Organization in AI design for real-time strategy games

Damien Ernst
University of Liège
191 PUBLICATIONS 4,067 CITATIONS
SEE PROFILE

Organization in AI design for real-time strategy games

Firas Safadi
University of Liège

Damien Ernst
University of Liège

Abstract

For the past two decades, real-time strategy (RTS) games have steadily gained in popularity and have become common in video game leagues. Without a doubt one of the most complicated genre, RTS games are challenging to both human and artificial intelligence. Regrettably, intelligent agents continue to pale in comparison to human players and fail to display seemingly intuitive behavior that even novice players are capable of. Working towards improving the performance of such agents, we present a clear and complete yet generic AI design in this paper.

1 Introduction

In a typical real-time strategy (RTS) game, players control units and structures on a battlefield and have to accomplish a specific objective, such as destroying all enemy buildings. During the game, additional units and structures can be produced at a cost. The typical winning strategy for a RTS game goes through securing specific areas on the map and gathering resources rapidly enough to build an army so as to crush opponents before they can amass a force of similar strength.

In most cases, some form of technological development is involved. Several unit types can be trained as long as their requirements are met. Units each have different attributes like hit points (HP), abilities and production cost. As players research new technologies, more advanced units are unlocked. They are thus repeatedly faced with multiple choices throughout the game and must constantly take decisions as to which units they require, how fast they require them, whether the current income is sufficient, whether they should attack or not, whether they should take control of new resource points (expand) or not, etc.

Clearly, making the right decisions is not possible without reconnaissance. In order to maximize their chances of winning, players need to know what their opponents have or intend to have so as to create an army most effective against theirs. They also need to assess the enemy's situation, detect temporary vulnerabilities and exploit them. This task of gathering information on opponents is often referred to as scouting and is necessary because players can only observe areas where they have deployed units.

One final and important aspect is related to diplomacy in the fact that participants in a game can choose to form alliances. Indeed, team play is often one of the most critical factors of success. In an effort to lighten the complexity of designing AI agents for RTS games, we will only focus in this document on the simpler one-on-one scenario.

The paper describes a generic agent architecture based on a bidirectional analysis applied to StarCraft. We separate long-term and short-term objectives (temporal analysis) as well as production-related tasks from combat-related ones (spatial analysis). The resulting design is composed of multiple managers organized in a task-specific hierarchical structure and grants the agent both performance and modularity.

After discussing some related work in Section 2, we briefly introduce the SartCraft game in Section 3. The architecture of our agent is detailed in Section 4. Section 5 exposes some of the results we obtained using our design and, finally, Section 6 concludes.

2 Related Work

During the last decade, RTS games have been approved by scientific communities as rich environments for AI researchers to evaluate different AI techniques. Development frameworks for RTS agents such as the ORTS[1] project (Buro and Furtak, 2004) appeared and research work started to tackle some of the challenges offered by the RTS genre. Due to the inherent difficulty of designing good RTS agents able to address the multitude of problems they are confronted to, most work has been concerned with specific aspects of the

[1]Open RTS

game.

The strategy planning problem is probably the one that has received the most attention with the success of case-based planning methods to identify strategic situations and manage build orders. An approach based on case generation using behavioral knowledge extracted from existing game traces was tested on Wargus[2] (Ontañón et al., 2007). Other approaches for case retrieval and build order selection were tested on the same game (Aha et al., 2005; Weber and Mateas, 2009a). A case retrieval method based on conceptual neighborhoods was also proposed (Weber and Mateas, 2009b).

The strategy planning problem has also been addressed with other techniques such as data mining (Weber and Mateas, 2009c). By analyzing a large collection of game logs, it is possible to extract building trends and timings which are then used with matching algorithms to identify a strategy or even predict strategic decisions. Evolutionary methods have been employed as well, mostly in strategy generation (Ponsen et al., 2006).

Meanwhile, some work has focused on lower-level problems like micro-management. Monte Carlo planning was among other things applied to simple CTF[3] scenarios on the ORTS platform (Chung et al., 2005).

Although the above-mentioned works all showed interesting results, none actually takes on all the aspects of the RTS genre. Other works have instead considered the entire problem and present complete agents. A cognitive approach was tested using the ORTS infrastructure, though it suffered from artificial human multitasking limitations (Wintermute et al., 2007). Another approach was based on an integrated agent composed of distinct managers each responsible for a domain of competence (McCoy and Mateas, 2008). Although the design was clear, it lacked hierarchical structure and unit management was largely simplified.

The design we present in this paper defines a clear and complete task-specific hierarchical structure and is generic enough to be adapted to most RTS games. We discuss the many aspects relative to the game genre and propose a global architecture integrating several managers operating on different levels.

[2]Wargus is a clone of Warcraft II, a RTS title published by Blizzard Entertainment in 1995.

[3]Capture The Flag

3 StarCraft: Brood War

StarCraft: Brood War is a sequel to Blizzard Entertainment's flagship RTS, StarCraft, released in 1998. Blizzard Entertainment is a famous video game developer and has recently accomplished an outstanding reputation with the successful World of Warcraft (WoW) title.

Even though StarCraft: Brood War is over a decade old, it continues to be the top RTS choice for many professional players, especially in South Korea, where the game has acquired enough fame to be considered a national sport.

StarCraft features three races: the Terrans, the Zerg and the Protoss. Each race has its own unique set of units and balanced strengths and weaknesses. The Terrans excel at defense and possess a wide range of specialized units. The Zerg usually overwhelm their opponents with massive armies and can expand their forces by infesting other lifeforms. The Protoss are a humanoid species with unmatched technology and individual psionic abilities.

There are two types of resources in StarCraft. Minerals are the most abundant one and constitute the primary costs of most basic units, while vespene gas is harder to obtain and is often required for advanced units.

In all three races, units require, besides their production cost, supplies. The amount of supplies used by a unit varies according to some parameters, including its size. When all available supplies are in use, more supplies must be produced before training any additional units. Conversely, when units are killed off, the supplies previously used are reclaimed.

Like many other RTS games, units in StarCraft have hidden attributes other than HP, energy or armor. For instance, a unit can be biological, mechanical or robotic. Some abilities only work on a specific type of unit. Units also differ in size and by the nature of the damage they can cause. They can be of small, medium or large size and can cause normal, concussive or explosive damage. Concussive damage works well against small units but is less efficient against large units, while explosive damage works in an opposite manner. There are other attributes such as speed, attack speed, whether the unit is aerial or not, etc.

This added complexity puts StarCraft in the top tiers of elaborate RTS games and poses a great challenge to players and AI researchers alike.

4 Agent design for StarCraft

Designing an agent capable of playing a complete game is no easy feat. Given the various tasks the agent must perform, the number of unit types to handle and the many different states a game can reach, it is clear that a blunt approach will not lead to satisfying results. In order to break down the complexity of the game, one must identify and consider tasks more individually, generalize the game state or focus on a substate, etc. We present below an approach based on a task-specific hierarchical design. Each module in the design performs a task while receiving instructions from the module above. Although the design is defined for StarCraft, most concepts can easily be extended to other games.

4.1 Identifying the tasks

In StarCraft, players perform tasks by giving units specific commands. Units in this context can be workers, military units or buildings. Workers can be ordered to move, to attack, to repair (Terrans only), to gather minerals from a mineral field or vespene gas from a refinery or to build a structure. Military units can generally be ordered to move or attack, to patrol, and in some cases, to use a special ability. Buildings can be ordered to train units, perform an upgrade or research a technology. Some Terran buildings can be lifted and moved. Defensive buildings can also be ordered to attack.

The first decomposition we did is spacial and divides actions into two categories: production-related actions and combat-related actions. Production actions include any unit production, technology research and upgrade, or any prerequisite action such as resource gathering or structure building. These actions can be taken by giving orders to workers or buildings. Combat actions consist of all other actions and can be given to any unit. They generally consist of move or attack commands given to military units as well as the use of special abilities. These orders may also involve workers and buildings.

Using a time-scale decomposition, decisions were separated into three different levels. On the topmost level, strategic decisions are taken. These are long-term, high-level decisions such as deciding to rush (i.e., quickly produce some units and attack the unprepared enemy), using a certain combination of units or expanding. The second level includes mid-term decisions such as primary target acquisition or squad compositions. The third level involves short-term decisions like ordering a set of units to attack a specific target or switching to a specific formation. Decisions below this level define the low-level AI behavior of individual units. To illustrate the different levels, let us consider the following. When a unit is ordered to attack a target, it is unlikely that the command is a result of an independent decision. Rather, the order results from a higher-level decision such as the squad the unit belongs to acquiring a target, which in turn is a result of a high-level decision to attack and destroy the enemy's base.

4.2 Resulting design

With two categories of actions and three planes of decisions, an example design as shown in Figure 1 can be adopted. As can be seen, production orders travel from the strategy manager down to the individual workers in the mining and construction squads through the production manager. Combat orders also originate from the strategy manager and reach military units through the combat manager. Of course, orders generated by the strategy manager are not directly transmitted to individual units. They are strategic orders such as *"build 2 factories"* or *"attack the enemy"*. This information is processed by the lower-level managers and eventually translates into direct orders to individual units. To better understand how this works, we will now detail one by one the different managers and define their role in the global design.

4.3 Strategy manager

In a RTS game, as its name would suggest, strategy is certainly the most important aspect. Regardless of how well a player handles units on the battlefield, if the strategy applied is poor compared to that of the opponent, the chances of winning are slim at best. Ideally, this should be the element to which most thought should be given when designing an agent. In practice, however, strategy development and processing is very low, if it exists at all, in agents because of the complexity of the task. Humans often use an intuitive approach to assess the current situation in a game and take appropriate strategic decisions. For example, a professional player can, by observing the first minutes of a game, estimate the composition of the forces a few minutes later, and in cases where one strategy is completely inefficient against another, al-

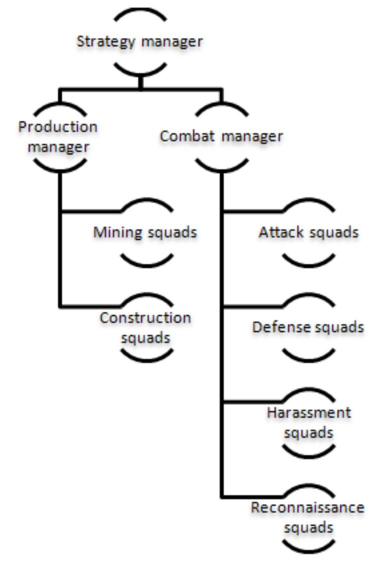


Figure 1: Example agent design.

ready determine the outcome of the match[4], all with high accuracy. To do this, an agent would require a database of strategies it might encounter along with a case-matching algorithm, which is difficult as existing strategies continue to evolve[5], new strategies appear, etc. Note that, at the time of the release of the game, such a database does not even exist yet. Instead, developers often implement a generic strategy which can be defeated after a couple of tries at most by novice players. For this reason, we believe that it is still early to bet on an agent's performance in this domain to pose a serious challenge to human players.

4.3.1 Role and importance

The strategy manager lies at the top of the hierarchy and essentially makes all the high-level decisions. It directly controls the production manager as well as the combat manager. Its role is to coordinate large-scale production and combat in order to defeat the enemy. It does so by:

- deciding what units to produce, technologies to research or upgrades to perform with or without assessing the enemy force
- deciding when and where to expand according to current income, sustainability and map-related information such as distance to primary base
- deciding when to attack, defend or scout

As previously discussed, the strategy manager is not likely to be the reason the agent wins against a human player and therefore, spending considerable development resources on it is unlikely to yield a satisfying increase in agent performance. Its behavior will remain rather simple and we will prefer using generic strategies based on generic triggers. The longer the agent stands its ground against a human player, the greater its advantage becomes as the number of units keeps increasing and multitasking starts to wear out the opponent. The agent, on the other hand, is capable of handling, for example, production tasks, attacks and scouting simultaneously with much higher efficiency.

4.3.2 Development

Before going any further, we would like to differentiate between two categories of agents: cheating agents and non-cheating agents. Cheating agents do not play the game in a fair setting. For instance, they can have access to the entire state of the game. This represents a tremendous advantage as it eliminates the need for reconnaissance. Cheating agents know exactly the opponent's position, units, wealth and any other information accessible through code at all times. Such agents also make it impossible for the opponent to fake a strategy by hiding its key elements (i.e., build a key factory outside your base in order to hide it from the enemy's scout when it gets there). This type of agent is the most common in RTS games. In StarCraft, all computer players are cheating agents. Non-cheating agents play, of course, the game fairly. We will focus on the latter category.

The first step that can be taken when designing the strategy manager is to consider one manager per

[4]An example of such scenario can be seen in the movie at the following address: http://www.youtube.com/watch?v=B7pOBHyl4W0. In this case, the Terran player goes for a rush strategy where a barracks is produced earlier than usual, thus sacrificing stronger economy to gain a temporary military advantage while the Zerg player goes for an early expansion, sacrificing military power for delayed economical advantage. As soon as both strategies become apparent (roughly a minute and a half after the beginning of the game), the commentator predicts the outcome of the match.

[5]Even though strategies are often characterized by some key elements, they can easily be morphed into close, atypical variants or completed revamped.

match type. That is, the manager is specific to its race and that of its opponent. Technically, the race of the opponent may not be known at the beginning of the match as it can be randomly selected by the machine. However, in most, if not all, cases in professional leagues, match types are known in advance as players tend to specialize in one race. This separation allows for clearer design as decisions and triggers can directly be race-specific. Using race-specific strategies is important and can increase the efficiency of the agent considerably. The opponent's race affects strategic choices such as the preferred unit types or abilities. For example, for a Terran player, the combination of Marines and Medics can be deadly against a Zerg opponent because most Zerg units are unable to withstand the damage dealt by a squad of Marines using the StimPack ability which greatly increases firepower long enough, but not so much against a Protoss opponent whose units are overall much tougher. A Terran player would also never consider researching the Ghost's Lockdown ability against a Zerg opponent as it can only be targeted at mechanical units.[6] The default AI included in the game does not select strategies based on the opponent's race.

The strategy manager can select a specific strategy at the beginning of the game which would translate into a number of production orders. However, in order to take other actions, it requires triggers based on the state of the game. Consequently, it must have some, inevitably simplified, representation of the game state and continuously update it. There are two ways information becomes available to the strategy manager. The first is through direct access to the game state and the second is through event notifications. For instance, whenever a unit becomes visible, an event is raised and the strategy manager is notified of the appearance of this unit. Other events involve the destruction of a unit, its disappearance, etc. The strategy manager can use either way to maintain its state representation up-to-date. One example of a simple representation is a vector including components such as the current elapsed time since the beginning of the game, the accumulated minerals and vespene gas, the current income, the number of total units owned, and some other enemy-related variables such as an estimation of the size of its army or the number of resource positions occupied. The strategy manager can then take specific actions as soon as certain state-based conditions are met. The state representation may also include static information such as map size. Such factors can affect the strategy selection process. Obviously, attempting a rush is less reasonable in a huge map where the units built early on are not likely to get to the enemy's base in time, before any defenses are set. Finally, random variables can be introduced in order to avoid predictability. The default AI for a Zerg computer player makes use of such techniques: at a small probability, it will attempt a rush which can take out many unprepared players.

One type of actions the strategy manager can take are production-related actions. These actions modify the state of the production manager and are represented by production orders composed of several parameters such as a task priority, a unit type and a quantity. An optional location can be set for construction orders. Orders can be revoked using similar commands. The strategy manager can therefore plan ahead and send production orders with different priorities to the production manager, but also interrupt running jobs for new, higher-priority ones. Consider a Terran agent playing against a Protoss opponent, both players being set on some strategy. If the Protoss player were to build an unexpected unit such a Carrier, the Terran agent could issue a high-priority production order for three Goliaths. When the high-priority task is over, previous tasks are resumed according to the production plan set by the strategy manager. This mechanism allows the agent to quickly build an appropriate counter to an enemy unit without having to completely alter the base strategy and can thus make the agent more adaptive in cases of unexpected behavior. It is also possible to use it with factories. If the Protoss opponent builds 7 Stargates, it is not necessary to wait until any units are produced to start building anti-aircraft unit factories. It is important, however, to carefully build the triggers for such mechanical reactions as they may not always be efficient. If the requisite technology is not even available, it may not be worth researching unless the enemy has at least a sizable amount of the unit type to counter.

Next to production actions, the strategy manager can take combat actions by modifying the state of the combat manager. The strategy manager interfaces with the combat manager primarily using flags. When the strategy manager decides to launch an assault, it raises an attack state flag in the combat manager, which translates into the combat manager either acquiring a new target and relaying orders to attack squads under its control or simply requesting that squads acquire targets on their own. Similarly, the strategy manager can raise a defense flag when the

[6] Zerg units are all biological.

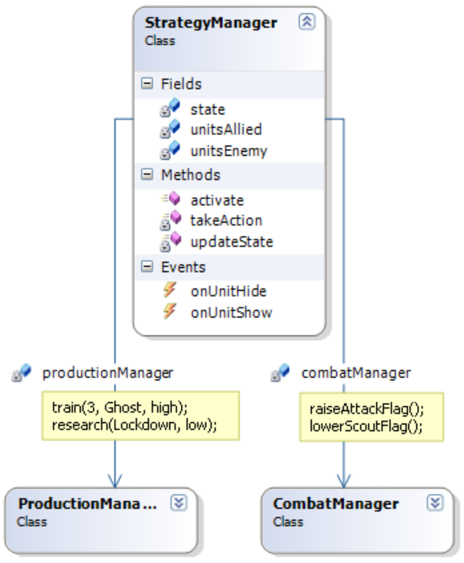


Figure 2: An illustrative example of the structure of a basic strategy manager.

agent is under attack or a scout flag when it requires information update on the opponent's situation. It may as well include force estimators to assess whether the agent's current force is a match for that of the enemy and determine opportunities. In a simple strategy manager, force estimators can be based on unit types and efficiency ratings. For each unit type the agent can build, an efficiency rating against every unit type the opponent can build is set. For example, Firebats will have a low efficiency against Dragoons, but a high efficiency against Zealots.

4.4 Production manager

Production is not a trivial process in StarCraft. It is subject to resource, supply, technology and terrain constraints. To train a specific unit, it must be first unlocked in the technology tree. For example, a Terran player must, after building a Starport, build a Science Facility and attach a Control Tower to the Starport before being able to train Science Vessels from the factory. The player must also meet the resource and the supply requirements. That is, a total of 100 minerals, 225 vespene gas and 2 supplies must be available for each Science Vessel. The unit production will then start, and the player must wait 80 game time units before it is ready. Terran factories can only train one unit at a time. Queuing other units during the current production will lock out resources, but not supplies. Note that in the design presented in this paper, it is up to the strategy manager to decide whether multiple factories are necessary. Another example to illustrate the terrain constraints is building placement. When a worker is given a construction order, a location must be specified. The terrain in a StarCraft map is usually diverse and made of many different materials (dirt, grass, water, asphalt, etc). It also can be elevated, occupied by a doodad, and so on. Therefore, beside the obvious placement limitation excluding the possibility of building structures over other structures, other limitations are directly related to the terrain and often determine constructible regions. Lastly, a special limitation is applied to centers (Terran Command Center, Zerg Hatchery or any of its evolutions, Protoss Nexus). These buildings cannot be placed in direct proximity to mineral fields or vespene geysers and can at best be set at a fixed, minimum distance.

4.4.1 Role and importance

Even though production tasks are decided by the strategy manager, the production manager needs to solve a number of underlying problems related to the constraints explained above, such as income optimization, build orders and building placement. Hence, its role is to coordinate production-related tasks in such a way that the resulting production is consistent with the demands of the strategy manager. It accomplishes that by performing the following tasks:

- determine a build order according to the priorities of the tasks given by the strategy manager
- resolve technological dependencies
- position buildings correctly
- assign workers to mining and construction squads according to current tasks.

The production manager handles mostly mechanical tasks, although some tasks such as building placement may sometimes require intelligence. For the most part though, it can be close to optimal and its development should therefore be rather straightforward, albeit tedious. Note that even with an optimal implementation, the production manager is not likely to give the agent an edge against a human player.

4.4.2 Development

Determining build orders is the most common process the production manager will have to go through. The objective is to produce units and buildings in the order preferred by the strategy manager. Preference is communicated by the latter in the form of priorities attached to production tasks. This approach is quite intuitive. High-priority tasks should not be slowed down by lower-priority ones. This, of course, does not imply that low-priority tasks cannot be carried out before high-priority ones. The reason is that production tasks are not instantaneous. This is best illustrated using an example. Consider the following two production tasks:

- *train(3, Ghost, high);*
- *research(Lockdown, low);*

Assume that only one Barracks is available and all technological requirements are satisfied. The order at which these four production orders will be executed depends on the minerals, gas, supplies and income. A Ghost costs 25 minerals and 75 vespene gas and requires 1 supply. The Lockdown ability costs 200 minerals and 200 vespene gas. A Supply Depot costs 100 minerals and provides 8 supplies. Let us examine some scenarios with different resources available:

1. **275 minerals, 425 vespene gas, 3 supplies:** the Lockdown ability will be researched at the same time the first Ghost is trained because the total requirements for both tasks are met.
2. **225 minerals, 275 vespene gas, 3 supplies, steady income:** the Lockdown ability will be researched at the same time the first Ghost is trained only if the income is such that by the time the first Ghost is completed, another 25 minerals and 75 vespene gas would be available.
3. **275 minerals, 425 vespene gas, 1 supply, no income:** the Lockdown ability will not be researched because an additional Supply Depot is required to train all three Ghosts.

Thus, the production manager estimates completion times and orders tasks according to priorities as well as both available and predicted resources. One way to achieve that is to distribute resources, current and future, in descending priority order. Income can either be estimated or measured (i.e., in minerals per frame).

Dependency resolving occurs when a task cannot be executed because of a technological limitation. In that case, it is up to the production manager to acquire the necessary technologies and proceed with the task. This can be done by breaking the pending production task into several sub-tasks with the same priority. For example, if the strategy manager requests the production of a Battlecruiser while the agent only owns a Command Center and a Barracks, the production manager will automatically add production tasks for a Factory, a Starport with an attached Control Tower and a Science Facility with an attached Physics Lab, all at the same priority as that of the Battlecruiser task.

Building placement can either be automatic or guided by the strategy manager. In most cases, no location is specified and the production manager simply scans for a valid position. This can easily be done by scanning the map in spiral starting from a set location such as the start location. Spacing parameters can also be specified. This can be useful when placing defensive structures. For instance, the strategy manager can specify a tactical location for the first defensive structure. Then, the location of subsequent defensive structures can be set to the same one with a zero spacing parameter. This results in stacking those structures as much as possible, turning the area into a heavily defended position. On the other hand, wider spacing can be used for factories and other buildings in order to allow units to easily travel across the base.

Workers are assigned to mining or construction squads depending on both the current production tasks and the current squad compositions. They are constantly reassigned depending on the resources required and on whether there are any constructions to initiate. Usually, an agent has at least enough workers to maintain optimal mineral and vespene gas gathering while handling one or two construction jobs.[7] Let us note that the production manager is notified of the appearance and disappearance of units in the same way the strategy manager is. Whenever the strategy manager is notified of such an event, it in turn notifies the production manager by triggering another event. Workers

[7]It is common for a player to have more workers than currently necessary when an expansion is planned in the near future or simply as a safeguard in case the enemy manages to eliminate some.

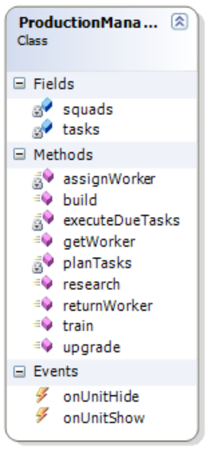


Figure 3: An illustrative example of the structure of a basic production manager.

are therefore caught and assigned by the production manager as soon as they are produced.

4.5 Combat manager

Combat management is without doubt one of the main difficulties for human players in RTS games. It requires both solid organization and high reactivity. It is all the more true for StarCraft with its limitation on multiple unit selection, quick-paced encounters and rich terrain. It is not possible to manage troops efficiently on the battlefield without organization. As players produce units, they will quickly find themselves in control of several units of different types and will feel the need to organize them in groups. For instance, tough, frontline units and vulnerable, high-firepower units are often separated into different groups in order to easily send in the former category first followed by the latter or to more carefully control the latter while the former simply acts as a *"tank"*. Caster units are usually isolated. Ground and air units can also be grouped separately as they are not subject to the same terrain constraints. More generally, players will want to group units together according to how likely they are to move to or to attack the same target. Moreover, players in StarCraft cannot select more than 12 units at the same time. With 40 Wraiths, a player must at least form 4 different groups and control each one individually. Added to this limitation is the fact that battle durations in StarCraft can be extremely short and thus, player reactivity must be correspondingly high. Even with huge armies, an encounter can easily last no more than a few seconds. For example, a Protoss player can lay waste to an entire Zerg swarm with some carefully placed Psionic Storms in a matter of seconds. Last of all, players need to take into account and react to terrain variation as it may either favor or disadvantage them. Accuracy reduction for units on low ground targeting units on high ground is one such mechanism.[8] Another illustration lies in the fact that ground units would have trouble fighting air units on a terrain with elevation variations as they move to and up or down ramps to reach their targets.

4.5.1 Role and importance

The role of the combat manager is to relay combat-related strategic decisions to military units and manage military squads. It determines squad compositions according to flag values as well as the units available. In other words, it must:

- act as an interface between the strategy manager and squad managers
- provide a centralized location for squad managers to access general combat information such as region focus or information on other squads
- assign units to squads according to flags, unit type and existing configurations
- constantly reevaluate squad compositions as flag values are toggled and units are created or destroyed.

Setting primary target regions or assigning the right units to the right squads are both delicate tasks that can greatly increase the performance of an agent when executed with care. As we've previously discussed, units differ in type and handling them as one group can lead to catastrophic results. Consequently, the quality of the combat manager is an essential point in the design of the agent and should be given considerable attention.

4.5.2 Development

We want the behavior of the combat manager to be largely influenced by the decisions taken by the strat-

[8] When a unit targets another on an elevated terrain, its hit rate is lowered to 70%. Its average firepower is therefore reduced by 30%.

egy manager. Doing otherwise would inevitably lead to overall erratic behavior. One could argue that since the combat manager is the entity best informed of squad compositions, it should be capable of accurately assessing forces and predicting whether an assault would be successful. Yet letting the combat manager decide when to attack is not likely to yield good results because the question does not solely depend on current forces and immediate outcome. This type of decision is closely tied to the strategy being applied, something the combat manager is not aware of. For example, if the agent decides to follow a defensive strategy in which it refrains from attacking and instead focuses on defense and expansion in an attempt to control as many resource points as possible and eventually defeat the opponent in macromanagement, launching an attack because the agent's current force is superior to that of the enemy would be an inconsistent move. There are many other examples. There are, for instance, scenarios where an attack is executed even though it is likely to result in a loss. Simultaneously attacking and expanding is one such scenario. By attacking, a player distracts the enemy and prevents an immediate attack, subsequently delaying the detection of the newly established base. As we can see, those decisions are not really based on squad compositions and the details are in fact quite irrelevant. Although we have discussed the possible use of force estimators in the strategy manager, it is clear that the latter does not necessarily require more than gross estimations. This is, of course, all in accord with our analysis. Long-term decisions should be handled by the strategy manager and timing attacks clearly falls in this category.

The combat manager may, however, organize an attack by specifying target areas and by controlling squad deployment. The combat manager can achieve this by confining target acquisition to different regions individually for each squad. It can focus on a single area or distribute squads on multiple areas according to tactical advantage. This could be used to attack a base on two or more fronts, or more often simply to attack two bases at the same time. This can be desirable when the enemy forces are concentrated in the primary base while a small outpost actively contributes to the opponent's economy. It is then interesting to attempt a synchronized attack and send troops to both locations in order to take out the outpost with a squad while other squads delay reinforcements. It goes without saying that this type of organization must be constantly refreshed. Should enemy units be encountered on the way to a target region, the target is renewed or at least adapted to include those units. The combat manager can also control squad involvement in battle by deciding whether a squad is operational or not. This is useful for preventing incomplete squads from getting involved prematurely. When a player creates an army and attacks, by the time units reach the enemy, new units have already been trained. Those are not usually sent along as the battle would be over before they arrive and they would not be able to stand their ground against enemy defenses without being numerous enough. In this case, the combat manager can simply assign those new units to new squads while restricting attack orders to squads marked as ready-to-go. More generally, the combat manager can decide whether a squad is able to perform a task.

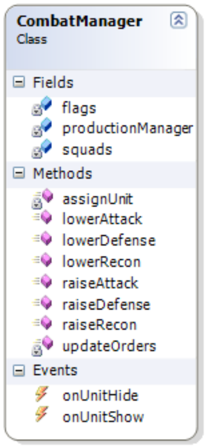
Assigning a unit to a squad involves two levels. The first one is to assign a task to the unit. Squads are grouped by task and units in the same squad necessarily perform the same task. Then, when the task is set, the combat manager must decide to which specific squad the unit should belong. The combat manager relies on flag states to assign tasks, though it may feature a set of override rules directly controlled by the strategy manager. Using a unit type exclusively for a task can indeed be part of a strategy. For instance, the Corsair spaceship can be used efficiently in both offense and reconnaissance tasks. In a case with both attack and reconnaissance flags raised, a new Corsair may end up assigned to either one of the two tasks, which might not be desirable if it was trained specifically for scouting. The agent can thus use a rule to avoid the undesirable outcome. Note that this mechanism is a simplified version of the more general task affinity system, in which units are attributed a set of task affinities which effectively dictate their general behavior.[9] In the example above, Corsairs could prefer, in order, scouting, attacking and defending. Consequently, using a combination of flags and task affinities leads to unambiguous task assigning: flags are used to determine the tasks available, while task affinities direct units to an appropriate task. Once the task is set, a specific squad performing the same task must be selected. Naturally, incomplete or new squads will be preferred to those marked as ready.

[9]The task affinity system associates a preference value to each task a unit can perform. When several tasks must be performed, a unit will be assigned to the task for which it possesses the highest affinity. This system can be found in other strategy games and is notably present in Dungeon Keeper, a title developed by Bullfrog Productions in 1997. Although affinities are usually static, they can be dynamically altered during a game to increase unit versatility.

A basic selection can be based entirely on unit type. That is, units of the same type are assigned to the same squad. Note that it is still possible to have multiple squads of the same unit type as squads can be considered complete. A more elaborate approach can involve the definition of templates. For example, an infantry squad could be defined as a squad of 10 Marines and 2 Medics. An alternative consists in defining ratios, such as a 4:1 ratio of Marines and Medics.

Obviously, squad configurations are not static and need to be updated as flag states and owned units change. When a flag is raised, a new task becomes available. The combat manager may then have to assign different tasks to existing units according to their affinities. Usually, we want units with similar affinities to be grouped together. This allows the combat manager to simply modify the squad task rather than reassigning units individually. Consider the following scenario. The attack flag is raised and the agent controls three squads. Two of them are attacking the enemy while the third one stands by in the base awaiting completion. If an enemy squad attacks the agent's base, the strategy manager will raise the defense flag, causing the combat manager to reevaluate squad configurations. Assuming all three squads are composed of units with a high affinity for defense, their task should be changed to defending. However, we have previously mentioned that the combat manager has the ability to exclude certain squads from certain tasks. In this case, distant squads such as the ones located in the enemy's base are not likely to get back in time to defend the agent's base. If the combat manager only reevaluates configurations for squads located in a set perimeter around the base when the defense flag is toggled, the attacking squads will not be recalled, which is a likely human behavior and in many cases an efficient one. Thus, in this scenario, the agent would continue to attack using the two attack squads in the enemy's base and defend using the previously idle squad in its base. Besides this type of update, squad configurations may need to be completely reset if too many squads have suffered heavy damage. One possibility is to reassign survivors to new squads.

Lastly, the combat manager may sometimes require workers to repair damaged structures or even to repair units or build defensive structures directly on the battlefield. Since workers are under the control of the production manager as soon as they are produced, the combat manager must communicate a request to


CombatManager
Class

Fields
- flags
- productionManager
- squads

Methods
- assignUnit
- lowerAttack
- lowerDefense
- lowerRecon
- raiseAttack
- raiseDefense
- raiseRecon
- updateOrders

Events
- onUnitHide
- onUnitShow

Figure 4: An illustrative example of the structure of a basic combat manager.

take control over a worker before it can use it. This can also be necessary if the defense flag is up and the combat manager does not control any units. This *"call to arms"* move where workers are used for defense is commonly used by players facing a rush and often helps avoiding defeat. Although this special case is not essentially relevant to the global design of the agent, it does show that communication between entities on the same hierarchical level in the design is not to be excluded. Another reason the combat manager should communicate with the production manager is the fact that some tasks require resources to be executed. Repairing units for Terrans or training scarabs in a Reaver or interceptors in a Carrier for the Protoss are such tasks. In order to maintain coherence, the combat manager must notify the production manager using, for example, production orders in a manner similar to that used by the strategy manager. Special parameters could be used for such production orders, as they merely serve to reserve resources and do not constitute actual orders executed by the production manager.

4.6 Squad managers

There is no ambiguity in the fact that micro-management currently represents the greatest strength an agent can possess against a human player. While human players struggle to control a dozen units separately, computer players effortlessly control hundreds of units across the entire battlefield. This area represents therefore an opportunity for AI developers looking to quickly build high performance agents.

Each unit in StarCraft has a low-level AI that defines its behavior in the absence of user commands. This AI includes routines for target acquisition in simple cases such as an enemy unit entering sight range, retreat when under attack and unable to retaliate, and also routines to move from one position to another on the map while being subject to terrain constraints. It is, however, primitive and not reliable, which often forces players to manually control units in a more efficient manner. For example, an experienced Zerg player would not send a group of Mutalisks to the enemy's base by giving them the *"attack-move"* command and count on the low-level AI to handle them efficiently. Instead, the Zerg player would control them every step on the way, making sure they do not fly over certain zones and carefully selecting enemy targets according to threat (i.e., High Templars) while avoiding as much enemy fire as possible. The basic AI does none of the above mentioned and thus, it is clear that the efficiency at which units can be used will drastically vary according to the quality of the user input. The default computer players in StarCraft largely discard this fact and rely on the low-level unit AI in combat.

When grouped together, units form a squad and are controlled by a squad manager. The idea of dividing units into groups is originally a compromise. One extreme is to consider all units as a single entity and give them the same orders. The other one is to consider each unit individually and give them different orders. The first approach would obviously yield disastrous results. The second one should theoretically yield the best results and is the only way to achieve optimal behavior. Unfortunately, it gives rise to several complications as units cannot not disregard each other in their behavior. In other words, if units individually acquire optimal moves without taking into account each other's moves, the resulting global behavior is unlikely to be optimal. With a large number of units, conflicting orders are bound to appear. For instance, a retreat order is prone to conflict, as a ground unit may not be able to do so if other ground units are behind it and do not receive a similar order. Consequently, moves must be coordinated across all units which can prove difficult as units are of varied nature and are not subject to the same constraints. In order to fragment this complexity, units are aggregated into squads. As a result, coordination is directed between units in a squad or between squads, but not between units across squads. Of course, the better the squad compositions are, the easier coordination becomes.

There are in fact many aspects to the management of units on the battlefield. Here, we explore some of the possibilities to improve the effectiveness of this task in one-on-one games using the concept of squad managers.

4.6.1 Target acquisition

Setting targets in a squad plays a crucial role in the overall squad efficiency. Heuristics for target selection are varied and can be based on static information, dynamic information or, more often, a combination of both. Examples of static information are a unit's race, type, size or speed. Dynamic information encloses any data that may be altered during a unit's lifetime. HP, energy, units in hangar (i.e., number of interceptors in a Carrier) but also modifiers applied through the use of special abilities such as Defensive Matrix, Plague or Blind as well as upgrades all fall within this second category. The first type of information can be used to select a target based on its inherent attributes. For instance, a Vulture is inherently more effective against a Zealot than it is against a Dragoon because of its range, speed and attack type.[10] A squad may thus detect capital enemy units, or more generally associate a priority to each enemy unit type. Using dynamic information allows squads to compare mutable attributes among units of the same type or across all units. For example, squads may select a close or severely damaged enemy unit. When coupled with static information, dynamic information allows squads to further refine a type-based selection process.

A simple mechanism to account for inherent strengths and weaknesses of units against each other is the use of priority tables. Essentially, each squad possesses a priority table associating a base priority for each type of enemy unit. In a basic implementation, the table can be based on the dominant unit type in a squad. This mechanism makes it easy to dictate target preferences based on type: Marines can prefer Lurkers to Hydralisks while Wraiths target Hydralisks first. It can also be used to make units target key buildings first.

Mutable attributes provide a means to differentiate

[10]The Zealot is a melee unit of small size, while the Vulture is a fast, ranged unit and deals concussive damage. On the other hand, the Dragoon is a large and ranged unit. Recall that concussive damage is fully effective against small units but suffers a 75% reduction against large units.

units regardless of their type and other static information. They can be used by squads to follow heuristics such as *"attack the target with the least HP"* to try to cut enemy firepower as quickly as possible or *"attack the closest target"* to minimize movement. Units that deal normal damage like Marines could indeed target enemy units with the lowest HP[11] as they do not need to worry about damage efficiency.[12] On the other hand, slow units like Reavers may prefer avoiding movement and firing at whatever units in range. Obviously, several parameters can be combined to achieve more accurate behavior (e.g. select the most damaged enemy unit within firing range).[13]

Although both information allow for interesting comparisons, none is usually useful enough on its own. If they are to perform efficiently, squads are bound to take both into account. It is clear that isolated static information cannot yield good selection: just think of the outcome if a squad manager selected a faraway target based on type rather than one of the targets at hand. This is also true for dynamic information and can be illustrated using the Marine squad example mentioned earlier. It is a fact that not all units are equally dangerous when facing Marines. For instance, Lurkers pose a greater threat compared to Hydralisks due to the area of effect of their attack and their ability to attack while burrowed[14] which Hydralisks do not possess. As a result, it is in the squad's best interest to eliminate a Lurker while it is still visible even if it is not the target with the least HP.[15] It is hence clear that the two types of data must figure in the selection process for it to be even reasonably effective. A basic example of such a process is presented in the listing below.

```
for marine in squad
  target = null
  subtarget = null
  for enemy in enemies
    if distance(marine,enemy) <= 128
      if target == null or
           priorities[enemy] > priorities[target] or
             priorities[enemy] == priorities[target] and
             enemy.HP < target.HP
        target = enemy
    else if target == null
      if subtarget == null or
           priorities[enemy] > priorities[subtarget] or
             priorities[enemy] == priorities[subtarget] and
             distance(marine,enemy) < distance(marine,subtarget)
        subtarget = enemy
  if target == null
    target = subtarget
```

Listing 1: Example of target selection process.

[11]Armor may be taken into account.

[12]Normal damage is not constrained by any size-related damage reduction.

[13]For units with an area of effect (AoE) attack such as Corsairs, a squad manager can also check enemy units within damage radius. AoE attacks can damage several units simultaneously, while regular attacks are focused on a single unit.

[14]Burrowed Zerg units can only be seen by detector units like Science Vessels.

[15]When undamaged, Lurkers have 120 HP and Hydralisks have 80 HP.

As can be seen, multiple parameters can be combined in different ways to achieve a particular heuristic. In the given example, targets in firing range are given top priority. Among those, the squad manager will prefer those with the highest priority and of those, it will select the one with the least HP. When no targets are in range, the squad manager will still prefer units with the highest priority but will go after the closest one. This selection process attempts to reduce movement, though it remains inefficient when there are many moderately close targets. If most local threats are very close to firing range, the detection of a far, high-priority enemy unit would cause the squad to ignore the local threats. Hence, the process can be tuned further by setting a secondary distance threshold past which priorities no longer hold any importance.

4.6.2 Behavior

Another important aspect in squad management is behavior. Although target acquisition is in part a result of an adopted behavior, the latter encompasses other elements such as involvement in battles, target sharing or navigation. It depends on a series of settings including the task assigned to the squad, the squad's composition and the current enemies.

Tasks inevitably affect the behavior of squads. Defense squads won't behave in the same way attack squads or reconnaissance squads do. Unlike attack squads, defense squads will not acquire targets too far from the established bases. Similarly, squads assigned to a scouting task will avoid acquiring targets altogether and attempt to preserve their life. It is thus possible to define behavior globally in terms of targeting constraints among other things. These general behavior patterns can then be specialized according to the remaining factors.

Squad compositions can play a significant part in behavior too. Depending on their units, squads will have different sensitivities to parameters involved in target acquisition, most notably distance. It is certain that a squad of Wraiths will weight distance differently compared to a squad of Siege Tanks. A Wraith can quickly close the distance between itself and a target because of its speed but also because it is not constrained by terrain and will therefore weight distance leniently. A Siege Tank is forced to use a strict weight if it is to avoid considerable waste of firepower due to movement. More generally, air units can afford flexible movement policies that are often inaccessible to ground units or that are at least severely limited. A *"fall back-on-hit"* trigger may yield great efficiency for a group of Wraiths because a Wraith can in most cases[16] fall back, forcing the enemy to follow it while the remaining Wraiths continue to attack. Ground units can less often use this technique because falling back is not always an option: other units may be located behind the unit under attack or the terrain behind it may simply not be walkable. This difference in turn affects a squad's predisposition to attack the same target. There are many instances in which it is advantageous for a squad to focus firepower on a target, among which whenever a squad detects the ability to kill an enemy unit with a single, synchronized attack. If the squad's total firepower exceeds an enemy unit's HP, it can constitute an interesting target.[17] This type of synchronization can be very effective with fast, air units like the Wraith. Thus, it is clear that different units are bound to exhibit distinct behaviors, at least if reasonable efficiency is aimed.

Another influential factor is, of course, the enemy a squad is facing. Units in a squad may assume different formations depending on the enemy. This is best illustrated using an example. Consider a group of 12 Wraiths. When the group is facing 2 Hydralisks, Wraiths can assume a stacked formation and make quick work of the Hydralisks one at a time. However, if the group is facing 12 Hydralisks, a stacked formation will certainly result in the destruction of the group because Wraiths cannot withstand that much firepower and will be quickly killed off. A better alternative is to use a scattered formation in which each Wraith attempts to stay out of every Hydralisk's attack range. A Wraith's ground weapon range is superior to a Hydralisk's air attack range.[18] Therefore, it is possible for a Wraith to use a *"hit-and-run"* technique against a Hydralisk. This second formation is more effective and can often turn the tide of the otherwise lost encounter. As an extra, we present below a simple algorithm to keep the Wraiths out of the enemy's attacks.

```
for wraith in squad
  target = null
  threats.clear()
  for hydralisk in enemies
    if distance(wraith,hydralisk) <= 128
      threats.add(hydralisk)
    if target == null or
        distance(wraith,hydralisk) < distance(wraith,target)
      target = hydralisk
  if not threats.empty()
    target = 2*wraith.position - threats.center()
```

Listing 2: Example of squad behavior prioritizing survival over attack.

Sometimes, squads may require to avoid dangerous areas altogether. This is frequent when attacking the enemy's base. The enemy could have set defensive structures and a squad may simply not be able to stand its ground against these constructions. Moreover, the workers located inside the base may constitute high-priority targets. A squad will thus want to bypass these buildings to reach the workers. Human players can avoid such targets intuitively. They may either control the squad while keeping a large distance between it and the structure or carefully control each unit when the former option is not available. For agents, this is not trivial to implement. It may nonetheless be possible to achieve with ease by forcefully altering the map state during the game: by reversibly rendering the tiles covered by the structure's attack radius non-walkable, a squad will be forced to avoid the dangerous zone without the need of any path planning. Units ordered to attack a worker will then find their way through the base as they would through difficult terrain or through crowded areas using their low-level AI. Obviously, for this to work, it must be possible to redirect map state queries to a copy of the map, as the alteration must not affect other squads and most importantly not be echoed on the opponent's game state. Wary game developers may choose to implement such mechanisms.

As we can see, there are many elements involved in behavior and the richer the implementation, the more likely squads are to perform better.

[16]Unless located on a map edge

[17]This instance can be generalized to the concept of optimal firepower distribution. Distributing firepower is useful when firepower waste can potentially be high. When two groups face each other, if one group focuses on a single unit in the opposing group, firepower waste can occur if the group's total firepower far exceeds the target's HP. In that case, it is interesting to distribute firepower in order to kill as many enemy units as possible during the time of a single attack.

[18]Without the Grooved Spines upgrade which increases a Hydralisk's attack range

4.6.3 Unit handlers

When squads compositions are mixed (i.e., not all units in the squad are of the same type), it can be interesting to use separate handlers for each unit type. This applies when a squad includes one or more caster units.[19] These units typically require more complex processing and, often, inter-squad communication. For instance, some degree of synchronization is necessary when selecting a target for a Ghost's Lockdown ability. When used, this ability causes the Ghost to fire a missile at the target which acquires the *"locked down"* attribute when it is hit by the missile. Hence, Ghosts must be coordinated in order to avoid multiple casts on the same target, a likely event given the lack of any markup during the period between the moment the Ghost uses the ability and the moment the target acquires the corresponding attribute. Another such example is the use of a High Templar's Psionic Storm ability. When cast over a target area, this ability causes all units within the area to gradually lose HP over time. Psionic Storms cannot be stacked to increase damage. High Templars must therefore coordinate their casts to hit the highest enemy density areas[20] while avoiding any overlaps. Thus, this added complexity may be isolated in a sub-handler.

Unit handlers may also be used when units in a squad are acting on their own. As we have already seen, squad managers can launch a synchronized attack on a specific target. However, under other behavior policies such as the *"hit-and-run"*, scattered formation mentioned in the previous section, units move and acquire targets[21] on their own and their processing can be relegated to a sub-handler. This way, squad-level decisions are clearly separated from independent unit-level decisions.

4.6.4 Production squads

We have focused on military squads because they are the predominant type. Managing production squads involves much fewer aspects than managing combat squads does and is as a result relatively easy.

[19]Caster units usually refer to units primarily used for their special abilities. Examples include the Ghost with its Cloaking, Lockdown and Nuclear Strike abilities or the High Templar with its Psionic Storm and Hallucination abilities.

[20]This may require a certain degree of prediction.

[21]Typically, the closest enemy unit

In mining squads, squad managers will have to perform tasks such as evenly distributing workers on mineral fields or sending a worker to the closest unoccupied mineral field. Squads can be given resource targets which squad managers will attempt to meet. For example, the production manager may set a 500 minerals, 50 vespene gas target in a squad. The squad manager will then distribute workers on mineral fields and refineries in such a way that the objective is achieved in a minimum amount of time.

For construction squads, tasks could include preemptively placing workers over construction sites so that pending construction jobs assigned to the squad may be started as soon as resource or technological requirements are met.

Lastly, production squads may be tied to a specific base in order to limit the scope of their activities.

5 Implementation and test results

StarCraft: Brood War is not open source but, due to the game's popularity and age, people have managed to open it up and figure out the memory addresses of the different objects in the game (i.e., state, units, attributes, ...) and eventually, a framework was built on top of this *"hacking"*.

5.1 The Brood War Application Programming Interface

The Brood War Application Programming Interface (BWAPI) is an open source C++ framework which allows AI developers to create custom agents by providing them with hooks to access the game state and issue commands. More information regarding the features and the design of the API can be found on the project's web page.[22]

5.2 The BWAPI Standard Add-on Library

The BWAPI Standard Add-on Library (BWSAL) is a set of add-ons for the BWAPI used to perform common tasks such as resource gathering or unit production. A list of the available components figures on the project's web page.[23]

[22]`http://code.google.com/p/bwapi/`

[23]`http://code.google.com/p/bwsal/`

5.3 The Brood War Terrain Analyzer

The Brood War Terrain Analyzer (BWTA) is an add-on capable of analyzing StarCraft maps and detecting key areas such as expansion sites or choke points. This information can then be used by agents during games to take into consideration map layouts. A complete description can be found on the project's web page.[24]

5.4 Implementation

A basic, limited agent based on the approach proposed in this paper was implemented to run some preliminary tests. The agent was built using the BWAPI as well as the mentioned add-ons. It does not strictly follow the presented design as the production part is largely handled by the modules provided by the BWSAL. Given the mechanical nature of the production plane, this has little effect on the agent's global behavior and is mostly transparent. Modules for strategy, combat and squad managers were developed and successfully integrated in the design. The agent is currently limited to Terran (agent) versus Zerg (opponent) games. It has been tested against Blizzard Entertainment's default Zerg AI.[25] A general, more complete implementation has yet to be evaluated against human opponents.

5.5 Results

The resulting agent is capable of playing full games as a Terran player facing a Zerg opponent. We present below some statistics as well as some screen captures[26] at the end of the document taken from several games played by the agent. A video displaying the agent's performance can be watched on http://www.youtube.com/watch?v=bZvFUsIAUnQ.

As can be seen in Tables 1 and 2, the agent controls units at a striking efficiency compared to the default Zerg AI. It managed to score an average of 3 units lost in 10 games versus the opponent's 84.5 average. It even succeeded in winning without losing a single unit in a couple of games.

[24]http://code.google.com/p/bwta/

[25]Remember that the default AIs in StarCraft are all cheating in that they have access to the entire game state. On the other hand, our agent plays the game fairly and only has access to enemy information it collects through reconnaissance.

[26]Debugging information has purposely been left on the images to reveal some details concerning the state of the agent.

Game	A	B	C	D	E
Units produced	73	62	72	69	68
Units killed	81	79	78	75	72
Units lost	10	2	0	0	1

Game	F	G	H	I	J
Units produced	77	76	63	85	78
Units killed	100	101	83	74	102
Units lost	2	1	0	13	1

Table 1: Unit statistics.

AUP	AUK	AUL	Games won	Total games
72.3	84.5	3	10	10

Table 2: Average units produced, killed and lost.

6 Conclusions and Future Work

In this paper, we have presented a task-specific hierarchical agent design for StarCraft: Brood War. Although the agent was developed based on the concepts present in StarCraft, the game's complexity and quality are such that the design is general enough and can easily be extended to other RTS games.

The results we have obtained demonstrate that the design we propose can be effective. During our analysis of the game and throughout the development of the design, we have realized that albeit being theoretically less important than strategy, unit management on the battlefield is a key factor to the success of an agent and can boost its performance considerably. Much to our expectation, the effect of focusing on the lower levels in our task-specific hierarchical design was extensively displayed in our tests.

As we complete our implementation, we turn to one of the ulterior motives which sparked our interest in AI research applied to RTS games: how can we create learning agents in such complex environments? By fragmenting complexity and clearly identifying and isolating tasks, it becomes possible to introduce learning aptitudes at different levels, notably in the individual managers. From strategic decisions such as countering an attack with the appropriate units to unit handling such as dynamically tuning the distance at which it is best to stay from the enemy all the way through intermediate management like primary target acquisition or formation switching, learning opportunities are seemingly numerous and varied in kind. It will be particularly interesting to examine how learning can affect the behavior of an agent and whether it can turn the odds against human players.

Of course, game developers have entirely different objectives and may not be interested in enhancing their AI routines unless it is deemed profitable. From a business point of view, it can seem like a waste of resources to improve the AI in a game if it ultimately fails to offer serious competition. Usually, in a RTS game, even novice players are able to spot and start exploiting weaknesses in an agent by just playing a few games against it. Unless the agent can adapt to the behavior of its opponents as quickly as they can, it will fail to provide a durable challenge and this is why human players end up turning to multiplayer options. Thus, without expecting a breakthrough in the field, game producers may not always want to grant AI development more resources than required for basic operation.

7 Legal Notice

References

Michael Buro and Timothy M. Furtak. RTS games and real-time AI research. In *Proceedings of the 13th Behavior Representation in Modeling and Simulation Conference (BRIMS-04)*, pages 51–58, 2004.

Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Case-Based Planning and Execution for Real-Time Strategy Games. In *Proceedings of the 7th International Conference on Case-Based Reasoning (ICCBR-07)*, pages 164–178, 2007.

David W. Aha, Matthew Molineaux, and Marc J. V. Ponsen. Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game. In *Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR-05)*, pages 5–20, 2005.

Ben Weber and Michael Mateas. Case-Based Reasoning for Build Order in Real-Time Strategy Games. In *Proceedings of the 5th Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE-09)*, 2009a.

Ben G. Weber and Michael Mateas. Conceptual Neighborhoods for Retrieval in Case-Based Reasoning. In *Proceedings of the 8th International Conference on Case-Based Reasoning (ICCBR-09)*, pages 343–357, 2009b.

Ben G. Weber and Michael Mateas. A data mining approach to strategy prediction. In *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG-09)*, pages 140–147. IEEE Press, 2009c.

Marc Ponsen, Héctor Muñoz-Avila, Pieter Spronck, and David Aha. Automatically Generating Game Tactics via Evolutionary Learning. *AI Magazine*, 27(3):75–84, 2006.

Michael Chung, Michael Buro, and Jonathan Schaeffer. Monte Carlo Planning in RTS Games. In *Proceedings of the 1st IEEE Symposium on Computational Intelligence and Games (CIG-05)*, 2005.

Samuel Wintermute, Joseph Xu, and John E. Laird. SORTS: A Human-Level Approach to Real-Time Strategy AI. In *Proceedings of the 3rd Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE-07)*, pages 55–60, 2007.

Josh McCoy and Michael Mateas. An Integrated Agent for Playing Real-Time Strategy Games. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 1313–1318, 2008.

Joseph Bates, A. Bryan Loyall, and W. Scott Reilly. Integrating Reactivity, Goals, and Emotion in a Broad Agent. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, 1992.

Michael Buro. Real-Time Strategy Games: A New AI Research Challenge. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1534–1535, 2003.

Danny C. Cheng and Ruck Thawonmas. Case-Based Plan Recognition for Real-Time Strategy Games. In *Proceedings of the 5th International Conference on Computer Games: Artificial Intelligence, Design and Education (CGAIDE-04)*, pages 36–40, 2004.

Paul R. Cohen. Empirical Methods for Artificial Intelligence. *IEEE Expert*, 11(6):88, 1996.

Alex Kovarsky and Michael Buro. A First Look at Build-Order Optimization in Real-Time Strategy Games. In *Proceedings of the 2006 GameOn Conference*, pages 18–22, 2006.

Blizzard Entertainment: StarCraft.
`http://www.blizzard.com/us/starcraft/`.

GosuGamers.
`http://www.gosugamers.net`.

iCCup – International Cyber Cup.
`http://www.iccup.com`.

StarCraft: Legacy.
`http://sclegacy.com`.

StarCraft Wiki.
`http://starcraft.wikia.com`.

Team Liquid.
`http://www.teamliquid.net`.

WGTour.
`http://www.wgtour.com`.

Glossary

Battlecruiser
The Terran air flagship capable of single-handedly destroying enemy squadrons, also equipped with a powerful, long-range nuclear weapon. 7

Carrier
A heavy Protoss command ship that shelters small interceptors automatically launched whenever enemy units are nearby. 5, 10, 11

Corsair
A Protoss anti-air spacecraft with the ability to disrupt the targeting abilities of ground units, effectively halting their attacks. 9, 12

Dragoon
A basic Protoss ranged ground unit. 6, 11

Firebat
A Terran infantry trained for close quarters combat (CQC). 6

Ghost
A specialized Terran infiltration unit capable of cloaking, disrupting mechanical units and calling down nuclear strikes. 5, 7, 14

Goliath
A Terran battle mech most notably used for its powerful Hellfire anti-air scatter missiles. 5

High Templar
A specialized Protoss psionic spellcaster capable of laying waste to entire armies with its deadly psionic storm ability as well as deceiving the enemy by creating illusionary unit duplicates. 11, 14

Hydralisk
A Zerg ranged and versatile fighter with the ability to mutate into a specialized anti-ground unit. 11–13

Lurker
A specialized Zerg anti-ground unit that can attack while burrowed. 11, 12

Marine
The basic albeit versatile Terran infantry whose efficiency can be temporarily boosted at the cost of some hit points. 5, 10–12

Medic
A Terran support infantry with the ability to heal, remove harmful statuses and blind enemy units. 5, 10

Mutalisk
A speedy Zerg air fighter whose attack can ricochet on nearby targets. 11

Reaver
A heavy Protoss robotic ground unit that can build and fire explosive scarabs on the battlefield. 10, 12

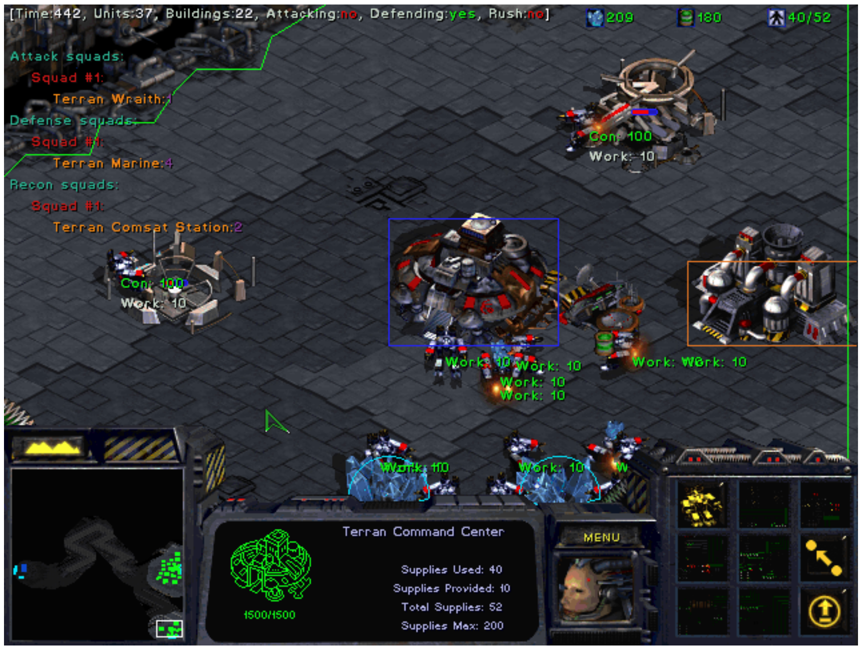
Science Vessel
A Terran support spaceship capable of deploying a temporary shield on friendly units, casting an EMP shockwave and irradiating biological enemy units. 6, 12

Siege Tank
A Terran armored tank which excels at both attack and defense thanks to its ability to deploy to siege mode. 13

Vulture
A fast Terran hoverbike that can lay spider mines. 11

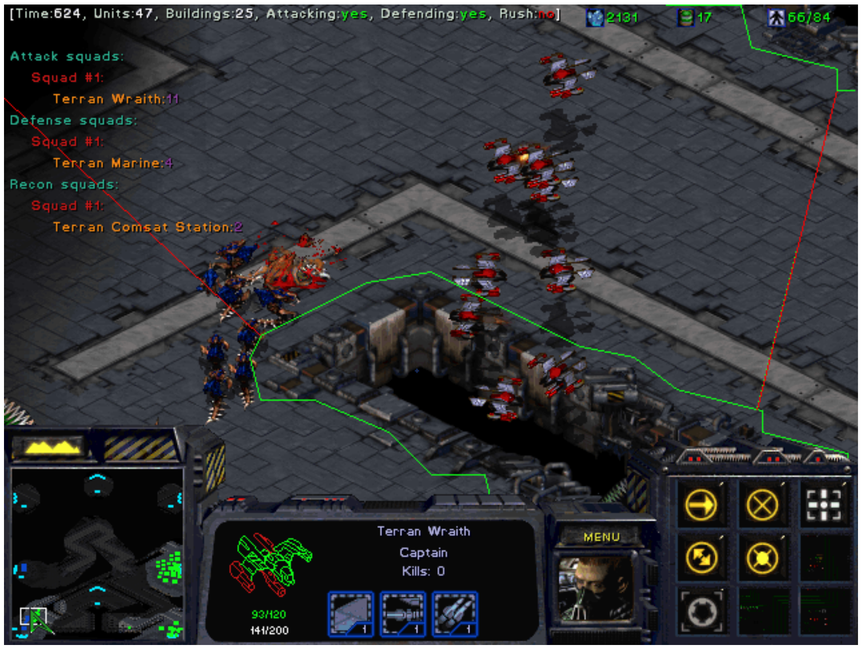
Wraith
A Terran air fighter with the ability to cloak. 8, 11–13

Zealot
The basic yet tough Protoss frontline infantry. 6, 11



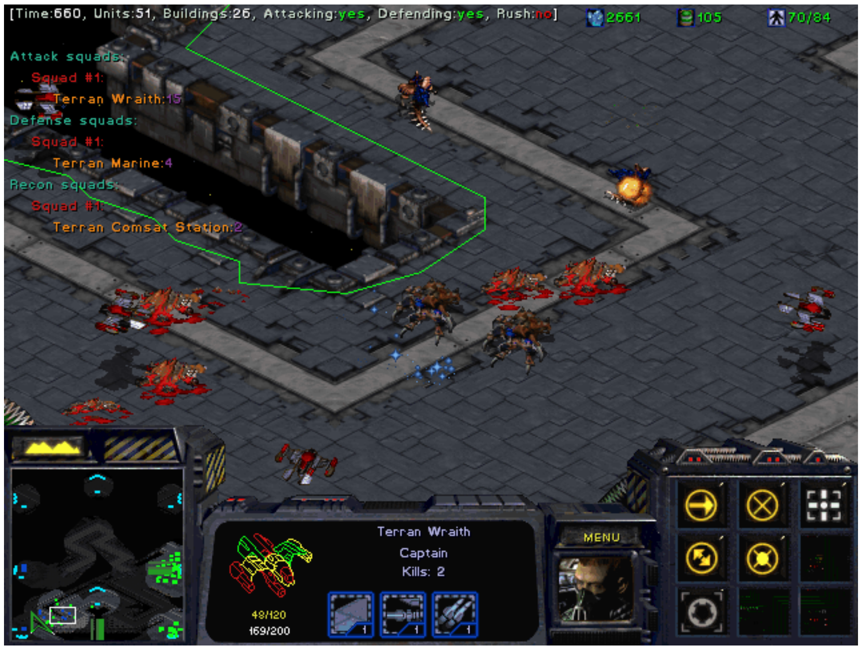
```
[Time:442, Units:37, Buildings:22, Attacking:no, Defending:yes, Rush:no]
Attack squads:
  Squad #1:
    Terran Wraith:1
Defense squads:
  Squad #1:
    Terran Marine:4
Recon squads:
  Squad #1:
    Terran Comsat Station:2
```

Figure 5: GS0121 - Agent establishing a nearby outpost.



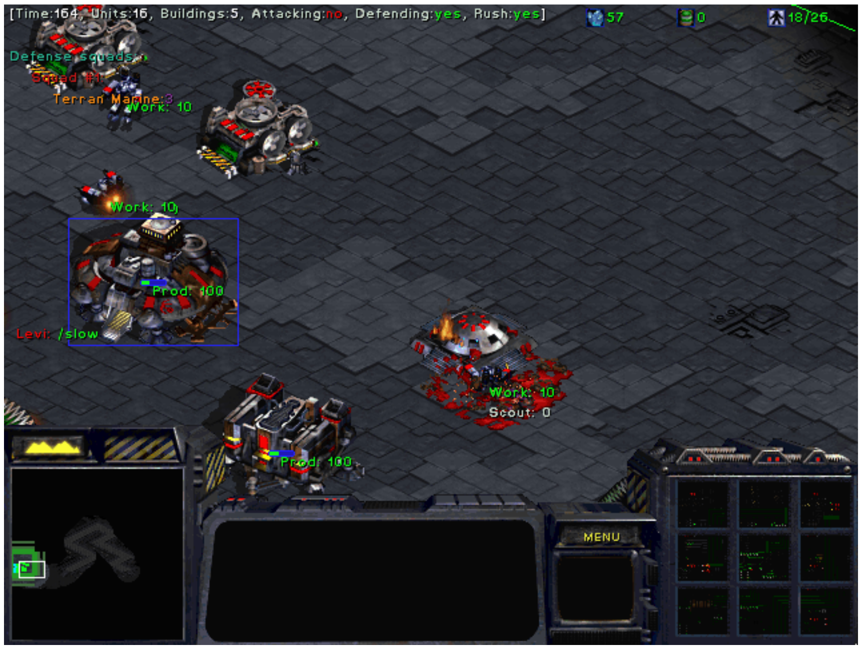
```
[Time:624, Units:47, Buildings:25, Attacking:yes, Defending:yes, Rush:no]
Attack squads:
  Squad #1:
    Terran Wraith:11
Defense squads:
  Squad #1:
    Terran Marine:4
Recon squads:
  Squad #1:
    Terran Comsat Station:2
```

Figure 6: GS0122 - Agent attacking the opponent's secondary base.



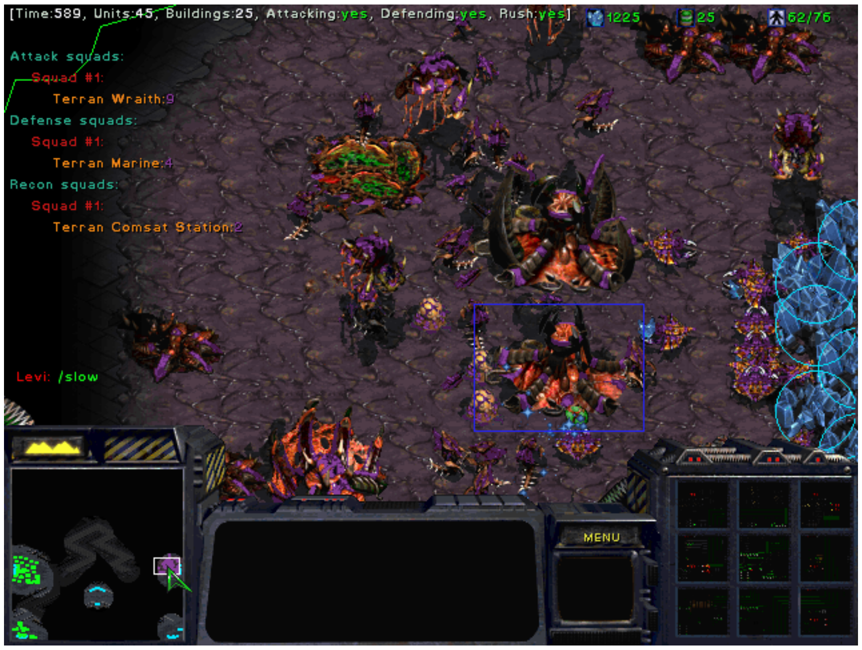
```
[Time:660, Units:51, Buildings:26, Attacking:yes, Defending:yes, Rush:no]
Attack squads:
  Squad #1:
    Terran Wraith:15
Defense squads:
  Squad #1:
    Terran Marine:4
Recon squads:
  Squad #1:
    Terran Comsat Station:2
```

Figure 7: GS0123 - Scanner Sweep used by recon squads on burrowed Lurkers.



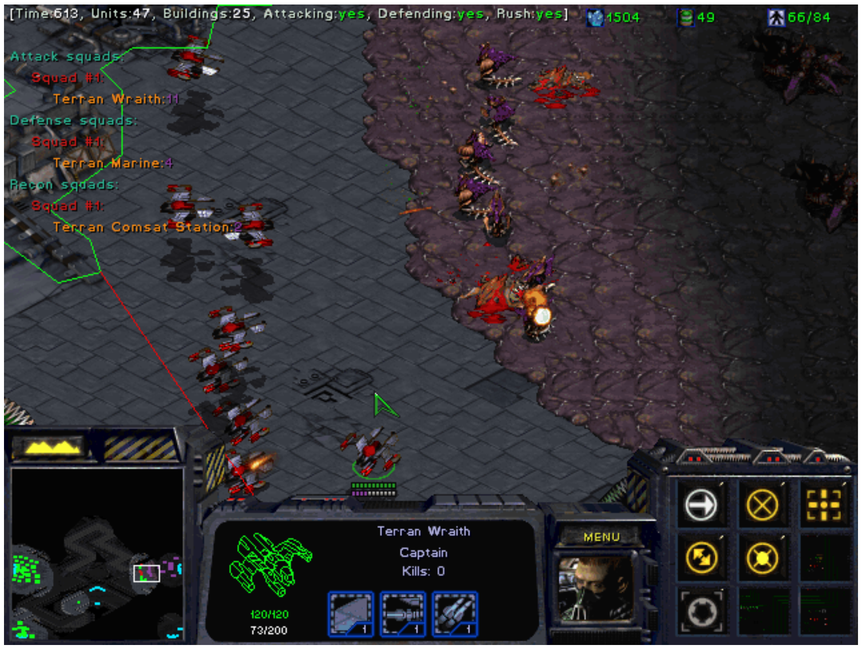
```
[Time:164, Units:16, Buildings:5, Attacking:no, Defending:yes, Rush:yes]
Defense squads:
  Squad #1:
    Terran Marine:3
```

Figure 8: GS0271 - Agent successfully defending against Zergling rush.



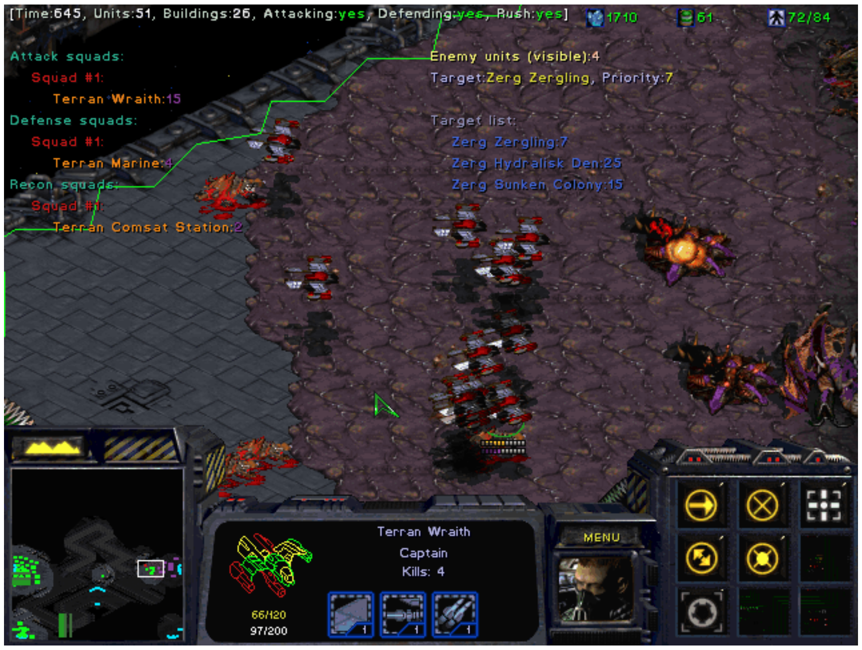
```
[Time:589, Units:45, Buildings:25, Attacking:yes, Defending:yes, Rush:yes]
Attack squads:
  Squad #1:
    Terran Wraith:9
Defense squads:
  Squad #1:
    Terran Marine:4
Recon squads:
  Squad #1:
    Terran Comsat Station:2

Levi: /slow
```

Figure 9: GS0272 - Recon squad locating the enemy's primary base.



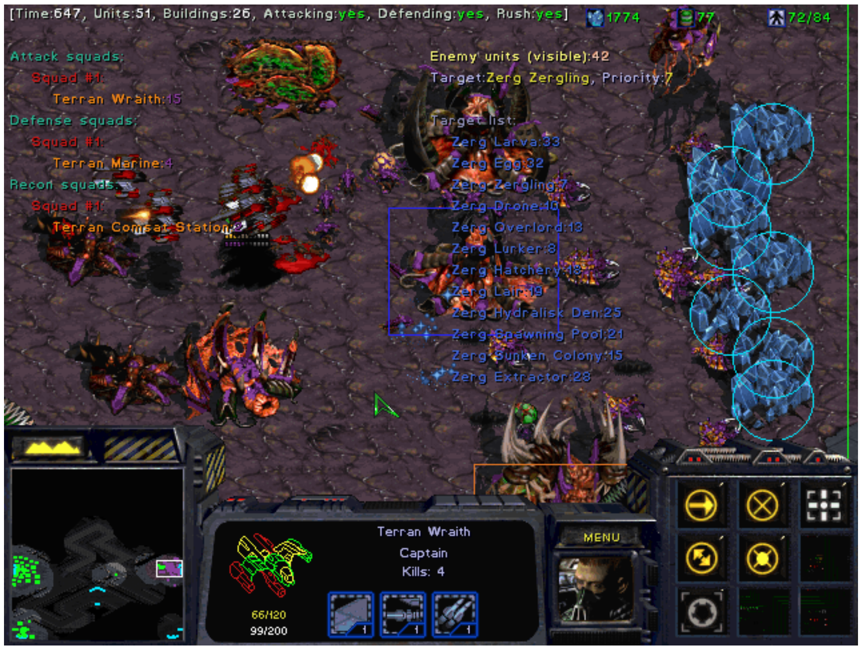
```
[Time:613, Units:47, Buildings:25, Attacking:yes, Defending:yes, Rush:yes]
Attack squads:
  Squad #1:
    Terran Wraith:11
Defense squads:
  Squad #1:
    Terran Marine:4
Recon squads:
  Squad #1:
    Terran Comsat Station:2
```

Figure 10: GS0273 - Wraiths avoiding enemy fire.



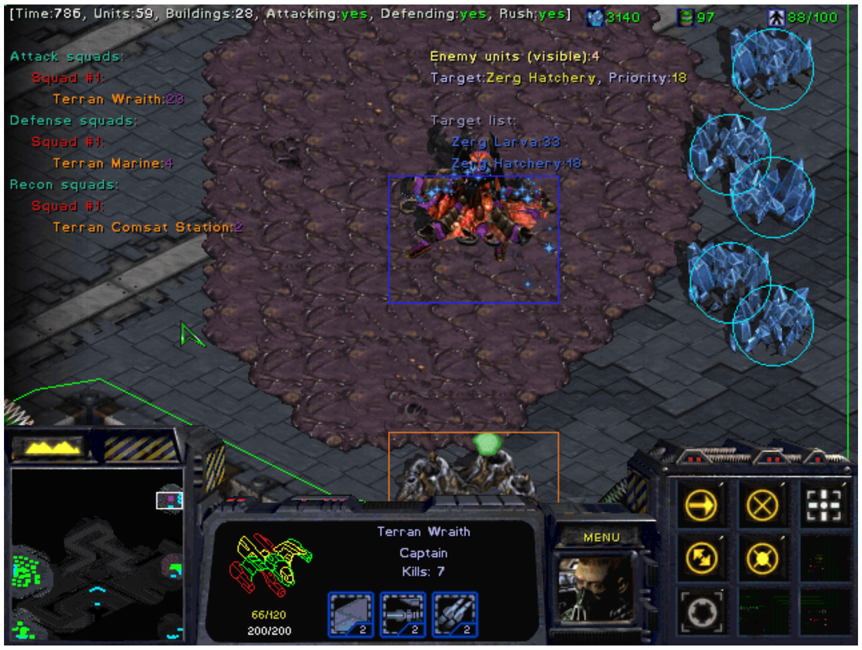
```
[Time:645, Units:51, Buildings:26, Attacking:yes, Defending:yes, Rush:yes]

Attack squads:
   Squad #1:
      Terran Wraith:15
Defense squads:
   Squad #1:
      Terran Marine:4
Recon squads:
   Squad #1:
      Terran Comsat Station:2

Enemy units (visible):4
Target:Zerg Zergling, Priority:7

Target list:
   Zerg Zergling:7
   Zerg Hydralisk Den:25
   Zerg Sunken Colony:15
```

Figure 11: GS0274 - Wraiths switching to a stacked formation.



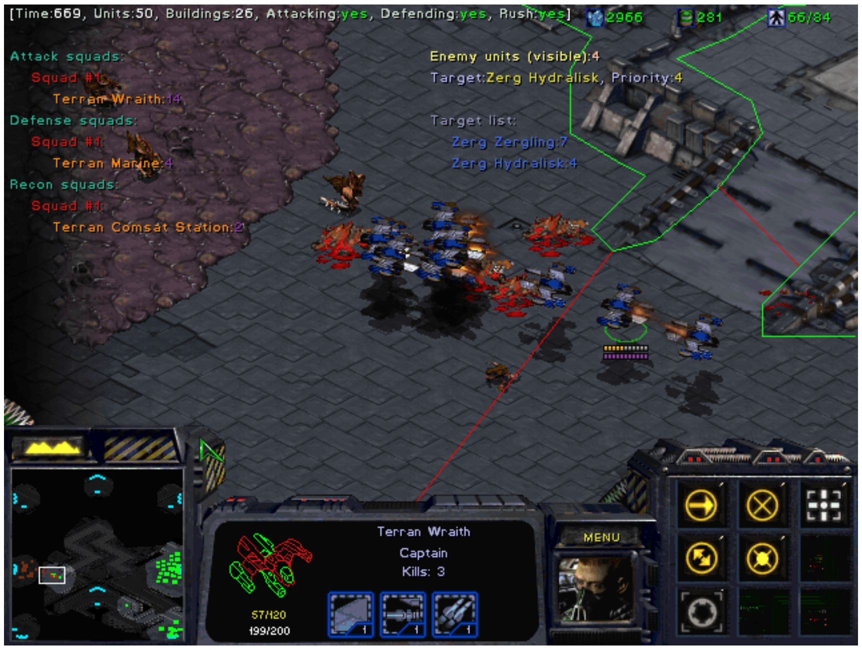
```
[Time:647, Units:51, Buildings:26, Attacking:yes, Defending:yes, Rush:yes]

Attack squads:
   Squad #1:
      Terran Wraith:15
Defense squads:
   Squad #1:
      Terran Marine:4
Recon squads:
   Squad #1:
      Terran Comsat Station:2

Enemy units (visible):42
Target:Zerg Zergling, Priority:7

Target list:
   Zerg Larva:33
   Zerg Egg:32
   Zerg Zergling:7
   Zerg Drone:10
   Zerg Overlord:13
   Zerg Lurker:8
   Zerg Hatchery:18
   Zerg Lair:19
   Zerg Hydralisk Den:25
   Zerg Spawning Pool:21
   Zerg Sunken Colony:15
   Zerg Extractor:28
```

Figure 12: GS0275 - Wraiths in the opponent's primary base taking out units according to priority.



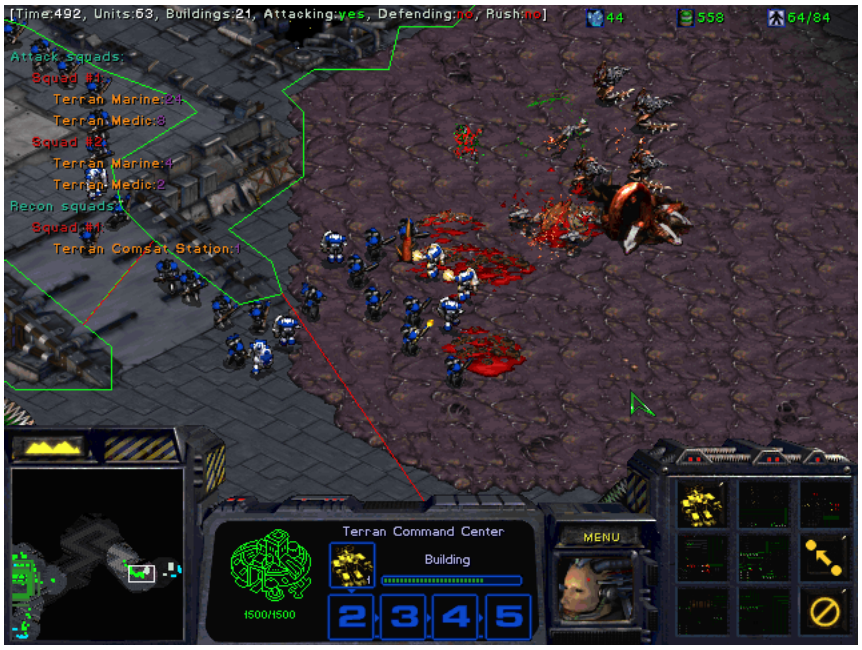
```
[Time:786, Units:59, Buildings:28, Attacking:yes, Defending:yes, Rush:yes]   3140   97   88/100

Attack squads:                                  Enemy units (visible):4
   Squad #1:                                    Target:Zerg Hatchery, Priority:18
      Terran Wraith:23
Defense squads:                                 Target list:
   Squad #1:                                       Zerg Larva:33
      Terran Marine:4                              Zerg Hatchery:18
Recon squads:
   Squad #1:
      Terran Comsat Station:2
```

Figure 13: GS0276 - Enemy outpost picked up by recon squad after the primary based has been destroyed.



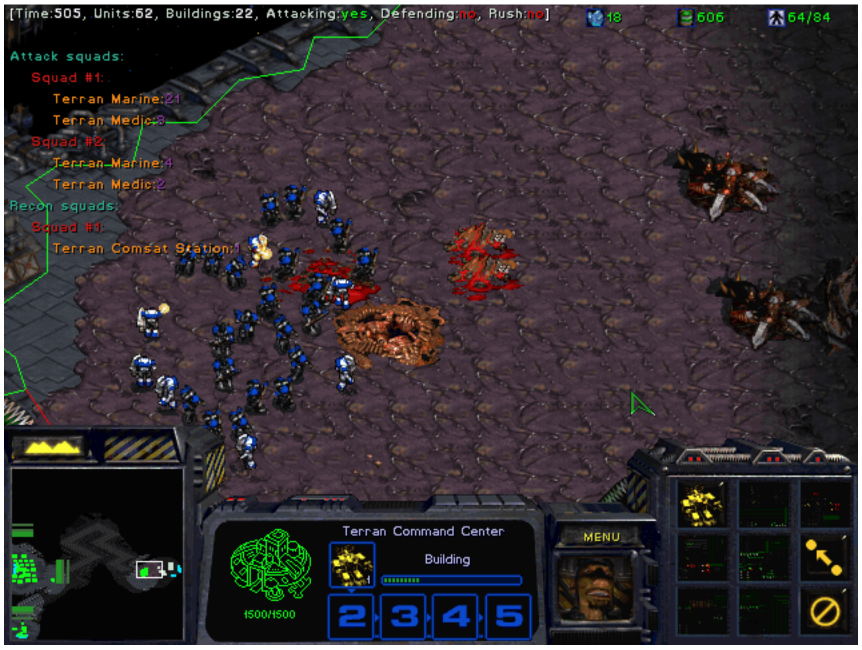
```
[Time:669, Units:50, Buildings:26, Attacking:yes, Defending:yes, Rush:yes]   2966   281   66/84

Attack squads:                                  Enemy units (visible):4
   Squad #1:                                    Target:Zerg Hydralisk, Priority:4
      Terran Wraith:14
Defense squads:                                 Target list:
   Squad #1:                                       Zerg Zergling:7
      Terran Marine:4                              Zerg Hydralisk:4
Recon squads:
   Squad #1:
      Terran Comsat Station:2
```

Figure 14: GS0341 - Fall back-on-hit trigger during stacked formation.



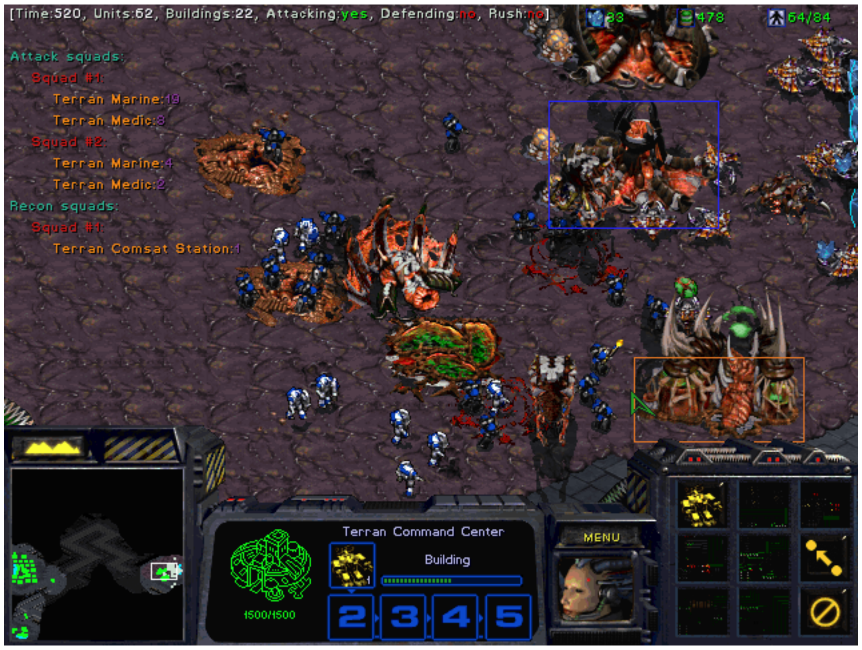
```
[Time:492, Units:63, Buildings:21, Attacking:yes, Defending:no, Rush:no]

Attack squads:
   Squad #1:
      Terran Marine:24
      Terran Medic:8
   Squad #2:
      Terran Marine:4
      Terran Medic:2
Recon squads:
   Squad #1:
      Terran Comsat Station:1
```

Figure 15: GS0391 - Agent using alternate strategy.



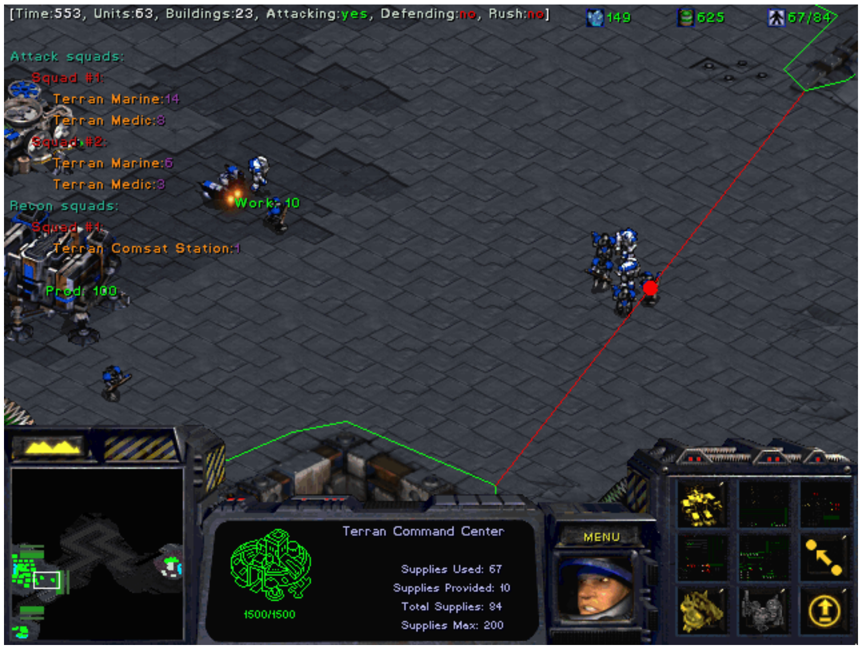
```
[Time:505, Units:62, Buildings:22, Attacking:yes, Defending:no, Rush:no]

Attack squads:
   Squad #1:
      Terran Marine:21
      Terran Medic:8
   Squad #2:
      Terran Marine:4
      Terran Medic:2
Recon squads:
   Squad #1:
      Terran Comsat Station:1
```

Figure 16: GS0392 - Front defenses breached.



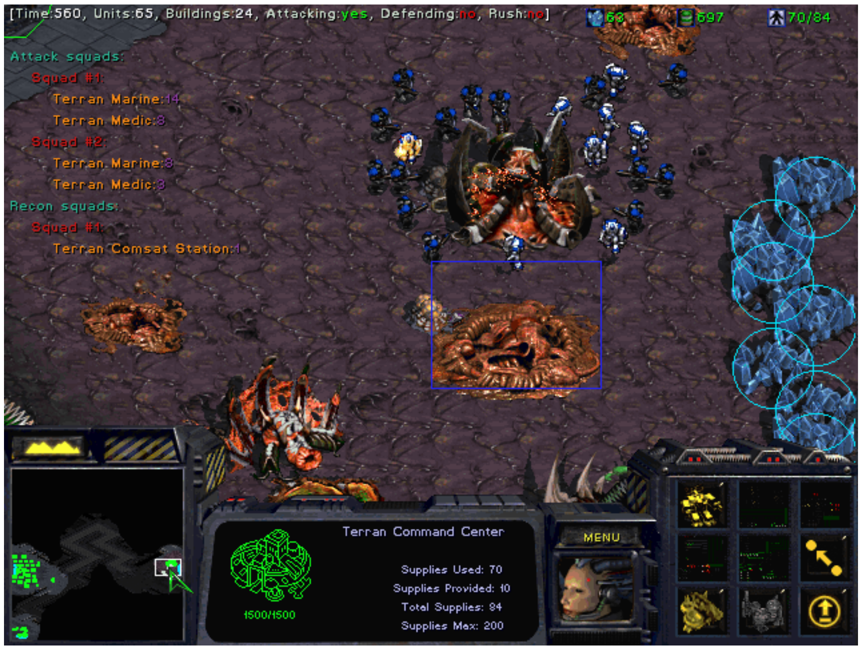
```
[Time:520, Units:62, Buildings:22, Attacking:yes, Defending:no, Rush:no]

Attack squads:
   Squad #1:
      Terran Marine:19
      Terran Medic:8
   Squad #2:
      Terran Marine:4
      Terran Medic:2
Recon squads:
   Squad #1:
      Terran Comsat Station:1
```

Figure 17: GS0393 - Marines running for the Lurkers.



```
[Time:553, Units:63, Buildings:23, Attacking:yes, Defending:no, Rush:no]

Attack squads:
   Squad #1:
      Terran Marine:14
      Terran Medic:8
   Squad #2:
      Terran Marine:6
      Terran Medic:3
Recon squads:
   Squad #1:
      Terran Comsat Station:1
```

Figure 18: GS0394 - Meanwhile, new squad awaits completion.



```
[Time:560, Units:65, Buildings:24, Attacking:yes, Defending:no, Rush:no]

Attack squads:
   Squad #1:
      Terran Marine:14
      Terran Medic:8
   Squad #2:
      Terran Marine:8
      Terran Medic:3
Recon squads:
   Squad #1:
      Terran Comsat Station:1
```

Figure 19: GS0395 - Agent finishing off the opponent.