An Artificial Intelligence system to help the player of Real-Time Strategy games

Renato L. de Freitas Cunha Luiz Chaimowicz
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais

Abstract

Real Time Strategy (RTS) games pose a series of challenges to players and AI Agents due to its dynamical, distributed and multi-objective fashion. In this paper, we propose and develop an Artificial Intelligence (AI) system that helps the player during the game, giving him tactical and strategical tips about the best actions to be taken according to the current game state with the objective of improving the player's performance. We describe the main features of the system, its implementation and perform experiments using a real game to evaluate its effectiveness.

Keywords:: Real-time Strategy, Artificial Intelligence

Author's Contact:

renato@renatocunha.com
chaimo@dcc.ufmg.br

1 Introduction

Real Time Strategy (RTS) games have become one of the most successful genres in the game industry. In these games, players have to manage a limited set of resources to achieve a particular goal. Normally, RTS games are played at an extremely fast pace and players have to deal simultaneously with several different objectives such as collect resources, construct bases, improve technology and battle against enemy armies. The real time, distributed, multi-objective characteristics of RTS Games make its gameplay very challenging, especially for novice players.

In this paper, we propose and develop an Artificial Intelligence (AI) system that helps the player during the game, giving him tactical and strategical tips about the best actions to be taken according to the current game state. The system evaluates the game state following pre-specified metrics, elaborates hypotheses on how to improve the player's performance and communicates this information to him formatted as a set of strategy tips. The information used by the system is the same available to the player, *i.e.*, it does not use any internal or privileged information when analyzing the game state. In other words, no cheating is allowed! The system can be considered like an experienced player that plays alongside the user. The main objective is to improve player performance during the game, helping him in dealing with the fast paced dynamics of RTS games.

The system was implemented in Stratagus, one of the most used open-source engines for RTS games. We developed an expert system using decision trees that were built based on knowledge available in strategy guides developed for RTS games. The engine was instrumented to infer the game state and give strategic and tactical hints to the player based on the decision tree. To evaluate the system, we performed a series of experiments with players using Wargus, a clone of Warcraft II implemented using Stratagus.

The remainder of this paper is organized as follows: the next section presents some related works in the field. Section 3 discusses the main features of RTS gameplay and the competencies that a player must have to master this kind of game. In section 4, we present the overall design of our system, discussing the main game strategies included in the expert system. Section 5 describes the main reasons for choosing Stratagus over ORTS, another commonly used RTS open-source engine. The instrumentation performed on the engine to implement the system is discussed in section 6. Finally, section 7 presents the experiments performed to evaluate the system while section 8 presents the conclusions and directions for future work.

2 Related work

RTS Games are very good testbeds for Artificial Intelligence (AI) algorithms. As discussed in [Buro 2004], the characteristics of RTS Games require the development of algorithms that deal with temporal and spacial reasoning, uncertainty planning, learning, and opponent modeling. Moreover, the algorithms have to cope with the dynamics and time constraints of the RTS games. Thus, in the last few years, several works have explored different aspects of AI in RTS Games. For example, the use of traditional planning approaches such as PDDL, SHOP and HTN [Alcázar et al. 2008; Lee-Urban et al. 2007; Hoang et al. 2005] for strategic planning, the implementation of evolutionary techniques for generating tactics [Ponsen et al. 2006], the use of multi-agent potential fields for path planning and unit control [Hagelbäck and Johansson 2008] and the use of different learning approaches for opponent modeling and strategy prediction [Baumgarten et al. 2009; Weber and Mateas 2009].

In spite of the large number of works exploring AI algorithms in RTS games (a more complete overview can be found in [de Freitas Cunha 2010]), the great majority focuses on the development of intelligent agents to play autonomously *against* the human player. In this paper we go in the opposite direction: we develop an AI system that plays alongside the player, analyzing the game state and displaying strategic tips that helps improving his/her performance. Very few works explore this type of approach. Two examples in commercial games are the tutorial system present in games such as Warcraft III, that helps the novice user to learn the basics of the game mechanics (not strategy or tactics) during a tutorial campaign and the board of advisors in the SimCity series, that gives some generic advice of the best actions to keep the city into a positive growth. Thus, we believe that the development of an AI advisory system for RTS games, that takes into account some of the complex characteristics of its gameplay, may be an important contribution to this field.

3 RTS Gameplay

To develop an advisory system for an RTS game, it it is necessary to understand the various aspects involved in its general gameplay. In this section we discuss the main competencies that are required for a good RTS game player. Despite the large number of different RTS games available, most of these competencies are common among them.

In RTS games the players engage in military combat to guarantee the supremacy of a 2D map. Because of that, there is the need to build an army with varying sizes whose utmost objective is to destroy the enemy forces. To maintain their armies, the players must engage in other activities, like resource extraction and management, and technological research. We believe that mastering all the activities involved in RTS gameplay is essential for any agent to become an expert in the game: one not only need to elaborate strategies to win, but also must pay attention to the various aspects of the game, like managing his own resources and replanning accordingly as new knowledge about the world is gathered.

Figure 1 shows, in the format of a mind map, a division of the competencies a player / AI agent needs to be good at RTS games. The competencies are directly connected to the central concept and the interaction between them are shown as gray lines in the background. Even though the strategic competence is directly related to all the others, this relationship was suppressed in the figure for simplification purposes. This competency classification is based on the design of the bot found in [McCoy and Mateas 2008], and will be briefly presented in the following paragraphs.

It is important to note that different players are better at different

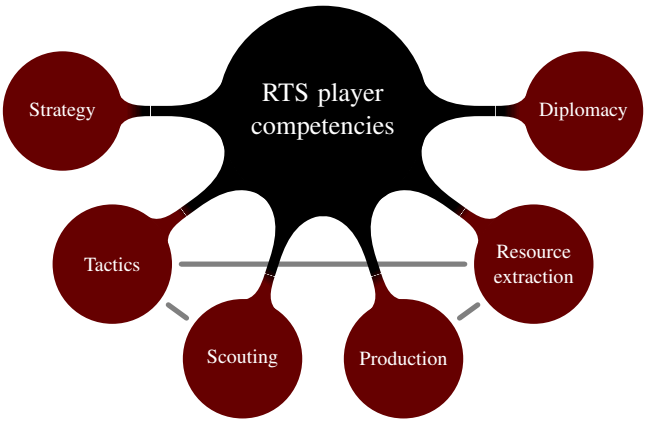

Figure 1: *Mind map showing the various competencies one is expected to find in a good RTS player. The straight lines connecting concepts indicate there is some kind of relationship between them.*

aspects of the game and, usually, being good at some key aspects, like combat tactics and resource management can give a player some advantage, even with him being not so good at other aspects.

3.1 Strategy

Strategy can be defined as the highest level of decisions that might be taken for one to achieve an objective. In the game, its role is to define what the player will do to achieve his objective. Strategies can vary during the game. The reason for that is that players will gradually gather information about their environments, and might need to adapt their strategies based on that information.

3.2 Tactics

Tactics handle the layout and maneuvers of troops during combat or in its imminence. This concept also involves the analysis of the surrounding environment to decide how to achieve a player's objectives. Unit micromanagement, which is a way to "fine tune" the behavior of individual units to perform optimally is also part of this competence.

3.3 Resource extraction

Resource extraction management is essential in an RTS game, because most resources are not renewable and are usually scattered along the map, meaning any player willing to extract them will be able to do so. Resources are needed to create units and buildings and to research technological upgrades. Because of their importance in a game, a player is required to balance their resource extraction with their investment in the army.

3.4 Production

Production management in RTS games is responsible for unit training, building construction, and technological research. Through the management of the production chain, it is possible to fulfill some strategic requirements. Knowing the requirements between the diverse technologies in a game, a player can sort the build order of his units to achieve a strategic objective faster. To exemplify, consider the technological graph shown in figure 2. In the figure, knowing that "Archers" are important to his strategy, a player can optimize the construction order of his units to achieve "Middle Ages" earlier in a game.

3.5 Scouting

Scouting is essential in RTS games because resources are non-renewable, thus it is necessary to keep searching for new sources of resources to maintain an army's productive chain. Additionally, exploiting more resources simultaneously helps in the development of the army by providing resources at an increased rate. It is also important to know where the enemy bases are located to plan an effective attack and to help inferring where the enemy attacks will come from.

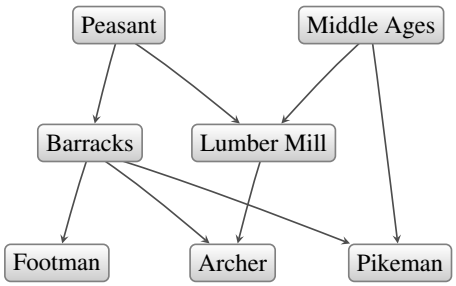

Figure 2: *Minimal technological graph that represents the dependencies between technologies of a synthetic RTS game.*

3.6 Diplomacy

Even though complex diplomacy is not a feature of many RTS games, it is present at online matches, represented by the creation and destruction of military alliances, resource exchange, and unofficial rules defined by players at the moment of the game setup. As the objective of online matches is usually to destroy all enemy armies, knowing when to propose and to break an alliance is an important aspect of gameplay that players should have.

4 Overall design

Knowing what characteristics an RTS player must master, we can start describing the design of the advisory system. The main idea is to use the knowledge found in strategy guides developed specifically for RTS games to develop an expert system able to help the player to learn the game faster. The steps taken to develop our system were:

1. Prospect the most used RTS game engines and choose one to implement the system;
2. Specify the tips to be used by the system based on the strategy guides;
3. Instrument the engine's code to work with the tip system;
4. Implement a subsystem to collect data about the state of the game;
5. Build and implement a decision tree encoding the tips;
6. Do tests to evaluate the system.

The discussion about the selection of the engine will be postponed until the next section. In this section we will describe the other aspects of the design.

For reasons that will be made clear in the next section, we implemented our system using Wargus, a Warcraft II clone that runs in the Stratagus engine. So, to implement the tip system, we decided to study Warcraft II's strategy guide [Blizzard 2010]. Being Wargus a Warcraft II clone, the rules and strategies developed for that game can be applied to Wargus.

In particular, some items were used as source of inspiration to create our knowledge base:

Workers are the key for development: the basic working units are responsible for assembling the whole game production chain. Therefore, they are the most important units in a game and are the key to victory.

The more workers, the better: unless the players are in a map with few gold sources, the more workers, the better, because that will increase the resource income ratio to maintain the production steady.

Train workers continuously: specially in the beginning of the game, it is important for the player to not wait for resources

Table 1: *Comparison of ORTS's and Stratagus' features.*

	ORTS	Stratagus
Successful implementation of the proposed task	✗	✓
Full RTS games	✗	✓
Documented code	✗	✓
Documented scripting language	✗	✓
High-level control of units	✗	✓
Low-level control of units	✓	✗
Clear separation between AI and engine code	✓	✗
Reference documentation	✓	✓
Open source	✓	✓

when building the initial base. Any delay in the beginning can mean defeat in the most advanced stages of the game.

Balance your gold and wood income ratios: in games like Wargus, a player needs to use both gold and wood to develop his army (and oil in water maps). Thus, it is important for the player to have both resources available, otherwise, his development will stale.

Scouting the environment is the key to victory: to react to the enemy's moves, one needs to know what the enemy is doing. This leads to scouting the map, one of the most important activities in the game.

Early-game order of construction: a common practice in board games like chess and checkers is to document standard opening and ending strategies to achieve good performance in the game. This is no different in RTS games, where players have cataloged the order in which a player have to build his constructions to have some structures as soon as possible in the game. Learning some of these orderings is essential to defeat expert players, and to guarantee the victory.

5 Engine selection

During the prospection phase, it was clear that most researchers in the RTS field were using either the Open Real Time Strategy (ORTS) engine or the Stratagus engine, with this last one commonly combined to the Wargus game. Therefore, these were the ones we chose to evaluate. In this section we will present the conclusions that we came to after evaluating these two engines and some suggestions on where these engines could improve.

Stratagus, best described in [Ponsen et al. 2005], is an open-source engine for building RTS games that supports both single player and multi player games. Stratagus was originally created as a Warcraft II clone and then grew to become a generic RTS game engine. Even though it was not created with an academic focus, it has been successfully used in AI research.

ORTS is also an open-source engine for building RTS games. Originally created to provide a hack-free RTS environment [Buro 2002], ORTS has also been used successfully in research. One of the key features of ORTS is that it uses a client-server model, in which only the server runs the game simulation and, consequently, prevents players from using map-revealing hacks.

To properly evaluate the value of both engines to our research, we decided to implement a simple task in them. The task we set was to implement influence maps to perform some map analysis activities.

While performing this task, we realized that even though ORTS already had some terrain analysis primitives, the lack of proper documentation made those barely usable to us. With Stratagus, although there was no initial support for influence maps, we were able to implement them without trouble.

An important aspect that is oftentimes overlooked when selecting an engine is the legibility and ease of editing of the engine's source code. Even though the sources of both engines are easily accessible, Stratagus feels like having more orthogonal and documented APIs, probably because it has been in development for more time than ORTS. The effect of ORTS's lack of documentation is that users of the engine tend to spend more time reading the code to figure out how it works.

Another important aspect is the scripting language used by the engine. While Stratagus uses Lua, a de facto standard for scripting in games, ORTS used, at the time of our evaluation, a mostly undocumented, home-brew scripting language. Again, the lack of documentation of this home-brew language makes ORTS a disservice, making the engine usage harder than it should.

With regards to the standard AI, Stratagus comes with the units' basic behavior already implemented, but access to lower level behaviors is denied to users, who are left with no ability to control all aspects of the AI without modifying the engine directly. ORTS's approach is equally problematic, because it only provides rather low-level controls, which means the user must program the most basic behaviors of the units before focusing in actually solving his problem. The ideal compromise in this question would be if engines could provide basic behaviors that could be overridden if needed.

In addition to the basic unit behavior, Stratagus also comes with some bot strategies implemented, which leverages the creation of single-player games and, perhaps for that reason, Stratagus also has some full RTS games implemented for it, making it a good platform for experimentation. This gives Stratagus a clear advantage over ORTS, which only had some "synthetic" games created for its AI competition: collaborative pathfinding, strategic combat, tactical combat and a minimal full RTS game. For the reasons aforementioned, we opted for using Stratagus in our work. Table 1 summarizes the information found in this section.

5.1 Wargus

Having selected Stratagus as our evaluation platform, we decided to base our system on the Wargus game rules. The main reason behind this decision is the fact that there are strategy guides written for Warcraft II, from where we could extract expert information. Since Warcraft II rules were used, it is worth defining some of the game's aspects for a better understanding of this paper.

Wargus is a Real-Time Strategy game set in the mythical kingdom of Azeroth. In this game, players are given the task of maintaining a thriving economy while building an army with which to destroy the enemy. There are two major races in Wargus from where the players can choose: humans, and orcs, a humanoid green-skinned race with broad noses and distinctive tusks. In this world, both races have access to melee, ranged, naval, aerial and spellcaster units.

Wargus allows users to play against AI opponents in separate human and orc campaigns and in stand-alone scenarios. The main objective of stand-alone scenarios is to destroy the player's opponents, and most campaign missions follow the pattern of collecting resources, building a base and units to, then, destroy the opponents. Other campaign missions feature specific objectives, such as rescuing troops, or escorting important characters through enemy territory. Regarding the game start conditions, in stand-alone scenario missions, players usually start the game with only a city center, (a "Town Hall" for the human race or a "Great Hall" for the orc race,) and a few worker units. Campaign missions, on the other hand, have different start conditions which depend on the story of that particular mission.

In Wargus, there are two essential resources that must be collected: gold, dug from gold mines, and wood, chopped from forests found in the game. Both resources are harvested by basic worker units, named "Peasant" in the human race and "Peon" in the orc race. These resources are delivered in the specific city centers of each race. Wood can also be delivered in lumber mills. Both wood and gold are required for the construction of most of the game's units and buildings. There is an additional resource in the game: oil, extracted from oil platforms in the sea. Oil can be considered a non-essential resource because it is only needed for the construction of water-related units and buildings.

Resources gathered, players can construct buildings and train units to build their armies. In the beginning of a match, only basic melee units can be trained in "Barracks". With the construction of new building types and the research of new technologies, new units, such as ranged and heavy melee units can be trained in the "Barracks". This pattern repeats throughout all games: every time a new building is constructed, there is the potential of unlocking a new unit unit. This unit will either be trained in an existing building or in the newly built one. As an example, as soon as a human builds an "Elven Lumber Mill", "Elven Archers" are unlocked and can be trained in a "Barracks". But when a "Gryphon Aviary" is built, the training of new "Gryphon Riders" will become available *at the "Gryphon Aviary"*.

6 Engine instrumentation

As Stratagus is more geared towards traditional RTS games and runs as a monolithic block, we had to adapt its code to extract information from the game state and to notify the user with the tips generated by the system. After studying its code, we came to the conclusion that hooking our functions to the engine's main loop would do what we needed without disrupting the operation of the engine. Specifically, our additions to the engine were: 1) an information-gathering system that scanned the engine data structures and cached them to be used by 2) the advisory system, that processed the game data and notified the user via the 3) notification system.

Recall that our system only uses the same kind of information that is available to the human player. Therefore, the information-gathering system must apply a filter to the game state data to remove all the information that would not be available to the human player before passing that data to the advisory system. The advisory system processing step just executes the encoded decision trees and, depending on the conclusion it comes to, a call to the notification system is made.

A minimalistic callback system was added to the construction behaviors in the engine to enable the system to know when a new unit was ready in case of processing the early-game building orders.

6.1 Information gathering

The basic data collection process is shown in algorithm 1. To make the notation more compact, set theory notation is used in the algorithm. These operations can be easily implemented in programming languages that support list comprehensions or can be easily translated to traditional *for loops*. The call to `ResetLocalState` is described in algorithm 2. The algorithms also make use of Iverson bracket, a notation that denotes a number that is one if the expression inside square brackets is satisfied and zero otherwise, as shown in equation (1).

$$[P] = \begin{cases} 1 & \text{if } P \text{ true,} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Algorithm 1 also references `IsLatticeBuilding()`. One can understand this as a function that returns True if a Wargus unit belongs to the set formed by the units displayed in Figure 3.

Figure 3 shows an abstraction of the game state used in this work. In it, it is assumed that the technological development of a player

Require: Stratagus player data structures
Ensure: Advisory system's current state updated
1: ResetLocalState()
2: Workers $\leftarrow \{w \mid w \in \text{PlayerU} \wedge \text{IsWorker}(w)\}$
3: VisibleE $\leftarrow \{e \mid e \in \text{Units} \wedge \text{IsVisible}(e) \wedge \text{IsEnemy}(e)\}$
4: Newenemies $\leftarrow$ (VisibleE $\setminus$ (Knownenemies $\cap$ VisibleE))
5: Knownenemies $\leftarrow$ Knownenemies $\cup$ Newenemies
6: Goldmines $\leftarrow \{g \mid g \in \text{Units} \wedge \text{IsVisible}(g) \wedge \text{IsMine}(e)\}$
7: IdleW $\leftarrow \{w \mid w \in \text{Workers} \wedge \text{IsIdle}(w) \wedge \text{IsRemoved}(w)\}$
8: Totalworkers $\leftarrow |$ Workers $|$
9: Newgoldmines $\leftarrow$ Goldmines $\setminus$ Oldgoldmines
10: Numtowers $\leftarrow | \{t \mid t \in \text{PlayerU} \wedge \text{IsTower}(t)\} |$
11: Goldpotential $\leftarrow$ sum($\{\text{GoldLeft}(g) | \ g \in \text{Goldmines}\}$)
12: Buildings $\leftarrow \{b \mid b \in \text{PlayerU} \wedge \text{IsLatticeBuilding}(b)\}$
13: Castlestage $\leftarrow$ [("Castle" $\in$ PlayerU) $\vee$ ("Fortress" $\in$ PlayerU)]

Algorithm 1: *Update stage of the advisory system. In this algorithm it is assumed that there exists a* `sum` *algorithm, capable of summing all the values of elements contained in a set. There are also used Iverson brackets, described in equation* (1)*, and operations for selecting a set's element.* `ResetLocalState()` *is described in algorithm 2.*

Table 2: *Naming convention used in figure 3. The first column shows the abbreviation used in the figure. In the second and third columns are shown the equivalent building names for each race of the game.*

	Race	
Convention	Human	Orc
Ap	Gryphon Aviary	Dragon Roost
Ba	Barracks	Barracks
Bs	Blacksmith	Blacksmith
Ca	Castle	Fortress
Kp	Keep	Stronghold
Lm	Elven Lumber Mill	Troll Lumber Mill
Mt	Mage Tower	Temple of the Damned
Th	Town Hall	Great Hall
Tm	Church	Altar of Storms
St	Stables	Ogre Mound

is a function of the different building types he has constructed. A consequence of this approach is that, every time a new building is constructed, a player's state is changed. This abstraction makes the game state space more tractable and simplifies the reasoning process. The naming convention used in figure 3 is shown on table 2. This abstraction is based on the one presented in [Ponsen et al. 2006].

The states shown in figure 3 were grouped in a way that divided the technological development of a player in four groups: an early-game stage, where the player has a very minimal base consisting of a Town Hall and a Barracks or less; a base construction stage, where the player is building his base and preparing the foundations to expand it; a middle-advanced stage where new and advanced units become available and, finally, an advanced, or "Castle" stage, in which a player already owns the most advanced city center ("Castle" for humans or "Fortress" for orcs) and, thus, has gradual access to all of the game's units and to complex unit combinations.

It is important to note that not all building types were used to construct the abstraction shown in figure 3. This is due to the fact that they are either required during all the game, such as farms, that must exist to provide food to the army and to permit the training of new units, or not entirely important for a match, such as towers, defensive buildings unable to move in the map or water-related buildings, which are only important in sea maps.

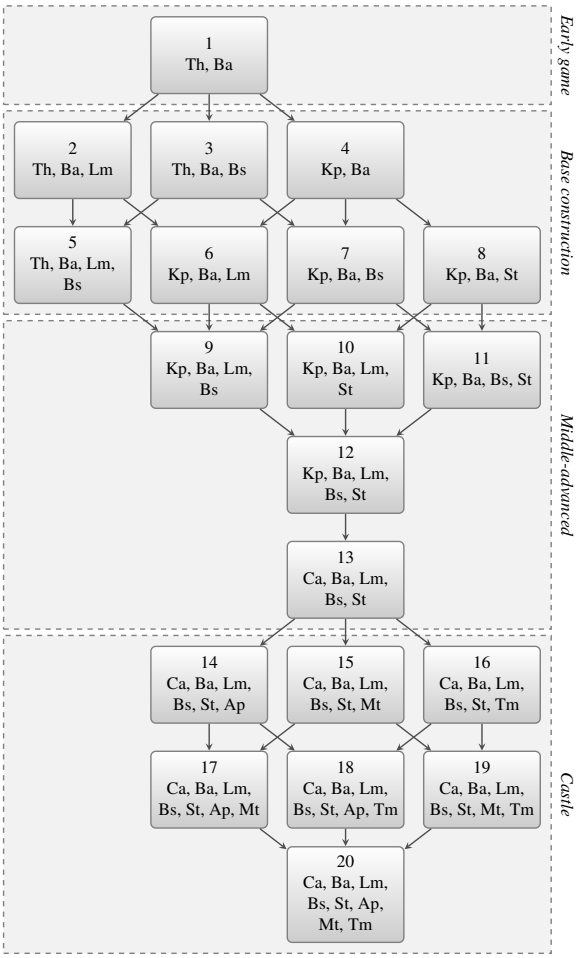

Figure 3: *Abstraction of the Wargus game state. In this figure, state transitions are triggered by the construction of a new building, and each state is defined by the types of buildings already built. The naming conventions are described in table 2.*

```
1  void CheckGoldThreshold() {
2    string msg = "Explore the map to find gold mines.";
3    if(PlayerData.goldPotential < GOLD_THRESHOLD) {
4      notifier.Notify(msg);
5      PlayerData.shouldScout = true;
6    }
7  }
8  // ...
9  void HandleIdleWorkers() {
10   if(GameCycle - LastWarning > IDLE_WORKERS_COOLDOWN) {
11     string msg = "You have idle workers.";
12     if(PlayerData.shouldScout) {
13       msg+=" What about using them to explore the map?";
14     }
15   // ...
16     notifier.Notify(msg);
17     LastWarning = GameCycle;
18   }
19 }
```

Figure 4: *Two excerpts from the coded expert rules. In the function `CheckGoldThreshold`, the system checks the amount of available gold mines and the remaining gold in them and stores it in the goldPotential variable. Whenever the gold potential reaches a low value, the system warns the player to explore the map. The `HandleIdleWorkers` function is called by the system whenever it finds idle worker units. In that case, one of the checks made by that function is if the player should explore the map. In positive case, the player is notified of that, with the advice of using the idle workers to explore the map. The `IDLE_WORKERS_COOLDOWN` variable is a configurable parameter of the system that defines the interval in which the check for idle workers should be done. `GOLD_THRESHOLD` is also a configurable parameter, but it defines how much gold should be considered low by its corresponding function.*

Require: Advisory system data structures
Ensure: Updated advisory system data structures
1: Buildings ← {∅}
2: IdleW ← {∅}
3: Newgoldmines ← {∅}
4: Oldgoldmines ← {∅}
5: Shouldchop ← false
6: Shouldmine ← false
7: Shouldscout ← false
8: Toomanytowers ← false
9: Totalworkers ← 0
10: Goldpotential ← 0
11: VisibleE ← {∅}
12: Allies ← Listofallies()
13: Enemies ← Listofenemies()
14: Oldgoldmines ← Goldmines
15: Oldenemies ← Enemies

Algorithm 2: `ResetLocalState`*, algorithm that clears the data structures used by the advisory system.*

6.2 Advisory system

The general guidelines presented in section 4 were encoded as a decision tree that considered some other aspects of the game, such as the time spent by the player since the beginning of the game and the constraints the player might have to follow those guidelines. Also, having a decision tree executed at every frame would result in the execution of the same actions until the state of the game is changed. To overcome this limitation, some actions were encoded to record the time they were last activated (as shown in lines 10 and 17 of figure 4). Together with nodes that checked for a timeout before recursing some branches, this simple system implemented a way of preventing constant execution of the same actions until the game state is changed. As one might expect, tuning the time parameters was needed to make the tree behave in a suitable way for execution.

Figure 4 shows an example of expert rules that were encoded for the usage of the advisory system. The figure shows two of the many functions defined in the system to handle the various rules described in section 4. The functions shown in the figure defined, respectively, a check for the gold extraction potential a player has and a function to handle idle workers when they are found. The purpose of the latter is that neither Wargus nor Warcraft II warn the player when he has idle workers. As the workers are essential for the development of an army, the advisory system not only checks if there are idle workers, but also tries to suggest what to do with these workers, such as exploring the map for more resources or allocating them to do other tasks.

6.3 Notification system

As a design decision, we defined that the messages displayed to the user would be made available in three ways: via the default notification interface provided by the engine, via a logging subsystem that cached all the messages shown in that game and via a speech synthesis system.

Being Stratagus a single-threaded application, the speech synthesis was made in a separate process. To synthesize the voice, we used the Festival [Black and Taylor 1997] Text-to-speech (TTS) system. By the premise that synthesizing all messages generated by the system would be boring to the player, we only synthesized the ones we considered important. Independent of the voice synthesis, all messages were rendered on-screen. Figure 5 shows how they looked for the user. In the figure, lines rendered in yellow were generated by the game, while lines in white were generated by the advisory system.

Figure 5: *Screenshot of the Wargus game with the notification system enabled. In the figure, the messages shown in white are the ones generated by our system, while the ones in yellow were generated by the game engine. For messages with some spatial meaning, the location related to it was rendered in the game mini-map, as emphasized by the red arrow.*

Table 3: *Usefulness of the advisory system, according to the users' opinions.*

Evaluation	Number of participants	Percentage
Useful	0	0%
Mostly useful	6	85.7%
Indifferent	0	0%
Mostly useless	1	14.3%
Useless	0	0%

7 System evaluation

Some tests were designed to evaluate the advisory system's performance with regards to the quality and utility of the generated tips. The objective of the tests was to evaluate if RTS game players considered if an advisory system was useful and the tips generated by our system were worthwhile. More specifically, the critical point of the user evaluation was to know, qualitatively, the opinion of the users with regards to the usefulness of the tips. To give objectivity to the test, only users who already knew RTS games were selected to playtest. That way, we could be sure the problems they could face would be more related to the advisory system than to the interface of the game.

The test consisted of letting the users play two matches of Wargus. In both, the task to be performed by them was to try to win the game. However, we made it clear to all users that the outcome of the match did not matter for the test, and this objective was set only for the users to have a clear goal. In one of the matches, the advisory system was turned off, at the other, it was turned on, in this order.

The tests were performed in a laboratory specially prepared to perform user tests, and all commands sent to the game by the user were recorded in Stratagus' internal log format in case a replay was needed. In total, seven users participated in the tests. The tests consisted of an interview to know the user's profile, followed by the testing sessions and, then, the users were invited to fill out a form with their opinions about the system.

All users that were submitted to the test were male, with a mean age of 26, and standard deviation of 3.2 years. From the results of the interview, we saw that most of the users considered themselves with some ability in RTS games, and had the profile we looked for, which means they played RTS games at least once and grasped the involved concepts, this doesn't mean they had to be good players.

Tables 3 to 10 summarize the answers given by the users at the post-test survey. For the results presented in tables 3 and 4, we estimate that the negative opinions are related to the fact that most tips generated by our system are related to resource management, what may have made them repetitive and, to some extent, inconvenient. Similarly, even though there were some rules that encoded scouting tips, those were triggered with a "good" frequency, and, when they were triggered, the users were unable to perceive them.

Table 4: *Users' opinions regarding the frequency of the tips. In other words, given the frequency of the tip presentation, the users were asked if they would change the tip frequency.*

Evaluation	Number of participants	Percentage
Much less frequent	0	0%
Less frequent	3	42.8%
Perfect	3	42.8%
More frequent	1	14.3%
Much more frequent	0	0%

Table 5: *Users' opinions regarding the advisory system. If the users thought the approach used was useful, they marked "Yes", and "No" otherwise.*

Evaluation	Number of participants	Percentage
Yes	7	100%
No	0	0%

Despite the negative opinions about the frequency of tips, users not only considered them useful, as shown in table 3, but also considered the approach taken valid (table 5). Some aspects of the system need improvement, given that the advisory system, as the users perceived it, concentrated its tips in resource management tips and basic tactical tips (table 9). The most interesting topics for the creation of new tips, according to the users, were in battle tactics, base layout, and defense tactics (table 10), which indicates that the users care about one of the most important aspects of the game: the combat.

Regarding the speech synthesis system, user's opinions indicate that this approach was pleasant to them (table 6), even though their opinions may have been divided between the synthesis, or not, of all messages (table 7). Despite the slight preference for the synthesis of all the messages, it is important to note that users were not subjected to tests in which all the messages were synthesized and, judging by the frequency of activation of certain branches of the tree during the implementation, users would probably be annoyed by the messages.

The results shown in table 8 indicates that most users had not seen systems similar to the one proposed in this paper. The ones that users pointed out as similar were already discussed in this text: SimCity's board of advisors and Warcraft III.

Through the responses of users and their reactions during the tests, we noticed that, for them, there is no clear distinction between

Table 6: *Usefulness of speech synthesis for the users.*

Evaluation	Number of participants	Percentage
Useful	2	28.6%
Mostly useful	3	42.9%
Indifferent	1	14.3%
Mostly useless	1	14.3%
Useless	0	0%

Table 7: *Should all tips be synthesized? Users were asked to answer "Yes" if they thought so and "No" otherwise.*

Evaluation	Number of participants	Percentage
Yes	4	57.1%
No	3	42.9%

Table 9: *Most useful tips according to the users. Users' evaluation regarding the usefulness of the tips given the competencies of an RTS game. Values sum more than* 100% *because people were able to select more than one option.*

Evaluation	Number of participants	Percentage
Resource management	7	100%
Basic tactical tips	4	57.1%
Scouting	1	14.3%
Counter-strategy	0	0%

Table 10: *Aspects of the game where more tips could be available. Given the set of tips generated, the users were invited to suggest where the system could benefit of having more tips. Values sum more than* 100% *because people were able to select more than one option.*

Evaluation	Number of participants	Percentage
Resource management	0	0%
Scouting	2	28.6%
Battle tactics	6	85.7%
Base layout tactics	5	71.4%
Defense tactics	4	57.1%
Counter-strategy	2	28.6%
Unit combinations for more powerful attacks	4	57.1%
How to invest resources	2	28.6%

Table 8: *Users were asked to say if they knew any system similar to the presented in this work and answered "Yes" if they knew them and "No" otherwise.*

Evaluation	Number of participants	Percentage
Yes	2	28.6%
No	5	71.4%

what would be tips on strategy (and therefore related to actions in the game) and tips on the usability of the game interface and its mechanics. As tips on the game's interface weren't part of the work presented here, some users found themselves frustrated by this design decision, mostly because of the lack of intimacy with the Warcraft II interface.

Some aspects are difficult to separate, though. For example: when a Lumber Mill is built near a forest and there are workers extracting wood from this forest, the wood income ratio would be greater than if the forest was farther. Even though this might be an obvious conclusion to experience users, this is not for novices. Therefore, different designers might come to different conclusions on what kind of tips should be encoded.

Users were asked to suggest changes to the advisory system. After analyzing their comments, we came to the conclusion that the changes suggested by them are more related to having more competencies of the game contemplated than to architectural changes in the advisory system, which might indicate that the current architecture of the system was pleasant to the users.

In the current implementation, the advisory system has only two options regarding the tips: they are either completely enabled or completely disabled. Comments from users indicate it would be interesting if the system had a greater level of granularity, allowing tips to be enabled or disabled individually. In particular, a tip that was enabled constantly, related to the existence of idle workers, annoyed some players, confirming the pertinence of this suggestion. The cause of the repetition comes from the structure of decision trees, which trigger actions whenever their conditions are satisfied and, therefore, it might be interesting to evaluate the effectiveness of the system using other data structures.

Other suggestions given by the users were to encode tips that did not leave the player in an idle state (and, therefore, in disadvantage against a non-idle player), and a higher level of interactivity with system, in a way that users could submit queries to the advisory system. One common complaint was that users did not know how to create some units, and wanted the system to tell them what to do to achieve their objective. This is a pertinent suggestion and the usefulness of the advisory system would increase if it had this feature.

Table 11: *Points awarded to the player that destroyed an unit of the given kind.*

Unit	Points	Unit	Points
Wall	1	Tower	95
Critter	1	Farm	100
Peasant	30	Lumber mill	150
Flying Machine	40	Runestone	150
Tanker	40	Barracks	160
Footman	50	Oil Rig	160
Transport	50	Blacksmith	170
Archer	60	Shipyard	170
Ranger	70	Foundry	200
Dwarves	100	Guard Tower	200
Knight	100	Refinery	200
Ballista	100	Town Hall	200
Mage	100	Stables	210
Demon	100	Inventor	230
Paladin	110	Church	240
Legendary Hero	120	Mage Tower	240
Submarine	120	Cannon Tower	250
Destroyer	150	Aviary	280
Gryphon	150	Keep	600
Battleship	300	Castle	1500

7.1 User performance

As mentioned, all the tests were recorded using Stratagus' internal log format. Therefore, we were able to analyze the performance of the players with and without the advisory system. As a performance function, we used the score attribution system used by Warcraft II, where each destroyed unit counted points to the player that destroyed them. Table 11 shows the score earned by a player for each unit destroyed.

For each test set, we plotted graphs that showed the performance of the users using the aforementioned metric. Figure 6 shows the performance of a specific user that had a performance similar to other users. Figure 7 shows a graph with an atypical performance plot. Without the system, the user lost, but a mere analysis of the graph might lead one to think that he might have won it. The fact is that the user built many defensive towers, and even when he had no combat units, his towers continued to destroy his attackers, earning him some more points.

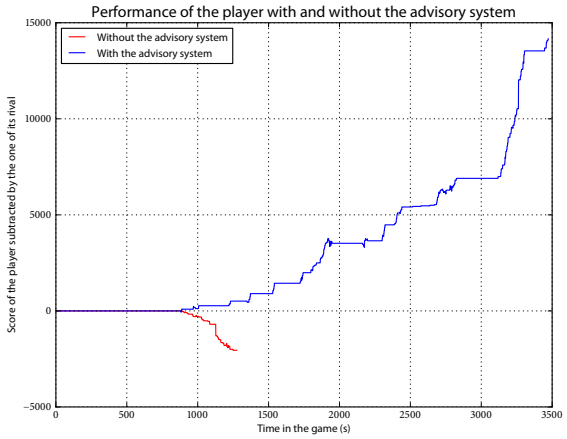

Figure 6: *Typical performance of a player with and without the advisory system. The value shown in the vertical axis is the difference between the user points and of that of his (computer) opponent and the plateaus show time intervals in which no combat occurred.*

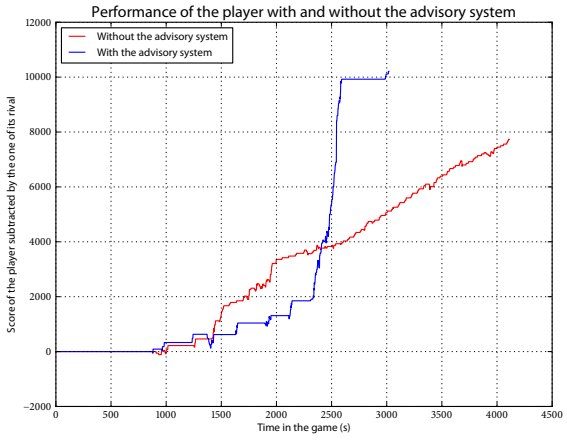

Figure 7: *"Anomalous" performance of a player with and without the advisory system. Even with his defeat without using the system, his performance was good. The value shown in the vertical axis is the difference between the user points and of that of his (computer) opponent and the plateaus show time intervals in which no combat occurred.*

8 Conclusion

In this paper we presented an expert system designed to help players in RTS games through the display of tactical and strategical tips. Basically, the system evaluates the game state following pre-specified metrics and traverse a decision tree built upon information available from strategy guides, giving the user some advice about the actions to be taken. Differently from some approaches used in games, one advantage of the proposed system is that is does not force the user to take specific actions. It works as an advisory system, in which advices can be taken at user discretion. Several experiments were performed to evaluate the system. A group of users played a game with and without the system and, in general, liked the approach, also giving valuable insights for its improvement. Performance tests were also executed and showed that the system does not compromise performance. Overall, we can conclude that this type of system has a good potential to be used in commercial games.

There are several paths for future work. First of all, some tuning is necessary to reduce the frequency of some tips which can be annoying to the user. A graphical interface allowing users to choose the content and frequency of the tips is in our plans. Also, the possibility of including new tips and conditions on the fly would be interesting for the system. In fact, a very promising, yet challenging, approach would be the use of machine learning and data mining techniques to learn from other games and infer tactics and strategies for the system, instead of relying only on precompiled information from the strategy guides. This would make the system more flexible and usable in different scenarios.

Acknowledgments

The authors would like to thank the financial support provided by CNPq, CAPES, and Fapemig in the development of this work.

References

ALCÁZAR, V., BORRAJO, D., AND LINARES, C. 2008. Modelling a RTS planning domain with Cost Conversion and rewards. In *ECAI*, A. Botea and C. L. López, Eds.

BAUMGARTEN, R., COLTON, S., AND MORRIS, M. 2009. Combining ai methods for learning bots in a Real-Time Strategy Game. *International Journal of Computer Games Technology 2009*, 10.

BLACK, A. W., AND TAYLOR, P. A. 1997. The Festival Speech Synthesis System: System documentation. Tech. Rep. HCRC/TR-83, Human Communciation Research Centre, University of Edinburgh, Scotland, UK. Disponível em http://www.cstr.ed.ac.uk/projects/festival.html.

BLIZZARD, 2010. Warcraft™ II strategy. http://classic.battle.net/war2/strategy.shtml, Abril. Acessado em 20 de abril de 2010.

BURO, M. 2002. ORTS: A hack-free RTS game environment. In *Proceedings of the International Computers and Games Conference*.

BURO, M. 2004. Call for AI research in RTS games. In *Proceedings of the 4th Workshop on Challenges in Game AI*.

DE FREITAS CUNHA, R. L. 2010. *Um sistema de apoio ao jogador para jogos de estratégia em tempo real*. Master's thesis, Universidade Federal de Minas Gerais.

HAGELBÄCK, J., AND JOHANSSON, S. J. 2008. Using multi-agent potential fields in real-time strategy games. In *AAMAS (2)*, IFAAMAS, L. Padgham, D. C. Parkes, J. Müller, and S. Parsons, Eds., 631–638.

HOANG, H., LEE-URBAN, S., AND MUÑOZ-AVILA, H. 2005. Hierarchical plan representations for encoding strategic game AI. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*, AAAI Press.

LEE-URBAN, S., PARKER, A., KUTER, U., MUNOZ-AVILA, H., AND NAU, D. 2007. Transfer learning of hierarchical task-network planning methods in a Real-Time Strategy games. In *Proceedings of the Workshop on Artificial Intelligence Planning and Learning*.

MCCOY, J., AND MATEAS, M. 2008. An integrated agent for playing real-time strategy games. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*.

PONSEN, M. J., LEE-URBAN, S., MUÑOZ-AVILA, H., AHA, D. W., AND MOLINEAUX, M. 2005. Stratagus: An open-source game engine for research in real-time strategy games. Tech. rep., Navy Center for Naval Research Laboratory.

PONSEN, M., MUNOZ-AVILA, H., SPRONCK, P., AND AHA, D. W. 2006. Automatically generating game tactics through evolutionary learning. *AI Magazine 27*, 3, 75–84.

WEBER, B. G., AND MATEAS, M. 2009. A data mining approach to strategy prediction. In *Proceedings of the IEEE Symposium on Computational Intelligence & Games*, P. L. Lanzi, Ed., IEEE.