Efficient Resource Management in StarCraft: Brood War

DAT5, Fall 2010
Group d517a
7th semester
Department of Computer Science
Aalborg University
December 20th 2010

Student Report

Title:

Efficient Resource Management in StarCraft: Brood War

Theme:

Machine Intelligence

Period:

Fall 2010

Group:

d517a

Group members:

Dion Bak Christensen

Henrik Ossipoff Hansen

Lasse Juul-Jensen

Kasper Kastaniegaard

Supervisors:

Yifeng Zeng

Jorge Pablo Cordero Hernandez

Abstract:

Resource management is a fundamental part of real-time strategy games, since a steady flow of resources is necessary in order to build an army thus win the game. We deal with resource management in the video game StarCraft, by developing an algorithm for predictable and efficient gathering of minerals in StarCraft. We present preliminary results on a scouting strategy for chosing base expansion locations. Our mineral gathering algorithm achieve a higher income rate than the built-in StarCraft mining approach, and we conclude that our approach is more predictable. We have shown a correlation between properties in an area of a map in StarCraft, and the location a human player is more likely to utilise for base expansions.

Circulation: 7

Pages: 46

Ended: December 20th 2010

PREFACE

This is the technical report made as documentation of the fall semester 2010, group d517a at Department of Computer Science, Aalborg University. The topic of the report is resource management within the real-time strategy game StarCraft by Blizzard Entertainment. The report documents related works in the field and the process of developing methods for solving problems within the topic.

Reading guide

Throughout the technical report, *bases* and expansions will be used interchangeably.

Code samples will be highlighted using a listing feature and referenced by numbers in the order of which they appear. Listings will be presented like this:

```
1 #include <iostream>
2
3 int main(int argc, char **argv)
4 {
5         cout << "This is a code sample." << endl;
6         return 0;
7 }
```

Implementation specific code inlined in sections will be presented in a `typwriter font`. Introductory concepts and words will be *emphasised*. References to figures, tables and listings will appear in the form of a chapter number and a counter. As an example, the second table in Chapter 3 will be referenced as Table 3.2.

The rest of the report is organised as follows:

Chapter 1 This chapter contains a general introduction to StarCraft as well as a general motivation into the problem addressed in this report. The chapter ends with the problem statement.

Chapter 2 This chapter contains a list of related works within the field of the problem domain. The chapter ends with a general summary discussing the relevance of all discussed related works, and how it may influence the problem.

Chapter 3 This chapter contains the first part of our solution to the problem; a predictable way to gather minerals during a game of StarCraft. The chapter will contain a formal solution as well as experiments with comparisons to the built-in StarCraft mining approach.

Chapter 4 This chapter contains the second part of our solution to the problem; how to scan and scout when searching for a new location to expand on the map. The chapter will first contain a discussion on which variables one should consider regarding base expansions. The chapter then goes on to study how to measure influence and how to design a decision model for the problem. Lastly, the chapter analyses the data gathered and summarises the achievements and open problems.

Chapter 5 This chapter contains the conclusion for the problem and the future works that can be done in order to attend to open problems.

The group would like to thank the open source community for making the two StarCraft APIs, BWAPI and BWTA, and for the support through an IRC channel. We would also like to thank *MasterofChaos* for providing the tool Chaoslauncher, which enables the injection of libraries into StarCraft. We thank *Bo Hedegaard Andersen* for his enthusiasm and cheerful company during the project period. Lastly we would like to thank *Michael Madsen* for his inputs to the various C++ curiosities we encountered throughout the scope of the project.

CONTENTS

CHAPTER 1

INTRODUCTION

This technical report deals with resource management in the video game *StarCraft: Brood War*, from this point on referred to as StarCraft. Resource management, defined in Section 1.4, deals with the issue of collecting resources and using these resources. The remainder of this chapter is divided into two parts. The first part will introduce the real-time strategy genre, introduce concepts and game mechanics in StarCraft, argue for the relevance of managing resources in a real-time strategy game, and give a brief introduction to the development tools provided in order to achieve possible solutions to the problem of the report. The second part offers a clarification of the problem, and defines the scope of the problem that is dealt with in this report.

1.1 Game mechanics of StarCraft

StarCraft is a classic real-time strategy game created by Blizzard Entertainment in 1998. In a real-time strategy game, gameplay consists of rapid multiple-unit actions requiring resource collection and control, base building, technology development, and military unit control [4]. StarCraft has a science fiction setting, in which players compete to win by controlling areas of the map and eliminating enemy forces. To help players achieve their goals, several types of units and structures may be constructed.

Even though a lot of time has passed since the release of StarCraft, and many real-time strategy games have been released since then, StarCraft is still the best selling real-time strategy game of all time [2]. The popularity of StarCraft in eSports may have influenced this, as StarCraft is very popular in those circles. In Korea, professional StarCraft players are comparable to athletes; idolized by fans and receiving six digit pay checks [7, 8].

The game contains three different races, each with their own strengths and weaknesses. The organic race Zerg has many small, fast building units and tends to focus on strength in numbers while the psionic race Protoss has big slow building units and tend to focus on quality over quantity. The human race Terran is in-between with a mix of slow and fast building units. For all races, structures and units require an amount of one or both of the two types of resources; minerals and vespene gas. Additionally there is a unit cap, which may be raised by creating a structure or training a special unit, depending on the race.

The use of StarCraft as an eSports game has put a lot of pressure on balancing the game, and though it is difficult to balance three very different races, StarCraft is arguably one of the

best balanced real-time strategy games. This can be observed as there is no general preferred race between professional StarCraft players and there is no single race dominating the top of the StarCraft ladder[1]. The developers have also worked towards balancing the game when problems have occurred. The most well-known of these cases is probably the *Zerg Rush*, in which a player playing Zerg would spawn zerglings very swiftly and attack the opponent as soon as possible. Blizzard Entertainment recognised that this strategy was too powerful and as a result this was made more difficult by a change in patch 1.08[2].

Resources are gathered by worker units which are able to harvest minerals and vespene gas, and build structures. The worker units gather minerals, by transporting them from their location to a resource depot owned by the player, native to their particular race. Vespene gas can be collected by first constructing a gas mine on a vespene geyser and then collecting gas from the mine in a similar way. Worker units can construct buildings by moving to a buildable area and starting construction. This works in three different ways, depending on race:

- Zerg buildings, with the exception of their resource depot–a Hatchery–and vespene gas refinery, can only be constructed on a carpet of biomass called Creep. Creep is generated by the Hatchery and expands as more structures are placed upon it. The worker unit is morphed to a building, sacrificing the unit in the process.

- Protoss have a special type of building–a Pylon–that provides their race with the supplies needed to construct units, while at the same time powering buildings and allowing the construction of more buildings in a small elliptic area from their centre. All Protoss buildings, with the exception of their resource depot, vespene gas refinery and the pylons, may only be constructed in these areas. While the buildings are being constructed, the worker unit can continue work elsewhere, collecting resources or starting construction of even more buildings.

- Terran buildings can be constructed in any buildable area without any additional requirements. The worker unit is however needed for the construction, and cannot be used elsewhere before the building is finished. A feat of the Terran race is that a large part of the standard Terran structures may be moved even after they are built.

Players compete to eliminate all enemy players by destroying all of their buildings. In order to do this, players must gather resources that can be used to construct units for gathering more resources, buildings for creating powerful units as well as combat units that can be used to destroy the enemy.

Resources on a level, also referred to as a map, are limited and decentralised, meaning that in order to gain these resources, outposts or expansions should be created to assist in dominating the resources. More resources mean the potential for a larger or more powerful army.

Unit producing buildings may produce one unit at a time (with the exception of Zerg who have *one* type of unit producing building that may make up to three at a time) and creating more unit producing buildings allows for faster unit creation, given that the player has the required resources. Some buildings are not used for constructing units, but rather to upgrade units. Upgrading is generally more time consuming than constructing a unit. When an upgrade has been completed, however, the upgrade is available to all existing units of the type that the upgrade was made for, as well as all future units of this type. Upgrades may be used for providing new abilities, more combat strength or a defensive bonus.

[1] `http://www.iccup.com/starcraft/ladder/1x1.html`
[2] `http://www.danielstaniforth.co.uk/StarCraft/patch.html`

A special type of building exists, referred to as a defensive structure. A defensive structure is a structure that may assist in defending a position like a base. Defensive structures will fire at any enemy within its range and may offer great support for defending units. Defensive structures have a good deal of damage resistance and will never count towards the unit cap. The catch is that defensive structures cannot be moved, and have construction times during which they may be attacked but cannot return fire.

An important distinction between units exists. Ground forces move on the ground and must therefore walk around obstacles to reach their destination. Flying units can move anywhere, but are more expensive or less powerful than ground units. All races have a flying unit that is able to transport some amount of ground units to another location, meaning that no region is completely inaccessible to ground units, though some regions are more difficult to arrive at.

The game contains several different combat units of varying strength and utility. Every unit is specific to their particular race. At the beginning of the game only the most basic units are available. In order to gain access to more powerful units, the player must first construct the buildings required for a particular unit. The buildings may also have the restriction that they cannot be constructed until another building has been constructed or upgraded. This ordering is known as a tech-tree, as the dependencies between buildings can be viewed as a tree structure.

The tech-tree enforces an evolution of the game. A strategy that works well in the beginning of the game, may not work well at a later point in time as the range of available units have increased for both players. The tech-tree also means that each player have a difficult choice to make. How should they prioritise moving through the tech-tree in comparison to constructing units that can be used for defence and attack? Too much emphasis on moving through the tech-tree will leave few resources available for defence and offense. This may result in being overwhelmed by the enemies' early game units or the enemy expanding unchallenged. Emphasis on units may be a good strategy in the beginning of the game. However, the enemy might be able to defend against these and construct more powerful units in the meantime. Note that this pattern of balancing one useful aspect of the game against another useful aspect exists for more than just unit production as it is evident in later sections.

Other than the combat and resource management, players also need to scout areas of the map in order to gain intelligence on the enemy. The game contains an inherent uncertainty, pertaining to the so called *fog of war*. Only areas of the map that are currently occupied by a friendly unit can be viewed by the player, while all other areas are covered by a grey fog. One of the basic strategies of the game is to remain out of sight and make surprise attacks on the opponent, while at the same time keeping as much information on the opponent as possible.

StarCraft is rarely one unending battle, but rather a set of skirmishes that may change the might of either player and in the end decide who have the upper hand. Resources in StarCraft exist around the map in the form of *vespene gas geysers* and *mineral fields*. Both play a large role in supporting the economy of a player.

Mineral Fields Minerals can be gathered by a worker unit by having the worker unit move to a mineral field, spend some time gathering the mineral and then transporting the mined portion back to a particular type of building called a *resource deposit*. Constructing a resource deposit as close as possible to a cluster of mineral fields will minimise the transport time for the workers and thereby yield a higher rate of minerals than a resource deposit positioned farther away. The amount of available resources for a player is a global value that will only decrease as the result of spending the resources on buildings or units.

A mineral field may contain any amount of minerals, usually $1,500$, and a worker may carry

up to 8 minerals at a time. Each mineral field may have at most one worker mining minerals from it at a given point in time. For this reason several trips back and forth is usually required to completely deplete a mineral field.

Mineral fields are typically clustered together. The mineral fields contained within a cluster are variable and differs greatly depending on the map.

Vespene Gas Geysers Vespene gas is gathered from vespene gas geysers which usually contain around 5,000 units of vespene gas. Vespene gas can be viewed as a slightly more sophisticated resource as it is generally not required for the most standard units and buildings. As a general rule, any unit that possess special abilities or is exceptionally strong will not only require minerals to construct, but vespene gas as well. The buildings that are required for these units may also require vespene gas in addition to minerals. Available vespene gas is, just like minerals, a global value for each player that only decreases as the result of spending vespene gas.

Vespene gas is in most cases sparser than mineral fields. On most maps, a base location will contain several mineral fields and zero or one vespene gas geyser. In order to gather vespene gas, a building–known as a refinery–must be constructed on top of a vespene gas geyser. Only one refinery may be constructed on a particular vespene gas geyser and only the player that has constructed the building may extract vespene gas from the vespene gas geyser.

When the refinery has been constructed, one worker unit at a time may enter it, spend some time inside and exit with an amount of vespene gas that is then returned to the resource deposit. Other workers attempting to extract vespene gas from the refinery will have to wait outside while another worker is extracting. Due to the sparseness vespene gas geysers, gathering vespene gas is a slow process when compared to gathering minerals. A player will often experience that the limiting factor when constructing powerful units is vespene gas rather than minerals. For this reason it is often advisable to use sufficient workers for extracting vespene gas such that the refinery is always being used.

Maps with unusual high amounts of resources either grouped together in a single cluster or scattered across the whole map are usually referred to as *money maps*.

1.2 Resource Management

This section will cover the subject of resource management in StarCraft. In this section, the elements of resource management are explained as well as the reasoning for the importance of this in regard to winning a game. The basis of the information contained within this section is expert knowledge from human players.

Success in StarCraft depends heavily on macro and micromanagement[3]. The ultimate goal is to eliminate all enemy structures, resulting in the last remaining player (or players in team games) winning the game. In order to destroy the enemy, an army of attack units is needed as well as a plan for destroying the enemy. Resources are required in order to construct an army, and the speed at which the army can be constructed depends on the amount of unit producing structures available. These structures also consume resources when constructed.

This means that given a high rate of resources, it is possible to construct a high amount of combat units rather fast. Resource management is about getting a high rate of resources and converting the resources into useable units and structures.

[3]In gaming, micromanagement describes minor, detailed gameplay elements, addressed by the player, while macro management refers to management of the overall game

1.2.1 Resource Gathering

In this technical report, resource gathering is defined as the act of transporting resources from the area in which they exist on the map to the resource deposit, such that the resources may be used by the player. The act of gathering resources is a basic element of resource management. As it have been mentioned in Chapter 1, resource gathering is performed by worker units. A larger amount of worker units gathering resources means a higher resource income rate. The StarCraft client allows worker units to gather resources automatically after having been given this order once. This is likely done to help a human player, as the amount of units and orders these units should receive, if no automation were present, is quite immense.

The automation does not promise to gather resources in the most optimal way, but it does ease the burden of a human player, leaving room to focus on strategic and tactical decisions.

1.2.2 Expanding

Maps in StarCraft are very diverse and with the build-in editor it is very easy for fans to create new maps. This means that maps can be very different and, in some cases, quite obscure. We have chosen to focus on maps that are considered balanced and fair for all participants–these are maps that are used in tournaments such as the International Cyber Cup[4]. These maps all share some general properties: Each player is given one resource deposit along with four workers and is placed at a start location near enough resources to build a base. The amount of workers given is not sufficient for fully utilising all of the mineral fields, encouraging players to construct more workers, but leaving room for players to decide how many additional workers should be constructed, before changing focus to something else.

In every game but the very shortest of games, it is necessary to expand to a new position, preferably before the resources at the old location run out. If a player focuses solely on the base given in the beginning they are bound to become overwhelmed later by a player that have expanded and created a much greater income rate. Controlling more resources than the opponent means the potential of creating a larger or better army than the opponent. A player can therefore be rather certain that the opponent have an interest in the resources and that the opponent know of the player's own interest in the resources.

It should be clear that expansions are something that should always be kept in mind, both one's own possible expansions and the opponents. Choosing a position for an expansion is not an easy matter either, as it expansions are often weak in their most early form. Players must decide what a good spot for an expansion is. Should it be close to an existing base or as far away from the enemy as possible? Should they go for the area with most resources or avoid the spots where the enemy is likely to expand? It is likely that there are a large amount of different considerations that should be taken into account when selecting a location for a new expansion.

1.2.3 Intelligence Gathering

Gathering of intelligence is quite important in a game like StarCraft. The actions of an opponent may reveal his strategy and make it possible for a player to create a counter-strategy. Gaining intelligence on the activities of an enemy may reduce or increase the desirability of expanding to a particular location as well as reduce or increase the desirability of attacking an enemy position.

Scouting for intelligence could be viewed as separate from resource management. However, scouting is only relevant if the information is used, just like gathering resources is only useful

[4] `http://www.iccup.com/`

if the resources are used for something. As it have been suggested above; intelligence gathering is not exclusively important for resource management, but is useful for attacking and defending as well.

1.2.4 Build Management

Build management deals with using the gathered resources and is important because unused resources do not contribute to winning. In StarCraft each race have several different structures that may construct different units. There is no order of buildings that can be said to always be the best order - it depends on strategy of the opponent, in the same way that there is no choice that is always correct when playing rock-paper-scissors. Much like other elements of StarCraft, optimal build management depends on adept intelligence gathering.

Build management is about what to build when and where. It is related to resource management as gathering resources makes construction possible and construction makes gathering resources possible. Expanding to swiftly may stretch a players defences too thin, leaving them open to an enemy attack. Expanding too late may result in the enemy taking advantage of the opportunity and creating several expansions for themselves and gaining a much larger income rate.

1.2.5 Summary

Resource management is an important aspect of StarCraft as it is a prerequisite for creating armies needed to destroy the enemy and ultimately win the game. It is difficult to separate resource management from other elements of StarCraft completely, as proper resource management depends on information from aspects such as intelligence gathering and build management. Conversely, build management depends heavily on resource management and it would seem like a lot of elements in StarCraft overlap in this manner.

1.3 StarCraft Development Frameworks

This section will describe the StarCraft development frameworks used for the project, along with some examples of their use.

1.3.1 BWAPI

The *Brood War Application Programming Interface*[5] (BWAPI) is an open source C++ framework allowing two-way communication with the RTS game StarCraft, meaning that an agent can observe events in the game and react upon these. BWAPI allows communication through objects and function calls, as opposed to input via human input devices. Blizzard Entertainment, creators of StarCraft, are not affiliated with the developers of BWAPI and strictly speaking, BWAPI is against the end user license agreement. That being said, *The AI and Interactive Digital Entertainment Conference (AIIDE) 2010* have hosted a StarCraft AI competition and have been granted permission by Blizzard Entertainment to do so[6], even though one of the rules of the competition is that BWAPI must be used for all submissions[7]. Though there is no indication that Blizzard Entertainment have actively supported BWAPI, they have in no way

[5] `http://code.google.com/p/bwapi`
[6] `http://eis.ucsc.edu/StarCraftAICompetition#Legal`
[7] `http://eis.ucsc.edu/StarCraftRules/`

acted against it. The framework allows the user to observe and react on virtually any observable information in the game. Any user-available command is made available by the framework along with some extra non-game related features, like drawing lines and figures and writing messages that are only visible on the computer executing the code. Thus, every action a human player can perform in the game can be done by a computer program as well. Every `Unit` in the game–unit being any player selectable object in this context–have a unique pointer and ID along with a range of methods and properties that the developer can use. There's also a `Game` class containing general information about the game, including a set of player references, game time, average fps etc.

The framework provides several events that a program using the framework may subscribe and respond to. As an example, the event `onStart` is raised once at the beginning of the game, and can be used as the main entry point initialising anything used by an agent. `onUnitCreate` is run every time a unit is being constructed, and `onUnitDiscover` is run when a unit is revealed. `onFrame` is run once every frame and is usually where the main logic is executed. It should be noted that the events of BWAPI are synchronous, meaning that the game cannot proceed until the instructions of an event hook have been executed. This generally means that all heavy calculations should be executed in a separate thread to avoid jerkiness in the game. Information on enemy units within fog of war is not available to a player in a game of StarCraft, and for this reason cannot be obtained through BWAPI either, under default circumstances. This information can be enabled using the cheat flag: `CompleteMapInformation`.

A simple example, in which the agent iterates through its own units, identifying worker units and ordering them to collect the nearest minerals, can be seen in Listing 1.1.

```
1  void ExampleAIModule::onStart()
2  {
3          for(std::set<Unit*>::const_iterator i=Broodwar->self()->getUnits().begin();i!=Broodwar->self()->
               getUnits().end();i++)
4          {
5                  if ((*i)->getType().isWorker())
6                  {
7                          Unit* closestMineral=NULL;
8                          for(std::set<Unit*>::iterator m=Broodwar->getMinerals().begin();m!=Broodwar->
                               getMinerals().end();m++)
9                          {
10                           if (closestMineral==NULL || (*i)->getDistance(*m)<(*i)->getDistance(
                                 closestMineral))
11                             closestMineral=*m;
12                         }
13                         if (closestMineral!=NULL)
14                           (*i)->rightClick(closestMineral);
15                 }
16         }
17 }
```

Listing 1.1: C++ Example using BWAPI

Line 3 in Listing 1.1 iterates through all units belonging to the player that the code is executed for, by using `self()->getUnits)` on the game object `Broodwar`. `self()` returns a pointer to the player that BWAPI controls, and `getUnits` is then used to get the units the player owns. In each iteration, the type of the unit is checked in line 5, to see if it is a worker unit, and line 8 to 14 then goes on to finding the mineral closest to the unit and ordering the unit to gather the mineral, by sending a right mouse button click order to it.

1.3.2 BWTA

Broodwar Terrain Analyzer[8] (BWTA) is an add-on for BWAPI that can be used to analyse a given StarCraft map and return various static information about the terrain. When analysed, the map is split into regions, where each region is a polygon containing no obstacles, meaning that units can go directly from one point within the polygon to another within the same polygon. Connections between regions are referred to as chokepoints. A chokepoint is a passable line that separates precisely two distinct regions.

A region contains zero or more base locations, where a base location is defined as a location with nearby resources. A base location will contain a fair amount of mineral fields and often also a vespene gas geyser. BWTA also reveals the set of starting locations. The set of starting locations is a subset of base locations, where a player may be positioned in the beginning of the game. For maps used in competitions, a starting location will contain several mineral fields and at least one vespene gas geyser. The information gained from BWTA can be considered valid strategic information, and can greatly simplify identification of key points on the map:

- Terrain information can be used to identify isolated island locations on the map
- Expansion locations can be used to find valid locations to gather minerals or scout for opponents
- Regions can be used to simplify information on strategic locations to a high level of abstraction and
- Chokepoints is especially useful when identifying places to set up a defensive parameter, and therefore also where the opponent is likely to do so.

Figure 1.1 shows a simple example, where the drawing functionality of BWAPI have been used to show BWTA's representation of the terrain. The visible SCV is positioned where BWTA have identified a chokepoint.

It should be noted that StarCraft contains a map editor that makes it easy for anyone to create a new map. There are very few rules that must be obeyed when creating maps and the result may therefore be unbalanced. Map creators do not need to adhere to the general rules of base locations and may create several separated mineral fields that have no obvious relation to a cluster. The wide range of possibilities mean that it is possible to create maps that BWTA is unable to fully analyse. For the purpose of this report, however, we have focused on standard, balanced maps used for competition which BWTA has no problems analysing.

1.4 Problem statement and scope of project

This technical report deals with the problems arising from managing resources in the game of StarCraft. We define resource management as the ability to sustain a steady economy in order to maximise throughput of military units to win the game–to balance income versus expenses [1]. To achieve this, three factors are considered: resource gathering, expanding base operations and defending units. These three factors are derived from expert knowledge by the authors. Spending resources is not considered. Resource gathering will be treated as the first thing to achieve efficiently, followed by expanding base operations and defending structures. This report deals with only the two first factors.

[8] `http://code.google.com/p/bwta`

Figure 1.1: BWTA example

Resource gathering

Resource gathering deals with the actions of sending out worker units to mineral fields, mining the mineral fields and returning with minerals to the closest resource deposit. From observations, the built-in AI in StarCraft for resource gathering is very unpredictable and doesn't always seem to choose the most logical mineral fields from a human point of view. This report addresses this problem by developing a more efficient algorithm for gathering these resources efficiently. In doing this, no opponents will be considered on the map, only mineral fields will be considered as a resource, a player will only have one resource deposit at any given point, and only the map *Astral Balance* will be considered. The reason only minerals are considered important is that the amount of vespene gas geysers are comparatively small. Scheduling workers for using a refinery is considered a relatively easy task, which we do not expect any gain from addressing.

Expanding base operations

Once a steady income is achieved for a player's starting location, they should consider expanding their operations to a new location, in order to satisfy an increasing demand for the production of military unit. This means that the player needs to select an appropriate place on the map to expand, from a known pool of possible expanding points. It is most convenient for the player to choose a location the enemy is not currently in control of. This means the player must scout areas of interest to find out where it is most appropriate to expand. This report will address scouting techniques and identify important factors to consider when expanding.

CHAPTER 2

RELATED WORKS

This chapter contains a description of related works, mainly conference papers. Related work covered by this chapter is found relevant to be studied due to its attention to StarCraft or its significance to the RTS genre in general. Each section will explain key concepts in depth and finish off by summarising the overall contribution from the authors. Each section will state each individual author's point of view. Lastly, this chapter will try to summarise to which extend the related works will influence the scope of the problem covered by this technical report. In this section, the points of view will that of the authors.

2.1 SORTS: A Human-Level Approach to Real-Time Strategy AI

Wintermute et al. [11] suggest the design and implementation of a real-time strategy agent with focus on human-like behaviour, called SORTS. Their reasoning is that in order to create an immersive game experience on par with a multi-player scenario, it is crucial that an agent mimics a human player. They make an implementation using Soar[1] for the AI architecture and use the open source programming environment for RTS games, *ORTS*[2], for the actual game play.

RTS games differ from other types of games, in that there are multiple simultaneous interacting goals and large amounts of perceptual data. Wintermute et al. goes on to tackle the perceptual system in a way similar to how humans reason from perception, forming two perceptual abstractions: grouping and attention. The abstractions are formed to decrease the amount of perceptual data from the game, and hence decrease the complexity of analysing it. At their core, these techniques resemble that of human behaviour.

2.1.1 Grouping

According to Gestalt psychology [5], humans use grouping to perceive objects of similar size and shape, located within close proximity. This is modelled in SORTS by grouping based on the unit type, unit owner, and the radius to other units. With a radius of 0, all units would

[1] `http://sitemaker.umich.edu/soar/`
[2] `http://skatgame.net/mburo/orts/`

be perceived individually. Grouping is also used to handle unit information such as remaining health for units, by summing the attributes within a group. This is common to RTS games, where a player might control units of the same type as a group.

2.1.2 Attention

Attention is an abstraction used to simulate the human visual system, in particular, the field of vision. In SORTS, its used to reduce the amount of information about the game world that will be handled at a given time, much like a human player during an RTS game: a player can't keep focus of the whole map at a given point in time. At this point, a human player might not know exactly which of their units are where on the map. SORTS implement this by dividing the map into a grid. When a given part of the grid is within focus, SORTS have all information available within the focus area. The unattended parts of the grid only expose information about how many friendly and enemy units are within a cell. This is illustrated in Figure 2.1.

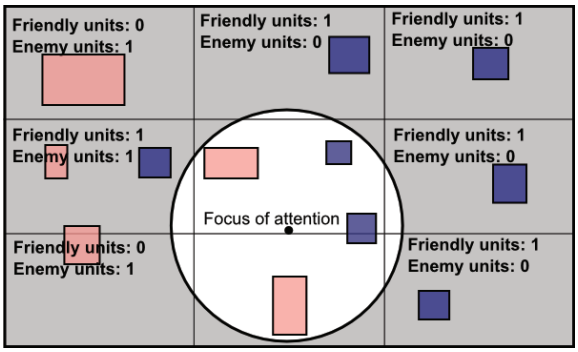


Figure 2.1: Using attention to obtain field of vision.

SORTS implement two other mechanisms to obtain human-like behaviour. A group order mechanism is implemented in order to obtain a behaviour, where a group of units are able to act and attack like a unitary whole. This is something that needs to be implemented partially because of the minimalistic execution system of ORTS. They call this global coordinators, in the sense that a mechanism tells a group of units what will be the best for them, and lets the units execute their orders themselves. They implement two coordinators: one for managing resources and one for managing attacks.

The other mechanism is implemented to mimic the human behaviour of waiting for a given task to finish, before continuing to the next task. They argue that this is somewhat uncommon compared to other AIs that are typically able to execute several subsequent commands. According to the authors, it is one of the main complaints from human players about AIs.

SORTS is tested within three different game scenarios: resource gathering, destroying the opponents base and a complete game. Their AI was entered in the AIIDE 2006 AI competition in all three categories. SORTS won the resource gathering competition, lost the opponent battle due to software bugs, and won in 60% of the games played of a complete game.

2.1.3 Summary

Wintermute el al. implemented the RTS agent SORTS, which took a human-like approach to game AI. The result is a vastly different, yet very capable, AI compared to conventional

approaches. They conclude that the approach helps to make a single player game feel more like it was a multi-player game, thus enhancing the game experience of the player. They achieve human-like behaviour with the help of grouping and attention, which they would like to integrate with learning to develop smarter agents for complete games.

2.2 Mining Replays of Real-Time Strategy Games to Learn Player Strategies

P. Strategies [8] argue that instead of using predefined rules made by developers, to design computer agents, emulation of human behaviours from existing game information should be sought after. In RTS games, this information can be gathered from replays of matches, and agents can then be trained accordingly. Their goal is to evaluate player behaviours, analyse player strategies and train an agent to learn strategies. They claim to use replays from highly skilled professional players collected from GosuGamers[3].

2.2.1 Analysing Replays and Learning

Replay data from StarCraft is stored as a sequence of actions in a binary format. Tools like BWChart Replay Analyser[4] and Lord Martin Replay Browser[5] are able to decode replays into a textual representation of the replay data. The actions in a replay file are a very simple form of replay representation, since the actual button presses made by the players are recorded. This means that there isn't actual information present in the replay files to tell exactly how many units a player has at any given point in time, unless the replay is simulated in the game client. This poses no problem to the authors, as their objective is to map player actions to a player strategy.

P. Strategies represent units and resources as game states. Building new units and making technology research are considered actions between two game states. A game state is evaluated from a set of features that tells how many buildings are in the state, how many military units, technology research done etc. Considering only two features in the game state, the game states and actions can be represented as

$$B_a = \{F_1^a, F_2^a\} \xrightarrow{\alpha} B_b = \{F_1^b, F_2^b\}$$

Where B_a is the game world before taking action α (that is, performing a new strategy), B_b is the game world after and F_i^j is a feature in the corresponding game state. To be able to measure features, the distance between features is calculated. This is normalised across all replays, in order to compare them. This normalising ranks two game states and their corresponding action to give a value of how good the action was.

2.2.2 Results

The data gathered is to be used in an attempt to correctly identify player strategies, in order to counter the strategy. A strategy can be visualised as a decision tree, where each node represents the state of the game world, and each link between nodes represents the action taken to get to the state. An example of one such decision tree can be seen in Figure 2.2, which shows combat

[3] http://www.gosugamers.net/general/
[4] http://bwchart.teamliquid.net/
[5] http://lmrb.net/

of a Terran player against a Zerg player. The thickness of links shows that the player is more likely to initiate and end combat in similar ways, but the core of the combat is often executed differently from one fight to the next.

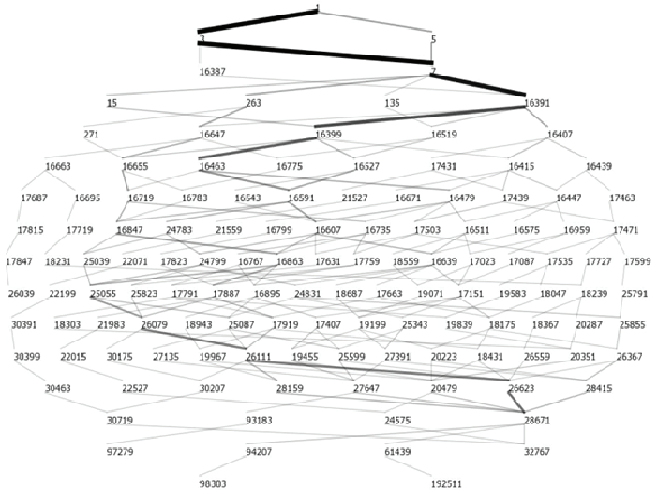


Figure 2.2: Example of a decision tree showing the building sequence of a combat between a Terran and Zerg player. Thicker lines shows more commonly chosen paths in the tree.

For testing, they divide their replays into two groups; one for training data and one for testing prediction accuracy of the opponents' player strategy. Results from their experiments show that they are able to produce an accuracy of between 84.9% and 88.6% depending on the race over the course of 300 rounds of gameplay. The accuracy greatly depends on the number of possible game world states of the race in question, with Terran having $2,234$ possible states, Zerg having 901 possible states, and Protoss only having 388 possible states, where the game world states represents the set of possible buildings.

2.2.3 Summary

Player Strategies have shown that they are able to learn individual player strategies from replay data. They conclude that their system needs further development and research in order to be perfected, namely an interface for executing actual decisions based on data in the system.

2.3 Applying Goal-Driven Autonomy to StarCraft

Weber et al. [10] implemented an agent called EISBOT which is designed to respond to unexpected events which may occur during a game, with respect to a goal. This is called Goal Driven Autonomy. Eisbot consists of four components: A discrepancy detector, an explanation generator, a goal formulator and a goal manager. EISBOT is implemented using the Broodwar API. BWAPI is further explained in Section 1.3.1. As of now, the EISBOT agent only plays the Protoss race.

2.3.1 Discrepancy Detector

The discrepancy detector is responsible for detecting any anomaly during execution of a plan. When the Goal manager component has chosen a plan to follow, it has also provided a set of

expectations, to be encountered during this plan. When these expectations are violated, the discrepancy detector provides a discrepancy to the explanation generator. There are 5 different discrepancies: unit, building, expansion, attack and force. The unit, building and expansion discrepancies are each sent when the discrepancy detector detects a construction that violates the set of expectations. The attack discrepancy is sent when the opponent has attacked, and the force discrepancy is sent when either player has a force advantage. Most of this information is only available to the agent if it is allowed to look at what the opponent has produced, as the agent does not perform active scouting. Using a cheat flag, this information can be accessed through the BWAPI framework.

2.3.2 Explanation Generator

The explanation generator takes as input a discrepancy and uses this to produce an explanation on why an expectation was violated. The explanation generator maps discrepancies to different explanations via simple if-then-else statements. There are seven different explanations: *opponent is teching*, *opponent is building air units*, *opponent is building cloaked units*, *opponent is building detector units*, *opponent is expanding*, *agent has force advantage* and *opponent has force advantage*. An opponent is teching if he pursues more advanced units that need specific technologies. An explanation will only be given to the goal formulator, if it violates the current high-level goal.

2.3.3 Goal Formulator

The goal formulation component is responsible for producing goals to pursue, based on different explanations. Depending on the specific explanation, the goal formulator will give a specific goal. The agent has 4 different goal types it can follow: *execute strategy*, *expand*, *attack* and *retreat*. The *execute* strategy can be a result of the agent discovering a cloakable unit, so it has to alter its strategy to also construct detector units. The goal formulator passes a goal on to the goal manager, which now carries the responsibility of making sure the goal is reached.

2.3.4 Goal Manager

The goal manager is in charge of realising the goal provided by the goal formulator. This manager is able to realise more than one goal at a time, but only one from each of the categories: strategy, economic and tactics [6]. The strategy manager is in charge of the high level strategy, such as which building to build, which units to train and what technology to research. The strategy manager has five high level strategies. As an example, one of these is focusing on air units, while another is concentrating on cloaked units. The income manager is in charge of new expansions, building new resource deposits and training, as well as assigning, new workers. The tactics manager is in charge of the *attack* and *retreat* goals, by either sending all combat units to the opponents base, or sending all combat units to its own base.

2.3.5 Summary

EISBOT wins against the built-in StarCraft agent in 73% of the games, and against humans on a competitive ladder in 43% of the games. It does, however, receive information from the framework which is not visible to a human, for example when it detects force advantages and expansions. This may be perceived as cheating, as the bot knows the entire game state.

2.4 A Data Mining Approach to Strategy Prediction

Weber and Mateas [9] suggest the use of data mining of replays in order to observe different strategies and use these to predict what strategy an opponent is using in a real-time strategy game. A vast number of replays for mining were downloaded from three homepages that are collecting such replays: GosuGamers[6], TeamLiquid[7] and ICCup[8]. The three homepages contain replays from professional and top amateur players, and a total number of 5,493 replays were downloaded, focusing entirely on one versus one game.

Lord Martin Replay Browser[9] was used to convert these replays into game logs, which is a collection of actions along with information on which player the action belongs to, as well the game time at the time of execution. The logs are then converted into feature vectors; one for each player in a game. The feature vector contains the actions and the point in time the actions were executed. Two different strategies can have the same initial build order, but the timing of the actions will make them divert. Each feature vector will be labelled with a strategy type, for example *fast air*. There are six strategy types, and if the strategy cannot be recognised, it is labelled *undefined*.

The feature vectors were exposed to different data mining algorithms using the *Weka toolkit*. Generally speaking, the Weka toolkit was used to make models that are able to predict what an opponents' overall strategy is, as well as predict the next action. As StarCraft is a game of imperfect information, models with artificial *noise* were also constructed. The noise was added, by increasing the time values in the feature vectors, simulating delayed information, as would be the case in a game of imperfect information. Setting the times in a feature vector to zero simulates a base that cannot be scouted. The more noise that was introduced, the less precise the models were.

Some of the models showed that the opponent strategy could be calculated with a precision of 70% just 5 minutes into the game, without any noise in the feature vectors.

2.4.1 Summary

This particular data mining approach for predicting the opponent strategy proved to be feasible. The difference between this approach and others like it is the use of a vast number of replays as basis for the data mining. As the data have been extracted from a vast number of replays, it will also span a variety of different maps, player types and strategies, thereby being able to predict many strategies in many different settings.

2.5 Requirements for Resource Management Game AI

De Jong et. al [3] states that where real-time strategy games can be used in the military to simulate strategic movements in battle, resource management games can be used as training by economics and managers. There is one main difference between a real-time strategy game and a resource management game; resource management games can be infinite and they concentrate on constructing a virtual society and maintaining this, utilizing a limited number of resources. Real-time strategy games are usually built up around finite games concentrating on destroying the opponent, while still utilizing a limited number of resources.

[6] `http://www.gosugamers.net/general/`
[7] `http://www.teamliquid.net/`
[8] `http://www.iccup.com/`
[9] `http://lmrb.net/`

The authors state that a player should be able to play by seven basic principles in order to play a resource management game intelligently; cope with resource dependencies, allocate space effectively, predict future game states, create a sound planning, handle competition, perform multi-objective reasoning, and deal with non-determinism. They go on to mention examples related to a simple resource management game, they have developed for their project. The goal of the game is to build factories and housing, striving for unemployment of zero while having earned as much money as possible within some given time:

Cope with resource dependencies Money can be turned into work places, thereby minimizing unemployment and generating taxes.

Allocate space effectively Placing factories close to the work force, but not so close that they complain about pollution.

Predict future game states Keeping track of the build time associated with actions.

Create a sound planning Making sure that any action is made at the right time and is not taken without being part of a higher sequence of actions to reach a goal.

Handle competition Handling opponents or other kinds of competition in the game

Perform multi-objective reasoning Often a player needs to keep more than one goal in mind at the same time. For example making sure his finances are as they should be, while building factories and making sure that there are workers for these.

Deal with non-determinism Dealing with the inherent non-determinism in the game stemming from unknown or random variables

The authors construct a hybrid AI agent, attempting to encompass all of these principles by combining symbolic and behaviour-based techniques. The hybrid utilises a behaviour based approach, using a neural network (where the input is the entire game state), and a search tree where the nodes are actions and the edges are preconditions and post conditions of that action. The tree is used as a planner, while the neural network is used as a heuristic function for choosing the optimal action in the tree.

Tests of the hybrid AI, performed in the custom game, shows that the hybrid approach outperforms approaches based on symbolic and behaviour-based techniques alone. They emphasise that when using a tree with nodes representing actions, a programmer must write their own heuristics, requiring a great deal of insight into the game. Instead, the hybrid approach creates its own heuristic function by updating values in the neural network by playing games.

2.5.1 Summary

The paper describes seven basic principles that any player has to be able to control, in order to successfully reach the end goal. They design a hybrid AI which is able to obey to the seven principles by combining two techniques, outperforming the techniques in examples where they are used separately. Some of the rules are the same as in the real-time strategy genre, and therefore may apply to some of the principles of resource management in these games. The focus of the paper is the seven principles and not the AI itself.

2.6 Relevance of related works

Wintermute et al. [11] implements the concept of global coordinators which is interesting with respect to mineral gathering. With this approach, it is worth looking into having one global coordinator controlling the units at a higher level–telling the units where to gather resources. A single unit would be in control of itself with respect to getting to the actual mineral.

P. Strategies [8] designs an agent trained with data gathered from a series of StarCraft replays. Their work shows that only limited data are available directly from reading a StarCraft replay. It works well for P. Strategies, since they only record actions taken. However, in a replay it's not possible to see whether or not an action was successful. This doesn't only lead to possible data skew, but it's impossible to tell things such as unit count and resource count at any given point in time during the replay, unless the replay is played in StarCraft.

Weber et al. [10] formulates several components of their agent. The most interesting part, however, is their goal manager, which states three goals to achieve; strategy, income and tactics. This approach acknowledges the splitting of economy–resource management–from other aspects of the game.

Weber and Mateas [9] also use StarCraft replays to identify strategies. They state, that a vast number of replays are needed in order to gain good results. They use the Weka toolkit to expose their data set to different machine learning algorithms.

Steven De Jong et. al [3] notices seven basic principles present in resource management games. Some of these seems to also apply to resource management within StarCraft, namely coping with resource dependencies, predicting future game states and creating a sound planning. At all times in resource management, you're faced with the question whether or not you should spend your resources now on more workers, or instead save to later in the game. This is especially the case when it comes to expanding–should we expand now with our current resources, or should we wait until we have enough resources to also put up defences? These are very important questions, and surely important when developing an agent for resource management in StarCraft.

CHAPTER 3

EFFICIENT MINERAL GATHERING

In StarCraft, a player usually does not take direct control of the workers that have been assigned to gathering resources. When a worker is ordered to gather a resource, the worker will not only mine the resources, but return to the nearest resource deposit as well and from there continue to gather resources until given a different order.

The behaviour of the workers is relatively simple. A worker will go to the resource it was ordered to gather, and if the resource the worker is moving to becomes occupied in the meanwhile, the worker will move to another currently unoccupied resource in the cluster, if such a resource exists. This may lead to the appearance of workers regretting their previous goal and choosing a new goal, in some cases leaving a mineral field that is becoming available shortly for a mineral field that will be occupied shortly. This behaviour can be considered suboptimal, as some travel time is wasted on moving between resources, before the actual mining takes place. The erratic movement may cause the mineral income rate to spike or drop every now and then, when a worker chooses a new path, making it difficult to predict the amount of minerals available at a later point in time. The standard behaviour negatively affects the income rate of a player as well as its reliability.

To avoid this, direct control can be applied to the workers, where the future availability of the resource is considered before moving.

3.1 Gather Minerals

Consider that for each mineral field, m, there exists exactly one queue Q_m containing the workers that will be mining this mineral field next. The queue is ordered such that the worker in front of the queue is the first worker that will be mining the mineral field, the second worker from the front will be mining the mineral when the first have completed and so on.

A mineral field queue is defined as a set $Q_m = \{D_0, D_1, \dots D_n\}$ where each element is a worker in the queue. The first element in Q_m, D_0, is the worker that may mine the mineral first. Each worker is assigned to at most one mineral field queue at a time. When a worker have finished mining, it is removed from the queue, as it merely has to return its cargo to the resource deposit. When the worker is ready for more work, e.g. when the worker has delivered minerals to the resource deposit, they are again added to a mineral queue.

By using mineral field queues, it is possible to determine which mineral field a worker should

use in order to minimise the time it takes for the worker to return to the resource deposit with an amount of resources. A factor in this is the time it takes for a mineral field to become available, meaning the time until all workers in Q_m have finished.

$$Work\,(Q_m) = \sum_{i=0}^{n} Work\,(D_i)\,|D_i \in Q_m \tag{3.1}$$

$Work(Q_m)$ in Equation 3.1 is the time it takes for a mineral field, m, to become available, meaning the amount of time before the last worker (and therefore all preceding workers) in the queue is done using the mineral. $Work\,(D_i)$ is the time that a worker, D_i, will need to occupy the mineral field. It is the time from the mineral field is available to D_i to the time D_i is done using the mineral field. Therefore $Work\,(Q)$ is the sum of $Work\,(D)$ for all workers in the mineral field queue.

$$Work\,(D_i) = max\left[0, Travel_{\rightarrow m}\,(D_i) - \sum_{j=0}^{i-1} Work\,(D_j)\right] + T|D_i, D_j \in Q_m \tag{3.2}$$

$Work\,(D_i)$ is the time needed for a particular worker D_i to finish using the mineral m, starting at the time the mineral field is available and ending when the worker no longer occupy the mineral field.

Equation 3.2 shows that the time it takes for a particular worker, D_i, to complete its work, depends on both the time it takes for the worker to move to the mineral field, $Travel_{\rightarrow m}(D_i)$, the constant time it takes to mine the mineral, T, and the time it takes for the workers in the front of the queue, $D_0 \ldots D_{i-1}$, to complete their work.

As the work of a worker is defined as the time from the mineral is available to the time the mineral is no longer in use by the worker, the actual travel time of the worker should not be considered when calculating $Work\,(D_i)$. The only part of the travel time that should be considered is the part of the travel that occurs while the mineral field is unoccupied. In other words, if a mineral field is in use by a worker, then other workers may move closer to the mineral field, without adding extra time to their work, as they may move at the same time as the mineral is in use.

Consider the following part of Equation 3.2: $Travel_{\rightarrow m}\,(D_i) - \sum_{j=0}^{i-1} Work\,(D_j)$. Here the work of those workers ahead of D_i is subtracted from D_i's travel time, such that only the time of the part of the travel occurring while the mineral field is available to D_i is added. If the sum of the work of those workers ahead of D_i is greater than the actual travel time of D_i, then D_i will be waiting for the mineral to become available and this part of Equation 3.2 will become negative. As no time is gained from waiting in line, the function guards against this case by enforcing a minimum value of zero.

Also note that the constant time to mine a mineral, T, is also added to the work of a worker.

$Work(D_i)$ is recursive as the work of D_i depends on the work of those workers ahead of D_i. The recursive part of the function will always terminate as the last worker, D_0, will always be reached and the sum of workers in front of D_0 is always zero as that is the worker that is first in line.

Consider an example with one mineral field, m, and two worker units, D_0 and D_1. Both workers are queued for m and D_1 is the worker in front of the queue, meaning that D_1 is the first that may mine the mineral field. This scenario can be viewed on 3.1.

In this example the travel time of D_1 is 71 time units, the travel time of D_0 is 104 time units and the constant mining time is 80 time units. In order to calculate the time required before the mineral field, m is available, the work of D_1 and D_0 is first calculated.

Figure 3.1: Illustration of a mining scenario.

$$\begin{aligned} Work\left(D_1\right) &= max\left[0, 71\right] + 80 = 151 \\ Work\left(D_0\right) &= max\left[0, 104 - 151\right] + 80 = 80 \\ Work\left(Q_m\right) &= 151 + 80 = 231 \end{aligned} \tag{3.3}$$

In Equation 3.3 the work of the two workers is calculated using Equation 3.2 as well as the total work of Q_m, using Equation 3.1.

The calculations state that worker D_1, being the first in line, will be finished after 151 time units. This is obviously true as moving to the mineral requires 71 time units and mining it requires 80 time units. The worker, D_0, is second in line. As it has been established that D_1 requires 151 time units to be finished, D_0 is able to move completely to the mineral field, as the travel time is 71, which is less than the time until the mineral field becomes available. The mining will not be possible until the mineral field has become available, meaning it cannot be done in parallel with D_1's mining, and the constant mining time, T, must be added to the work of D_0.

The total time until the mineral field is available is 231, which is exactly the time before both workers has finished mining.

By using these equations it is possible to calculate the round-trip time a worker, D, will have if added to a particular queue, Q_m, assuming the worker is not in a queue. The round-trip time is the time is takes for a worker to move to a mineral, m, potentially wait in line, mine the mineral and return with some of the resource. A complete round-trip causes the worker to collect an amount of minerals and become ready for a new round-trip.

The time it takes a worker, D, to move to a mineral, m, mine the mineral and return with the cargo (round-trip) is $R(D, m)$ defined as can be seen in Equation 3.4.

$$R\left(D, m\right) = Travel_{\rightarrow m}\left(D\right) + max\left[0, Work\left(Q\right) - Travel_{\rightarrow m}\left(D\right)\right] + T + Travel_{\leftarrow m}\left(D\right) \tag{3.4}$$

Just like Equation 3.2, $Travel_{\rightarrow m}\left(D\right)$ is the time required for the worker, D, to travel to the mineral field, m. Assuming that a worker will always be added last in line, the time for all preceding workers in the queue will be the sum of the work for all workers currently in the queue, $Work\left(Q_m\right)$. The time worker D will stand in line is the total work time of the queue $Work\left(Q_m\right)$

subtracted by the time that has already been spent on travelling. The constant mining time T is also added along with the travel time to return to the resource deposit, $Travel_{\leftarrow m}(D)$.

Minimalizing the round-trip time for a worker will cause the worker to always pick the mineral field that will allow for the fastest delivery of an amount of minerals in a greedy fashion (Mind that the method doesn't guarantee the highest possible income rate given a set of workers, see Section 5.1. Furthermore, by using these rules for each worker, the actions of the workers can be simulated to calculate the amount of resources that will be gathered over time.

$$R_{min}(D) = R(D, m_i) \,|m_i \in M, \forall m \in M : R(D, m) \geq R(D, m_i) \tag{3.5}$$

$R_{min}(D)$ is the minimum round-trip time for a worker D, given a set of minerals, M. Equation 3.5 should be read as: The minimum round-trip time for D is the round-trip time for D, m_i, where m_i is a mineral field in M and for all m in M the round-trip time of D, m is greater than or equal to the round-trip time of D, m_i. Selecting the mineral m_i for D means that D will have the fastest round-trip time and will deliver an amount of resources as fast as possible.

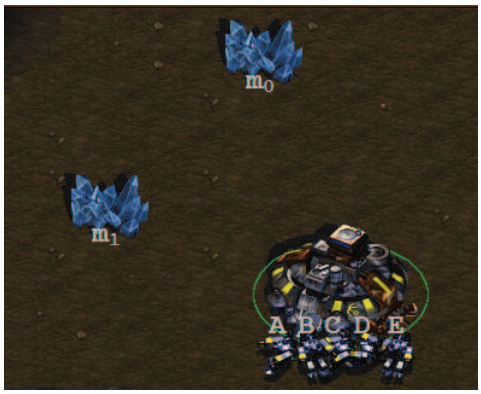


Figure 3.2: Illustration of a relatively large mining scenario.

Consider the example seen on Figure 3.2. In this particular scenario there are five workers, one resource deposit and two mineral fields. The workers are identified as A, B, C, D and E respectively. The mineral fields are m_0 and m_1. In order to visualise the actions of the workers, a trace can be viewed in Table 3.1.

The traces contain the time of each action, the worker affected, the action performed by the worker, the travel time needed for an action (in parenthesis), the state of the two mineral queues and the current amount of minerals. Two numbers are important during mineral collection; the travel time from point to point and the mining time of a mineral. The mining time is a constant, measured as 80 time units. The travel times from and to every point used in the trace will not be listed, instead the times will be stated in the rows, below the actions, to which they belong. The trace describes 3 workers, each moving to one of two minerals, mining the mineral and returning to the resource deposit. As the workers return to the resource deposit they are assigned to a new mineral. The actions of the three workers constitute a round-trip for each of them. On Figure 3.2 it should be noticeable that both the workers A and B have a shorter path to the mineral m_1 than to the mineral m_0. However, on the second row of Table 3.1, it may be observed that the worker B, moves to the mineral m_0. The reason for this is that the worker, A, is already in the queue for m_1 and the time B would consume if placed in Q_{m_1}, would be greater than

Time	Worker	Action	Queues	Minerals
0	A	Move to m_1 (110)	$Q_{m_0} = \emptyset$ $Q_{m_1} = \{A_0[work \leftarrow 190]\}$	0
0	B	Move to m_0 (126)	$Q_{m_0} = \{B_0[work \leftarrow 206]\}$ $Q_{m_1} = \{A_0[work \leftarrow 190]\}$	0
0	C	Move to m_1 (126)	$Q_{m_0} = \{B_0[work \leftarrow 206]\}$ $Q_{m_1} = \{A_0[work \leftarrow 190], C_1[work \leftarrow 80]\}$	0
190	A	Move from m_1 (90)	$Q_{m_0} = \{B_0[work \leftarrow 16]\}$ $Q_{m_1} = \{C_0[work \leftarrow 80]\}$	0
206	B	Move from m_0 (108)	$Q_{m_0} = \emptyset$ $Q_{m_1} = \{C_0[work \leftarrow 64]\}$	0
270	C	Move from m_1 (90)	$Q_{m_0} = \emptyset$ $Q_{m_1} = \emptyset$	0
280	A	Deliver & move to m_1 (90)	$Q_{m_0} = \emptyset$ $Q_{m_1} = \{A_0[work \leftarrow 170]\}$	8
314	B	Deliver & move to m_0 (108)	$Q_{m_0} = \{B_0[work \leftarrow 188]\}$ $Q_{m_1} = \{A_0[work \leftarrow 136]\}$	16
360	C	Deliver & move to m_1 (90)	$Q_{m_0} = \{B_0[work \leftarrow 142]\}$ $Q_{m_1} = \{A_0[work \leftarrow 90], C_1[work \leftarrow 80]\}$	24

Table 3.1: A sample trace for the scenario on Figure 3.2

choosing m_0 as the mineral, due to the time spent standing in line for the mineral.

$$\begin{aligned} R(B, m_0) &= 126 + max[0, 0 - 126] + 80 + 90 = 314 \\ R(B, m_1) &= 118 + max[0, 204 - 118] + 80 + 90 = 374 \end{aligned} \tag{3.6}$$

Equation 3.6 shows the calculations made before choosing an action for worker B in the second row of the table, using the function R from Equation 3.4. The set of minerals in the scenario is $M = \{m_0, m_1\}$, so $R_{min}(B)$ is $R(B, m_0)$, as it may be verified by Equation 3.5, meaning that having B move to m_0 will give B the lowest round-trip time possible. The trace shown on Table 3.1 is rather straight forward, as the workers choose the same mineral field in their second run (chosen when dropping minerals off), as they did in the first run. For a larger scenario with more workers and possibly more mineral fields, it is not uncommon for workers to change mineral fields between runs, as one particular mineral may not necessarily keep providing the lowest round-trip time. Implementation and usage of the algorithm is covered in Section 3.3.

3.2 An Incremental Learning Method for Travel Time

The mining algorithm explained in the previous section depends heavily on a function providing the time it takes to go from the resource deposit to a mineral field and back. Movement in StarCraft works by the path finding algorithm finding a path after which the unit will move through this path to the destination. The chosen path is deterministic given the same starting position, destination and obstacles, though Blizzard Entertainment have not released the details of the algorithm. Units do not move at a constant speed. A unit, given a move order while standing still, will first accelerate towards a maximum speed and decelerate before reaching its destination.

As the travel time will only be used for deciding the travel time between a resource deposit and the mineral fields the amount of starting positions and destinations are relatively few. For this reason the time needed to travel from one position to another is recorded and the recordings are then used. Every time a worker unit move from the resource deposit to a mineral field the time of the travel is recorded (in number of frames). Information on the travel is saved, including the starting position, the destination and the time.

The idea is that whenever another worker unit is moving in a similar path to the one recorded, the time is already known. Given enough data, all possible paths a worker may use when mining minerals, is known for a specific map. So for known source/destination pairs this method will be very accurate. Though travel time for unknown distances will have to either be approximated or remain unknown.

The information has been gathered by using the mining algorithm and defining the travel function as a function that returns a previously calculated value. If no data matching the source and destination existed the function would return a low value to convince the unit to take the path and thereby gather previously unknown data. This method has been used on a particular starting location for a specific map and a lot of data have been gathered. The behaviour is very obviously affected as more travel information is gathered. Instead of returning a low value, when no travel time has been recorded, the distance from source to destination could be calculated and a travel time could be estimated, but then it will take longer to learn a travel time.

3.3 Experiments and Efficiency of the Resource Gathering Algorithm

The mining algorithm as described in Section 3.1 should improve the overall income in comparison to a standard approach of mining minerals. In addition to this, the algorithm is expected to provide a stable and predictable income rate. In this section an approach based on the algorithm will be presented, as well as the standard approach. The outcome of each approach is measured and the results are presented, compared and discussed.

3.3.1 StarCraft AI Approach

The most usual and simplest approach for a player to make their workers gather minerals is by selecting an amount of workers and right clicking any mineral. When the player have right clicked any mineral the worker will move to that mineral and try to mine it, if the mineral is occupied it goes to another mineral close by and try to mine this. If no minerals is unoccupied in the mineral cluster the worker go to a field and wait until it is available. When the worker have successfully mined the resources it will return to the resource deposit with the mined minerals. After having dropped off the minerals it will return to the mineral it have lastly mined to mine this again. This process is automated by StarCraft.

The efficiency of the mining algorithm described in Section 3.1 will be compared to a mining agent acting the way that have just been described. As a computer agent is able to make immediate actions, and a human likely has some latency, a computer agent is also used for the StarCraft AI approach. Using two computer agents makes them more even in terms of speed of actions, and the comparison therefore relates only to the method used. The StarCraft AI mining agent will order the units to mine the closest mineral, the first to arrive will mine this mineral, the others will automatically find another mineral field if more are available. The agent will construct new workers as soon as possible with respect to the amount of minerals and supply.

The tenth worker will on its creation construct a supply depot next to the resource deposit to allow for the construction of an additional 8 workers, and after constructing this supply depot the worker will start mining.

3.3.2 Mining Algorithm Implementation

The mining algorithm has been implemented using two worker events. Whenever a worker is done mining, an event is fired, as well as when a worker delivers minerals to the resource deposit. At the beginning of the game each worker is assigned to the mineral field giving the lowest round-trip time. Every time a worker has finished mining an event is called and the worker is removed from the queue that they previously belonged to. When a worker returns to the resource deposit with a mineral another event is called and again the mineral field with the lowest round-trip time for the worker is assigned. Like the StarCraft AI approach, workers are created as soon as possible and a single supply depot is constructed by the tenth worker. The lowest round-trip time requires the travel time from point to point. This travel time is recorded as described in Section 3.2.

3.3.3 Test Setting

All tests will be carried out on the map, *Astral Balance*. Any map could have been used but *Astral Balance* has been chosen due to being the first listed standard map for two players. Though the difference between starting positions is usually considered negligible, in order to ensure fairness, tests will always be performed in the upper right corner. The starting positions lie next to a mineral cluster consisting of eight mineral field each holding 1500 minerals, as well as a single vespene gas geyser.

The time will be measured in number of frames, as this is a time measurement available directly through the BWAPI framework. The agents will per 500 frames output their current amount of minerals. This makes it easier to compare the efficiency of the two approaches, with respect to the actual amount of minerals gathered in the game, the rate for each 500th frames and the differences in this rate. From these results the gain from using one method over the other should be clear. Tests will be run several times, and averages of the outcome of these will be used as a basis for the results and analysis of the test.

3.3.4 Results

The test were carried out as described in Section 3.3.3. A test with the described setup ran 10 times where it collected data at the same time, but the mining algorithm as described in Section 3.1 ran one time prior to the data collection, to collect data on the travel time. The setup ran for 8000 frames and per 500 frames the number of minerals where printed. From these data the rate could be calculated.

The following three graphs show the difference in the two mining approaches first with respect to the amount of mined minerals, secondly with respect to the rate at which they collect minerals per 500 frames and lastly the standard deviation is shown for the two approaches.

Figure 3.3 shows the amount of mined minerals, with the number of minerals on the Y-axis, and the number of frames on the X-axis. Both algorithms increase the amount of minerals over time, but as it is visible, the line of the proposed mining algorithm is smoother than that of the standard StarCraft AI approach. There are some fluctuations in the amount of minerals for the standard StarCraft AI approach. These fluctuations represent an unavailability of resources for some point in time, followed by a burst that seems to catch up to the expectations. At no

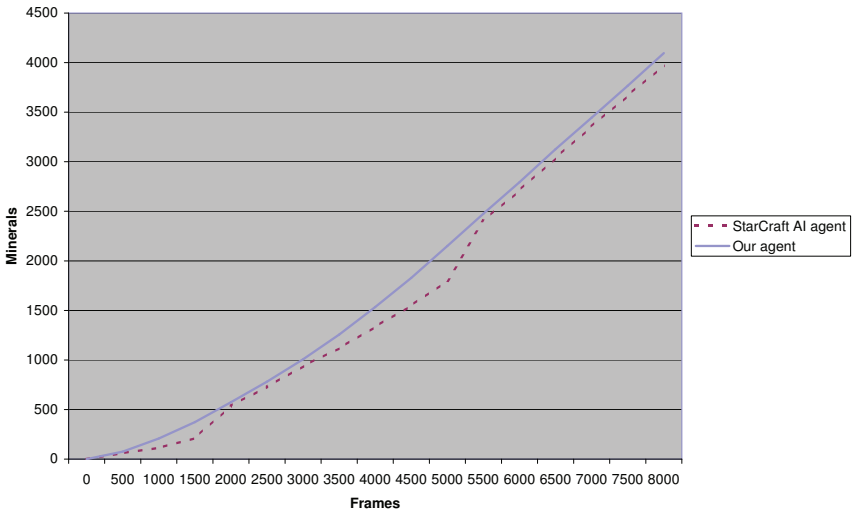


Figure 3.3: Amount of minerals gathered.

point in time, within the measured limits, have the standard StarCraft AI gathered more than the proposed algorithm. At the last measured $(8,000)$ the proposed mining approach had in average mined $4,096$ minerals, whereas the StarCraft AI approach had mined 3971 minerals. The proposed mining approach mined roughly 3% more minerals than the StarCraft AI approach.

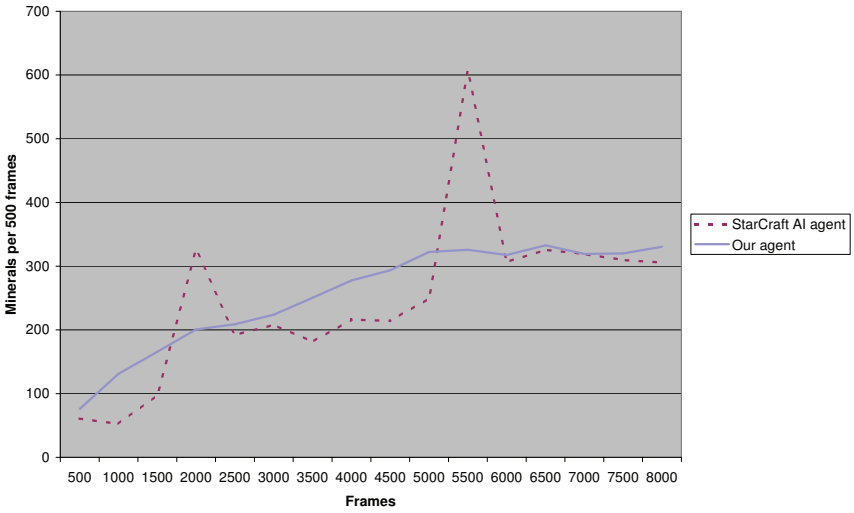


Figure 3.4: Rate of mineral gathering.

Figure 3.4 shows the rate with which the two approaches collect minerals with a time span of 500 frames. With the number of minerals mined per 500 frames on the Y-axis, and the number of frames on the X-axis. Both approaches roughly increase their rate over time, as more workers are created. However, the rate of the standard StarCraft AI approach has some relatively large changes in rate over the course of the $8,000$ frames, a sudden drop in the rate followed by a large increase. This was evident in Figure 3.3 and more so in Figure 3.4. It is also interesting to note that the slope of both graphs seems to decrease over time. This indicates that the gain from creating a worker may depend on how many workers are already working the mineral fields. One

would expect this to be the case as the more workers are present for the set of mineral fields; the more time is wasted on standing in line for an unavailable mineral. Over time the graph is likely to reach a horizontal asymptote as the amount of workers reach a point where the mineral fields are completely occupied at all times, meaning that adding more would not yield a larger rate of minerals.

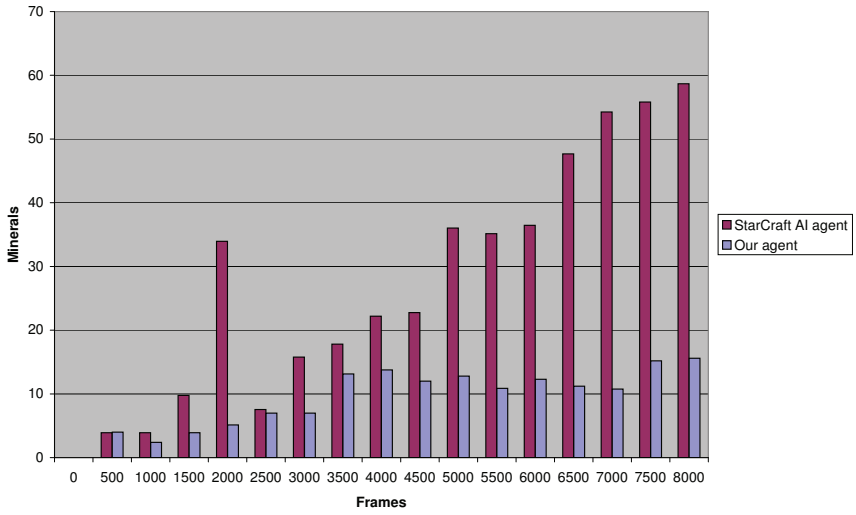


Figure 3.5: The standard deviation of the collected data.

Figure 3.5 shows the standard deviation of the two mining approaches. The Y-Axis is the standard deviation in number of minerals. The X-axis represents the time through number of frames. At each interval there are two columns, the left is the standard StarCraft AI approach and the right is the results for the proposed algorithm. At the early points the deviation is relatively small, however, as time progresses the deviation seems to grow. The deviation of the standard StarCraft AI approach clearly grows at a much higher rate than the proposed algorithm. As this is the case, it seems that the proposed algorithm is the more reliable of the two. This is important when aiming to predict the amount of minerals available at a later point.

3.3.5 Conclusion

From the previous result two things can be concluded. The first is that the proposed approach seems to collects slightly more minerals than the StarCraft AI approach. The second is that the proposed model is the more predictable of the two. This is true both in terms of stability of mineral income rate and the confidence that may be attributed to this rate.

3.4 Summary

In this section, a mining algorithm has been suggested as an alternative to the standard mining mechanism. The algorithm works by having each worker choose the mineral field that will allow the fastest mineral delivery. Each time a worker returns with an amount of minerals this is calculated and the worker is assigned to the proper mineral. In this respect the mechanism should be considered greedy for each worker, meaning that it gives no guarantee to being the optimal solution over time.

In order to work, the algorithm needs access to the travel time from point to point of working units. This has been implemented by measuring the travel times of worker units beforehand and using this data for the function afterwards.

The results from Section 3.3.4 on page 27 show that the proposed approach is very predictable as the curves are smoother and consistent from time to time. Observation shows that the rate increases with time and the number of workers. This is useful when calculating expected income for an amount of workers over time, as well as the potential gain from creating an expansion. Furthermore the proposed approach increases the income rate.

CHAPTER 4

EXPANDING: SCANNING AND SCOUTING TECHNIQUES

This chapter will propose a set of variables that may be important when a human player is expanding their base operations–such as the enemy position, resources on the map and race of the player–and a decision model for this will be discussed. A tool will be discussed for the analysis of StarCraft replay matches, used to measure the influence of the proposed variables. The chapter will conclude with a discussion of the retrieved data, and the importance of variables with respect to expanding base operations.

4.1 Variables for Choosing Expansion Bases

Section 1.2.2 argues that the concept of expansions is a key component in winning a game of StarCraft and that gaining knowledge about your opponents is an important factor in this. Expansions that can mine minerals uninterrupted by enemy attacks is far more valuable than one that is destroyed shortly after its creation. Therefore it is useful to consider all the possible expansions before making a decision. When a human player considers which expansion site to expand to, a lot of factors are involved. As some base locations contain more resources than others, some of them can be preferable to others. However, it may not be so easy to map properties such as resources of a base location to its desirability, as this does not consider the theory of minds.

A player can choose a base location that is not as immediately attractive as another, if this means the opponent is less likely to discover the expansion. The problem of the theory of minds–and two agents that may both knowingly apply this to the other player–is the recursion that occurs. At the first level we say that a player has an idea of what the opponent thinks, at the second level a player has an idea of what the opponent's idea of the player is and so forth. It should be kept in mind that a player may not always choose the most obvious site for an expansion for any reason, which is an always-present uncertainty.

In StarCraft, knowledge of an area requires a presence of units which the player controls. When units exit an area the area is covered by fog of war, meaning that from the perspective of the player, the state of that area does not change. It is not possible to observe resources being removed, units being moved away from or through the area, new buildings being constructed

etc. as long as fog of war is present. In order to gain information on these areas, the player can scout the area in a range of different ways, depending on the race of the player. Scanning an area where there is no enemy present gives the information that there is no enemy presence in that particular location. However, scanning an area where the enemy is present, does not only provide the information of enemy presence in the scanned location; as expansions are costly, it may be possible to determine an estimate of the total number of expansions, meaning that finding an expansion will tell the player a lot about the likelihood of other expansion sites containing an enemy expansion.

By creating a model of how good an expansion site is for a given player, a player may determine which locations are preferable for expansions. Apart from this, a model will also make it possible for a player to determine how good a given location is for an opponent player. This information can be used as the guideline for which locations should be scouted first.

In order to mimic this reasoning for a computerised agent, it is necessary to formalise and identify the variables that a human player considers important when expanding. Expert knowledge and personal belief have been used in identifying variables that may be important and as an effect some of these variables may be unimportant while others are essential. This section will cover these variables superficially and the most promising of them will be described in the experiments section. The following is a list of the variables:

Variables relative to the location itself

Amount of gas The amount of gas available on the location.

Number of minerals The amount of minerals available on the location.

Number of chokepoints The amount of chokepoints, as defined in Section 1.3.2, in the area. In general, the fewer chokepoints in an area, the less entrance points exists for enemy ground units.

Presence in shortest path between enemy and own base A key point to surviving and making surprise attacks on the enemy, is to remain hidden for as long as possible. If an expansion is placed in an area that will need to be passed through on the way between two opposing base locations, there is a chance that the expansion will be discovered by chance. On the other hand, the area may also be a key strategic position to put up main defences, since it can be located on the only route between two opposing base locations.

Distances from bases to the area Various distances from both own and enemy bases to the area. The lower the distance from the nearest enemy base, the faster the opponent can bring reinforcements to invading forces. In the same way, the lower the distance from the nearest ally base, the faster reinforcements can be brought to the defenders. The distances can be split up in four categories:

Own/enemy flying distance The flying distance from the nearest enemy/ally base to the area.

Own/enemy ground distance The ground distance from the nearest enemy/ally base to the area, taking into consideration the environmental obstacles that may be on the way.

Open sides on the base In StarCraft, every map has a fixed size on both axes, with an impenetrable wall that no units may pass. In effect, an area close to one or more of

these axes has fewer sides to defend from enemy forces much like what is the case for chokepoints, but in this case also counting enemy flying units.

Variables relative to the map and the general status of the match

Number of own/enemy expansions The number of enemy expansions vs. the number of own expansions may tell something about the status of the match. Roughly speaking, the more expansions a player has, the higher his income and hence the better his position in battle.

Number of possible expansions Every map in StarCraft is different, and the amount of possible locations to set up a base for resource gathering varies from one map to another. This number may directly affect the importance of other factors like the number of own/enemy expansions, since 2 expansions in a map with 16 expansion locations might be less important than the same in a map with 4 expansion locations.

Highest presence in shortest path between enemy and own base In order to clarify the importance of the presence in shortest path between enemy and own base, as described earlier, a normalization factor is needed. A choice for such a variable may me the *highest* presence possible between the choices of expansions.

Game time The amount of time that has passed since a match began may be important for several reasons. For example, since defensive structures takes time and resources to build and maintain, an early expansion may increase the income of a player, but will leave both the starter base and the expansion vulnerable until defences has been set up.

Best number of gas/minerals Variables describing the highest number of gas/minerals possible for all choices of expansions. Both may be used as normalisation factors for other variables.

Own/Enemy race Each race in StarCraft has its own strengths and weaknesses, so the presence of different enemy and ally races may proof a factor when deciding where to expand.

4.2 A Decision Model for Base Expansion

Given each of the variables mentioned in Section 4.1, their importance needs to be identified and utilised. Since some of the variables may affect each other, a structure such as Bayesian Network that can represent this correlation, seems like a viable choice of representation. With such a structure, the choices human players make, when deciding where and when to expand can be boiled down to choosing the action with the highest occurrence, given the variables that are present when an expansion needs to be made. This network can then be used by a potential computer player to make decisions on where to expand at a given time in the game. The same network may be used to estimate where the opponents have expanded at a given point, if they were to use the same approach. Using the logic presented in Section 4.1, this information may then in turn be used to know where to scan for information on the opponent.

4.3 Using Replays to Measure Influence of Variables

In order to determine the importance of the variables in Section 4.1, states of the variables, given lots of different game states, are needed. Since information on the enemy is limited in

the start of the game, due to the fog of war, a generalised Bayesian Network seeking to match all players–regardless of their playing style–needs to be constructed, in order to know where to search for the opponent.

Due to the shear amount of data needed, gathering data from live battles made specifically for the sake of gathering data for this project is infeasible. Luckily, due to the popularity and age of StarCraft, a lot of replays has been made during the years, and have been collected on certain websites, like the ones mentioned in Section 2.4.

Using these replays, the actions made by thousands of players on a great range of different StarCraft maps, can be gathered for analysis. Unfortunately, due to the nature of the replay format used in StarCraft, the variables that have been identified in Section 4.1 cannot be collected directly from the replay file [8]. In order to retrieve the variables, a simulation of the game, from which the replay was made, needs to be analysed. Since StarCraft currently is closed source, the only way to be sure the simulation is correct, is to use the original replay launcher from the game. Using BWAPI along with BWTA (See Section 1.3) to hack and communicate with StarCraft, the variables can be retrieved from the replays and saved in a format that can later be used for analysing the data.

4.3.1 Implementation of Replay Analyser

BWAPI is used for collecting data from replays, among other things, by utilising terrain analysis through BWTA. The purpose of the data is to locate positions that an enemy is likely to occupy as well as identify locations that are well suited for expansions. This is done by tracking every time that a player in a replay creates a resource deposit in a new region. When this happens the properties of the region is saved as an entry tagged *BUILD*, indicating that a player have chosen to build an expansion on this region. Every unoccupied region at the point in time where the player constructed a resource deposit is considered a potential location that was not used. For each of these locations an entry tagged *NOT BUILD* is created, with the properties of the region. The purpose of this act is to have a representation of which properties indicate a good place to expand and which properties means that a location is not so good. The properties used in this analysis is described in Section 4.1.

It should be noted that the starting positions of each player is also considered to be an expansion, in order to ensure that the likelihood of an enemy occupying the location is not underrepresented. For each set of entries written for a particular time, only one will be a *BUILD* entry, the rest will be *NOT BUILD*. In most cases this will create an overrepresentation of *NOT BUILD* entries, but as long as the data is used as a guide for where to build and not whether or not to build, we do not foresee this to be a problem.

Some properties depend on the amount and locations of existing expansions. In order to know the locations of the expansions of each player, ownership has been defined. Ownership is gained by placing a resource deposit on a region that is owned by no one. Ownership is lost when the last building, able to produce units on the region, is destroyed. Ownership is defined because controlling a region in StarCraft does not necessarily mean controlling the region for the entire length of a game. A list of ownerships is maintained and used whenever a property requires the information.

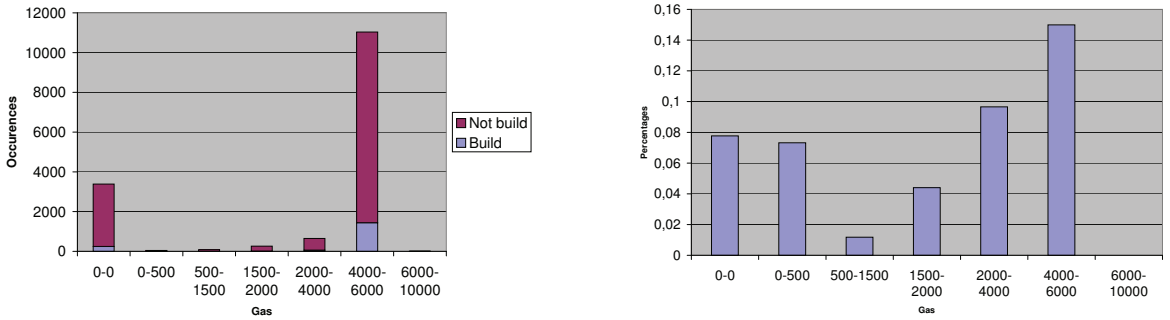
4.4 Analysis of Data

In order to perform an analysis on the collected data from the replay parser, the data mining tool *Waikato Environment for Knowledge Analysis*[1], Weka, is used. Weka is a tool that enables most common machine learning and data processing algorithms to be run on data sets [12]. The primary purpose for using Weka on our data set is to gain knowledge on the proposed variables in Section 4.1, in order to know which variables are important when deciding where to expand base operations to a new place on the map. Weka is also able to produce a Bayesian Network based on the data set, which matches our goal of Section 4.2.

Using a small data set based on 250 replays, with 15,468 instanced based on 19 attributes, some tendencies are observed. Other than the 18 variables already mentioned in Section 4.1, the 19th variable is indicating whether or not an expansion is built in a region, whenever they expand. Entries have been captured in the way described in Section 4.3.1, the entries are distributed between races as follows: 201 Terran entries, 161 Zerg entries and 194 Protoss entries. The graphs in this section that are bicolored contains the entries for both *BUILD* and *NOT BUILD*. As explained in Section 4.3.1, the number of *NOT BUILD* will in most cases be larger than the number of *BUILD* entries, but the ratio of $BUILD/NOTBUILD$ can be used as a comparison between properties, as it is the ratio describing the actual amount of expansions, to a location with some property, compared to the number of times such an expansion has been possible. This section will only cover variables with interesting results. As an example, walking distance has been skipped, as the general tendencies seem to be the same as that of the flying distance variable.

Amount of Gas

The variable *amount of gas* determines the total amount of vespene gas in the region. Vespene gas is used mostly for nonstandard units and upgrades, while minerals are used for constructing everything. Given that this is the case, the requirement for either resource may not be the same.



(a) Gas relation.

(b) Gas factor.

Figure 4.1: (a) Relation between building and not building on locations with various amounts of gas. (b) Percentages of building versus not building on locations with various amounts of gas.

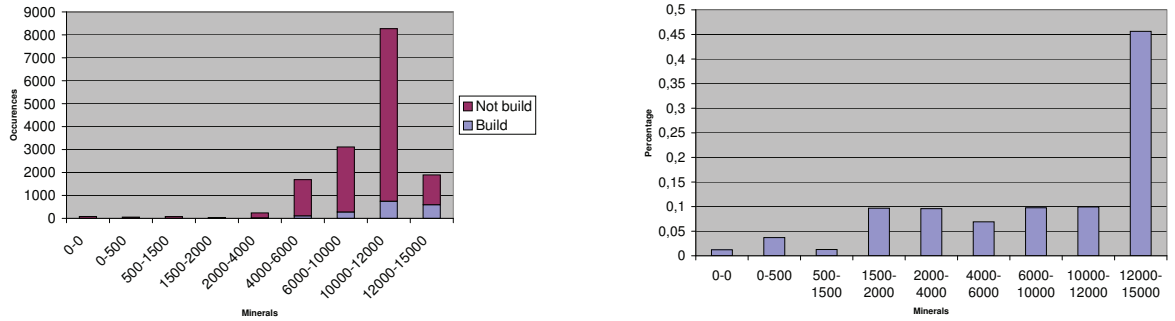
As it can be seen in Figure 4.1a and Figure 4.1b, the likelihood of building at a location with no vespene gas is somewhat high. The likely cause is that these instances represents cases where the region does not and have not, at any point in time, contained vespene gas. In these cases, the players interest have been the minerals at the location and the player have disregarded the

[1] `http://www.cs.waikato.ac.nz/ml/weka/`

lack of vespene gas at the location. In the case where there is an amount of vespene gas higher than zero in the region, players seem most interested in the locations where there is a rather high amount of vespene gas. It seems reasonable that players will either disregard vespene gas completely, if that was not the purpose of the expansion, or choose a location with at least a decent amount of vespene gas, to justify building an expansion at all.

Amount of Minerals

The variable *number of minerals* determines the number of minerals in a region. Minerals are the core resource, used for constructing everything. In most maps there will not be any region with a vespene gas geyser and no mineral fields. Either there are only mineral fields, mineral fields and one or more vespene gas geysers or no resources at all.



(a) Mineral relation.

(b) Mineral factor.

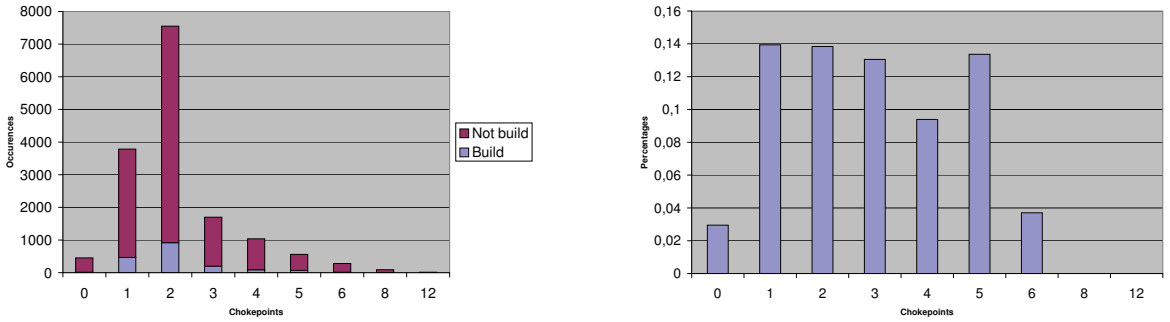
Figure 4.2: (a) Relation between building and not building on locations with various amounts of minerals. (b) Percentages of building versus not building on locations with various amounts of minerals.

As it can be seen in Figure 4.2a and Figure 4.2b, players seems to primarily build expansions at locations where a lot of minerals are present. This is perhaps not very surprising as it is expected that players will expand to retrieve resources and the primary resource is minerals. Apart from that, it is not uncommon that starting positions have the maximum amount of resources in a map.

Number of Chokepoints

The variable *number of chokepoints* determines the number of entrances to a region from different regions. The more chokepoints a region contains, the more points must be considered when defending against ground units. A location with zero chokepoints is referred to as an island, because it is inaccessible to anything but aerial units.

The data from Figure 4.3a and Figure 4.3b shows that islands are sometimes used for expansions, however, they do not seem as attractive as other locations with a low amount of chokepoints. It should be considered that the starting area of players is, in most cases, not islands. The reason for this is to allow early game ground units to be effective. Constructing a building on an island requires a worker to be transported to the island. It is true for all StarCraft races, that the technology to do this is not available right away, so island expansions are most likely mid- or late game expansions. There does not seem to be much distinction between regions with two, three or four chokepoints, which would indicate that players does not distinguish between these, or at least not in the same degree that islands are considered distinct.



(a) Chokepoint relation.

(b) Chokepoint factor.

Figure 4.3: (a) Relation between building and not building on locations with various amounts of chokepoints. (b) Percentages of building versus not building on locations with various amounts of chokepoints.

Game Time

Figure 4.4 shows the percentage of total builds conducted at specific time intervals. As the starting position is included in this, each player always constructs a base in the beginning of each game, which is why the amount of constructed bases for around zero game time is rather high. It would seem that there is no golden point in time for constructing an expansion. However looking at the data it seems like early game is perhaps more focused on attack, defence and getting the current base running, postponing the first expansion. At late game there is a light drop in the amount of expansions constructed. This may be due to the game being almost settled, causing players to focus primarily on combat in order to exterminate the last bases of the enemy. Generally, it seems like expanding may happen through the core part of the game.

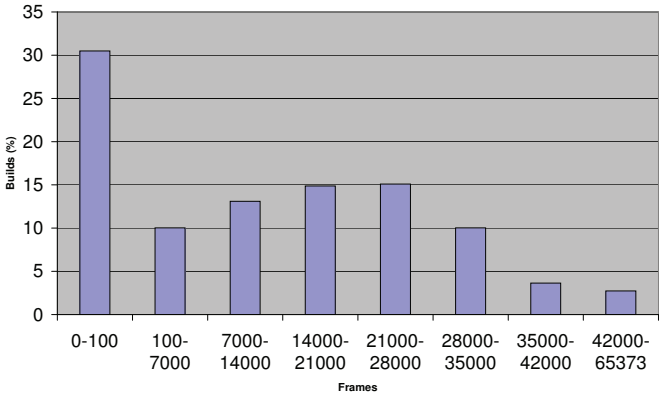
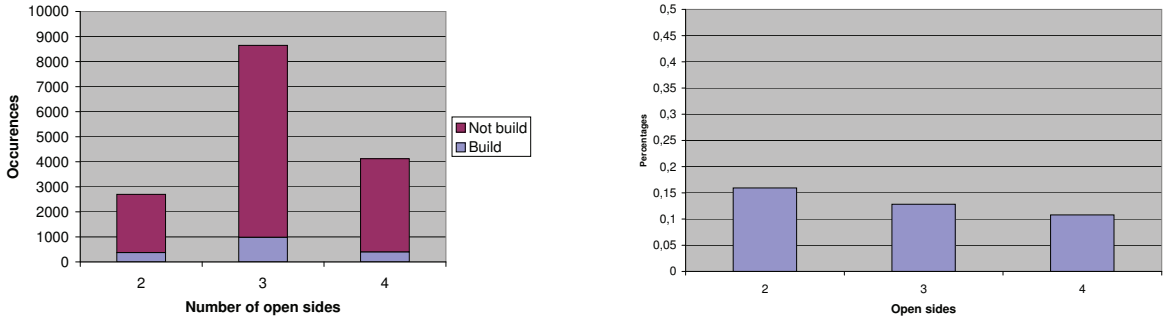


Figure 4.4: Percentage of total builds conducted at specific time intervals.

Open Sides

The variable *open sides* shows the number of open sides of a base location. A side is defined as the north, south, east or west border of a base location and an open side is a side that is not in close proximity to the edge of the map. In practice, this means that an open side is a direction in which the base location is not protected by the map border. Air and ground units are likely

to attack from an open side, where an attack from a closed side is either impossible or restricted to aerial units. Due to the way a side has been defined, there can be a maximum of four open sides. This is the case when a base location is positioned in the middle of a large map, in this case, the base location will be open to attacks from all sides. Three open sides means that the base location is positioned near the border of a map and two open sides means that the base location is in one of the four corners of the map.



(a) Open sides relation.

(b) Open sides factor.

Figure 4.5: (a) Relation between building and not building on locations with various amounts of open sides. (b) Percentages of building versus not building on locations with various amounts of open sides.

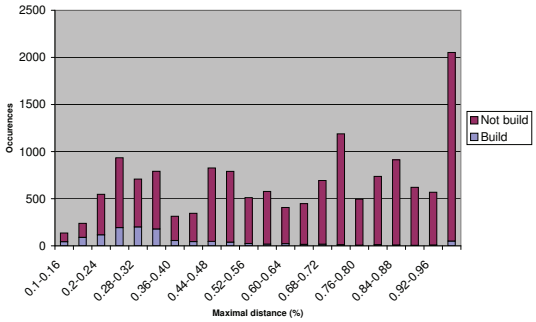
As it can be seen in Figure 4.5a and Figure 4.5b, the collected data shows a slight tendency towards base locations with few open sides. These locations are often easier to defend and that may be part of the incentive for constructing an expansion on this type of base location. However, it should be considered that starting positions are often located in the outskirts of a map, meaning that the player's first base will often have at least one closed side.

Own Flying Distance

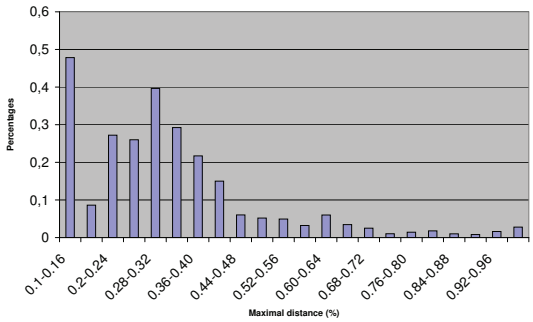
The variable *own flying distance normalised* is a property determining the direct distance from the players nearest already built base to the location being considered. The distances that are zero represents the starting locations of the players. Figure 4.6a and Figure 4.6b shows that players seem prone to build expansions in locations that are close to existing bases. This makes a lot of sense in that it is likely much easier to defend bases that are close together, compared to bases that are far from each other or where there does not exist a path between. For scanning, it may be worth considering that if an enemy base has been found, it is possible that there are more nearby.

Enemy Flying Distance

The variable *enemy flying distance normalised* is also the direct distance, but in this case the direct distance to the nearest enemy expansion. It should be noted that the players in the replays used, may not always have been aware of the locations of enemy expansions. That being said, it can be seen from Figure 4.7a and Figure 4.7b that players generally build expansions rather far away from their enemy. Superficially, this seems logical as one would imagine expansions close to the enemy may be discovered and/or destroyed. An interesting thing to note is that in some cases bases have been built very close to the enemy, but seemingly not in a midrange distance from the enemy. This would indicate that if a player is building a base close to the enemy, it



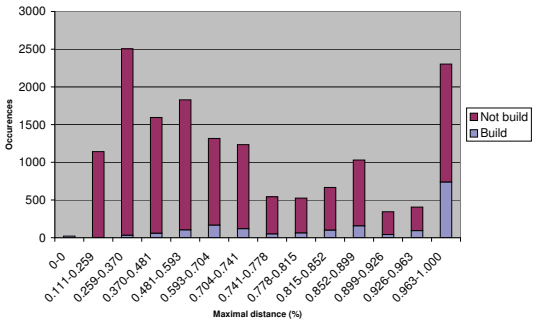
(a) Own flying distance relation.



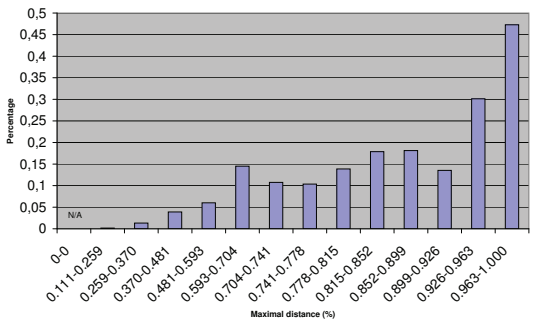
(b) Own flying distance factor.

Figure 4.6: (a) Relation between building and not building on locations with various distances from own base expansions. (b) Percentages of building versus not building on locations with various distances from own base expansions.

will be very close. As the number of cases this happens are relatively low, it is not unlikely that this type of expansion belongs to a specific strategy. Alternatively it should be considered that there may be some specific game event taking place. Our replay analyser does not discriminate between cases, where a player is eradicating an enemy expansion and constructing a new base at the location even before the enemy has been completely removed, and the case where the player is simply building a resource deposit inside the opponents existing base. The result for the distance of 0 in Figure 4.7b (N/A) stems from the concrete values, 9 builds and 0 not builds. Though building inside the enemy base happens very rarely, it still does happen. There are no cases of *not building* inside the enemy base, due to the way the variables are collected. Since a base is occupied by an enemy, it is regarded as not being interesting for players when they consider new expansion locations and hence, *not build* will in no case increase.



(a) Enemy flying distance relation.



(b) Enemy flying distance factor.

Figure 4.7: (a) Relation between building and not building on locations with various distances from enemy base expansions. (b) Percentages of building versus not building on locations with various distances from enemy base expansions.

Expansions

The variable *expansions* shows the number of current expansions at the time of constructing a new expansion. As an example, consider a case where a player has seven expansions and constructs another. This will be the player's eight expansion, meaning that the column marked

7 is the number of times a player has constructed their eighth expansion.

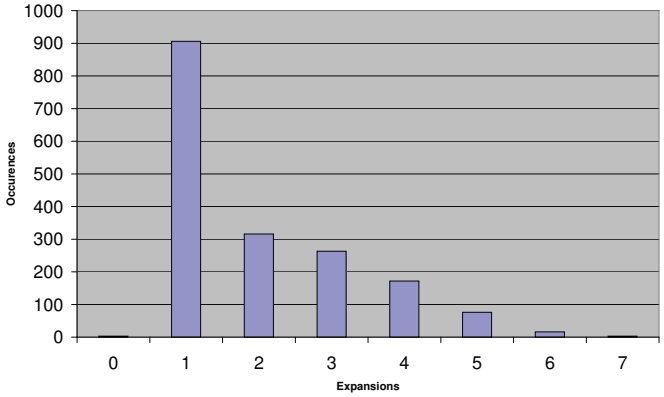


Figure 4.8: Amount of occurences for various amounts of expansions.

From Figure 4.8 it can be observed that the more expansions a player has, the rarer building an additional expansion is. This could be applied when scouting, considering that the more enemy expansions have been found, the less likely it is to discover another. One should be careful not to interpret the data, as it being unlikely for a player to create more than one expansion. A player may create an expansion while only having the base at the starting position. The player may then have this expansion destroyed by the opponent and choose to construct a new one. In this case, both expansions are constructed while having one existing base, though only one of them should be considered the *first* expansion. There are a few instances of players constructing an expansion without having any expansions. This may happen when a player loses ownership of their last expansion but manages to utilise a worker to create a new expansion before the last of their buildings are eliminated. This seems to be a rare occurrence, possibly because a player may be willing to surrender, when they lose ownership of their last base. 250 replays have been collected with more than 900 cases for building an expansion while having one already. For this reason, it would seem that constructing expansions is a very common element of StarCraft, as would have been expected.

4.5 Summary and Open Problems

Weka has been useful in identifying relations between the recorded properties and building or not building an expansion. The amount of resources, vespene gas and minerals, seems to be a large factor when choosing a location, hinting that players build expansions for the profit they may bring and not merely to surround an enemy or get control of strategic key points of a map. Is also interesting to observe that the distances to both a player's existing bases and the distance to the enemy is important factors as well. Players tend to construct expansions near their own expansions, possibly because it makes the expansions easier to expand. Players will generally also construct their expansions far away from the enemy, possibly for the reverse reason. Variables have only been evaluated on how they directly influence the *BUILD/NOT BUILD* relation. As an example, this means that it is not considered how the importance of variables changes with game time. Some of the variables that have been excluded from this section, due to not showing an interesting relation with *BUILD/NOT BUILD*, may turn out to have a clearer influence on this, if coupled with another variable. The gathered data contains the start base of each player as an entry on par with the player-chosen expansions. In some

cases it may have been more useful to distinguish between pseudo randomly selected starting positions and consciously chosen expansions. As an example this can assist in classifying how valuable a player considers an area with a high amount of resources, by examining the number of cases a player have actively constructed an expansion on a resource dense area.

CHAPTER 5

CONCLUSION AND FUTURE WORK

We have addressed some of the problems concerning mineral gathering and base expansion in StarCraft. We have designed an algorithm for efficient mineral gathering and performed preliminary work on identifying scouting strategies when choosing a new expanding location.

We successfully constructed an algorithm for mineral gathering, which queues workers for minerals. The algorithm minimises the round trip time for each worker in order to maximise income. The algorithm was implemented and tested against the built-in StarCraft mining approach. These experiments showed that an agent based on the algorithm gathered 3% more minerals than the built-in StarCraft mining approach. It was shown that it is difficult to predict the amount of minerals gathered, using the built-in StarCraft mining approach. In comparison with this, the standard deviation of gathered minerals by an agent based on our algorithm is low, making it more predictable. The increased predictability increases the confidence in the calculations made for the amount of minerals gathered in some finite time. This can in turn be used when considering at what point in time it is viable to create an expansion, as the potential gain can be closely approximated.

We analysed a range of variables, some of which influences a human player's selection of an expansion location. Data on these variables was gathered by analysing 250 replays. Identifying the most influential variables is required in order to develop a model for strategic scouting.

The experiments suggested that the distance between a location and existing expansions, is an important factor. Players are inclined to build expansions close to their own expansions and far away from the enemy's.

The amount of resources contained within a region also seems to be a significant factor when choosing expansion locations. Players had a preference for resource-dense locations, suggesting that the main purpose of expanding is likely acquisition of resources.

5.1 Future Work

The mining algorithm is greedy in the sense that each worker selects the option, allowing the fastest mineral gain. It might be beneficial to make the algorithm distribute workers by having them act as a group, increasing the total income over time instead of the income per worker.

It would be beneficial to use a greater amount of replays for the analysis since this would generate more accurate data, as stated in Section 2.4.

In this technical report we have considered a number of variables, which we believe may be related to whether or not an expansion at a given location is desirable. It is possible that there may exist influential variables, which we have not considered. Due to this, it may be beneficial to perform further experiments. It may also be beneficial to utilise the knowledge of expert players, in order to identify further variables.

So far data entries have only contained *BUILD/NOT BUILD* tags. Locations already containing an expansion do not create a new entry. It is possible that creating a tag for *ALREADY BUILT* will allow a more accurate depiction of the relation between expansions. As an example, a player may have a low preference for a resource-dense location, if they already occupy several of such locations. Introducing an *ALREADY BUILT*-tag will also allow for a more accurate representation of the starting base of players, making this a special case that does not have a *BUILD*-entry.

BIBLIOGRAPHY

[1] M. Buro and T. M. Furtak. Rts games and real-time ai research.

[2] V. Chartz. Software totals. `http://www.vgchartz.com/`. [Online; accessed November 25th 2010].

[3] S. De Jong, P. Spronck, and N. Roos. Requirements for resource management game ai. In *In Proceedings of the IJCAI 2005 Workshop on Reasoning, Representation, and Learning in Computer Games*, 2005.

[4] B. Geryk. A history of real-time strategy games. `http://www.gamespot.com/gamespot/features/all/real_time/`. [Online; accessed December 5th 2010].

[5] M. Kubovy and A. O. Holcombe. On the lawfulness of grouping by proximity. *Cognitive Psychology*, 35(CG970673):71–98, 1998.

[6] J. McCoy and M. Mateas. An integrated agent for playing real-time strategy games. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*. The AAAI Press, 2008.

[7] K. Olsen. South korean gamers get a sneak peek at 'starcraft ii'. USA Today Online, `http://www.usatoday.com/tech/gaming/2007-05-21-starcraft2-peek_N.htm`, May 2007. [Online; accessed November 25th 2010].

[8] P. Strategies. Mining replays of real-time strategy games to learn player strategies.

[9] B. Weber and M. Mateas. A data mining approach to strategy prediction. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, 2009.

[10] B. G. Weber, M. Mateas, and A. Jhala. Applying goal-driven autonomy to starcraft. In *Proceedings of the Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*. The AAAI Press, 2010.

[11] S. Wintermute, X. Joseph, and J. E. Laird. Sorts: A human-level approach to real-time strategy ai. In *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference*. The AAAI Press, 2007.

[12] I. H. Witte and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2nd edition, 2005.