Strategic reasoning in complex domains:

A comparative survey on scientific AI techniques to improve real-time strategy game AI.

Author: Kasper van Mens
Bachelor Thesis
Supervisor: Jan Broersen
May 14, 2012

Universiteit Utrecht

Abstract:

Over the past decade, scientific artificial intelligence research has picked up real-time strategy video games as testbed for research. The large amount of data, the dynamic environment and strategic depth of the game makes the genre a challenge for scientific artificial intelligence research. This thesis is a comparative survey of two artificial intelligence techniques from scientific research, Case-based reasoning and Dynamic Scripting, to create a strong real-time strategy game playing agent.

Contents:

1 Introduction

The computer game industry has grown enormously over the past years. One genre of games in this industry is real-time strategy (RTS). In a typical RTS game, players fight over resources, build bases and try to defeat the enemy forces. A player can choose to play versus another human player or to play versus a computer controlled opponent, called an Artificial Intelligence (AI). The AI in RTS games is of very low quality. Creating a strong AI for RTS games is very difficult [4]. The main focus of commercial game developers is to entertain the player. Therefore most of the research in improving the quality of games is done on the graphics. Games are developed under sever time constraints and most game designers do not get their hands dirty on a complex task, such as creating a human like AI [13]. However, 15 years after a machine defeated the world chess champion Kasparov, how hard can it be to create a strong RTS game AI?

To understand the complexity of a RTS game, imagine playing chess on a 512x512 board, where players can have over a hundred pieces. All pieces may move simultaneously and can have several complex abilities. Players may create new units and buildings and only have visibility within a small area around their own pieces. In AI terms, this means that players have to deal with a very complex game state, a very large action space and with imperfect information. The commercial game industry uses a script as AI in these complex environments. A script is a sequence of actions that the computer controlled player must execute. Scripts are developed by the game designers in a very late stage of game development [18]. The complexity of RTS games makes it very plausible that game designers cannot foresee every situation in the game and it is very likely that the scripts contain weaknesses [27] . Unforeseen situations by the game developers will be poorly handled by the AI and this will eventually lead to inferior game-play. Besides being weak in new situations, scripts are static [3]. The static nature of the scripts results in predictable behavior that will be easily exploited by human players. Weak, predictable and non adaptive scripts are common problems in state-of-the-art game AI [17].

Over the past decade, AI researchers have picked up RTS games as topic for their research. Scientists have argued that there is a need for scientific research on RTS game AI [4,6,9]. One of the reasons RTS games are a good testbed for AI research is because they provide a well defined environment to test and evaluate AI techniques. Most RTS games allow researchers to research loose aspects of a full game AI independently, like resource management or scouting behaviors. Commercial game developers can benefit from scientific AI research, because high level AI increases the playing challenge and is a potential selling point for a game [10,18]. Besides the benefits for commercial game developers, knowledge gained from AI research can be used for other applications such as military drones [6].

This thesis is a comparative survey of different AI techniques that are being used by scientific researchers to improve RTS game AI. Different designs are compared and the most fruitful ones are discussed. The main problem of RTS game AI is the complexity of the game. Strong RTS AI needs to be able to handle new situations and be unpredictable for the opponent. The agent must perform multiple tasks at once, in a complex and highly dynamic environment. Case-based reasoning and Dynamic Scripting have been proposed as techniques for strong RTS game-play. The question is, to what extend are the techniques suitable for the complex task of playing high level RTS game-play?

2 Real-time strategy games

2.1 The genre / history

Real-time strategy (RTS) games are a sub-genre within the genre of strategy computer games. The game that has arguably been the most influential to the genre is Dune II (Westwood,1992). There were a few games before that could be considered as real-time strategy, but Dune II defined the RTS mechanics we know today [43]. The term real-time means that players are allowed to make actions independent of the other player, in other words: they do not have to wait their turn. The real-time aspect makes every RTS game a race against the clock: who gets the most actions done in the same amount of time. Being inactive is detrimental for a player. Professional players execute over 200 actions per minute [42].

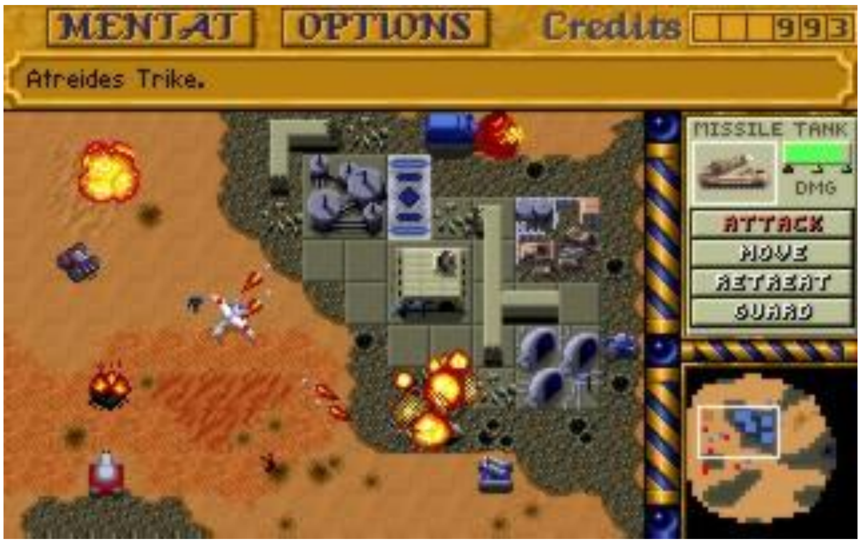

Figure 1: A screenshot of Dune II (Westwood,1992) [43]

2.2 Game-play (Starcraft)

Starcraft (Blizzard Entertainment,1998) is often chosen as number one strategy game of all time and can be seen as the prototype RTS game [43]. Even fourteen years after its release it is still played by millions of players online. In South Korea there is a professional league devoted to the game, where players earn six figure incomes and three television channels stream 24 hours of Starcraft a day [42]. One of the reasons for the popularity of the game is the strategic depth and balance. Throughout the years, the developers have continued updating and balancing the game by patches and this process has contributed to the high level competitive scene there is today.

2.2.1 World representation

The world of a RTS game is called a map and players usually view the map from an isometric perspective. The map is divided into tiles which can represent several types of terrain. Different maps are of different sizes. The screen of a player only shows a part of the total map, but players have access to a minimap at the bottom of the screen, where the whole map is visible. However, what is on the map is only partially visible due to the fog-of-war. Fog-of-war is an important aspect of RTS games. Fog-of-war means players have visibility only in a small area around their own buildings and units. Players therefore need to make decisions under uncertainty.

2.2.2 Resources

Gathering and managing resources is a key element of RTS games. Buildings and units cost resources and those resources need to be gathered with workers. At the start of a game a player usually starts with very limited resources and only a few units. Throughout the game more workers are generated in order to boost the players income. A player with an economic lead is most of the times ahead in the game. However, investing too much in economy leaves the player with a weaker military force. An aggressive opponent could sacrifice some of his economic growth in order to have a bigger army than the more economic oriented player. In this way the aggressive player has the military advantage and can exert some pressure on the economic player. Sometimes a player sacrifices all of his economic growth and performs a 'rush-attack' on the opponent, overwhelming the opponent with inexpensive units. If a player does not notice this rush intention of the opponent, he could lose the game very quickly. A key aspect of RTS is the balance between investing in economy and military.

2.2.3 Buildings

A player starts with one 'main' building and five workers. Throughout the game new buildings are constructed in order to generate advanced military units, research technologies or expand to new resource locations. The sequence and timing of buildings constructed is called a build-order. Build-orders often contain the sequence of training units and researching technology too. Strategies are defined by the sequence of the construction of buildings, units and technologies.

2.2.4 Units

Player make units with their buildings. Units cost resources and have a shadow cost. Examples of shadow costs are: the time it takes to produce a unit, the technology required for the unit or the amount of supply the units takes in the army. The size of an army is limited by the supply cap. Different unit types have different properties and abilities. Properties of units are for example health, attack damage and attack speed. The different unit types defined by the abilities and properties of a unit, behave in a rock-paper-scissor fashion. This means that there is no absolute best unit or army. Every army has a counter. A counter strategy is a strategy that is cost-effective on the strategy that the opponent is executing. Units of the scissor type will be very effective on units with the paper type, these units 'counter' the paper type ones. Because every strategy has a counter strategy, there is no nash-equilibrium. Players need to be highly dynamic and react to what the opponent is doing all time. Building a model of your opponent is therefore very important.

2.2.5 Tech-tree

A tech-tree is a model of the research path a player can take (Figure 2).The tree is a one-dimensional directed graph, where each node represents a type of building. The tree is traversed by constructing a new type of building. Buildings generate units and technology. The state of a player in the tech-tree reveals the different types of buildings a players ownes and therefore defines the possibilities a player has. This tech-tree is very important for strategy planning and opponent modeling.

2.2.6 Macro

Macro management (macro) is everything that involves your economy and building your base. You have to create additional workers for a long period of the game in order to increase your economy. Buildings have to be created in order to produce units and research important technologies. Resource management is an important aspect of macro, deciding where to spend the resources on: army, tech or economy. Inactivity or bad planning leads to banked up resources. Every resource that is not spend and could have been spend on something useful means either; a smaller army, later technology or being behind in resource income.

2.2.7 Micro

A very important aspect of a high level RTS player is micromanagement (micro). Micro involves all the commands you give to your individual units. Examples of micromanagement behavior in combat are: choosing to focus fire with your troops on one particular unit, or to retreat single units that are low in health, or to flank or surround an enemy. Terrain is very important for micromanagement, spatial reasoning is needed to position your army efficiently and benefit from the terrain.

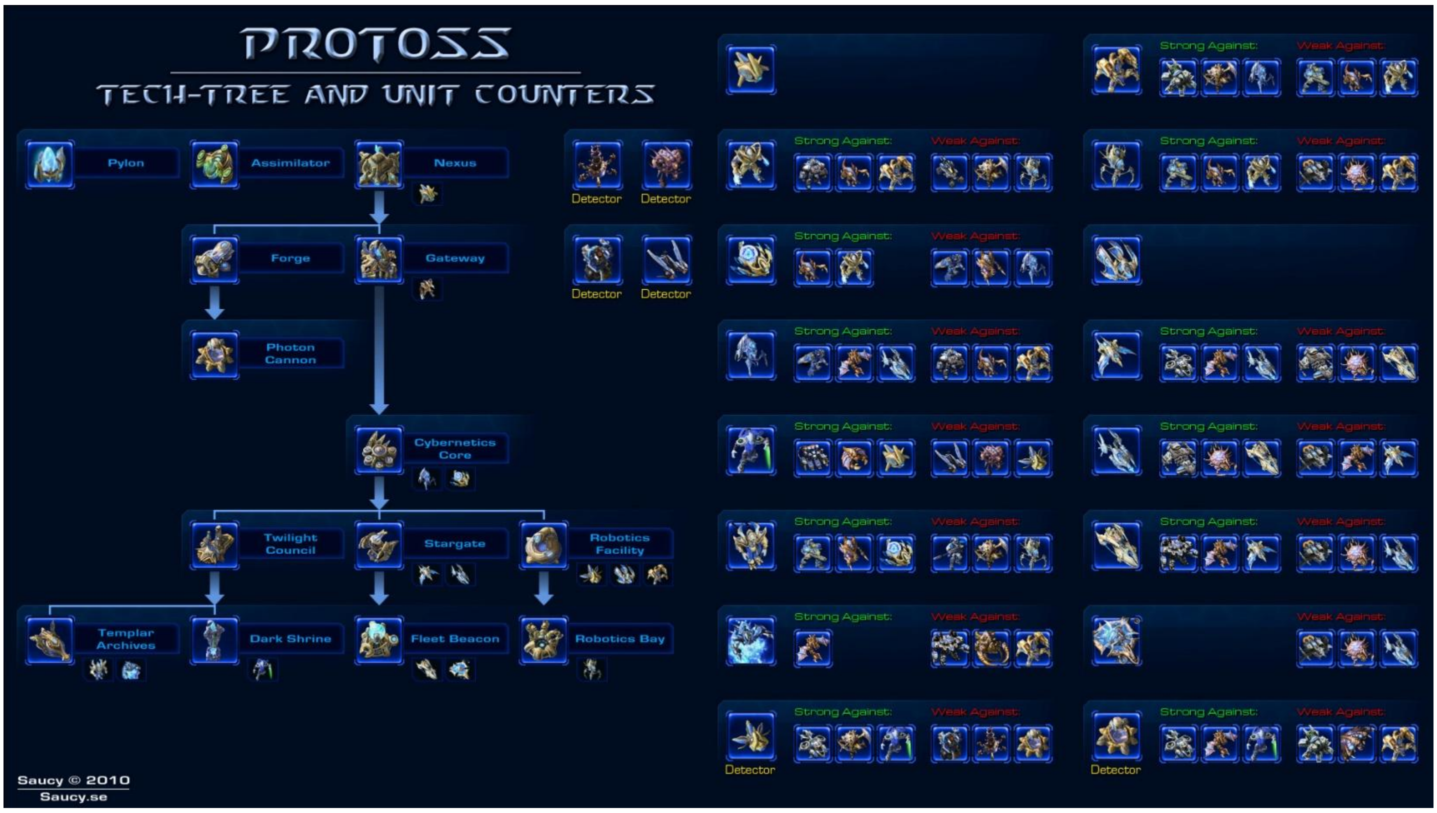

Figure 2: Tech-tree (left) and unit counters (right) for the Protoss faction of Starcraft 2 (Blizzard Entertainment, 1998) [42]

2.3 Low level AI and High level AI in RTS games

There is a difference between the game engine domain AI (low level) and the player domain AI (high level). The former one is everything that is needed to *run* the game, like path-finding algorithms for the units. The latter one is the intelligence that is needed to *play* the game. The RTS AI referred to in this thesis is the high level RTS AI, as if we replace the human player with a computer controlled player, a bot.

3 Complexity of RTS games

A RTS game playing agent must have the ability to deal with the complexity of RTS games. Dealing with the complexity involves being able to handle the large amount of information, making decisions in a very large action space and perform multiple tasks at once. The task of winning an RTS game involves highly reactive play, which means predicting the strategy of the opponent and react to it with a counter strategy. For an agent to be able to reason in such complex environments, abstractions of certain aspects of the game are needed. Human players make abstractions of the data of RTS games naturally [11,26].

3.1 Large action space

A problem for RTS AI, is that there are many different complex unit types in a RTS game. Ponsen *et al*. have analyzed the size of the action space in the RTS game Wargus [25,45]. They formulated a formula for the action space which results in a space of 15.000 moves for a simple early game. For comparison, chess has a decision complexity of 30 moves. In chess only one piece can be moved at a the time, while in RTS games all units can move simultaneously. With the action space of Wargus, it is impossible to systematically search the actual action space for good moves, because the space is simply too large to search under the real-time constraints [19].

Scientific research on RTS AI often makes abstractions of the action space in order to reduce the size of it. Aha *et al*. group several game actions together and call it a tactic [2]. Instead of searching for possible next-moves, their agent chooses the best next-tactic. By reducing the action space, planning becomes manageable for the agent. Another way of making abstractions to action space is to reason about army squads instead of individual units [23]. One thing to notice is that abstractions mean a loss of information. It could be discussed if essential RTS game-play is lost if micromanagement over individual units is neglected.

3.2 Goal Oriented Action Planning

Another way to conceptualize the action space is by using goal oriented action planning. Classical decision making techniques use state-action rules. In a given state the best set of action is chosen and performed. This technique is static and is unable to cope with highly dynamic environments [22]. Goal oriented action planning (GOAP) is a technique to improve the performance of game agents. The basic idea of GOAP is that in a given game state, the agent does not decide what action to perform, but what goal to pursue. In every game state, the agent first chooses one or more goals to pursue and then what actions to perform in order to reach these goals. By reasoning about goals the AI becomes less predictive. It is possible for the agent to achieve the same goal in different manners on different occasions.

Another way a goal driven autonomy strengthens the RTS game AI, is by integrating knowledge about the intentions of the opponent [38]. If actions are coupled to goals, the agent is able to build a model of the opponent based on the intentions of the opponent. Instead of figuring out the next move the opponent, the agent reasons about the goals the opponent is pursuing. The large number of possible next-moves an opponent can make, is too large to reason about. Conceptualization of actions to goals reduces the action space. Human players do not think and reason about actual game actions, but make conceptualizations and group actions together that have the same intention.

3.3 Complex state space

One of the aspects of the complexity of RTS games are the complex and dynamic game states. The actual game state would be defined by the position of all the units and buildings on the map, which has an average size of 512x512 tiles. If a player moves one unit, the actual game state would be changed. Players move units all the time. Interpreting the raw data of a RTS game, to determine the game state, is problematic [23]. One way to overcome this problem is to categorize all the possible game states in fewer meta-states of the game.

Ponsen *et al*. created a state lattice with meta-game states and reduced the total amount of game states of Wargus. Each meta-game state is defined by the sequence of the building types that have been constructed. Constructing a new type of building means that the game state has transitioned into another state. Their motivation for this type of categorization is the tech-tree of the game, which humans use to determine the position and possibilities of a player. Multiple instances of the same type of building are discriminated by this definition of meta-game state. Buildings that do not unlock new features, like walls or defensive towers, are also left out of the definition. With this abstraction the total amount of possible game states of Wargus is only 20 [23].

It is easy to see that by reducing the state-space to 20 a lot of information about the game situation is lost, like the distance to the opponent, the amount of units of both players and the positions of the armies. However, using the tech-tree as fundament for the game sate has become a standard in AI research [2,19,33].

3.4 Multiple levels of reasoning

The task of winning a RTS game involves multiple subtasks, like setting up an economy and fighting battles. Different independent subtasks need to be optimized simultaneously by the player. There is often more than one goal a player needs to pursue. The design of a RTS game playing agent has to be suited for multitasking. A classic method for implementing AI behaviors is a finite state machine (FSM). A FSM is a simple, effective and expressive method used by many AI developers [1]. However, the design does not allow reasoning about multiple goals at once or the execution of different independent subtasks simultaneously.

The design of the agent needs to have different independent working modules, each specified to a subtask of RTS game-play. Each subtask needs to be hierarchically structured from the high level strategic plans that entail all available military units to the low level individual unit commands. The advantage of a modular hierarchical design is that the search space for each subtask is reduced. Different independent managers work on a subtask with specific levels of abstraction, most suited for that domain. Another benefit is that each manager can use its own AI technique, most suited for the type of subtask. Research has shown that statistical AI techniques are more appropriate for micromanagement tasks and symbolic techniques are more suitable for strategic decisions [32].

Safadi *et al*. propose a simple hierarchical model to improve RTS game AI [26]. Their model is based on knowledge from human experts. Wintermute *et al*. use a hierarchical architecture called SOAR to play complete RTS games[40]. The architecture of SOAR is based on human perception. The research shows how to design a hierarchical modular agent, but the framework did not show good results in game-play. Weber used a reactive planner for his Starcraft playing agent. This agent is very similar to the SOAR architecture but differs in two ways. First, there is no claim that the model is similar to the human cognitive brain. Second, there is more coordination between units in the reactive planner than in the SOAR architecture[39].

3.5 Domain Knowledge

For strong RTS game play, a lot of knowledge about the game is required. There are different ways to acquire this domain knowledge. Common sense and personal game play experience are the most simple ones. Expert human knowledge in the form of strategy guides or online communities are another way to acquire domain knowledge. Research done with the game Wargus has studied the built-in AI scripts to acquire domain knowledge [2,23,25]. Wargus did not have a competitive, professional scene. There was not an online community to learn from. Knowledge about the game was scarce. More recent research is done with the Brood War Application Programming Interface (BWAPI), which is a free and open source C++ framework for creating AI modules for Starcraft [35,38,44]. With Starcraft as new testbed for AI research, a lot of domain knowledge becomes available. The professional scene and live online community provide endless information about the game in the form of strategies, rules of thumb, map analyzes and more. Tons of expert demonstrations are available in the form of replays, which are recorded game logs from human players.

One interesting approach to acquiring domain knowledge was performed by Ponsen and Spronck. They created an evolutionary algorithm which could automatically generate new tactics for the game Wargus. The idea behind the algorithm is to acquire domain knowledge which is not yet available by the designers of the game. The domain knowledge encoded by AI designers is likely to be sub-optimal. Game designers are unable to try every option in every situation, but an offline learning algorithm is able to.Test results show that the algorithm is capable of finding counter tactics to a given strategy. However the algorithm is only tested on a few very basic strategies [24].

4 Planning in RTS games

4.1 Classical Planning

The classical planning techniques, like a state-space search are impractical for the domain of RTS games. Classical planning is done in fully observable, deterministic, finite, static and discrete environments [1]. RTS environments are not in this category. The fog-of-war, real-time aspect and the huge amounts of raw data pose a problem for a classic planning approach. The game states are too complex and the action space is too large to handle with a systematic search. Another problem for a systematic search for the best move is that a single move is very difficult to evaluate. The evaluation of the performance of a single action is problematic because most of the single actions do not directly lead to winning the game, in most cases not even to winning a local fight.

4.2 Case-based Reasoning

4.2.1 Definition of Case-based Reasoning

Case-based reasoning (CBR) is a technique that uses old experiences to understand new problems [14].CBR techniques are able to handle domains where classical planning is computationally undoable. In RTS games it is very hard to define general relationships between problems and solutions. The CBR techniques deals with this problem by using specific knowledge learned from previous situations to solve a new problem. This method is similar to way humans interpreted a new situation, with the use of old experience. The psychological plausibility motivates the CBR approach [28].

Case-based reasoning uses a database of problem situations called cases. Each case describes a problem situation and a used solution, called the case-representation. New problem situations are solved by using solutions from similar problems in the past (Figure 3). If the solution from the past problem does not yet fit the current problem, adaptation methods are required. Adaptation methods transform the old solution to fit the new problem. After the adaptation, the solution is used to solve the new problem. The next step is to evaluate the perfomance of the (adapted) solution to the new problem. If the evaluation is positive enough, the solution is then again stored in the case-memory for later use. This last step of CBR makes it an online learning technique by continuously updating the memory-case.

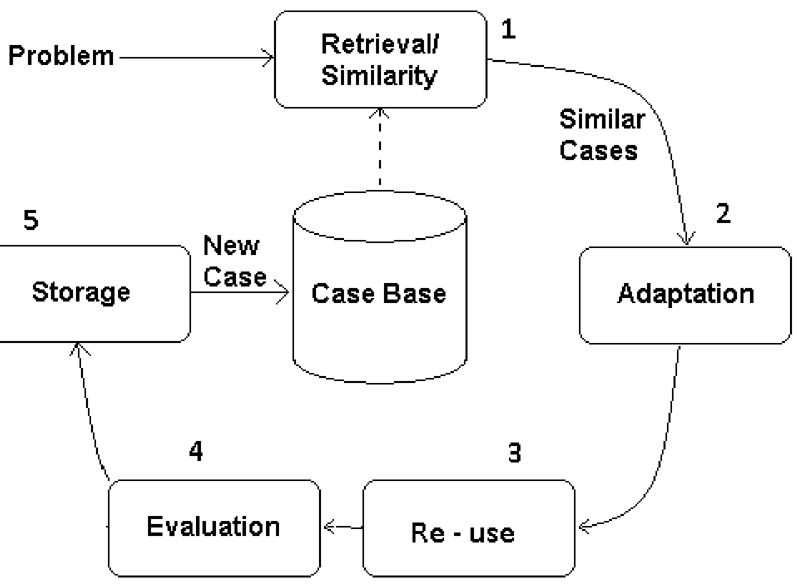

Figure 3: Schematic representation of a case-based reasoning cycle

4.2.2 CBR in RTS game AI
Different researchers use different designs for their CBR system. Some designs have left steps out of the system, like adaptation or evaluation. Adaptation methods are complex and a lot of knowledge is required to define efficient adaptation methods for RTS games [33]. Evaluation and storage requires a correct evaluation function of short-term game actions, which is difficult to define for RTS games. For a CBR technique to work, a lot of domain modeling is required. Cases are labeled with a game state (problem situation) and information about the next-moves. A major part of the efficiency of CBR depends on the way the cases are represented, indexed and searched

4.2.3 Basic Case representations
For a CBR technique to work in the domain of RTS games, abstractions of the game state have to be made. Abstractions mean loss of information. How much information is used to label a case, is called *the richness* of the case-representation. A problem situation in a CBR technique applied on RTS games is the game state. A very rich case representation is for example, a game state that is defined by the exact location and properties of every unit and building owned by a player. Such a representation would probably be inefficient, because it contains too much data and changes every time a unit moves, which is almost every instance of the game. Very rich case-representations allow more complex reasoning [19]. One downside of rich case-representations is the effort it takes to create a large database of rich cases. The search and comparison of similar cases is computationally heavier too. Another problem is, that for rich case-representations a lot of domain knowledge must be at hand [34].

Aha *et al*. were the first researchers who used a CBR technique for strategy planning in Wargus [2]. Their system is an online learning technique with a very simple case representation. The game state is based on the meta-game states defined by Ponsen and Spronck [23]. The solutions stored in the cases are a set of actions which they call a tactic. No adaptation of tactics take place, but an evaluation of the tactics is stored. The best suited case for a new problem is found by comparing similarity of the game state and the performance of the tactic stored in the case. When the most similar case, with the best performing tactic, is found, the tactic is executed. After the execution of the tactic, the system updates the performance of the tactic, or if the situation was new, a whole new case is stored (Figure 4).

Although the agent was able to defeat the built-in scripts of Wargus 80 percent of the time, some critique can be given. First of all, using only one algorithm for a full game playing agent is probably not the best design. RTS game play is a typical example of multi-level reasoning. To make their single algorithm work for a domain that needs such complex reasoning, huge abstractions have to be made. Also, several important features of RTS game play are ignored. Moreover, the agent does not build a model of the opponent, nor does it have any spatial or temporal reasoning. A smaller point of critique is the fact that the game state estimation of the agent cheats, it uses information that is not visible due to the fog-of-war. One of the major challenges of AI research in this area, is dealing with imperfect information.

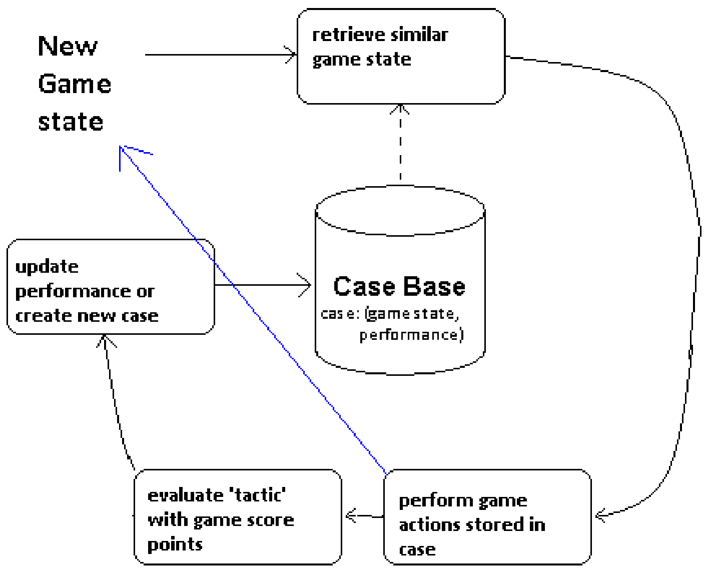
New Game state
retrieve similar game state
Case Base
case: (game state, performance)
update performance or create new case
evaluate 'tactic' with game score points
perform game actions stored in case

Figure 4: Schematic representation of the CBR technique applied on a RTS game. The blue arrow indicates that the state of the game has changed after the execution of the actions in the stored tactic. A new game state means a new problem situation and the process starts over.

4.2.4 Complex case representations

The first step of CBR is interpreting a new problem with the use of old experiences. The technique searches for the most similar case. In RTS games, the first step involves searching for game states that are similar to the current game state. With complex case representations it is more likely that there does not exist an exact match in the case memory. The more information used in the definition of the game state, the more likely it is that one variable is not the same. Generalization methods are required to generalize variables and compare the game states. Adaptation methods are required to transform the stored solution to fit the new game state.

Aamodt *et al*. used CBR on the application of micromanagement in battles of the strategy game Warcraft 3 [30,46].Their system is capable of fighting a battle efficiently by retreating low health units and focus fire enemy troops. The problem situation is determined by the army composition of the two players. These army compositions are complex and most of the time differ from the cases stored. Several generalization and adaptation methods are therefore used to match the situation and to adapt the rules that are in the case. The system of Aamodt *et al*. is able to constantly defeat the hardest built-in AI of Warcraft 3. However, the system uses several trigger conditions to activate certain important behaviors, like the retreat of low health units. This part of the system is basically a finite-state-machine. It is hard to evaluate if the CBR techniques has improved micromanagement of units in Warcraft 3, but the research shows that CBR is able to handle rich case representations with the use of generalization and adaptation methods.

Another research that used generalization and adaptation methods is done by Weber and Mateas, who used CBR on build-order planning in Wargus [33]. The CBR technique of Weber and Mateas uses a complex game state definition and therefore rich case representations. However, not all the information is used in every situation. Generalization- and adaptation methods are created with the use of knowledge about the game. It depends on the type of decision the agent has to make, how much information is used by the agent.

The agent uses huge abstractions of the game data where possible and reason about fine nuances where needed. The downside of this approach is that a lot of domain knowledge must be at hand to create the methods and it takes a lot of time to create a case memory of cases with a rich case representation.

4.2.5 Integrating domain knowledge

The first researchers on RTS game AI used game actions performed by the built-in scripts of Wargus as starting domain knowledge[2,23]. The machine played a game according to the rules defined in a script, the game log was recorded and individual cases were extracted. Every time a game state changed, a new case was made and all the actions performed until the next state change were stored in the case as a solution.

Ontañón *et al*. built a full game playing agent with the use of a CBR technique with very rich case representations [19]. A complicated goal oriented architecture allows their agent to reason about goals in Wargus. The case memory is filled by human demonstrations, expert players playing games of Wargus. Cases are labeled with a game state and actions performed in the game state. Additional knowledge is inserted in each case by manually annotating in every case which action contributed to achieving which goal. The agent not only knows what actions were performed, but why they were performed. Manually annotating game logs takes a lot of time and the knowledge about the goals must be at hand. This research used only a few demonstrations and the agent was only tested on 2 customized maps. The technique showed good results on the two customized test maps, but problems occurred when tested on a larger and more varying map set [20]. The case memory of this agent was relatively small due to the time consuming nature of manually annotating game logs.

Instead of manually annotating the demonstrations, researchers have tried to do it automatically. Data mining is a technique for discovering patterns in large databases to guide future decisions. This approach lets the machine process very large data sets with minimum input from a human user. Research on the game Starcraft used data mining on replay databases [35]. The approach seems fruitful, if the machine can automatically learn domain knowledge, it can browse through all the tons of expert demonstrations available in the form of professional Starcraft replays. This should result in a relatively large case memory. However there is a difference between the game demonstrations used in earlier research on Wargus and the professional Starcraft replays. The demonstrations used in Wargus were made solely for AI training purposes [2,19]. The demonstrations consisted of prototype strategies where the intention of every move is clear. Starcraft replays are not made for AI training purposes and are stored in a proprietary binary format specified by Blizzard, so researchers have no control over the data and the replays have to be converted to a useable format. The replays are noisy and contain a lot of spamming actions. Spamming actions are extra unnecessary actions of a player. Professional Starcraft players often execute more than 200 actions per minute, which results in more than 3000 actions per replay. Therefore annotating goals and other information is very hard [35].

Weber and Ontañón have done research on automatically annotating game traces [36]. First they automated the process of converting expert replays into cases. A goal ontology is defined to specify the intentions of a player during a replay. The idea is to group sets of actions to goals and build cases based on these goals. A case consists of a game state, coupled with a goal. While playing the game, the CBR part of the agent receives a goal as input and retrieves the most suitable case to achieve this goal. The agent did not win against the built-in AI of Starcraft but showed a promising start to automatically annotate replays for a case-based planner. Improvements to their system can be made with a richer case representations, allow the system to pursue more than one goal at once and better methods for adaptation.

4.3 Dynamic Scripting

4.3.1 Definition of Dynamic Scripting

Dynamic scripting is an online machine learning technique that is based on reinforcement learning. Dynamic scripting was originally developed to create scripts for computer controlled opponents in computer role-playing games (RPG). When a human player encounters a battle with a computer controlled character, a script is generated on the spot. This script defines the combat behavior of the computer controlled character and consists of a set of rules extracted from the rule database. The probability that a rule is chosen for a script is influenced by a weight attached to the rule. When the battle between the human player and the computer controlled player is over, rules are evaluated and the rule database is updated. Rules that lead to successful outcomes increase in weight and rules that lead to failure decrease in weight. After the adaptation of the rules, a weight redistribution function is used so that the sum of all weights in the database is constant. One of the advantages of the dynamic scripting technique is the use of domain knowledge. Rule databases are made with the use of knowledge about the game. In the original development of dynamic scripting, each type of computer controlled character in the RPG had its own rule database. The scripts made for warrior characters were made from the rules specified in the warrior rule base and for the wizard characters from the wizard rule base.

4.3.2 Dynamic Scripting in RTS game AI

Dynamic scripting is implemented in the RTS game Wargus by Ponsen [23]. The difference between dynamic scripting applied to a RPG game and a RTS game is that in the RTS variant there exists a rule base for each game state instead of for each character. A script is created when the AI controlled player enters a new game state. At the start of the game, the agent is in state one and selects several rules from the database of state one. When a executed rule converts the game state into another state, a new script is generated out of the rule base belonging to the new game state. After each state transition, the performance of the script is evaluated, where good evaluation of rules means a weight increase, which means a higher chance to be chosen next time the agent is in that game state.

Ponsen made abstractions to the game state and action space [23]. They defined twenty possible meta-game states. With conceptualizations of the action space the space is reduced to fifty rules per rule base. With this abstraction, micromanagement of units is never performed. The idea of this technique is that the agent is able to adapt to a new situation by exploring all the different rules. If a generated tactic does not work in a given situation, chances are unlikely that the tactic is chosen again due to the weight decrease. Each time the unsolved situation is encountered, different combinations and sequences of rules can be tried until a working tactic is found. The agent should be dynamic by always stochastically choosing rules and not allowing a zero percent chance of choosing a rule.

The agent of Ponsen is only tested against the four built-in scripts of Wargus, where the technique did not perform well [23,25]. Two of the four strategies, soldier rush and knight rush, won about 99 percent of the games versus the dynamic scripting player. After improvement of the rule databases the dynamic scripting player won eventually 29 percent of the games versus the soldier rush and 13 percent of the games versus the knight rush.

One point of critique is the fact that Ponsen and Spronck had one counter strategy to the knight rush which was developed by their evolutionary algorithm and added the whole strategy as a single rule to a rule base. In other words, they scripted a full counter strategy to the knight rush. This counter strategy was a knight rush itself, slightly optimized with a fast armor upgrade [23]. Dynamic scripting is developed to be unpredictable and able to adapt to new situations. Hard coding a strong rush tactic does not contribute to these principles. The results against the other 2 built-in scripts increased after improving the rule bases. The problem is, that adding a full hard coded strategy to a system that is developed for being adaptive and unpredictable, seems counter intuitive.

A second point of critique is that the agent does not build a model of the opponent. The technique is developed as adaptive game AI and RTS game-play relies on reactive, adaptive play. However, DS adapts to its opponent in a delayed fashion. Weights are updated after the evaluation of a tactic. Against static opponents, the DS technique will eventually learn how to counter the strategy of the opponent. Against dynamic opponents, it is very doubtable if DS will be able to adapt. The DS technique will adapt after every game to the evaluation of the last played game. If the opponent changes its strategy every time, the DS technique will always learn the wrong tactics. Eventually, after many games, the technique will probably develop an all-round strategy that generally counters every strategy a little. First of all, this would then be a static all around strategy, second of all it is very likely this strategy will be weak. One of the challenges of RTS game AI research is to find a way to deal with imperfect information. Imperfect information is only a problem for opponent modeling, when this aspect of RTS game play is ignored, research in RTS-game AI looses one of its challenging aspects.

4.3.3 Evaluation

One of the problems of dynamic scripting applied to RTS games is the evaluation function. Ponsen *et al*. explain that dynamic scripting may be successful applied to game AI if the game AI can be scripted, domain knowledge on the characteristics of a successful script can be collected and an evaluation function can be designed to assess the success of the functions collected [25]. The problem with RTS games is the evaluation function of a successful script. It is very hard to evaluate short-term tactics, let alone single game actions. Ponsen *et al*. used the game score as an evaluation function for their technique. The game score consists of military points gained by killing enemy units and production points by creating buildings and units. The score neglects spatial and temporal factors. For example, it could be efficient for a player to sacrifice several units in order to gain control over the map. The points gained for killing and producing units are only based on the cost of the unit in the game. Shadow costs like production time or the technology required, are not taken into consideration.

A second problem with the use of the military point system is that the points gained for killing units are based on the value of the units. The value of a unit is determined by its utility. Some units rely heavily on micromanagement. These units, like archers or catapults, can do a lot of damage when controlled correctly, but die instantly if sent blindly into battle. In the experiments of Ponsen *et al*., both AI controlled players do not micromanage their units, which makes those units inefficient. Nevertheless, the points rewarded for killing those units are still based on the efficiency of the units with proper control. The rewards are relatively too high and result in a distorted game score. Another thing with the wrong use of units, is that the in-game unit costs are also based on the usefulness of the units, the effectiveness of a unit in a game. Without micro, the AI will pay relatively too much for units that they do not use cost effectively. Scientific research that aims to create strong RTS AI must take this into account and prevent the AI for spending valuable resources on worthless units.

4.3.4 Goal-directed hierarchical approach

One way to improve the dynamic scripting technique is to design a goal-directed hierarchical approach for dynamic scripting [8]. The idea of this approach is to decouple the rules from the game state and couple rules to goals and then the goals to game state (Figure 5). There are weights between a game state and a set of goals and between each goal and a set of rules. In each game state a goal is selected out of the goal database belonging to that game state. When a goal is selected a script is generated from the rule base belonging to the goal and game state. In this design there are separate learning spaces for goals and rules. It should enhance the learning speed by applying additional domain knowledge. The design is not implemented in a RTS game playing agent and the actual usefulness of the goal-directed hierarchical design for dynamic scripting on RTS games remain to be examined.

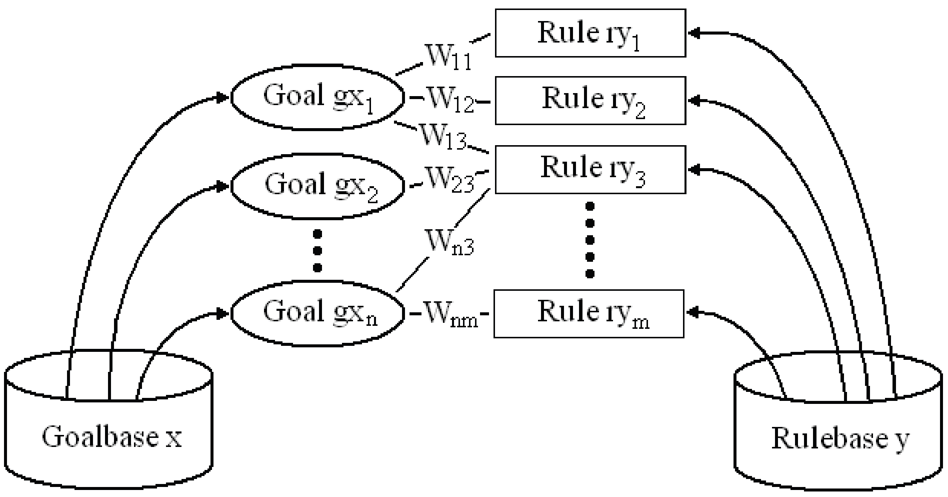

Figure 5: A schematic representation of goal-directed dynamic scripting (Dahlbom,2006)

5 Conclusions

The complexity of RTS games makes it difficult to create a RTS game AI that is able to compete with high level human game-play. The commercial game industry deals with the complexity by creating complex scripts for the computer controlled players. These scripts are static and often contain weaknesses, which players exploit very easily. Scientific research on RTS game AI has experimented with new techniques to improve RTS game AI. The techniques proposed have to be able to deal with the large amount of data in a highly dynamic, partial observable environment. Agents need to reason in a very large action space and perform multiple tasks simultaneously. The question is, to what extend are the scientific AI techniques suitable for this complex task?

5.1 Scientific answer to the large amount of data

Scientific techniques rely on abstractions of the game data. With the use of correct abstractions, scientific techniques are suitable to handle the large amount of data. One of the fundamental structures that defines the game state is the tech-tree [24]. However, defining meta-game states solely on the tech-tree position of a player is too simple. It only defines what the player is doing, not when or where. Without temporal and spatial information, an agent is not suitable for strong RTS game-play, so this information must be taken into the definition of the game state. The current position of a player can be defined as the path that a player has traversed in the tech-tree together with the elapsed game time at which each transition to a new node first occurred. Additional information about the size of the map and the distance to the enemy base must be added to allow spatial reasoning.

Besides a complex game state, there are too many game actions to take into consideration when planning the next action. Therefore scientists have made conceptualizations of the action space. A good way to reduce the action space is by grouping actions together that lead to the same goal [22,38]. This way, the agent reasons about goals and intentions instead of actual game actions. Another advantage of symbolic goal representations is the ability to communicate between different managers. Playing a RTS game involves performing multiple subtasks simultaneously. An agent design with multiple independent managers is able to handle multiple subtasks [26]. Due to the fact that the managers use different algorithms internally, a communication language between the managers is required, namely symbolic goal representations [39].

A multiple hierarchical agent design is suitable for the task of playing RTS games. It models the multiple levels of reasoning that are involved in RTS game-play. Another reason to choose such a design is that specific domain knowledge can be integrated in each internal manager. Each manager can run on a different algorithm, one that works best in the sub-domain of the agent [32].

5.2 Scientific answer to be adaptive

In RTS games, it is very likely that an agent encounters a problem situation it has never encountered before. Scientific research has tried to create an agent that is able to efficiently deal with these new situations.

Case-based reasoning (CBR) has been proposed as a technique that is able to effectively handle new situations in RTS games [2,19,33]. With specific knowledge from a solution to a similar problem, CBR adapts to new situations. However, strategic decisions in RTS games require a rich case representation, with a lot of information to base decisions on. The problem with rich case representations is the fact that building a large case memory is very time consuming and the knowledge about the game must be at hand [19,20,33]. CBR is able to learn this domain knowledge by storing new solved problems in the case memory. CBR design with this last step are not suited for RTS games because it requires an evaluation function for short-term game actions [2,14]. The evaluation function currently used for short-term tactics is based on the game score [2,23]. The game score only discriminates between produced units and killed units. Not all tactics have the goal to produce or kill units, the goal may also be to research a lot of technological upgrades very fast or to effectively position an army somewhere on the map [42]. The advantages of such tactics are not rewarded by the game score.

Without a correct evaluation function, machine learning is problematic [1]. However, a CBR technique without learning has been successfully applied to build-order selection and opponent modeling [33,34]. Build-order planning and opponent modeling do not require the same information as strategic reasoning. Therefore a more simple case representation can be used and an automatic method is able to build the case memory from expert replays [35,36]. The problem with a non-learning variant of CBR is that it does not create new cases online and all the knowledge must be in the case memory on forehand. This technique is only suitable for RTS games where a lot of domain knowledge is known.

Another technique proposed to effectively handle new situations is Dynamic Scripting (DS) [8,23]. DS is a form of reinforcement learning and the technique has problems with the evaluation of short-term game plans. A correct evaluation function is fundamental for DS to be efficient. DS has only been tested to play full games of Wargus. The results of this research showed that DS is able to learn to choose the correct rules in a given situation. However, it has not shown results that indicate a future in RTS game AI. The technique is not suited for strong RTS game-play because of the huge abstractions that have to be made to the game information. The meta-game states are oversimplified and therefore the agent is not able to perform spatial or temporal reasoning. The technique is proposed as a solution for adaptive game AI. However, DS does not build a model of the opponent. Opponent modeling and reactive game-play are the essence of RTS games. DS builds a model of its opponent in a delayed fashion and this way of opponent modeling will most likely not work against highly dynamic opponents. The biggest eyesore of the research done with DS, is the implementation of a full game script called the ‘knight-rush’ [23]. This addition of a static, full game script, goes against all what DS stands for, creating dynamic scripts on the spot.

5.3 Scientific answer to be dynamic

Goal oriented action planning has been proposed as structure so that the agent does not always perform predictable actions. The agent first chooses a goal it wants to achieve, then selects the best actions to get there. There are often multiple ways to achieve a goal and the agent can vary in the actions he chooses to reach the same goal at different moments [22,38]. This technique is suitable for RTS game AI. Case-based reasoning is dynamic with the use of adaptation methods [30,33]. The downside of adaptation methods is that they require a lot of domain knowledge. The method of choosing rules to generate scripts in DS ensures that there is always a possibility that a script differs from time to time. Even if a script has been proven to be the best solution in a certain situation, there is a possibility that the agent does not choose this script and diverges from the best known tactic. The effectiveness of this mechanism is arguable. Being unpredictable for your opponent is an advantage in RTS games. However, performing an optimal, but predictable strategy can win you games too.

One thing to notice, is that due to the fog-of-war, not every part of the map is visible. An enemy cannot predict what you are doing, unless he performs some form of reconnaissance. If a player denies the scouting of the opponent, it is very hard to predict his next move. This shows that AI techniques do not have to switch between actions all the time in order to be unpredictable. Human players often perform optimal predictable strategies, but do everything they can to deny the opponent to get the information [42].

6 Future research

There are several directions for future research. First, a more complex evaluation function must be identified. Research can be done to determine the actual unit values in AI games. Temporal difference learning methods have been used in scientific research to determine the values of pieces in chess. Another research direction that involves the game score is to integrate the shadow costs of units. Units are relatively more expensive if they take a long time to produce, or a lot of technology is required to produce the units. These properties should be represented in the score gained for producing or killing such a unit. A third improvement of the game score can be made by integrating a point system for tactics that gain technological- or spatial advantages for a player. With a more precise and correct evaluation of tactics, or even game actions, more research can be done on machine learning algorithms in RTS game AI.

Current research on RTS game AI has problems with testing the performance of the agent. Playing full RTS games is very time consuming. If the only evaluation of the agent is the outcome of the game, it is hard to determine if the agent has a strong opening, mid- or end game. It is also hard to determine the performance of the individual subtasks performed by the agent. Multi agent designs often want to research if the implementation of another technique into the agent will improve the game-play of the agent. It is very hard to measure if the performance of an agent is improved. New, better methods for evaluating the performance of an agent should be examined. Chess- and checker bots from scientific AI research are often tested in a competition with an ELO rating. Such a competition exists online for the game Starcaft. With an ELO rating, better comparisons of different agents of scientific research can be made. ELO ratings can accurately measure improvement in the performance of players. So, future research should test the strength of their agents in these ELO competitions.

Another direction for future research is to identify more methods to automatically integrate domain knowledge. RTS AI research is very time consuming and research on data mining methods may be able to find a solution to this problem. With the use of algorithms that are able to learn from expert demonstrations, a lot of domain knowledge can automatically be integrated. An agent with a lot of knowledge about the game and a correct evaluation function that allows machine learning, could potentially be a strong RTS player. Future research must demonstrate if an agent, armed with these properties will outperform humans in this challenging and popular domain.

7 References

7.1 Books

[1] S.J. Russell, P. Norvig; “Artificial intelligence: a modern approach”; Prentice hall; 2009.

7.2 Articles

[2] D.W. Aha, M.Molineaux, M. Ponsen; “Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game”; *Lecture notes in computer science*; 2005.

[3] M. Brockington, M. Darrah; “‘How not to Implement a Basic Scripting Language”; *AI Game Programming Wisdom*; 2002

[4] M. Buro; “Real-time strategy games: A new AI research challenge”; *International Joint Conference on Artificial Intelligence*; 2003

[5] M. Buro; “ORTS: A hack-free RTS game environment”; *Lecture notes in computer science*; 2003

[6] M. Buro; “Call for AI research in RTS games”; *AAAI;* 2004

[7] D.C. Cheng, R. Thawonmas; “Case-based plan recognition for real-time strategy games”; *Proceedings of the Fifth Game-On International Conference*; 2004

[8] Dahlbom, Niklasson; “Goal-directed hierarchical dynamic scripting for RTS games”; *AAAI*; 2006

[9] C. Fairclough, M. Fagan, B.M. Namee, P. Cunningham; “Research directions for AI in computer games”; *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science*; 2001

[10] K. Forbus, J. Laird; “AI and the Entertainment Industry”; *IEEE Intelligent Systems*; 2002

[11] J. Graham, L. Zheng, C. Gonzalez; “A Cognitive Approach to Game Usability and Design: Mental Model development in Novice Real-Time Strategy Gamers”; *Cyber Psychology & behavior*; 2006

[12] T. Kent; “Multi-tiered AI layers and terrain analysis for RTS games”; *AI Game Programming Wisdom;* 2004

[13] J.E. Laird, M. Lent; “Human-level AI’s Killer Application”; *AI Magazine*; 2002

[14] J. Kolodner; “An introduction to case-based reasoning”; *Artificial Intelligence Review;* 1992

[15] M. Molineaux, D.W. Aha, M. Ponsen; “Defeating novel opponents in a real-time strategy game”; *IJCAI 2005Workshop on Reasoning, Representation, and Learning in Computer Games*; 2005.

[16] M. Molineaux, D.W. Aha; P. Moore; “Learning Continuous Action Models in a Real-Time Strategy Environment”; *Proceedings of the Twenty-First Florida Artificial Intelligence Research Conference*; 2008

[17] A. Nareyek; “Intelligent Agents for Computer Games”; *Lecture Notes in Computer Science*; 2002

[18] A. Nareyek; “AI in Computer Games”; *ACM Queue;* 2004

[19] S. Ontañón, K.Mishra, N. Sugandh, A. Ram; “Case-Based Planning and Execution for Real-Time Strategy Games”; *Lecture notes in computer science;* 2007

[20] S. Ontañón, K. Bonnette, P. Mahindrakar, M.A. Gómez-Martín, K. Long, J. Radhakrishnan, R. Shah, A. Ram; “Learning from human demonstrations for real-time case-based planning”; *IJCAI STRUCK;* 2009

[21] S. Ontanón, K.Mishra, N. Sugandh, A. Ram; “On-Line Case-Based Planning”; *Computational Intelligence*; 2010

[22] J. Orkin; “Applying Goal-Oriented Action Planning to Games”; *AI Game ProgrammingWisdom 2;* 2003.

[23] M. Ponsen; “Improving Adaptive Game AI with Evolutionary Learning”; *Master’s thesis Delft University, Netherlands;* 2004

[24] M. Ponsen, H. Muñoz-Avila, P. Spronck, D.W. Aha; “Automatically Generating Game Tactics through Evolutionary Learning”; *AI Magazine*; 2006

[25] M. Ponsen, P. Spronck, H.Muñoz-Avila, D.W. Aha; “Knowledge acquisition for adaptive game AI”; *AI Magazine*; 2006

[26] R. Safadi, R. Fonteneau, D. Ernst; “Artificial Intelligence design for real-time strategy games”; *NIPS Workshop on Decision Making with Multiple Imperfect Decisoin Makers;* 2011

[27] J. Schaeffer; “A Gamut of Games”; *AI Magazine*; 2001

[28] R. Schank; “Dynamic memory; a theory of reminding and learning in computers and people”; *Camebridge University Press*; 1982

[29] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, E. Postma; “Adaptive Game AI with Dynamic Scripting”; *Machine Learning;* 2006

[30] T. Szczepanski, A. Aamodt; “Case-based reasoning for improved micromanagement in Real-time strategy games”; *Proceedings of the ICCBR 2009 Workshop on CBR for Computer Games*; 2009.

[31] P. Tozour; “The Perils of AI Scripting”; *AI Game Programming Wisdom*; 2002

[32] B. Weber; “Integrating Expert Knowledge”; *AAAI*; 2010

[33] B. Weber, M. Mateas; “Case-Based Reasoning for Build Order in Real-Time Strategy Games”; *Proceedings of the Artificial Intelligence for Interactive Digital Entertainment Conference;* 2009

[34] B. Weber, M.Mateas; “Conceptual Neighborhoods for Retrieval in Case-Based Reasoning” *Proceedings of the International Conference on Case-Based Reasoning*; 2009

[35] B. Weber, M.Mateas; “A Data Mining Approach to Strategy Prediction”; *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games*; 2009

[36] B.Weber, S. Ontañón; “Using Automated Replay Annotation for Case-Based Planning in Games” *To appear in Proceedings of the ICCBR Workshop on Computer Games*; 2010.

[37] B. Weber, M. Mateas, A. Jhala; “Case-Based Goal Formulation” *To appear in Proceedings of the AAI Workshop on Goal-Driven Autonomy;* 2010

[38] B.Weber, M.Mateas, and A. Jhala; “Applying Goal-Driven Autonomy to StarCraft”; *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference;* 2010

[39] B.Weber, P.Mawhorter,M.Mateas, A. Jhala; “Reactive Planning Idioms for Multi-Scale Game AI”; *Proceedings of the IEEE Conference on Computational Intelligence and Games;* 2010

[40] S. Wintermute, J. XU, J. E. Laird; “SORTS: A Human-Level Approach to Real-Time Strategy AI”; *AAAI;* 2007

7.3 Websites

[41] J. Orkin; “Applying Goal-Oriented Action Planning to Games”; *www.jorkin.com*

[42] www.teamliquid.net

[43] www.gamespot.com

[44] http://code.google.com/p/bwapi/

[45] http://wargus.sourceforge.net