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

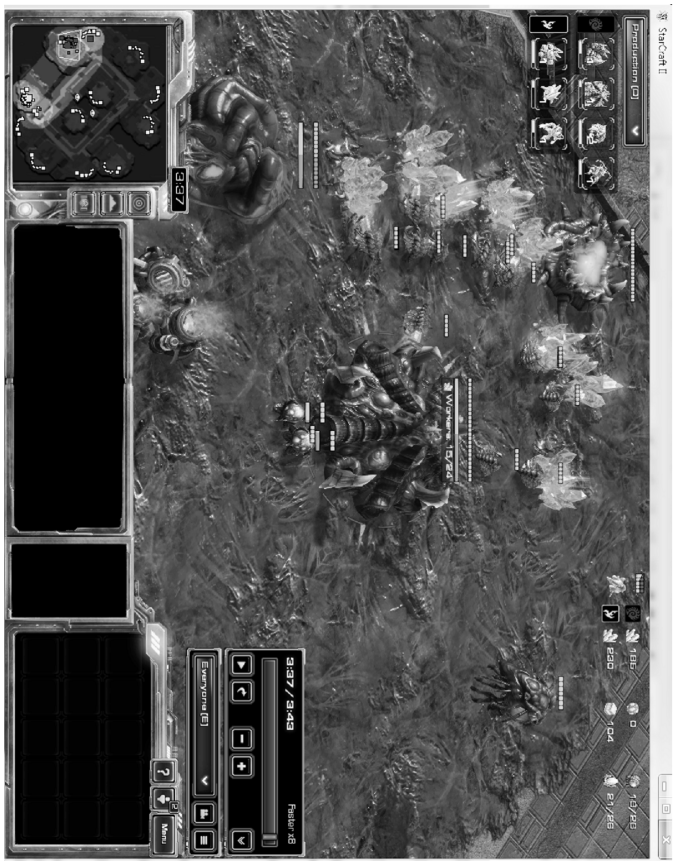


Figure 2. Player 1 - Matching production templates

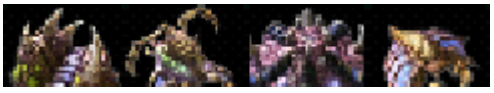


Figure 3. Player 2 - Matching production templates

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.

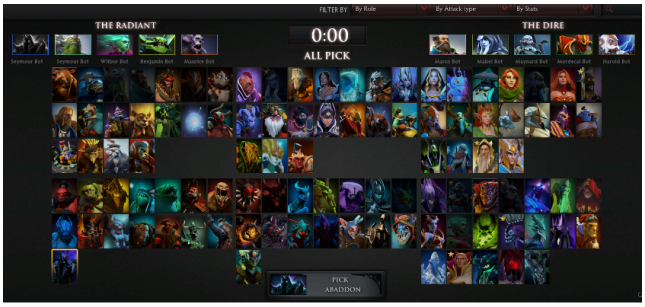
Heroes fall into general categories based on their abilities and how they interact with other heroes within the game. The picking process leads to a diverse set of combinations that can be formed between the 2 teams. However, some of these combinations are weaker than others due to the interaction of 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

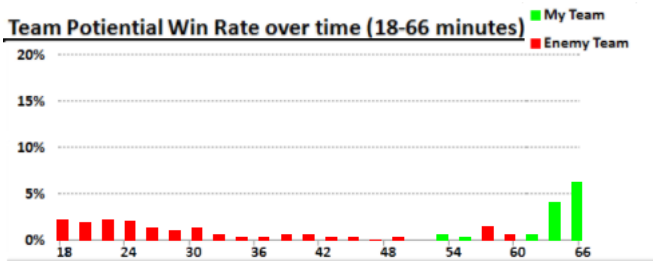
Figure 7. DotA 2 Hero Selection Screen



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).