluated the quality of resulting stories by measuring their similarity to "source" stories (input human-made stories). Compared to the approach in this paper, they were solving a problem of generation of the stories rather than the analysis of the stories.

This paper is organized as follows: First, we will describe the story domains we used in the experiments, then we will discuss updates of the method for narrative analysis and afterwards we present results of the new experiments. We will conclude the paper with discussion and future work.

Figure 1. SimDate3D Level Two screenshot showing Thomas and Nataly in the park with emoticons above their heads having a conversation about music.

2 DOMAINS

The experiments detailed in this paper have been conducted on IS system SD Level Two detailed in [5] and MOSS system [6].

SD game (Figure 1) is a 3D dating game taking place in a virtual city, with three protagonists: Thomas, Barbara and Nataly. The characters communicate through comic-like bubbles with emoticons indicating the general topic of the conversation

[1] Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic.
Email: {michal.bida,cerny.m}@gmail.com, brom@ksvi.mff.cuni.cz

(see Figure 1). The user partially controls one of the characters actions (typically Thomas). The users' goal is to gain the highest score by achieving certain kind of things, e.g. Thomas kissing one of the girls. The game features four possible endings.

The MOSS system [6] developed by M. Sarlej generates short stories with morals (e.g. greed, retribution, etc) in three domains (animals, family and fairytale). Each moral has its own emotional pattern that is used to generate stories with moral of a particular category. Internally the system uses Prolog abstraction to generate the stories, which is then translated to human readable text with Perl scripts. We worked directly with the internal prolog representation of the stories, which we parsed and analyzed with the system.

3 METHOD

Here, we will briefly overview the method we evaluated (which is given in detail in [4]). The main idea is to cluster the resulting narratives of a given IS system into groups of similar stories. The human designer then needs to see only several stories from each group to gain sufficient understanding of all the stories the cluster contains, saving development time. The clustering is done with the k-means algorithm. In the previous work, the clustering was based on two general features of stories: a) story action sequence and b) story tension (dramatic) curve.

The story action sequence is created by taking the sequence of actions done by all the characters in the story. Each of the actions available in the domain is assigned a letter and the sequence of these letters forms the *action string*. This way, standard string distance metrics (*Levenshtein*, *Jaro-Winkler* and *Jaccard* distances) are applicable to measure similarity between *action strings* representing different stories. In previous work, Jaccard distance has been shown to be of little use for story clustering in SD domains and is therefore tested here only for MOSS stories.

The *tension curve* is extracted from emotions experienced by the story protagonists. In SD this is straightforward as the characters are equipped with emotion model. The tension in SD is computed as follows: Every 250 ms we make a snapshot of all characters' emotions. Then we take the sum of these emotions where every positive emotion is counted with a minus sign and every negative emotion is counted with a plus sign. The resulting number encodes the tension value at the moment. The *tension curve* is then simply the piecewise linear function defined by these values.

In the MOSS system the emotions are also defined explicitly as a part of the generated stories. We again take the sum of positive and negative emotions at each time point of the story and the resulting value is the tension value at the specific time point of the story.

We propose two new features for clustering the SD stories: *sub-scene sequence string* and *condensed tension difference curve*. A sub-scene is a time span in the story where a) the set of characters that are in the proximity of the main protagonist do not change and b) the location of the main protagonist does not change. Let us suppose that Thomas (the main protagonist) is with Barbara (character) at the restaurant (place) – this is one sub-scene. After 5 minutes, Nataly arrives and joins them. At this moment, the old sub-scene ends and a new one begins. The new sub-scene features Thomas, Barbara and Nataly at the restaurant. Sub-scenes are extracted automatically from the story logs. The time span of sub-scenes varies from 5 seconds (enforced lower limit) to the whole duration of the story.

To measure distance between sub-scene sequences we assign strings to sub-scenes in the following way: one letter represents a location of the story (e.g. P for park) and the consecutive letters represent characters in the sub-scene (e.g. T for Thomas; one letter per each character present). For example, the "TBR" string represents a sub-scene where Thomas is with Barbara at the restaurant. The sub-scene sequence string is simply a concatenation of the individual strings. We then apply string distance algorithms as is the case with action strings.

Condensed tension difference curve is extracted from sub-scenes. We look at the tension value at the beginning and at the end of the sub-scene. The difference between these two values represents the tension difference for respective sub-scene. The condensed tension difference curve is defined as a sequence of all of these differences.

We have not implemented sub-scenes for MOSS stories, because the MOSS stories are already relatively short and composed of at most two sub-scenes. To check whether the clustering really captures non-trivial properties of the stories, we also tested difference in story length as distance metric for the MOSS domain.

All pairwise distances between stories have been computed, normalized and standardized prior to clustering.

3.1 Story space exploration with a random user

IS systems are often interactive, requiring a human user in the loop. Exploring the story space of such systems may be problematic as one needs many users and many story runs to get a reasonable coverage of the story space. For semi-automatic analysis the designer would benefit from an algorithm that would be able to explore parts of the story space automatically. For SD we have implemented a random user that is able to play the game alone. In addition, the random user tries to steer away from a given set of stories. Hence exploring parts of story space not covered in the given set of stories revealing previously unseen parts of the story space to the designer. This is achieved as follows: The random user (controlling Thomas) extracts the sub-scene sequences from the given set of stories and then tries to achieve a different sub-scene sequence in the story he is playing in. E.g., if the random users detects that most of the given stories started with characters at the restaurant, he will try to change location in the story by inviting the characters for example to the cinema and so forth for the second and the n-th sub-scene in the sequence. The random user has simple domain-specific knowledge that limits the actions he considers only to those contextually appropriate (e.g. he does no try to become intimate with a girl at the restaurant).

3.2 Evaluating clustering quality

As there is no generally accepted method for evaluating the quality of a clustering independent of the application, we use ad hoc method suitable for our scenario. Intuitively, a clustering is good, if stories in the same cluster have many features in common. Let us have a feature function $f: S \rightarrow V$, where S is the set of all possible stories and V is a finite set representing possible values of a feature the designer might be interested in.

For a cluster $X \subset S$ we define *precision with respect to f* as the proportional size of its largest subset sharing the same value of the feature:

$$precision(X, f) = \frac{\max\{|M| : M \subset X, \forall m, n \in M : f(m) = f(n)\}}{|X|}$$

In other words, precision of 0.62 means 62% of stories in the cluster produce the same value for *f*. The precision of the whole clustering is simply the average of per-cluster precisions. A system that clusters stories can be considered useful, if it provides high precision across multiple domains and multiple features.

In the experiments, we tested three features: the ending of the story (Experiment 1), the type of user (random vs. human) that generated the story (Experiment 2) and the MOSS moral of the story (Experiment 3).

As k-means depends on random initialization we ran each analysis 100 times to get robust results. In further text, we always report the average precision of these 100 clustering runs. To provide a simple baseline to the measurements, we also tried assigning stories to clusters at random. Once again an average of 100 random assignments is measured.

To provide a more robust evaluation of the methodology, it would be best to measure precision with respect to similarity of stories as perceived by humans. This however poses multiple methodological issues. In our view, a biggest obstacle to human evaluation is finding a useful dataset. Since humans cannot effectively cluster more than a handful of stories, the dataset needs to be small, which is usually unsuitable for machine clustering as the algorithm can easily pickup artifacts in the data. We left this as a future work.

4 EXPERIMENT 1

In Experiment 1 we analyzed an extended dataset of 70 human play sessions of SD Level Two using additional features – sub-scenes sequence string distance and condensed tension difference curve based on sub-scenes. Precision is measured with respect to the ending of the story. A graph of the results is presented in Figure 2.

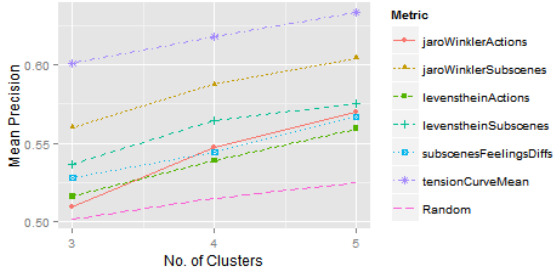


Figure 2. SD Level Two clustering results. Cluster precision weighted averages can be seen for three, four and five clusters (this is chosen arbitrarily based on that there are four possible endings). The results are averaged over 100 clustering runs with different initial cluster positions. The precision is calculated with respect to story ending.

As in previous work [4] we see that the tension curve outperforms other approaches in mean precision (0.6 for three clusters to 0.63 for five clusters). The interesting observation is that the sub-scene string sequence (metrics marked as "Subscenes" on Figures 2, 3, 4) outperform action strings (metrics marked as "Actions" on Figures 2, 3, 4) on this dataset. This indicates that sub-scene sequence is a meaningful feature in SD domain, relevant to story ending. Also note that Jaro-Winkler distance on sub-scenes (average 0.58) slightly outperforms Levenshtein (average 0.56). This is somewhat unexpected as Jaro-Winkler distance is usually a sub-par choice for clustering as it does not satisfy the triangle inequality. However this distance gives more weight to differences between first four characters of the string. The good performance of Jaro-Winkler on sub-scene sequences may then be explained by a large impact of the beginning of the story on its ending. Assigning higher weight to story start and/or story end might be an interesting extension of the approach as it would reflect the way stories are perceived by humans.

The compressed tension difference curve (metrics marked as "subscenesFeelingDiffs" on Figures 2, 3) scored on par with action strings distance metrics (average 0.55), but did not match the uncompressed original tension curve.

All metrics scored significantly better than the random cluster assignment. However compared to previous results [4] the addition of more stories resulted in lower precision for all previously measured metrics (tension curve and action strings). This might be partly caused by the larger size of the dataset, but it indicates that the metrics need to be made more robust.

Examples of the stories from this dataset and their clustering can be found in the appendix.

5 EXPERIMENT 2

In Experiment 2 we analyze a dataset containing 41 original human play sessions (as analyzed in [4]) and 66 randomly selected play sessions gathered from the random user. Precision is measured with respect to the type of the user that generated the story. A graph of the results is presented in Figure 3.

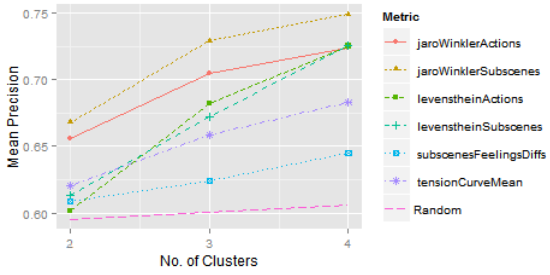


Figure 3. Experiment 2 clustering results. Figure shows the average precision of clustering with respect to the users that created the stories as a function of number of clusters.

The best metric for distinguishing between human and random user is Jaro-Winkler distance on sub-scenes (with precision 0.67 on two and 0.72 on four clusters). This can be explained again by the feature of the algorithm putting more weight on the first characters of the string. The random user tried to achieve different sub-scene sequence than the human users. Even though the story always begins the same (the first sub-scene is always the same), the random user immediately tried to change the sub-scene, so the second one differed from the average done by human users. This was picked up by Jaro-Winkler resulting in better performance of the algorithm.

The tension curve performed worse on this task (average 0.65). This is understandable as different sub-scene sequences in the story may produce similar tension curves. However this also indicates that the problem of similarity of the stories is multi-layered and to grasp this properly a combination of features is likely to be required.

6 EXPERIMENT 3

In Experiment 3, we ran the method on stories generated by the MOSS system. We have analyzed 3000 stories from fairytale domain of MOSS with recklessness, retribution and reward morals (1000 from each). Half of the stories comprised of two dramatic actions, and the other half comprised of four dramatic actions. In both cases, the resulting stories contained about 30 atomic actions. The precision was measured with respect to the moral of the story. A graph of the results is presented in Figure 4.

We can see that the precision of clustering is very high for almost all clustering metrics. For MOSS stories of length four, tension curve achieved precision of 0.99 on three clusters. The sum of normalized story length and Levenshtein on action strings was the second best scoring 0.93 on three clusters. On MOSS stories with length two, these two metrics performed a bit worse. The best was Levenshtein on action strings which averaged on 0.94 and the tension curve with 0.88 precision on average. The story length metric was outperformed by almost all other metrics and it also did not bring significant improvements to the Levenshtein distance indicating that the MOSS generating process did not produce artifacts in story length. Similarly to previous results on the SD domain, Jaccard distance did not perform well.

This overall good performance is caused by the fact that stories in MOSS are generated through templates that use emotional patterns. Stories in one domain exhibit the same or very similar emotional patterns resulting in similar tension curves. This is picked by the tension curve metric really well. The comparable performance of string metrics on action strings is likely caused by the presence of emotional actions in the action strings. The overall slightly worse performance on stories with dramatic length two is probably caused by the fact that less dramatic actions in the story offer less space to distinguish the stories from each other (however the performance was still remarkably good).

Examples of the stories from this dataset and their clustering can be found in the appendix.

7 CONCLUSIONS AND FUTURE WORK

We have presented new data for a methodology for semi-automatic evaluation of interactive storytelling systems based on clustering of similar stories. We have reproduced and refined previous results in the area.

New results showed that the method can be transferred successfully to other domain. However we need to take this with a grain of salt as the MOSS story generator abstraction was very favorable to the method as it uses emotional patterns to define categories of the stories.

Next, we have added new feature of stories, sub-scene sequence, that was used in the implementation of random user designed to explore unvisited parts of the story space of SimDate3D domain and we have shown the performance of the method on distinguishing random user from the human users. Some of the metrics scored worse than expected indicating that to grasp story similarity properly a combination of features will be required.

The semi-automatic exploration of the story space with a random user proved useful and will be further investigated in future work.

We have also shown the performance of the method on an extended dataset from SimDate3D Level Two. Although we

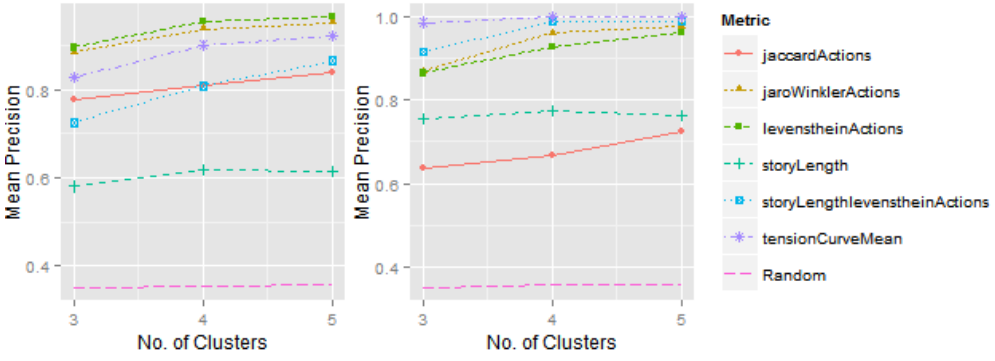


Figure 4. Experiment 3 MOSS domain clustering results. On the left there are precisions of clustering for three, four and five clusters when distinguishing between stories of the dramatic length two with particular moral. On the right there is the same for stories with the dramatic length four. All results were averaged over 100 clustering runs.

have reproduced the performance ordering of the metrics, the overall results were worse than in previous paper. The reason may be that the metrics do not accurately represent story similarity and pick a large amount of noise. A detailed analysis of stories in the same clusters could shed more light onto this and it is planned as future work, including comparison with story similarity as perceived by humans.

In line with conclusions from previous work, the tension curve provided best overall results across domains and feature functions, but as it did not work very well in Experiment 2 it cannot be considered universal and better metrics are needed. A combination of tension curve and one of the string distances might prove useful.

Other future work includes experiments with combination of distance metrics for the clustering algorithm and further enhancements and additional experiments with the random user. Finally, it would be beneficial to experimentally determine, how humans would cluster some of the stories.

ACKNOWLEDGEMENTS

This research is partially supported by SVV project number 260 224 and by student grant GA UK No. 559813/2013/A-INF/MFF.

REFERENCES

[1] M. Mateas, and A. Stern. Façade: An Experiment in Building a Fully-Realized Interactive Drama. In: *Game Developer's Conference: Game Design Track*. (2003).

[2] R. Aylett, M. Kriegel, and M. Lim: ORIENT: Interactive Agents for Stage-Based Role-Play. In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, vol. 2, pp. 1371–1372 (2009).

[3] R. Aylett, M. Vala, P. Sequeira and A. Paiva: FearNot! – An Emergent Narrative Approach to Virtual Dramas for Anti-Bullying Education. In: *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling*, LNCS Vol. 4871, Springer, pp. 199-202, (2007).

[4] M. Bída, M. Černý, C. Brom: Towards Automatic Story Clustering for Interactive Narrative Authoring. In: *Interactive Storytelling*. LNCS, Vol. 8230, Springer, pp. 95–106. (2013)

[5] M. Bída, M. Černý and C. Brom: SimDate3D – Level Two. In: *Proceedings of ICIDS 2013*. LNCS, Vol. 8230, Springer, pp. 128-131. (2013)

[6] M. Sarlej, M. Ryan.: Generating Stories with Morals. In: *Interactive Storytelling*. LNCS, vol. 8230, Springer, pp. 217-222. (2013)

[7] P. Weyhrauch: Guiding Interactive Drama. PhD. Thesis, Carnegie Mellon University, Pittsburgh (1997).

[8] S. Ontañón, and J. Zhu: On the Role of Domain Knowledge in Analogy-Based Story Generation. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pp. 1717–1722, (2011).

APPENDIX – EXAMPLE STORIES

Here, we present several examples of stories from SimDate3D and MOSS domains and show examples of the clustering of stories using the tension curve metric. In both cases we provide simple handcrafted natural language representations of the actions in the story.

A. SimDate3D Domain

Story 1: Thomas went with Barbara to the cinema. After the movie, he was rude to her. They have parted ways. Thomas went to Nataly's home to pick her up. They went out for a walk, but they did not speak much. Thomas insulted Nataly. They met Barbara. An argument started and both girls left Thomas.

Story 2: After the movie, he was rude to her. They have parted ways. Thomas went to Nataly's home to pick her up. Thomas was rude to Nataly. They went out for a walk and Thomas was rude to Nataly. They met Barbara. An argument started and both girls left Thomas.

Story 3: Thomas spent a long time with Barbara in the cinema, then he was very rude on her. Nataly was in the restaurant alone. Then Thomas and Barbara got very angry on each other, but continued talking. Nataly noticed them on their way from restaurant and she run towards them. An argument started and Thomas ended up with Nataly.

Stories 1, 2 and 3 get clustered together in most cases. Stories 1 and 2 are extremely similar and end the same, while story 3 is an example of a story that is relatively similar to the other two, but does not end the same.

Story 4: Thomas and Barbara were on a way to cinema. Thomas asked Barbara to kiss him and to cuddle, she refused. Then they've run into Nataly, argument started and Thomas ended up with Barbara.

Story 5: Thomas and Barbara were going to the cinema. Thomas was making jokes on the way. Before they've get to the cinema they've run into Nataly, argument started and Thomas ended up with Barbara.

Stories 4 and 5 on the other hand are also very similar and end the same but were almost never clustered together.

B. MOSS Domain

Story 1: A wizard gets hungry. He picks up a rose. A troll kidnaps a princess. The troll also kidnaps a dwarf. A knight rescues the princess from the troll. (Generated as an example for recklessness)

Story 2: A wizard gets hungry. He picks up a rose. A troll kidnaps a princess. The troll also kidnaps a dwarf. A dragon gives a treasure to the dwarf. (Generated as an example for recklessness)

Story 3: A dwarf kills a princess. A troll kidnaps the dwarf. A dragon tries to kidnap a unicorn, but fails. Fairy gives magical dust to the dragon. Dragon gives the dust back to the fairy. (Generated as an example for retribution)

All those stories are from the same cluster. While it is clearly visible, how stories 1 and 2 are extremely similar, story 3 seems very different.

aMUSE: Translating Text to Point and Click Games

Martin Černý [1] and **Marie-Francine Moens** [2]

Abstract. In this demo we will show aMUSE — a system for automatically translating text, in particular children stories, to simple 2D point and click games. aMUSE consists of a pipeline of state-of-the-art natural language processing tools to analyse syntax, extract actions and their arguments and resolve pronouns and indirect mentions of entities in the story. Analysed text serves as data the game mechanics operate on, while the story is represented graphically by images the system downloads from the Internet. The system can also merge multiple stories from a similar domain into a branching narrative. Users will be able to both play games created by aMUSE and create games from their own texts using the aMUSE editor.

1 INTRODUCTION

Video games are a powerful media for telling stories and for transferring experiences and feelings in a more general sense. Games are different to most other art forms in that they require active collaboration on the receiver's part. Thus adapting a story to the video game genre requires more than visualisation of the story events on screen: The game mechanics must also be designed to support the story or actively convey parts of the experience.

Recent research has shown that both game design and adaptation of text to game can be, to some extent, performed automatically. Most of the work so far either a) focuses on the game mechanics and does not consider the story of the game, or b) uses a large amount of domain-specific knowledge.

In this demo we will show aMUSE — a system that can automatically translate stories given in natural language to simple games without using any domain-specific knowledge. As our focus is on the story, we have chosen to generate games in the 2D point and click adventure genre. Games in this genre are inherently story-driven and consist of the player clicking on various objects to trigger interactions. If the correct interaction is found, the story progresses further. We have chosen this genre as it allows for a very direct mapping between the story and the game mechanics.

2 RELATED WORK

A system called Angelina can fully automatically design simple 2D and even 3D games [3, 2]. Game-o-matic [9] uses common-sense knowledge databases to generate 2D arcade games involving given topics. Our work is orthogonal to these efforts as it translates a story written in a natural language to a predefined game mechanic instead of generating the mechanics.

In the context of adapting a text to an interactive experience, De Mulder et al. [4] discuss transforming patient guidelines into educational 3D experiences. The authors use a large domain-specific knowledge base to provide common-sense grounding to the fragmentary information present in the text.

Some progress has been made on generating 3D scenes from text to be later used in a whole interactive experience [5]. However, the system is not fully automatic, as it relies on crowdsourced domain-specific knowledge to correctly position the entities in the scene and does not produce playable experiences yet.

3 THE SYSTEM

The aMUSE system consists of four parts: editor, translator, server and frontend. The editor is a graphical application that lets the user enter stories, group stories to form projects and control the execution of the translator. The translator is responsible for finding an interactive representation of the story which is passed to the frontend. For fast startup of the translator and due to some technical aspects of the technologies used, some of the tasks performed by the translator are carried on a dedicated server. The frontend is a simple game engine written in Flash that visualises the game provided by the translator.

To translate a story, the translator first passes it to the server. The most important part of server-side processing is *semantic role labelling* (SRL) using the Lund pipeline[3]. SRL builds upon syntactic features of the sentence to discover *semantic frames*. A frame represents a concept in the sentence (the *root*) and annotated arguments of the concept (the *roles*). We use frame definitions given in PropBank[4].

For example the sentences "The city was taken by the Romans" and "The Romans took the city" have different syntax, but both contain the frame $take.01(taker : Romans, thingTaken : city)$. The numbered suffix to the frame root distinguish between various meanings of the same word: e. g., "I cannot take it anymore" would resolve to $take.02(tolerator : I, thingTolerated : it)$. The Lund SRL was trained on news texts, so we used transfer learning [8] to adapt it to handle stories better.

The last crucial part of server-side processing is coreference resolution using Stanford CoreNLP [6]. Coreference resolution links all mentions of the same entity (pronouns, in particular) throughout the whole story. The annotated text is then returned to the translator.

The translator uses the semantic frames to find possible interactions for each sentence of the story. In our case, interaction is an agent-action-target triplet, where either agent or target may be omitted (but not both). All frames with roots that are verbs are candidates for interactions. Simple hard-coded heuristics are used to choose the agent and the target among the frame's roles.

Now, every story is represented as a linear multigraph with sentences as nodes and possible interactions as edges from the previous sentence to the sentence that defined the interaction. Optionally, the translator can merge multiple stories to form a non-linear story

[1] Charles University in Prague, Czech Republic, email: cerny.m@gmail.com
[2] KU Leuven, Belgium, email: sien.moens@cs.kuleuven.be

[3] http://nlp.cs.lth.se/
[4] http://verbs.colorado.edu/propbank/

graph. To achieve this we check all pairs of sentences A, B. If they are from different stories, but have similar frames then for each interaction (X, A) we add (X, B) to the graph and vice versa, i.e., at these nodes the game can switch to a different story, depending on the interaction chosen by the player. This approach was inspired by the story generation process described in [7].

The translator then lists all the entities present in the story and schedules at which point in the story they should appear. As coreference resolution is not flawless, we make the simplifying assumption that two entities with the same name are the same and merge the respective entity mentions. The translator then requests images for the entities from the server which uses Spritely [1] for this task.

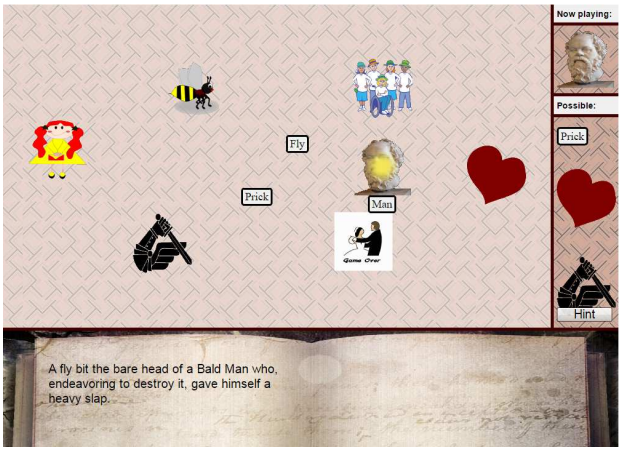


Figure 1. Screenshot of the aMUSE frontend.

The frontend then uses the story graph as the basic structure to guide gameplay. It keeps the current node in the graph and when the user performs an interaction corresponding to any of the outgoing edges, the story progresses to the edge's target, i.e., every action of the user corresponds to progressing the story one sentence further.

Originally, we intended that the user will represent the protagonist of the story and perform only the interactions where he is the agent. In this case, the other interactions would be performed by the system automatically as a kind of a cutscene. This however led to a large number of non-interactive nodes, so we decided to alter the game design a little: the user is no longer a character in the story; he represents a disembodied entity, whose single goal is to make the story happen. To do this, the user can take control of any active entity and act (click on objects) on its behalf. The resulting interactions are very abstract and it is almost impossible to decipher the story from the interactions themselves. To allow the player to follow the story, the original text of the sentence is shown in a stylized book. The screenshot of the frontend is given in Figure 1.

So far, we have not been able to finish our work on extracting spatial relationships between the entities from text, so the entities only float around the screen without any structure.

4 CONCLUSION

Our system is capable to automatically translate stories written in natural language into a specific type of playable experiences. While many of the interactions that the system produces make sense, it also produces absurd options, mostly due to imperfections in natural language processing (NLP). To some extent, this can be enjoyable from the user perspective, but there is definitely room for improvement.

The system works reasonably well on short stories targeted at very small children, as the vocabulary and syntactical structure is simple. However, the main reason that short stories work better than longer ones is that the gameplay is very limited and it is not fun to click through a longer story. Although longer stories also degrade accuracy of coreference resolution. Semantic and syntactic complexity of the text is currently the most limiting factor for our tool. We tested the system on Aesop's fables, where the resulting gameplay was still more often relevant to the story than not. However, when run on fairy tales collected by Andrew Lang, which have long and complex sentences and archaic language style, only a minority of the resulting interactions were reasonable. Further issues arise from incorrect association of words with images.

Our system can serve as a demonstration of the power (and remaining deficiencies) of the contemporary NLP technology. We believe that NLP is at the level where it can improve games and gaming experience. While we are aware of game-related research using syntactic analysis of texts, we are not aware of usage of SRL in this context, although there are high possible benefits.

Examples of games created by the system can be played online[5] and the system itself is fully open-source.

ACKNOWLEDGEMENTS

This research is partially supported by the EU FP7-296703 project MUSE, student grant GA UK No. 559813/2013/A-INF/MFF and by SVV project number 260 224.

REFERENCES

[1] M. Cook. Spritely — autogenerating sprites from the web. `http://tinyurl.com/spritelypost`, (2013). Last checked: 2015-01-16.
[2] M. Cook and S. Colton, 'Ludus ex machina: Building a 3D game designer that competes alongside humans', in *Proceedings of the Fifth International Conference on Computational Creativity*, pp. 54–62, (2014).
[3] M. Cook, S. Colton, A. Raad, and J. Gow, 'Mechanic miner: Reflection-driven game mechanic discovery and level design', in *16th European Conference on Applications of Evolutionary Computation*, volume LNCS 7835, pp. 284–293. Springer, (2013).
[4] W. De Mulder, Q. Ngoc Thi Do, P. Van den Broek, and M.-F. Moens, 'Machine understanding for interactive storytelling', in *Proceedings of KICSS 2013: 8th International Conference on Knowledge, Information and Creativity Support Systems*, pp. 73–80, (2013).
[5] R. Hodhod, M. Huet, and M. Riedl, 'Toward generating 3D games with the help of commonsense knowledge and the crowd', in *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, pp. 21–27, (2014).
[6] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, 'The Stanford CoreNLP natural language processing toolkit', in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60, (2014).
[7] N. McIntyre and M. Lapata, 'Plot induction and evolutionary search for story generation', in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 1562–1572. Association for Computational Linguistics, (2010).
[8] Q. Ngoc Thi Do, S. Bethard, and M.-F. Moens, 'Text mining for open domain semi-supervised semantic role labeling', in *Proceedings of the First International Workshop on Interactions between Data Mining and Natural Language Processing*, pp. 33–48, (2014).
[9] M. Treanor, B. Blackford, M. Mateas, and I. Bogost, 'Game-o-matic: Generating videogames that represent ideas', in *Proceedings of the The Third Workshop on Procedural Content Generation in Games*, p. 11, (2012).

[5] `http://tinyurl.com/amuseExamples`

Data Collection with Screen Capture

Jason Traish, James Tulip and Wayne Moore[1]

Abstract. Game traces are an important aspect of analysing how players interact with computer games and developing case based reasoning agents for such games. We present a computer vision based approach using screen capture for extracting such game traces. The system uses image templates of to identify and log changes in game state. The advantage of the system is that it only captures events which actually occur in a game and is robust in the face of multiple redundant commands and command cancellation.

This paper demonstrates the use of such a vision based system to gather build orders from Starcraft 2 and compares the results generated with those produced by a system based on analysing log files of user actions. Our results show that the vision based system is capable of not only automatically retrieving data via screen capture, but does so more accurately and reliably than a system relying completely on recorded user interactions.

Screen capture also allows access to data not otherwise available from an application. We show how screen capture can be used to retrieve data from the DotA 2 picking phase in real time. This data can be used to support meta-game activity, and guide in-game player behaviours.

1 Introduction

Game traces allow researchers to follow the evolution of game state as a game is played. Retrieving game traces is necessary to further understand decisions made by players in different game states, and to support the development of AI agents.

Data for board games such as GO [4] and Chess [7] are obtained from a sequential list of user interactions with the game. In GO and chess the user interactions with the game are very limited and the effect of any player action in the game is deterministic. For example, in GO it is known that when a player places a stone such that an opponent's stones are surrounded, then the opponent's stones will be eliminated. However, in commercial RTS games such as Starcraft 2 [1], the set of user interactions for is often far larger than in a classic board game, and the effect of player actions on game state is uncertain. A user can move a camera, move units, construct buildings, train units, buy upgrades and much more. Some of these commands (eg camera movement) have no effect on game state, and for others, (eg unit movement) the effect is indirect. Furthermore, the actual internal game state is inaccessible.

In both GO and Starcraft 2 a game is recorded as a sequence of user interactions. However, while in GO this sequence corresponds directly to changes in game state, in Starcraft 2, multiple redundant commands may be issued in a short space of time, many may never have effect, or they may be cancelled before they are enacted. The only way to tell what actually happens is to play the user interactions back through the game environment.

The other problem that occurs, particularly in RTS games, is that the rules determining how player actions affect the game environment can change due to developers tuning and rebalancing game play. This makes it unfeasible to recreate game state based on user interactions because their effects are constantly changing. In the case of Starcraft 2, while we can access the list of raw commands given by the user via game replay files, access to this list does not allow recreation of game state unless it is used to replay the game using the actual game environment. Once again, the solution is to directly monitor game state changes rather than user inputs.

Lack of access to internal game state makes it difficult to develop AI agents, and much game AI agent research is based on the use of appropriately instrumented simulators. Unfortunately, many of these are highly simplified versions of the original game. Samothrakis [8] suggests that a screen capture approach would solve this problem. Screen capture also offers a standardized way to provide AI agents with input. This is necessary to meaningfully compare the performance of AI agents. Screen capture also allows retrieval of game state from closed source commercial games, and so enables testing of agents using the original game rather than a simulator.

This paper demonstrates the use of a screen capture system to analyse Starcraft 2 build orders. Build orders describe a player's sequence of creating units, buildings, and upgrades to reach a specific strategic goal. The efficiency of a player's build order can significantly impact their chance of winning, and there is considerable research in build order optimization [6]. We demonstrate the extraction of build orders using screen capture and compare the results generated with those produced by Sc2Gears [3]. Sc2Gears calculates build orders based on a log of user interactions. We also demonstrate the real-time use of screen capture to monitor hero selection in the online game Defense of the Ancients (DotA 2) [2]. DotA 2 is a multi player online battle arena (MOBA) game where players select heroes with various characteristics and do battle in teams. Hero selection and team combinations have a large impact on team success, and there is much interest in predicting game outcomes based on hero combinations [9].

2 Screen Capture

The screen capture system models a human observer tracking and recording changes in a game. It identifies areas of the screen which display information relevant to game state and then monitors changes in those areas, interpreting them in terms of game state.

Initially, the area of the game window that the relevant information will appear is specified. Then all patterns showing the information to be recognised in that area of interest are recorded as a set of templates. All templates have the same dimensions to simplify and

[1] Charles Sturt University, Mining Lab, Australia, email: {jtraish & jtulip} @csu.edu.au & wmoore@lisp.com.au

speed up matching. After all templates have been loaded into the system, PCA [5] is used to compress each set of templates down to 30 descriptors per template. This enables a reduction in the number of comparisons for each template and facilitates real-time analysis of captured images. The application is then started with the replay for game trace retrieval. The windows contents are captured via the Windows API and stored in an OpenCV image as often as the refresh rate allows. The image is then decomposed into the identified areas of interest such as the game timer, player's production icons, progress bars, and resource supply. Game specific heuristics are then used to extract information from the screen using template matching and to monitor game state events. The processed game state information is then stored as a game trace.

The screen capture system can be summarised as taking the following steps:

1. Load pre-labeled templates.
2. Decompose templates into basic descriptors.
3. Open the application's associated replay file.
4. Capture the game window using the Windows API.
5. Store and decompose the windows contents into areas of interest.
6. Match templates against areas of interest using a multi-threaded framework.
7. Process the results and store the resulting game trace.
8. Repeat from step 3 to analyse further games.

3 Starcraft 2 Build Orders

Figure 1 displays the replay interface in Starcraft 2 that was used to retrieve game traces. Before starting the process, the system must be aware of where to look for which templates. The templates are stored in sets, one each for the production icons of each player selectable race, and an extra one for other GUI elements. This reduces the number of comparisons necessary as a player can only produce items for their chosen race.

To retrieve the build order we now identify what is displayed using our library of PCA refined descriptors. The top left hand corner of Figure 1 shows seven units/buildings in production. Each item of production shows an identifying image, a number showing how many units are being produced simultaneously, and a green progress bar reflecting the completion percentage of that item. Each different production icon indicates that a build queue is active within that area of interest. In this case we would say that 4 build queues are active for player 1 and 3 are active for player 2. The icon positions are then posted to different worker threads which compare the captured image with an assigned template set. Figures 2 and 3 show the matching templates used to identify production icons collected from the scene shown in Figure 1. Each template is labelled with the name of the production icon.

After identifying the production icon, the game trace heuristic then finds the number on each template as shown in Figure 4.Numbers are identified using a relatively naive yet accurate method. Because numerals are imprinted against a production icon's image they contain a small amount of noise. The noise is reduced by only accounting for pixels that are very similar to white. Then the filtered image is compared against a set of number templates where the closest is selected as the matching number. Following this process identifies the digit shown Figure 4 as matching the template shown in Figure 5.

The completion percentage shown in the production icon is then determined (Figure 6). For this we simply perform a threshold check for predefined pixel values along the length of the progress bar, returning when an empty pixel is found.

Figure 1. Sample screen capture

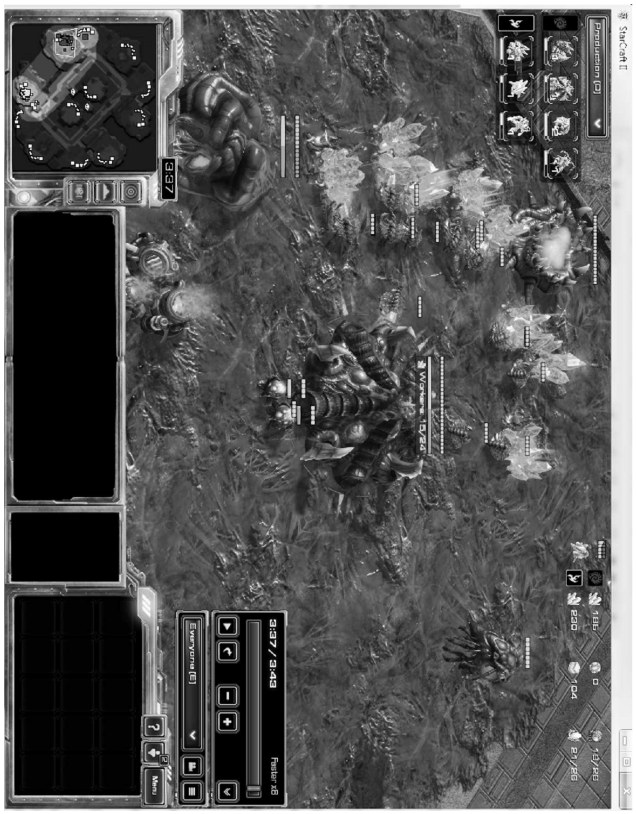


Figure 2. Player 1 - Matching production templates

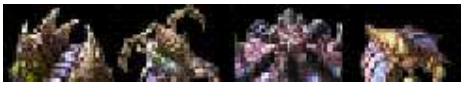


Figure 3. Player 2 - Matching production templates

Figure 4. Digit with noise

Figure 5. Pre-labeled Digit (Matching Template)

Figure 6. Progress bar

Since each template comparison is independent, all template comparisons can be run in separate threads. Once all threads have finished analysing each real-time acquired production icon image the information is used to update each players build order along with the game state, and the game time at which the image was retrieved.

As each player's build order is updated, it is possible that a previously recorded production item is cancelled. If a production item is not listed but was less than 97% complete when last identified then that item is assumed to have been cancelled and is removed from the recorded build order. Within a game of Starcraft 2, this can occur at any point in time when a user selects a production item and cancels it. A cancellation is also noted if the number of items listed within the production icon drops while the current completion percentage is under 97%. This leads to the flaw in the current game trace heuristic that if a production item is almost complete then any number of production items of the same type can be cancelled and they will be falsely recorded as completed. In practice, this rarely occurs.

When a new production item type appears or the production count increases then the game trace heuristic appends that item to the build order. If the number of items in production recorded by a production icon number remains the same for longer than the time to create that item, then another production item of that type is appended to the build order. This deals with the case of when a series of probes/workers are queued. Since they are created one at a time, a constant production count of 1 appears over an extended period. Thus, keeping track of how long it is from when a production icon first appears we can determine when an item repeats production. The exception is when production is halted or paused which can be detected when the progress bar is halted.

After a game has completed, each players build order is recorded to file and the next game is opened and the process repeated. The replay interface is controlled by sending Windows API keyboard messages to Starcraft 2 to display the production icons and accelerate the play back. The replay playback is accelerated to the maximum of eight times the normal playback rate.

4 Comparison with User Interaction Logs

An experiment was conducted to evaluate how the screen capture system performs in capturing a build order in comparison with the established tool Sc2Gears [3]. Sc2Gears applies the user interaction approach to analyse build orders. Both systems were tested using a set of 100 public Starcraft single player versus single player ladder games. Comparisons were made only on the first 10 minutes of game play so that replays of diverse lengths would not affect the results significantly. Each of the 100 games was also processed by a human to generate a ground truth set of build orders. The accuracy for the automated systems was calculated as the number of matching build order steps compared with the human verified sequence.

Table 1 shows that the screen capture technique was able to significantly reduce the number of errors in calculated build orders compared with an analysis based on raw user interactions. The screen capture system still generated a small number of errors in cases where actions were cancelled on the last frame (thus appearing to have actually been completed). Table 2 shows an example of an opening build order extracted using screen capture compared with one using Sc2Gears from the same game. The extracted information is significantly different. Sc2Gears incorrectly identifies the creation of three probes and an additional pylon. In this case, the player requests production of an additional Probe without the necessary resources, a situation that can only be determined by running the game replay. The extra pylon identified by Sc2Gears was the result of the player ordering construction of a pylon and then moments later changing the location of its construction. These errors highlight the issues encountered when using user interaction methods to extract game traces.

Table 1. Error Rates

Error	Screen Capture	Sc2Gears
Mean	0.39%	30.71%
StdDev	0.96%	27.75%

Table 2. Example Game Trace

Sc2Gears	Screen Capture
1. Probe	1. Probe
2. Probe	2. Probe
3. Probe	3. Probe
4. Probe	4. Pylon
5. Probe	
6. Probe	
7. Probe	
8. Probe	

5 Hero Selection in Defence of the Ancients 2

The screen capture technique was also applied to Defence of the Ancients 2 (DotA 2) to test its real-time capabilities of the screen capture framework, and its capacity to generalise beyond Starcraft 2. This section re-enforces the application of the screen capture frame work in retrieving data from a 2D display interface. The data from the DotA 2 interface is retrieved without error and thus no comparison against other methods is given, instead a potential use of the retrieved data is given. The experiment with DotA 2 shows flexibility and versatility of the screen capture approach.

DotA 2 is a multi player online battle (MOBA) game that involves 2 teams of 5 players. Each player must pick a hero, and after a hero is selected and locked in it can not be picked by any other player. Players can select a hero they intend to pick before locking it in, and this is referred to as shadow picking. A shadow pick will only display to the allied team, and is important in influencing the heroes other members of the team will select.

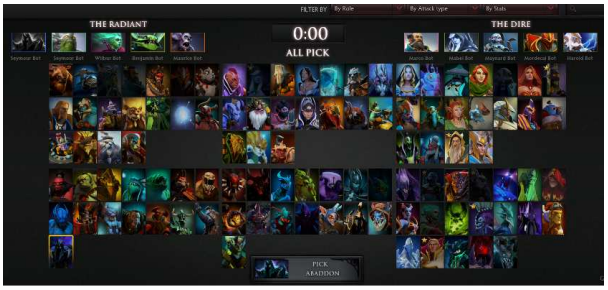
Heroes fall into general categories based on their abilities and how they interact with other heroes within the game. The picking process leads to a diverse set of combinations that can be formed between the 2 teams. However, some of these combinations are weaker than others due to the interaction of hero's strengths and weaknesses. Each hero has synergies with certain allied heroes and/or are able to exploit weaknesses in particular enemy heroes. Thus, it is an interesting problem to see how players adapt their choice of hero during the 1 minute picking phase. It is also interesting to see how these picks can be used to predict the winning team and what rate of success they might have.

In DotA 2, there is much interest in real-time capture of game actions since such a capability offers the potential to support real-time

guidance on hero selection. It also provides information useful to calculating the likelihood of final outcomes. Screen capture potentially can achieve this while user interaction logs are available only after a game has ended.

Figure 7 shows a standard DotA 2 'all pick' mode selection screen. It can be seen that all players have locked in their hero choices except for the player shown on the upper left . This player's portrait is rendered in grey scale to show that it the depicted hero has only been shadow picked. During the picking phase we use the screen capture

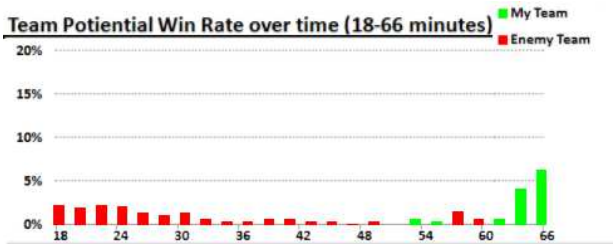
Figure 7. DotA 2 Hero Selection Screen



framework to identify which heroes have been locked in or shadow picked. This data is then analysed using a statistical algorithm based on hundreds of thousands of games of DotA 2.

The current program then displays the win rate for any point in time during the game as shown in Figure 8. This graph can be used loosely to identify when one team is stronger than another and can be used as an indicator for players to become more aggressive within the favoured time zones. It can also be used by lower skilled players to help better identify hero picks that complement their team, and to see what effect their pick would have on the progress of the game. Figure 8 shows that the enemy team has a small advantage that decreases over time until around the 60 minute mark, at which point My Team increases substantially in strength.

Figure 8. DotA 2 Predicted Game Balance



6 Discussion and Further Work

The Starcraft 2 experiment shows that the screen capture approach can help generate more accurate build orders than conventional systems based on logs of player actions. Its application to analysing hero selection in DotA 2 shows that the principles can be applied generally to any game, and for any analytical purpose, using different sets of image templates and different analytical heuristics. The technique can be applied to almost any application where a streaming 2D display record is available. Furthermore, no access to game code or proprietary APIs is required. This opens up data collection and analysis for previously inaccessible games and other applications. The high performance provided by the simplified PCA based image descriptors and parallel template matching allows the development of real-time in-game decision support systems, once again without access to game code or proprietary APIs. The screen capture system takes advantage of using the game display to retrieve actual game events while user interaction logging methods can result in noisy data that can detrimentally affect further analysis.

However, currently screen capture has only been applied to applications where the state is represented with scale and rotation invariant 2D images. There would be considerable challenges in applying the technique to applications that display their state in 3D.

The technique could also be extended to live game data retrieval, such as a Starcraft 2 commentator agent. An agent could be set up to watch two players play a competitive game, giving viewers predictions and feedback in a similar way to how real commentators perform.

The screen capture system could also be used to track in game auction house item prices. The retrieval of the changing value of game items could allow systems to graph, analyse and predict market trends in online worlds.

It could also be used for non game applications such as watching a user's screen and determining the time spent interacting with different windows. This could help system analysts trace work flow and productivity in given applications without access to the source code.

While analytic techniques relying on replays to retrieve game data have to wait until a game has been played and recorded before analysis can be applied, a screen capture system can be used to analyse live games, allowing interested parties to use the data in prediction systems or other applications.

7 Conclusion

Screen capture data retrieval offers great advantages to researchers and applications looking to gather data from complex environments with 2D displays. The system is flexible and more accurate than user interaction logs for such applications.

REFERENCES

[1] 'Blizzard entertainment'. http://www.blizzard.com.
[2] 'Valve corporation'. www.dota2.com.
[3] 'Sc2gears'. https://sites.google.com/site/sc2gears, (2012).
[4] T. Bossomaier, J. Traish, F. Gobet, and P. C. R. Lane, 'Neuro-cognitive model of move location in the game of go', in *(IJCNN), The 2012 International Joint Conference on Neural Networks*, pp. 1–7.
[5] Ian T Jolliffe, *Principal component analysis*, volume 487, Springer-Verlag New York, 1986.
[6] Matthias Kuchem, Mike Preuss, and Günter Rudolph, 'Multi-objective assessment of pre-optimized build orders exemplified for starcraft 2', in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pp. 1–8. IEEE, (2013).
[7] Peter CR Lane and Fernand Gobet, *Using chunks to categorise chess positions*, 93–106, Springer, 2012.
[8] S. Samothrakis, D. Robles, and S. Lucas, 'Fast approximate max-n monte carlo tree search for ms pac-man', *Computational Intelligence and AI in Games, IEEE Transactions on*, **3**(2), 142–154, (2011).
[9] Pu Yang, Brent Harrison, and David L Roberts, 'Identifying patterns in combat that are predictive of success in moba games', *Proceedings of Foundations of Digital Games*, (2014).

Cognitive Navigation in PRESTO

Paolo Calanca and **Paolo Busetta**[1]

Abstract. The PRESTO project has developed an AI infrastructure and an agent framework called DICE for the creation of game-independent, modular NPC behaviours based on a BDI (Belief-Desire-Intention) approach enriched with cognitive extensions for human simulation. Behavioural models can be combined via end-user development tools to form the behavioural profiles of NPCs in a game. Furthermore, PRESTO is producing a set of behavioural models targeted at its pilot project's needs or expected to be of common use. This paper focuses on a fundamental building block: navigation of (human and non-human) characters, implemented as the interplay between a set of behavioural models encapsulating higher-level decision making concerning e.g. speed control, activation of gates, replanning when faced with the impossibility to going forward and lower-level modules for path planning, steering and obstacle avoidance that focus on performance and simpler perception-driven choices. These lower-level modules are embedded into the PRESTO infrastructure and contain a few novel algorithms. The higher level navigation behavioural models in DICE can encapsulate very different physical and emotional profiles; they deal with short-term memory and background knowledge concerning spatial knowledge and impose constraints on path planning based on physical as well as cognitive considerations (e.g. risks or threats). DICE provides the coordination between body-controlling behavioural models (for navigation as well as posture, facial expressions, actioning) and decision-making models representing e.g. the standard operating procedures of professional roles, the cognitive appraisal of events and perceptions, the modality of reaction to unplanned events occurring during a game.

1 INTRODUCTION

PRESTO (Plausible Representation of Emergency Scenarios for Training Operations) [2] aims at adding semantics to a virtual environment and modularising the artificial intelligence controlling the behaviours of NPCs. Its main goal is to support a productive end-user development environment directed to trainers building scenarios for serious games (in particular to simulate emergency situations such as road and industrial accidents, fires and so on) and in general to game masters wanting to customize and enrich the human player's experience. The framework for behavioural modeling in PRESTO, called DICE, was inspired by a BDI (Belief-Desire-Intention) [1, 9] multi-agent system with cognitive extensions, CoJACK [10, 6]. PRESTO offers powerful end-user development tools for defining the parts played by virtual actors (as end user-written behaviours) and the overall session script of a game. PRESTO supports a specific virtual reality, XVR from E-Semble, a well known tool in use for Emergency Management and Training (EMT) in a number of schools and organisations around the world, as well as Unity 3d and, at least in principle, is agnostic with respect to the game engine in use.

The rest of this introduction briefly explains the motivations behind PRESTO with an example and gives an overview of the system. The following sections are dedicated to its navigation subsystem, first discussing lower-level facilities for path planning and steering and then introducing a higher-level layer that takes into account cognitive aspects including memory and appraisal of the perceptions according to the semantics of the environment and the NPC's own psychological profile.

Directing NPCs as virtual actors in a virtual stage. Serious games have the potential to dramatically improve the quality of training in a number of fields where the trainee has to face complex and potentially life-threatening situations. In particular, open-world 3D simulations (also called ''sandbox" or ''free-roaming" games) have been used for quite a long time by the military, with a few products reaching a significant market success, and are becoming common in civilian emergency training because they allow the rapid construction of scenarios for the rehearsal of safety procedures. The main limitation of current technology concerns NPCs, whose behaviour may be quite sophisticated when performing predefined tasks but is often unaffected by context; further, a professional programmer is required for the implementation of any procedure that cannot be described with the simple selection of a few waypoints and the choice of a few actions, let alone introducing variants due to psychological factors. These issues lead to repetitive and hardly credible scenarios and to the slow and costly development of new ones when many NPCs are involved.

As an example, consider a fire breaking in a hospital ward during daytime with patients with different impairments, visitors of various ages and professionals with different roles, experiences and training. In this scenario, which is taken from the pilot project of PRESTO, most characters are NPCs while the human players, i.e. the trainees, are either health professionals that could be in charge for a ward at the time of an accident or emergency staff called to help. A training session would require two apparently conflicting abilities from NPCs. From the one hand, they should act autonomously according to a variety of parameters concerning e.g. their physical and psychological state, their current position, their capabilities; e.g. visitors may act rationally and follow well-marked escape routes or flee panicking to the closest exits, nurses at the start of their shift are fully responsive and careful while at the end of the shift fatigue may lead to errors, and so on. On the other hand, in order to make training effective and engaging, the trainer supervising a simulation session should be able to temporarily suspend it (e.g. to give feedback to the trainees), change the course of events or affect the way certain characters behave (e.g. to introduce more drama or rehearse different procedures), as well as introducing or removing characters in following runs of the same scenario. Hardcoding all possibilities, assuming that this is

[1] Delta Informatica Spa, Trento, Italy, email: name.surname@deltainformatica.eu

supported by the game in use, is a laborious task to say the least.

The objective of PRESTO is to allow NPCs to act as "virtual actors" because they are able to "interpret" a part written at a higher level of abstraction than with common scripting languages, with additional modalities (that may correspond to, e.g., levels of skills or psychological profiles) that can be selected at the beginning but changed during a game as a result of the application of rules or by explicit user choice. The game's master (i.e. the trainer) is empowered to become a "director" able to "brief" virtual actors, that is, to define the parts the artificial characters have to play by means of a language aimed to non-programmers that composes more fundamental even if potentially very complex behaviours into game-specific sequences. Key enablers are end-user development tools [7] and the ability to mix and match behavioural components taken off-the-shelf from a market place (similar in principle to asset stores in popular gaming platforms such as Unity).

Semantics and NPC programming in PRESTO. PRESTO provides facilities for the semantization of the game environment in order to support decision-making based on game- and scenario-independent properties. Most importantly, ontologies are used for the classification of objects and locations and for annotating them with properties and states (called "qualities") that allow abstract reasoning, while navigation areas can be annotated with various properties [5]; some of these aspects are discussed in Sec. 3.

DICE (Fig. 1) supports multi-goal modeling of NPC behaviours, where navigation, body postures and facial expressions, manipulation of objects and decision-making concerning tactical and long-term objectives are controlled by concurrent threads (implemented, in BDI speak, as intention trees achieving independent hierarchies of goals and subgoals). Furthermore, decision-making in DICE happens at two levels, controlled by independent "planned" and "reaction" intention trees. A decision-making behaviour started in reaction to an event pre-empts and blocks the execution of a planned behaviour until it is fully completed, at which point the planned behaviour is resumed. This allows, for instance, to have short-term reactions to perceptions (such as hearing a noise) that partially change the NPC state (e.g. by pointing the head towards the source of the noise) while not affecting navigation or longer-term procedures if not required. All behaviours in the body-controlling intention trees and in decision-making can be overriden by new behaviours at any time, e.g. as new perceptions are processed, as part of a decision-making routine, as a user choice from a GUI, as a command from a PRESTO session-controlling script; at any time, no more than one behaviour for each intention tree is active.

Changes in behaviours due to emotions, fatigue or other non-rational factors can be dealt within DICE in various ways, of which the most novel (and dramatic) is by defining behavioral rules that select alternative models according to the current cognitive state of the NPC. These rules can be defined directly by the end user, who is enabled to change the behavioural profiles of her characters according to the evolution of the game or even in real-time by explicit choice and from the session-level script. As in CoJACK [10], cognitive states are represented in DICE by moderators (i.e. numeric values modeling specific factors such as fear and fatigue levels) and a set of cognitive parameters computed from those moderators (modeling e.g. reactivity and accuracy), even if greatly simplified with respect to the original. Any behavioural model, including navigation, can use moderators and cognitive parameters to tune its own internal parameters, e.g. to decide the speed of execution of action or memory fading. Changes to moderators are normally performed by behavioural models for cognition according to appraisal rules (concerning e.g. the perception of threatening things) and time; however, it is possible to force the value of moderators at any time from any behavioural model (e.g. because of the realization of a dangerous situation) or from the session-controlling script, thus allowing the trainer to fully control the overall behaviour of an NPC during a game.

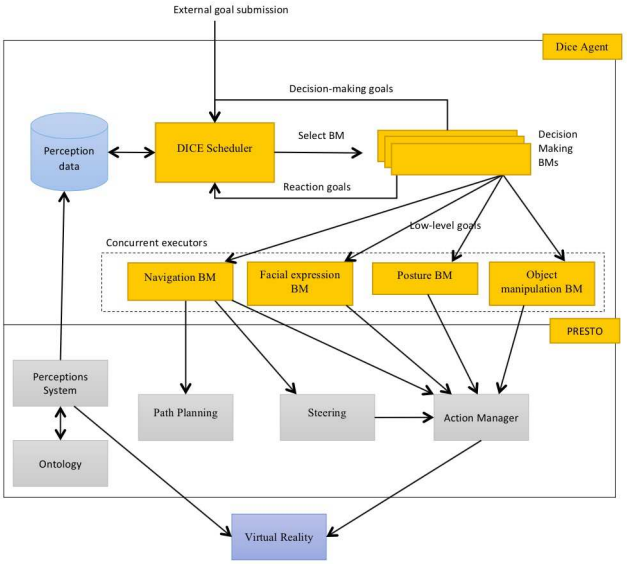


Figure 1. Simplified DICE architecture with navigation highlighted (BM: Behavioural Model)

One of the implications of the DICE approach on navigation is that, at any time, the travel direction (decided by a behaviour) can be changed and may be resumed later (e.g. when a reaction is completed). The APIs make programming this concurrent machinery a straightforward business, while the end-user development tool for behaviour modeling (called the DICE Parts Editor) provides an extremely powerful yet intuitive way to write scripts that affect one or more intention trees at each step [8].

As mentioned earlier, PRESTO has a facility to edit and control session-level scripts inspired by interactive books. A session script is composed by a set of scenes connected as a graph. At each scene, goals can be given to NPCs, their internal state changed (including emotions) and objects manipulated. The trainer starts a script at the beginning of a training session and advances it by manually navigating the graph of scenes or letting PRESTO choose the next one e.g. when certain events happen or when a timer expires. This allows a large, potentially unlimited number of different sessions to unfold from a single script with no need to reprogram the NPCs once equipped with all required behavioural models. In the hospital ward example presented earlier, the initial scene would command visitors, patiens and nurses to accomplish their routine goals; the script may continue with alternative scenes such as "fire breaking in a patient room" or "fire breaking in a surgical facility", each with different people involved, and then with sequences that may lead e.g. to smoke filling the area and visitors fleeing or an orderly managed situation with the intervention of fire fighters, chosen according to the decisions of the trainer and the events occurring during a session.

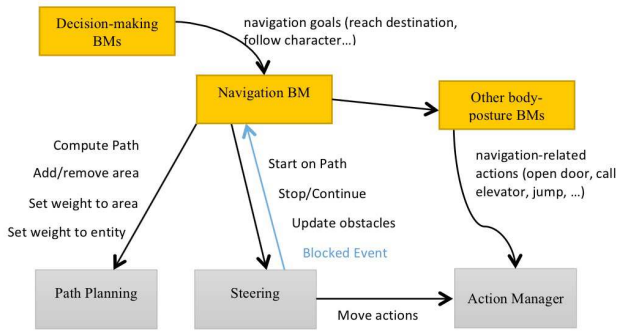


Figure 2. Navigation subsystem architecture

2 NAVIGATION ARCHITECTURE

The overall architecture of navigation within a DICE agent, shown in Fig. 2, closely resembles a standard model [3], with a path planning module, a steering module looking after actual body movements and simple obstacle avoidance, and the navigation behavioural model calling the path planner and the steering modules according to the goals provided by decision-making (e.g., of reaching a destination, of following another character, and so on).

The path planner uses a navigation graph which is instantiated for each agent and modified by the navigation behavioral model to reflect memory, navigation decisions and specific capabilities. From this graph, the path planner can compute one ore more paths to the desired destination and the behaviour can choose which one to follow based on any attached information. Once a path is chosen, the steering module is invoked by the behaviour to move along it. State information on the steering activity for a specific path, including an explanation in case of unsuccessful conclusion (e.g., facing a gate, impassable obstacles, aborted by another steering request typically generated by a reaction), is used by the behaviour to track progress and possibly perform actions to resume navigation. Analogously, the state of a goal given to the navigation behaviour is reported on a tracking object that allows higher-level decision-making behaviours to know when the goal has been satisfied or the reason for failure, including abort caused e.g. by a reaction submitting a different navigation goal.

The flow of perceptions goes to steering as well as to all behavioural models to update their own internal state. As a consequence, the navigation goal being currently pursued may be changed because e.g. of a reaction or the decision to take a different course of actions.

3 MESHES, AREAS AND SEMANTICS OF THE ENVIRONMENT

Configuration information affecting navigation is distributed in three main data structures, two of which concern meshes and are directly used by the navigation modules while the third is related to semantics for the decision-making layer.

Navigation meshes and navigation areas. PRESTO uses navigation meshes (that is, sets of adjacent convex polygons that share edges and cover a walkable / drivable / otherwise navigable surface) [11] to compute safe and efficient paths through the environment, avoiding walls, obstacles and precipices. Navigation meshes can be automatically built from the environment geometry and from parameters including the navigating object's radius, height and max acceptable steepness, so it is possible to generate meshes specialized per character type (including non-humans, e.g. vehicles).

Semantics data on the navigation meshes, such as the terrain type and traffic constraints (permitted directions, reserved paths, ...), can be added with a tool that allows the creation and annotation of navigation areas by selecting polygons of a mesh. Furthermore, as discussed below, behavioural models manipulate areas rather than polygons of a mesh.

Locations of Interest and navigation-affecting entities. PRESTO allows the end-user to classify and annotate locations of interests and objects within the environment with semantic information taken from an ontology. This is composed of a domain-independent core and one or more domain-specific extensions [5] and determines which behavioural models can be used in a specific game; for instance, the current PRESTO pilot project contains a hospital ontology that is used by models of nurses and doctors while a generic safety ontology is used by fire fighters. A small part of the semantic annotations is directly managed by the navigation subsystem as discussed later, most importantly the property of being a "gate", i.e. anything that has a state of openness that can be manipulated by a character. Being a gate is not automatically related to the classification of the object (e.g., a door is not a gate if it is permanently closed) and may even change dynamically. Anything else that may affect what the character does during its movements is handled by other behavioural models and especially by decision-making models. This separation of concerns relies on the possibility offered by DICE to stop and change navigation goals at any time, possibly as reactions that simply delay rather than abort the procedure being executed by a character.

4 LOWER-LEVEL NAVIGATION FACILITIES

Higher-level behavioural models and lower-level facilities share a navigation graph, manipulated by behaviours and used by the path planner, and status information on the current steering activity. A set of APIs allow behaviours to affect the navigation graph, invoke the path planner and trigger steering.

Navigation graph and path planning. The Path Planning module uses a navigation mesh to build a polygon adjacency graph, which in turn is used as navigation graph shared with the behavioural models. While navigation meshes are generated off-line and shared by all agents, a navigation graph is specific for each agent since it is based on the background knowledge of the agent, its capabilities, its memory and its decisions. For instance, the configuration of the background knowledge of an agent specifies which mesh to use and how much of it is known at the beginning of a game; furthermore, behavioural models can add or remove navigation areas (converted in polygons by the Path Planning API).

Edges in the navigation graph carry a weight, by default representing the euclidean distance between the centroids of the two polygons correspondent to two graph nodes. These weights can be manipulated by behavioural models to convey preferences to the path planner; this is done by specifying the weight for an entire area, which is like altering the area's distance from the remaining navigable areas.

The path planner computes the shortest path from a source point to a destination point by using the weigths and applying the well known A^* algorithm.

Steering and obstacle avoidance. The steering module moves the NPC controlled by the agent along a path computed by the path plan-

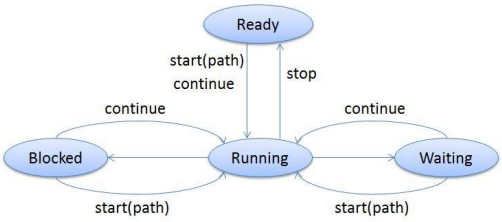


Figure 3. Steering FSM

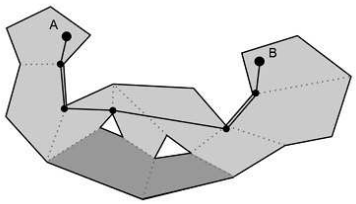


Figure 4. Trajectory generated by the Funnel algorithm

ner. To this end, it computes and updates a trajectory that avoids obstacles and moves the NPC along the points of the trajectory. While the path is computed from the start point to the destination point, the trajectory is computed locally, that is, from the current NPC position up to a maximum distance. The trajectory is frequently updated so that it continuously adapts to changing conditions. The trajectory is computed inside a "global path", i.e. the sequence of polygons computed by the path planner forming a tunnel in the selected navigation mesh. Only obstacles inside this tunnel, perceived by the agent and close to the current position of the NPC are considered by steering, which considers also their semantic properties; most importantly, objects classified as gates and in a "closed" state are not avoided. When the agent perceives that an obstacle has moved then the trajectory is immediately re-calculated.

Steering is a Finite State Machine, illustred in Fig. 3. The agent (that is, its navigation behavioural model) can query its state and send inputs that will cause state transitions; in particular, the behaviour can start steering on a selected path, stop it and later resume it on the current path or re-start it on a different one.

While Running, steering moves the NPC by calling PRESTO's "MOVE" action, which in turns controls the body's animation concerning legs or other moving parts (e.g. wheels), translate the NPC in space at the desired speed and adjust the NPC position on the ground. MOVE modifies the speed according to its initial value, providing any required acceleration; a complementary STOP action decelerates the NPC.

The Blocked state is entered when steering fails in computing a trajectory because the path is obstructed by too many obstacles. As discussed below, it is left to the behaviour to take a decision, e.g. waiting and later resuming or temporary removing the obstructed polygon from the agent's navigation graph and recomputing the path.

The Waiting state is entered when the NPC cannot go further because it is in front of a closed gate. Steering moves the NPC to an appropriate distance before entering Waiting. At this stage, the behaviour has to take an action depending on the gate's type, for example a door must be opened or an elevator must be called. Once the action has been performed, steering can be resumed. Note that the behaviour may decide to abort steering and change path because, for instance, the opening action fails for some reason not under navigation's control (e.g., the goal of opening a door cannot be achieved because a key is required and not owned by the NPC).

Steering trajectory computation. The trajectory is first computed ignoring ostacles, using the Funnel algorithm [4]. This algorithm is also known as "string-pulling" because the trajectory being generated is like a string pulled from the two extremes (Fig. 4).

The generated trajectory is modified to avoid obstacles, represented with simple geometries, like circles and rectangles, enlarged by the agent radius; an example of the algorithm is in Fig. 5. As first

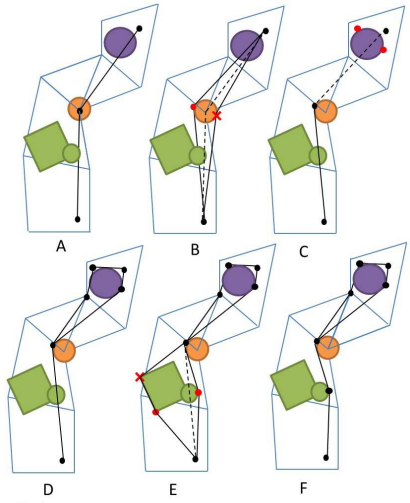


Figure 5. Obstacle avoidance algorithm, A: the output of the Funnel algorithm. B: the trajectory point is inside the orange obstacle, the right side is rejected. C: the new segment intersects the violet obstacle. D: the trajectory is recomputed to attach the two sides, but the first segment intersects the green obstacle cluster. E: the left side is rejected because a point is out of the path. F: the two final trajectories.

step, obstacles that intersect each other are clustered; each isolated obstacle forms a cluster by itself. Then each cluster is checked for intersections with the trajectory segments. If a segment intersects the cluster, the segment is discarded and two poly-lines are computed from its starting point to its ending point, passing to the right side and to the left side of the cluster. If no poly-lyne is within the path, the steering state is set to Blocked and the algorithm is stopped, eventually invoking the higher-level behavioural model. If exactly one of the computed poly-lines is inside the path, then the intersecting segment is substituted with that one. If both the poly-lines are inside the path, then the trajectory is duplicated. At this stage the checking process is repeated recursively on the resulting trajectories to handle further intersections with other clusters. The final output of the algorithm, if successful (i.e. if the Blocked state is never reached), is one or more trajectories; one is eventually chosen at random, to prevent the oscillations that typically arise when NPCs facing each other use the same deterministic steering algorithm.

5 HIGHER-LEVEL NAVIGATION BEHAVIOURAL MODELS

Navigation control in DICE is split in two types of behavioural models. One type, identified as "navigation BM" in Fig. 1 and 2, satisfies the navigation goals submitted by decision-making behaviours (e.g., of reaching a destination); slightly different navigation models are provided that depend on the main physical features of the NPC, e.g. of being a human rather than a vehicle, and consequently on the NPC's ability to move and affect the environment. As mentioned above, the navigation BM runs in its own intention tree (thread of execution) concurrently with decision-making and other body-controlling behaviours. The navigation BM calls path planning and controls steering, acting according to the latter's indication in particular when entering the Blocked or Waiting states. A number of different decisions can be taken according to the model and to the semantics of gates or obstructing objects, which may in turn cause goals to be submitted to other body-parts behaviours (e.g. opening a door, calling a lift, and so on).

A second type of behavioural model, referred to as "navigation capabilities" and included as a decision-making module in DICE, looks after some of the cognitive aspects of navigation. In particular, the navigation capability of an NPC decides which mesh to use on creation, then changes the default speed, default animations and so on according to the current sub-rational state of the agent (i.e. its moderators and cognitive parameters). Thus, PRESTO can provide capabilities specialized e.g. for quiet or excited people, for permanent or temporary physical impairments, for different types of vehicles, and so on. Navigation capabilities may access the cognitive state to tune their parameters (e.g. speed or animations); furthermore, behavioural rules may be defined to switch navigation capabilities entirely during a game depending on the NPC's moderators. For instance, a high level of fear may select a model whose default speed is running and movement animations jerky, while a high level of fatigue may select a model doing exactly the opposite. Furthermore, the navigation capabilities satisfy goals concerning path selection, such as "stay out of sight of entity E" or "don't go thru location L" (which may have been classified as dangerous by a decision-making model according to the appraisal rules of the agent), by taking note of what to avoid and manipulating the navigation graph accordingly, based on current knowledge and the flow of perceptions.

Behavioural models in DICE have their own configuration parameters, called "background knowledge". As mentioned above, the background knowledge of the navigation capability of an agent determines how much the agent knows *a priori* about the environment – it can be everything or being limited to a few areas; the navigation graph is created accordingly. The flow of perceptions arriving from the PRESTO infrastructure includes also the visible navigation polygons of the various meshes; this data is used by the navigation capability to update the navigation graph. The cognitive model of DICE, not discussed here, looks after short-term memory management, which includes calling the navigation capability to purge the navigation graph; that is, the agent literally forgets about where to navigate according to timing and frequency of perceptions from the environment. Out of scope of the navigation subsystem, and not discussed here, is a "search" behaviour, which is a set of decision-making procedures that can be started when a navigation goal fails with an "unknown path" error.

In the hospital fire scenario presented in the introduction, the navigation capability of a patient on a wheel chair would use a different mesh than the one selected for a visitor with normal walking capabilities, e.g. to avoid steps and stairs. The patient's background knowledge would include the navigation areas of the entire ward (since she has been there for a while) while the visitor's knowledge would be initially empty and populated while she moves in the ward; a decision-making procedure of the visitor that invokes a goal such as "go to patient room nr. 3" would initially fail because, indeed, no path can be computed and a search behaviour would need to be invoked allowing the progressive discovery of the navigation areas of the selected mesh. If, at any time during the game, a fire alarm starts ringing, its perception on both visitor and patient would trigger a (decision-making) reaction that is handled different according to the currently active behavioural models, which in turn may depend on cognitive states such as fear. The perception of smoke and fire would submit goals such as "don't go thru that area" handled by the navigation capability as mentioned above. A rationally-behaving NPC that knows the position of a location ontologically classified as "fire exit" would navigate to the latter, with a speed and a modality that depend on the currently active navigation capability (excited / not excited, walking / pushing the wheel chair); an NPC that doesn't know about fire exits or that it's too fearful to act rationally would run to the closest exit.

Queuing and other coordinated behaviour. Steering looks after obstacle avoidance and thus somehow takes care of certain crowding behaviours. However, proper coordination is a matter for decision making at least partially outside of the scope of navigation. Work is in progress on game-theoretical descriptions of queuing and access to shared resources that allow the definition of policies at a very abstract (meta-) level. This exploits the support in DICE for introspection, semantic tagging of goals and plans, dynamic assignment and aborting of goals and intentions as well as the ability to dynamically manipulate semantic tags of any entities (including NPCs) offered by PRESTO. The specification of policies is expected to substantially reduce the coding required by models and allows the reuse of the same coordination patterns in many different situations, e.g. for queuing to pass through a gate (which will be part of the navigation BMs) as well as for queuing at the entrance of an office or at the cashier in a supermarket (which are decision-making behaviours not related to navigation goals).

6 CONCLUSIONS AND FUTURE WORKS

At the time of writing, testing and performance evaluation are still in progress. Initial results show that the navigation meshes are surprisingly small even in very large and complex indoor and outdoor environments; in turn, this makes the maintenance of per-agent navigation graphs and path planning computationally well affordable. Other work in progress concerns coordinated behaviour, as discussed above.

While the navigation algorithms described in this paper contain a few novelties, we believe that the most interesting part of the PRESTO approach is the coordination among navigation behaviour, other concurrent body-controlling intentions and the two-level decision making, all affected by cognitive elements such as short term memory management and emotions. When combined with its semantic facilities and end-user development tools for the creation of NPC behavioural profiles, PRESTO represents an interesting improvement to the state-of-the-art of game platforms, especially for serious game development.

ACKNOWLEDGEMENTS

We thanks all other members of Delta Informatica's technical team (Matteo Pedrotti, Mauro Fruet and Michele Lunelli). PRESTO has been funded by the Autonomous Province of Trento (PAT), Italy.

REFERENCES

[1] Michael E. Bratman, *Intention, Plans, and Practical Reason*, Harvard University Press, November 1987.

[2] Paolo Busetta, Chiara Ghidini, Matteo Pedrotti, Antonella De Angeli, and Zeno Menestrina, 'Briefing virtual actors: a first report on the presto project', in *Proceedings of the AI and Games Symposium at AISB 2014*, ed., Daniela Romano, (April 2014).

[3] Alex J. Champandard, *An Overview of Navigation Systems*, volume 2 of *AI Game Wisdom*, 131–139, Charles River Media, Massachusset, 2004.

[4] Xiao Cui and Hao Shi, 'An overview of pathfinding in navigation mesh', *IJCSNS International Journal of Computer Science and Network Security*, **12**, 48–51, (December 2012).

[5] Mauro Dragoni, Chiara Ghidini, Paolo Busetta, Mauro Fruet, and Matteo Pedrotti, 'Using ontologies for modeling virtual reality scenarios', in *to appear in Proceedings of ESWC 2015*.

[6] Rick Evertsz, Matteo Pedrotti, Paolo Busetta, Hasan Acar, and Frank Ritter, 'Populating VBS2 with Realistic Virtual Actors', in *Conference on Behavior Representation in Modeling & Simulation (BRIMS)*, Sundance Resort, Utah, (March 30 – April 2 2009).

[7] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf, 'End-User Development: An Emerging Paradigm', *End User Development*, **9**, 1–8, (2006).

[8] Zeno Menestrina, Antonella De Angeli, and Paolo Busetta, 'APE: end user development for emergency management training', in *6th International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2014, Valletta, Malta, September 9-12, 2014*, pp. 1–4. IEEE, (2014).

[9] Anand S. Rao and Michael P. Georgeff, 'Bdi agents: From theory to practice', in *IN PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS (ICMAS-95*, pp. 312–319, (1995).

[10] Frank E. Ritter, Jennifer L. Bittner, Sue E. Kase, Rick Evertsz, Matteo Pedrotti, and Paolo Busetta, 'CoJACK: A high-level cognitive architecture with demonstrations of moderators, variability, and implications for situation awareness', *Biologically Inspired Cognitive Architectures*, **1**, 2–13, (July 2012).

[11] Paul Tozour and Ion Storm Austin, *Building a Near-Optimal Navigation Mesh*, 171–185, AI Game Wisdom, Charles River Media, Massachusset, 2002.