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.

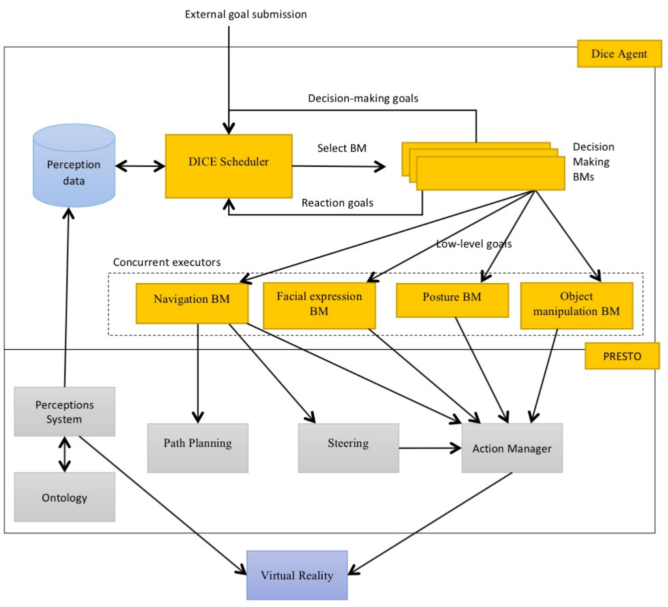

Figure 1. Simplified DICE architecture with navigation highlighted (BM: Behavioural Model)

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.

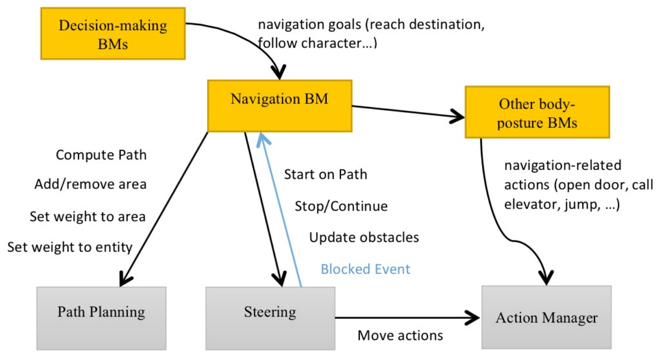

Figure 2. Navigation subsystem architecture

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-

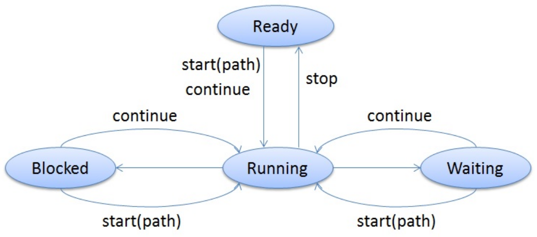

Figure 3. Steering FSM

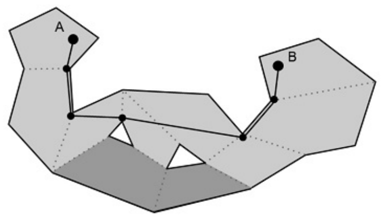

Figure 4. Trajectory generated by the Funnel algorithm

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

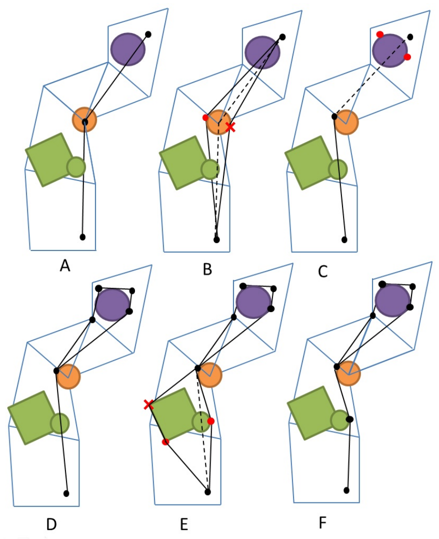

Figure 5. Obstacle avoidance algorithm, A: the output of the Funnel algorithm. B: the trajectory point is inside the orange obstacle, the right side is rejected. C: the new segment intersects the violet obstacle. D: the trajectory is recomputed to attach the two sides, but the first segment intersects 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.