Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

THE NEED FOR A REAL TIME STRATEGY GAME LANGUAGE

Roy Hayes
Peter Beling
William Scherer

University of Virginia
System and Information Engineering
151 Engineer's Way
Charlottesville, VA 22904, USA

ABSTRACT

Real Time Strategy (RTS) games provide complex domain to test the latest artificial intelligence (AI) research. In much of the literature, AI systems have been limited to playing one game. Although, this specialization has resulted in stronger AI gaming systems it does not address the key concerns of AI research, which focuses on the development of AI agents that can autonomously interpret, learn, and apply new knowledge. To achieve human level performance, current AI systems rely on game specific knowledge of an expert. This paper proposes a RTS language in hopes of shifting the current research focus to the development of general RTS agents. General RTS agents are AI gaming systems that can play any RTS game, defined in the proposed RTS language. The structure of the RTS language prevents game specific knowledge from being hard coded into the system, thereby facilitating research that addresses the fundamental concerns of artificial intelligence.

1 INTRODUCTION

Over the last decade researchers have begun focusing on artificial intelligence (AI) for real time strategy (RTS) games. RTS are popular computer games, which feature developing and supporting an army of different unit types and buildings. Players perform their actions in real time, as opposed to turn-based decision making in classic games such as chess. Players simultaneously control many, perhaps hundreds, of units and buildings. To increase the difficulty of these games players' knowledge of the playing map is restricted to a small area around their own and allied military assets.

Researchers have generally limited their AI implementation to subcomponents of games, such as resource management or path finding (McCoy and Mateas 2008). That is to say, researchers have sought to develop a solution to a single aspect of RTS game. In recent years there has been a push to develop AI systems capable of playing the full game. However, these implementations are limited to playing a single RTS game and, consequently, may leverage expert knowledge and game specific information. Game-specific research may yield powerful outcomes for specific games, but it does not address the root concern of AI research, which is to develop agents that can autonomously interpret, learn, and apply new knowledge (Kaiser 2005). We assert that, in many threads of published literature, AI RTS agents have exhibited improvement because of more efficient programing and better human implemented strategy and not because of advancements in artificial intelligence.

We believe there is a present need for an RTS language. Such a language would allow for the creation of an agent that can autonomously learn how to play a previously unknown game. Of particular interest is the class of General Game Playing (GGP) agents originally developed for relatively simple games such chess and tic-tac-toe and more recently extended to poker (Thielscher 2011). By allowing these GGP agents

978-1-4799-7486-3/14/$31.00 ©2014 IEEE

to play against one another over a variety of games it becomes apparent, which AI system is more adept at learning and applying knowledge.

There are several advantages to developing general RTS agents that leverage GGP methodology. Improvements in the performance of these agents, as measured by win percentage over both human and other AI opponents, would be due to their ability to gather and apply new knowledge, and so would demonstrate an improvement in our collective AI knowledge. Additionally, the creation of general RTS agents would accelerate development of new RTS games. Developing the AI scripts found in today's games is a resource intensive project. The speed of game development would increase if AI systems become capable of learning how to play new RTS games. Lastly, general RTS agents will improve player experience. Traditional game AI systems are non-adaptive. Therefore, once the player has determined a weakness it becomes trivial to win. A general RTS agent would implement an adaptive learning system, thereby mitigating this problem (Synnaeve and Bessiere 2011).

This paper proposes a RTS language that would support the development of a general RTS agent. The authors appreciate the complexity of RTS games and therefore future iterations are expected to expand on the language.

The outline of the paper is as follows. To orient the reader we provide an overview of previous research in real time strategy AI and general game language. Then we present a proposed general RTS language, followed by a discussion of the broader impacts. Lastly, we conclude with final thoughts and suggestions for future work.

2 RELATED WORK

2.1 Real Time Strategy Artificial Intelligence Research

Real time strategy (RTS) games are noticeably more complex than traditional board games, such as Chess and GO. They require that players master several subtasks, including resource management, high level strategy, and opponent prediction. Researchers have developed AI systems that attempt to solve these problems but to date no AI system can beat expert human players (Ontanón, Synnaeve, Uriarte, Richoux, Churchill, and Preuss 2013). This is partially due to the nature of the design goals for AI in RTS games, which center on challenging but not necessarily dominating human players. Additionally, traditional RTS AI systems leverage procedural scripts, and humans have proven adept at exploiting the predictability and other weakness of such systems (Dahlbom 2004).

Some academic research has been devoted to developing AI systems that can rival expert players in principal RTS tasks. Notable in this regard is the task of resource management. A primary challenge in RTS games is to gather resources, which support the construction of buildings and training of units. Additionally, players must decide how to allocate their resources, deciding what buildings to make and units to train. Van der Bloom *et al.* developed an AI system that augments its behavior based on resource densities found in the current game map (van der Blom, Bakkes, and Spronck 2007). The AI system leverages apprenticeship learning and decision trees. The internal nodes of the decision tree are based on a feature such as the distance between resource deposits and the main base. Branches correspond to feature values and the terminal nodes correspond to an action to be taken. The decision tree is generated using an ID3 algorithm, similar to recursive partitioning, where actions are separated based on map conditions.

The formulation in (van der Blom, Bakkes, and Spronck 2007) allows the AI system to adapt its action based on map conditions that it has not seen. Because the system implements an apprenticeship learning methodology, however, its efficacy is limited by the knowledge of the expert. This method is beneficial in developing a stronger AI system that can adapt to different terrain scenarios but it does not address the problem of autonomous learning. Additionally, experts will still have to develop scripts to generate the training scenarios that the algorithm will use.

High level strategic thinking consists of determining when and how an army should attack the opposing force. To address this problem Sailer *et al.* implemented an AI system that utilized simulations and

Nash-optimal strategies (Sailer, Buro, and Lanctot 2007). Their AI system simulated a set of strategies against each other, resulting in an *n*x*n* matrix that allows the AI system to compare strategies. An evaluation function was used to determine the winning strategy. This function was relatively simple and relied on win percentage, game length, and military strength.

The AI system in (Sailer, Buro, and Lanctot 2007) can defeat static script AI. However, similar to the method in (van der Blom, Bakkes, and Spronck 2007), this system relies on expert knowledge and obfuscates a large number of subtasks. The AI systems effectiveness is limited by the expert defined strategies. General RTS agents will have to adapt their strategies as they learn to play the game, providing them more flexibility and potentially improving performance.

Opponent modeling is an important part of RTS games. High level strategic thinking and resource management decisions are dependent on the perceived opponent strategies. Well balanced RTS games implement a rock-paper-scissor methodology, where one unit has advantages over specific unit types and disadvantages against another. Therefore, the effectiveness of an attack is not only dependent on how it is carried out but also the unit composition of the opposing forces.

When planning an attack the AI system must predict an opponent's high level strategy and military composition. Several researchers have addressed the topic of opponent modeling. Bakkes *et al.* utilized opponent modeling to augment their RTS AI system (Bakkes, Spronck, and Jaap van den Herik 2009). Opponents were described by a set of features such as unit types and buildings observed. Using K-means clustering, opponents were segmented into strategy groups and the AI system chose an appropriate counter strategy. Rashad expanded on this methodology by using a rough set theory and neural networks for opponent modeling. Rough set theory identifies dependent variables, thereby decreasing the feature space (Rashad 2012). Neural networks provide better classification accuracy than K-means. However, neural networks are considered black boxes; therefore an operator cannot determine why groups are classified together.

In traditional opponent modeling the expert identifies a set of features that will be important in RTS games, such as number of a specified unit type. The system than uses a training set of games, played by experts, to identify common strategies. General RTS agent methodology alleviates the need for experts to play a set of games because the training set is generated by self-play. Self-play is the process by which an autonomous agent plays itself in an attempt to learn successful strategies. Under self-play an expert is only required to list features that may be relevant for any RTS game.

Researchers have combined the aforementioned subtask to create AI systems that can outperform script RTS agents. McCoy and Mateas published one of the first papers devoted to examining an AI system that combines multiple AI subcomponents to play RTS games (McCoy and Mateas 2008). Their agent was comprised of distinct AI managers, each of which was responsible for at least one of subtasks identified in their analysis of human play. It is important to note that the strategy each manager implements is augmented based on real time game data. However, the strategies that are implemented are written by experts and not autonomously discovered by the AI system.

Recently Weber *et al.* directly expanded on McCoy and Mateas's work by developing a system that was capable of micromanagement, terrain analysis, and more reactive to real time updates of game states. Weber *et al.*'s AI system was able to rank as an amateur StarCraft player in the International Cyber Cup (Weber, Mateas, and Jhala). However, the strategies implemented by the system were developed by human expert and not derived through an organic learning process. The next section describes the Game Description Language.

2.2 General Description Language

AI practitioners have focused on developing better game playing agents, but in most case have relied on game specific knowledge and increases in computing power. The underlying AI has not been made smarter, rather endowed with expert knowledge and faster processors. Deep Blue, the world's best chess system, beats human opponents by calculating 18 moves ahead. In essence the AI gaming community has become

the proverbial steam engine, able to perform certain cognitive tasks faster than humans. However, this line of research has not tackled the most fundamental question in AI, namely how to create a system that autonomously learns (Finnsson and Björnsson 2008).

General Game Playing research attempts to design computer agents that learn how to play games, without human intervention. These agents are given a description of a game and through a variety of techniques learn both the rules and a winning strategy for the game. Designing a program that can interpret any game prevents biases from being structured into the code. In other words, the software cannot be design to perform well for a single game.

Games are described using a first-order logic language known as the Game Description Language (GDL) (Thielscher 2010). This language allows General Game Playing computer agents to understand the rules of a game they have not seen before. This, in turn, facilitates the construction of evaluation models without human intervention. It is important to note that there are two specifications: GDL-I and GDL-II. GDL-I specifies only deterministic and complete information games, while GDL-II extends GDL-I to specify incomplete information and stochastic games.

A game can be defined as any finite horizon decision problem. A game can be made up of multiple decision makers (players) and multiple decisions (moves) (Thielscher 2010). The outcome of the decision can depend on one or multiple players' moves. A sub-class of games consisting of only one player is known as a puzzle. Puzzles can be used to represent simple games such as the Tower of Hanoi or more practical decision problems such as supply chain management.

Games can be segmented into categories based on the information that a player receives. If a player is privy to all the information about a state, then the game is known as perfect information game, otherwise it is known as an imperfect information game. An example of a perfect information game is checkers, while an example of an imperfect information game is poker. Games can be further segmented into deterministic and stochastic games. A deterministic game is where the outcome of a move is known beforehand (chess) and a stochastic game is the converse (Dice Roll).

The Game Description Language describes an entire game using only 12 keywords. The rest of the information is game specific. This is an important distinction because game specific information can change and there is no standardize format to write this information in. For example a cell on a checkers board could be referred to as any one of the following symbols: a1, (a, 1), (0, 1), or VAS. Therefore, dictionary lookups are not capable of determining the rule. Instead higher level techniques are required. The list of 12 keywords and their meanings can be found in (Thielscher 2010).

The game description language was originally designed for turn based games. Each turn a player is given propositions, a set of statements which are true in the given state. Additionally, the player is given possible steps that can be made from that state and how they will affect the game. The problem with this formulation is that as the game increases in complexity so do the number of propositions needed to describe the game state. In addition, the number of possible actions and transitions also increase with complexity.

Real time strategy (RTS) games are not conducive to the original format of GDL. Turn based updates are unfeasible for RTS games because they are played over a continuous time-frame. Furthermore, a player can control hundreds of buildings and units, leading to an exponential growth in possible actions and state transitions. Therefore, defining propositions for a RTS game's possible states and transitions is impracticable.

For an autonomous system to be competitive it must be able to act based on an internal view of the world. Parsing updates takes time, which limits the autonomous agent's ability to perform useful actions in the RTS game. Lewis *et al.* found a correlation between winning the game and the action rate of the players (Lewis, Trinh, and Kirsh 2011). A player that performs more actions than their opponent improves their position faster and thus giving them an advantage. Although, this is correlation and not causation it highlights the need for efficient updates, which is not possible under the current GDL language format.

An autonomous system that can efficiently reconcile any errors in its internal view with periodic update would need a basic understanding of the game, prior to learning game specific information. In other words,

the system should understand it is playing an RTS game and can assume that an army is made up of units, which will occupy a world. This allows for time saving domain knowledge to be added into the system and increases efficiency.

As mentioned, there currently is no autonomous agent that can beat an expert human in RTS games. Therefore, domain knowledge should be available to the autonomous agent. Leveraging domain knowledge limits the amount of information the autonomous agent needs to learn and may result in better performance. Additionally, making domain knowledge available to the agent does not violate the core goal of AI research, which is to develop agents that autonomously learn and utilize knowledge. An agent will still need to learn and utilize game specific information. To develop such an agent a RTS language needs to be created. The next section proposes a Real Time Strategy language.

3 PROPOSED REAL TIME STRATEGY LANGUAGE

3.1 Language Syntax

Similar to the Game Description Language, the Real Time Strategy language leverages a set of key words that will be utilized by all real time strategy games. However, to counter act the previously mentioned deficiency of the Game Description Language, the Real Time Strategy language will be implemented in a XML format, which allows for rapid interpretation of game specific information. The general RTS agent is given a game description prior to playing the game. This description details the possible buildings and units that can be constructed. Once the game begins updates are issued by the request of the agent. These updates allow the autonomous agent to change its internal view to reflect the true game state. It is not possible to detail the language in its entirety in such a small paper. However, examples and supporting documentation can be found in the full language documentation (Hayes, Beling, and Scherer 2014).

The description of an RTS game can be divided into two high level categories, a description of the environment and a description of the armies. Higher level tags can be thought of as entities representing factions, buildings, and units. The elements within the tags can be considered attributes of these entities. For example, by adding elements within a knight tag, the can be given properties such as attack strength and speed. The uses of key words allow the RTS AI to interpret game specific information. The paper will utilize Warcraft 2 as an example implementation of the Real Time Strategy Language. Warcraft 2 is published by Blizzard Entertainment. The bold words in the example are keywords, the remainder is game specific information.

Real Time Strategy games have common elements among them. They usually have several playable factions and several resources that need to be collected to allow for the creation of buildings and units. Therefore, factions and resources are made keywords. The name of the specific factions and resources is game specific information. Within each resource elements a player is given the amount currently available of that resource. This format is demonstrated below.

< **Factions** >
Humans
Orcs
< /**Factions** >
< **Resources** >
< *Wood* > 100 < /*Wood* >
< *Gold* > 100 < /*Gold* >
< *Oil* > 10 < /*Oil* >
< *Food* > 5 < /*Food* >
< /**Resources** >

This simple example illustrates a combination of key words and game specific information. There are two factions in the game, one corresponding to Humans and the other to Orcs. The resources tag states the game specific resources available in the game.

Factions can possess different buildings and units, with different abilities. Game specific information can be used as tags to specify the properties of that object. Below is an example of a building and a unit specification. To save space the conventional XML format has been shortened.

< *Humans* >< **Building** >
< *TownHall* >
< **UniqueID** > *TownHall*1 < /**UniqueID** >
< **Health Point** > 1200 < /**Health Point** >
< **Terrain** > *Snow* < /**Terrain** >
< **Action** > *Idle* < /**Action** >
< **Shape** >
< **Square** > 2 < /**Square** >
< /**Shape** >< **Position** >
< **X,Y** > 120, 120 < /**X,Y** >
< /**Position** >< **Vision** > 1 < /**Vision** >
< **Build Speed** > 30 < /**Build Speed** >
< **Enemy** >< /**Enemy** >
< **Require** >< **Resource** >
< *Wood* > 800 < /*Wood* >
< *Gold* > 1200 < /*Gold* >
< /**Resource** >< /**Require** >
< **Upgrade** > *Keep* < /**Upgrade** >
< **Purpose** >< **Process** >
< **Resource** > *Wood*, *Gold*
< /**Resource** >< /**Process** >
< **Build** > *Peasants* < / **Build** >
< /**Purpose** >< /*TownHall* >
< /**Building** >< /*Human* >

The building described above is town hall, which is a human building. Town halls have a health point of 1200. A shape must be specified, along with a description of how many cells it takes up. The town hall is a square that is 2 units wide, giving it a total area of 4 grid cells. There are a set of defined shapes, which can be found in the full language document (Hayes, Beling, and Scherer 2014). The building is given an x/y position and a vision of 1. This means it can see 1 unit length away from it, allowing the town hall to generate a list of enemy units and buildings that are currently in its field of view. The town hall must be built on snow and requires both wood and gold resources for construction. Currently the town hall is idol. Additionally, the town hall can be upgraded to a keep. The building specializes in processing wood and gold, adding them to the player's reserves, as well as training peasants. Similar to buildings, a unit is often army specific, with specialized functions. Below is an example of the Elvin Archer unit in Warcraft 2.

< *Human* >< **Unit** >
< *ElvinArcher* >
< **Health Point** > 40 < /**Health Point** >
< **Build Time** > 15 < / **Build Time** >
< **UniqueID** > *Archer*1 < /**UniqueID** >
< **Armor** < *Shield* > 4 < /*Shield* >
< /**Armor** >< **Shape** > **Circle** < /**Shape** >
< **Size** > 0.5 < /**Size** >< **Enemy** >< /**Enemy** >
< **Action** > **Idle** < /**Action** >
< **Position** >
< **X,Y** > 120, 120 < /**X,Y** >
< /**Position** >< **Terrain** > *Snow* < **Terrain** >

< **Attack** >< *Arrow* >
< **Range** > 4 < /**Range** >
< **Damage** > 3 – 9 < /**Damage** >
< **Recharge** > 2 < /**Recharge** >
< **Shape** > **Point** < /**Shape** >
< **Terrain** > *Air*,*Snow* < /**Terrain** >
< /*Arrow* >< /**Attack** >
< **Vision** > 5 < /**Vision** >
< **Speed** > 3 < /**Speed** >
< **Require** >< **Resource** >
< *Gold* > 500 < /*Gold* >
< *Wood* > 50 < /*Wood* >
< *Food* > 1 < /*Food* >
< /**Resource** >< /**Require** >
< /*ElvinArcher* >
< /**Unit** >< /*Human* >

The Elvin Archer is an early game unit for the human faction. The archer has relatively low health points and armor but boast larger attack damage, greater speed and vision than other early units. The archer can only traverse snow, meaning it cannot enter a tile that is covered in wood. However, the archer can attack units that currently occupy snow and air regions.

Maps describe the environment in which the game is played. This is where different terrains are specified, as well as the amount and location of different resources. Below is a brief example of the map specification system.

< **Map** >< **Name** > *Hills* < /**Name** >
< (0,0) >
< **Terrain** > *Snow* < /**Terrain** >< *Gold* > 1000 < /*Gold* >< /(0,0) >
< (0,1) >
< **Terrain** >< *Wood* > 300 < /*Wood* > *Snow* < /**Terrain** >< /(0,1) >< /**Map** >

The above specification demonstrates several complex features. In the first cell the terrain is specified as snow but there is a gold deposit worth 1000 units on it. In the second cell notice the resource is specified within the terrain tags. This indicates the terrain starts as wood but once the 300 units of wood are removed it becomes snow, this allows for the generation of temporary obstructions.

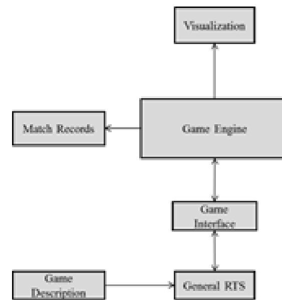
An update about the state of the game has the potential to contain a large amount of data. For example, in a standard 128 X 128 map there is information about 16,384 grid cells, as well as numerous buildings and units. However, this data is limited by the vision constrains of the agent's current buildings and units. Furthermore, the XML structure allows for rapid parsing of the data into relevant information.

3.2 Game Management

The RTS game management system has already been developed for artificial intelligence competition. The game management system accepts the connection of an AI system. The AI system designates which faction it intends to play as. The game manager selects a map, informs the AI system of their opponent's faction, and sends a signal when the match begins. The game management system maintains the true state of the game. That is to say the game management system keeps track of units, buildings, resources and map conditions. By querying the game management system for updates about the current state of the map, an AI internal state can be brought into alignment with current state of the game. Below is a diagram of the system.

The authors contend that there is no need for the underlying system to be modified. Rather the messages should be standardized, thus allowing an AI to interact with any RTS game. For standardization purposes updates of the game state should be given in the aforementioned formats. That is to say, unit, building,

Figure 1: Game Management Diagram



and map information should follow the previously mentioned XML format. Game designers can determine what information to issue in updates. For example, if the Archer always has a vision of 5 then the game designer may choose not to send vision information in updates, as the AI system should already be aware of it.

Developing a standard format for issuing the orders will allow AI systems to effectively play multiple games. This standard format will be interpreted by the game interface and passed to the game engine. Below is the proposed format for a standard command set.

1. Construct Building - Construct(Building Name, Unit Unique ID, X-position, Y-position)
2. Move a Unit or Building - Move(Unique ID, X-Position, Y- Position)
3. Train - Train(Building or Unit Unique ID, Name of unit/upgrade),
4. Gather Resource - Gather(Unit Unique ID, X-Position, Y-Position)
5. Attack - (Allied Unique ID, Attack Type, Enemy Unique ID)
6. Action - (Game Specific Action, Allied Unique ID [], Enemy Unique ID [], X-Position [], Y-Position [])
7. Update

The autonomous AI system must specify the desired building, construction location, and worker unit to correctly issue a construction command. Depending on the games both buildings and units may be allowed to move. In games where buildings are not allowed to move the general RTS agent will not apply the move command to buildings. Training encompasses building units and their associated upgrades. As such, both buildings and units may be required to implement the training function. Gathering resources requires the worker unit to be identified and the location of the resource. When issuing an attack both the autonomous system unit/building and the enemy unit/building must be identified. Action is a function that allows for game specific actions to be implemented. The effects of these actions will be described in the game description. The [] symbol denotes that these variables are arrays, which may be left empty in the event the action does not require them.

4 BROADER IMPACTS

Developing a Real Time Strategy Language will allow the development of general RTS AI systems. These systems will be required to learn and apply knowledge in a large, dynamic environment. While general AI systems could lead to the development of better RTS AI systems, they also can have an impact outside of video games. As the United State population grows older there is growing research into the cognitive ability of the elderly (Ball et al. 2002). Specifically researchers are concerned about the decline in cognitive function as people age. Basak *et al.* have found that playing RTS video games can improve an elderly person's performance in task switching, working memory, visual short-term memory, and reasoning (Basak, Boot, Voss, and Kramer 2008).

Basak *et al.* theorized that the improvement seen in cognitive task could be caused, "[because the game] keeps the player on his or her toes; one is always changing priorities." If the improvement in cognitive ability is due to the elderly person strategizing under uncertainty then we assert the benefits of the RTS game will decrease as a person becomes more familiar with the game. As previously mentioned modern RTS games leverage scripts, which humans have proven adept at exploiting the predictability and other weakness of such systems. Therefore, as an elderly person plays the game longer there is less uncertainty and thus the benefits of the game will decrease. A general RTS AI system must have the ability to learn and adapt, thereby preventing a player from capitalizing on the predictability of the AI. A general RTS AI will be able to sustain the cognitive benefits longer than a script base AI system

This framework can also be applied to domains outside of video games. As mentioned earlier Real Time Strategy games are made up of several sub-tasks. The selecting the order units and buildings construction is multi-objective a resource constrained scheduling problem. A general RTS AI will implement a simulation optimization approach to solve this complicated problem. This framework can be adapted to solve real world multi-objective resource constrained scheduling problem, such as the construction of large instillations (Long and Ohsato 2009). Additional solutions to other real-time strategy game sup-problems, such as opponent modeling, can also be expanded to other domains.

5 CONCLUSION AND SUGGESTIONS FOR FUTURE WORK

Real time strategy research is moving in the direction of Deep Blue. In the future, AI systems will be able to beat the best players at games like StarCraft. However, if the research continues on the current path this will be accomplished by implementing game specific knowledge and human based strategies. That is to say, no progress will be made on making an AI system that autonomously learns.

By proposing a RTS language the authors hope to shift the current research focus to the development of general RTS agents. It should be noted that, unlike their GGP counterparts, these agents will not be able to play a wide class of games. Instead these AI systems will be limited to playing RTS games, which is analogous to an AI system capable of playing only symmetric complete information board games, such as chess, checkers, and go. This will allow programmers to leverage domain knowledge, while preventing the use of game specific knowledge.

The authors contend that leveraging domain knowledge is acceptable due to the highly complex nature of these games. AI systems will still need to learn strategies for each new game. A solution to this problem will lead to the first AI system capable of perform adequately over multiple RTS games.

Currently the authors are engaged in implementing the proposed RTS language for several games including StarCraft and Warcraft 2. The goal is to demonstrate that it is possible to develop a general RTS system that can interpret two separate games and devise an adequate strategy for both.

REFERENCES

Bakkes, S., P. Spronck, and H. Jaap van den Herik. 2009. "Opponent modelling for case-based adaptive game AI". *Entertainment Computing* 1 (1): 27–37.

Ball, K. et al. 2002. "Effects of cognitive training interventions with older adults: a randomized controlled trial". *Jama* 288 (18): 2271–2281.

Basak, C., W. R. Boot, M. W. Voss, and A. F. Kramer. 2008. "Can training in a real-time strategy video game attenuate cognitive decline in older adults?". *Psychology and aging* 23 (4): 765.

Dahlbom, A. 2004. *An adaptive AI for real-time strategy games*. Ph. D. thesis, University of Skövde.

Finnsson, H., and Y. Björnsson. 2008. "Simulation-Based Approach to General Game Playing.". In *AAAI*, Volume 8, 259–264.

Hayes, R., P. Beling, and W. Scherer. 2014. "Real Time Strategy Language". *Arxiv* 1401.5424v1.

Kaiser, D. M. 2005. "The structure of games". In *Proceedings of the 43rd annual Southeast regional conference-Volume 1*, 61–62. ACM.

Lewis, J. M., P. Trinh, and D. Kirsh. 2011. "A corpus analysis of strategy video game play in starcraft: Brood war". In *Proceedings of the 33rd annual conference of the cognitive science society*, 687–692.

Long, L. D., and A. Ohsato. 2009. "A genetic algorithm-based method for scheduling repetitive construction projects". *Automation in Construction* 18 (4): 499–511.

McCoy, J., and M. Mateas. 2008. "An Integrated Agent for Playing Real-Time Strategy Games.". In *AAAI*, 1313–1318.

Ontanón, S., G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. 2013. "A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft". *IEEE Transactions on Computational Intelligence and AI in Games* 5 (4): 293–311.

Rashad, M. 2012. "A Rough–Neuro Model for Classifying Opponent Behavior in Real Time Strategy Games". *International Journal of Computer Science* 4.

Sailer, F., M. Buro, and M. Lanctot. 2007. "Adversarial planning through strategy simulation". In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, 80–87. IEEE.

Synnaeve, G., and P. Bessiere. 2011. "A bayesian model for opening prediction in rts games with application to starcraft". In *IEEE Conference on Computational Intelligence and Games*, 281–288. IEEE.

Thielscher, M. 2010. "A General Game Description Language for Incomplete Information Games.". In *AAAI*, Volume 10, 994–999.

Thielscher, M. 2011. "The general game playing description language is universal". In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, 1107–1112. AAAI Press.

van der Blom, L., S. Bakkes, and P. Spronck. 2007. "Map-Adaptive Artificial Intelligence for Video Games.". In *GAMEON*, 53–60.

Weber, B. G., M. Mateas, and A. Jhala. "Building human-level ai for real-time strategy games".

6 AUTHOR BIOGRAPHIES

ROY L. HAYES is a PhD Candidate in the Department of Systems and Information Engineering at the University of Virginia. His previous research leveraged agent-based modeling and machine learning techniques to model high frequency trading. His current research examines reinforcement learnings applicability to real time strategy games. His email address is rlh8t@virginia.edu.

PETER A. BELING is an Associate Professor in the Department of Systems and Information Engineering at the University of Virginia. Dr. Belings research interests are in the area of decision making in complex systems, with emphasis on Bayesian scoring models, machine learning, and mathematical optimization. His research has found application in a variety of domains, including lender and consumer credit decision-making, analysis of high frequency trading strategies, and man-machine decision systems. His email address is pb3a@virginia.edu.

WILLIAM T. SCHERER is a full professor, who has served on the University of Virginia Department of Systems and Information Engineering faculty since 1986. He has authored and co-authored numerous publications on intelligent decision support systems, combinatorial optimization and stochastic control. His current research focuses on systems engineering methodology, financial engineering and intelligent transportation systems. His email address is wts@virginia.edu.