CMPUT 350 Project Report

Motivation

Our goal for this project was to develop a bot for StarCraft 2 using the StarCraft C++ API with the intention to make a defensive build. As none of our group members have prior experience with StarCraft 2, we hoped that this project would allow us to learn about the game and its strategies, and the experience of developing this bot. We also hoped to apply our knowledge of algorithms and C++ to create this program. The main strategy we wished to implement was one that relied on maintaining influence of the map through several node objects to maintain control of our various bases, hoping for a bot that could live long and maintain defensive capabilities.

Prior Work

There have been research papers related to StarCraft 2, researching many of the different components to accommodate an automated strategy of a StarCraft bot.

Churchill, D., & Buro, M. (2011). Build Order Optimization in StarCraft. *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Retrieved from https://www.aaai.org/ocs/index.php/AIIDE/AIIDE11/paper/view/4078

Discusses algorithms to help determine and automate the optimal build order to determine the shortest sequence of actions required to achieve a particular goal. Particularly such that they would be automated by a bot playing the game in real time, allowing it to make decisions that would allow it to accomplish a particular task.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., … Tsing R. (2021). Starcraft II: A New Challenge for Reinforcement Learning. *ResearchGate.* Retrieved from https://www.researchgate.net/publication/319151530_StarCraft_II_A_New_Challenge_for_Reinforcement_Learning

Discusses the challenges of using StarCraft II as a reinforcement learning target because of its number of actions, the depth of its state tree, and its status as an incomplete information game.

Čertický, M., & Churchill, D. (2021). The Current State of StarCraft AI Competitions and Bots. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, *13*(2), 2-7. Retrieved from https://ojs.aaai.org/index.php/AIIDE/article/view/12961

Goes over some AI competitions and tournaments and some of the strategies of some of the bots that were entered into them. These include bots that use evolutionary algorithms, reinforcement learning, or hierarchical task networks to determine build and action sequences, leading to very unique and interesting Machine Learning-based decision-making.

Our Approach

General

We had decided to choose the Zerg race for our project. The strategy for our bot involves mostly around map control, and abusing the low individual unit cost that the Zerg class offers. Our bot's main goal is to spread itself across the map, increasing production rates of units and by this to increase our attack power. Because the Zerg have access to high mobility while over creep, this strategy also allows units to be distributed across the map, and still have easy access to nodes that need defending. Initially, we planned on only using zerglings for their low unit cost and high mobility, but we discovered that towards the late game this strategy broke down because of an inability to hit flying units, and also the weak damage of the zerglings. Because of this, we transitioned to training more powerful units, and units with the ability to hit flying enemies.

Bot Structure

The core of the bot is the BasicSc2Bot class. Within the BasicSc2Bot class is a vector of the class Node. The BasicSc2Bot class interfaces with the StarCraft 2 API, and acts as a bridge between the API and the various nodes we have created. Each Node instance carries out the work done in a single base, acting as a "sub-bot", partitioning the work done by the "dom-top". This allows the program to be modular, so that the bot's territory can expand easily. Tags are used to assign units to each corresponding base in a vector within the node class.

As the Node class cannot directly access the StarCraft 2 API, the BasicSc2Bot class also updates the information known by each node for each update. Any time a unit is created it is added to the closest node, unless that unit is itself a hatchery, which gets added to its own new node.

Base Creation

To build up the base, the node class generates new units and buildings according to conditions set in the program per update. Buildings are placed on a random location

located somewhere in the vicinity of the base. These buildings will also be in the unit vector of a certain node. Nearby vespene gas locations are also obtained, and extractors are developed onto it for each base if possible.
Throughout the game, the number of units for each unit type related to each node is tracked. This number helps determine what buildings need to be constructed, and how many units to make and when to send out attacks.

Expansion

To expand territory, a new hatchery is built when the necessary conditions are met. Building a new hatchery will create a new node object, in which the new hatchery will be the center of. Hatcheries must be placed in locations nearby vespene gas and minerals to ensure efficiency. So, for each map, predefined locations are used to place hatcheries onto. As the map is rotationally symmetrical, only saving the displacement of each hatchery location from the starting location was necessary.

Unit Control

To control units, each Node delegates orders to the units related to itself. In each update, all the units under that node are looped through and given their “regular task” when needed. For example, larva will morph into a certain unit and queens will generate larva. However, these tasks can be overwritten by special tasks, such as attacking, moving defense, and creating a new base.

The units we had decided to use for the bot include:

- Drones for collecting materials, and transforming into buildings.
- Overlords to increase food capacity.
- Zerglings for scouting and early-game defense.
- Spine Crawlers and Spore Crawlers for defending the bases against attacks.
- Queens, which create creep tumors, generate larva and defend.
- Roaches, mutalisks, hydralisks, corruptors, and ultralisks are used as parts of the army that will attack the enemy base.

The buildings necessary to create these units are constructed throughout the game. Unit morphing is prioritized such that more advanced units able to be created will be created first (given that there are enough materials). A balance of ground and flying units is ensured.

Drones are also managed per each node in order to balance the amount collecting minerals and the amount collecting vespene gas. Drones always obtain minerals from the nearest mineral deposit from its respective base.

At the beginning of the game, we send a single zergling to scout out the map initially in order to locate the location of the enemy base. The zergling checks all the possible enemy spawn points, and if the zergling is killed, it is determined that the base is near that location. This information will then be used when attacking. A scout Zergling will continue to be sent around the map for the remainder of the game, to discover any other enemy bases that could be located at expansion nodes.

In order to ensure that we actually produce early game units in the early game, and late game units in the late game, we keep track of the time each node has existed for. We use this value to set a variable to keep a minimum amount of minerals around when we create a new unit. Depending on how high the cost is, and how important to the late game the unit is, this minimum can be subverted. In this way, low value unit production in the late game is kept to a minimum. For example, we may only be able to build a zergling if it would leave us with 300 minerals left over, which is enough to make roaches and hydralisks as well.

Defense

In order to create a strong defense, we had decided that units should move to the location being attacked to ensure that the enemies are quickly defeated without destroying too many of our bases' structures. To implement this mobile defense in our bot, the node which has received the most amount of damage (the ratio of the hatchery's current health and max health) is determined per update within the BasicSc2Bot class. Whenever the most damaged Node has changed, defense units, consisting of spine crawlers, spore crawlers and zerglings, are moved to the location of the corresponding node. Defense is also moved whenever a new base is constructed as these bases are closer to the center of the map, and thus are more likely to be attacked.

Offense

In order to attack the enemy base, we decided to attack early and often. In the early game stage, we gather small groups of army units to disturb the enemy's economic development. While doing so, our base prepares for the later major attacks by producing stronger army units and investing in researching unit upgrades.

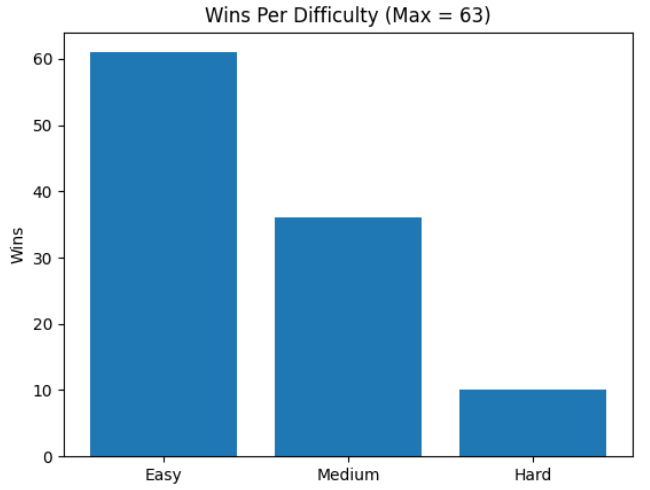
The Ambush function, which is called when there are enough army units, sends an army of a certain amount towards the expected enemy location. The SearchAndAmbush function is called when the number of attackers is above 50, and sends an all-out attack to completely clear remaining enemy units at the end of the game.

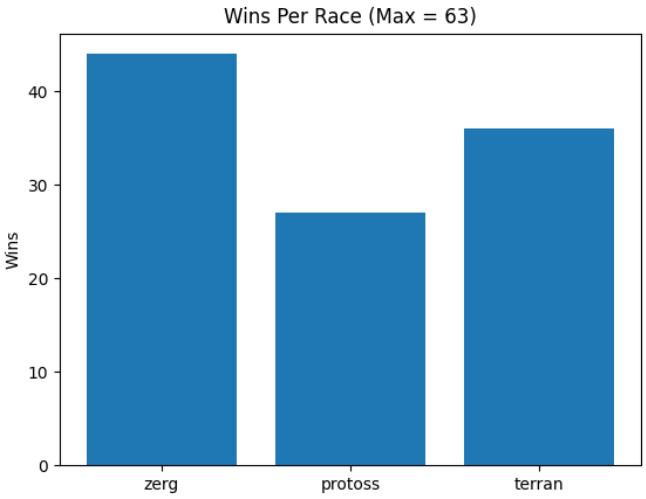
As the game progresses, the necessary amount of army units needed to call an attack increases. This is done to scale with the expected size of the army at that point in the game, so that the base is not left completely undefended.

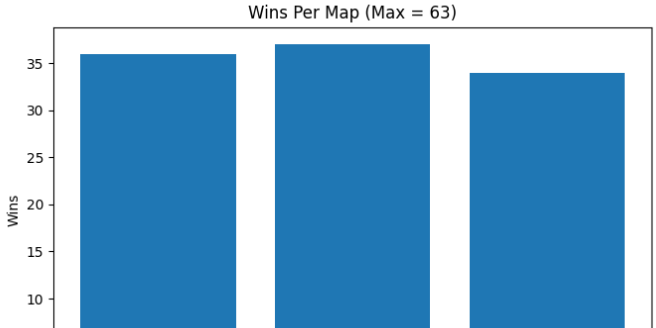
Software Modules

The main BasicSc2Bot class is found in `BasicSc2Bot.h` and `BasicSc2Bot.cpp`. The Node class is defined in `Node.h` and `Node.cpp`.

For testing, the script file `Tests.py` and batch file `RunTests.bat` are written to collect win rate and performance statistics which also includes time taken to finish each matchup. Additionally, `ParseTests.py` is used to more easily collect test results and aggregate data.







Evaluation

To evaluate our bot, we used a python script to run the bot against an AI of each race on each map that will be used for the tournament. We performed tests against the built in StarCraft II AI on difficulty levels Easy, Medium, and Hard, and recorded the results to text files. Each of these was run for as many iterations as we had time for, which ended up being a combined 7 test runs per case. This means that, when looking at the effect of any factor individually, our sample size was n = 63 iterations. We also wrote a second python script to parse the data contained in the text files, and create these graph visualizations.

We observed that our bot performed very well against the Easy AI, with a 97% win rate, and only 2 losses of 63 games played. We performed on par with the medium AI, with a 57% win rate and 36 games won, but performed poorly against the Hard AI with a 16% win rate and only 10 games won. This was expected, and confirmed that our bot performs about on par with the medium AI.

We observed no significant difference in win rate when playing on different maps, but we did

observe a difference in performance when playing against the various races of opponents. We performed best in the Zerg matchup with a 70% win rate, and we performed worst against the Protoss, with only a 43% win rate. The Terran had a 57% win rate. This difference could have been caused by the majority of our testing of the bot in development being against the Hard Zerg AI, and we tested against Protoss and Terran relatively little.

Because we had the goal of developing a defensive strategy, we also measured performance by recording how long games took to finish. Our results were as follows:

Average Time per Win: 781 seconds (~13 minutes)
Average Time per Easy win: 744 seconds
Average Time per Medium win: 843 seconds
Average Time per Hard win: 782 seconds

Average Time per Lose: 957 seconds (~16 minutes)
Average Time per Easy lose: 1229 seconds
Average Time per Medium lose: 1216 seconds
Average Time per Hard lose: 814 seconds

These results indicate that our strategy was successful at surviving for a long time, even in matches where we ended up losing. We observe that when we win, it usually happens quicker than we lose, indicating that our bot is resilient to early game rushes and ‘cheese’ strategies. We also have an average game length of 16 minutes in the games where we lose, which is longer than most AI vs AI matches.

We did a t-test using our results to compare with our goal of exceeding 12 minutes, and we saw significant results that we were able to achieve this, getting a p-score of 0.000345 for the timings of our losses when fighting the hard difficulty AI, showing that our results were statistically significant.

Advantage and Disadvantage

One advantage of our program is that it is greatly effective in the early stages of the game because the ambushes that we release near the start of the game are strong and quick, meaning that we can weaken the enemy very early on.

Our bot is also effective at living long, as we aimed for a defensive build, with the main strategy of dominating the map by spreading out our bases to multiple locations across the map. By having multiple bases, we are less likely to be wiped out completely by the enemy in a short amount of time.

However, the bot has a couple of disadvantages. As the placement of hatcheries is hardcoded in to depend on the map, node expansion does not automatically work on any map and must be pre-programmed into the bot. This is not ideal, and an algorithm that determines the best placement could be more responsive to the enemy's actions. Our defensive strategy can also be problematic. Spore Crawlers and Spine Crawlers are generally slow at moving, are unable to attack while uprooted, and take a while to root back onto the ground. As such, this movement time makes both the bases and the units vulnerable to enemy attacks.

Future Work

For the future, we would like to develop a more solid build order that can be used to support our strategy to lead to the faster development of stronger units. This can be beneficial to optimizing our defensive and offensive tactics, thus leading to us having a more consistent strong strategy that arises from having stronger protection earlier in the game from ambushes that may be released by the enemy at any moment.

Another development we would like to focus on is finding a way to incorporate researching and unit development to further focus our defensive capabilities. Spreading creep and burrowing are two Zerg mechanisms that have not been heavily incorporated to their greatest extent, so we would like to develop those in order to allow us to defend our bases and units more effectively.

Conclusion

Overall, while the bot is not better than the best integrated Starcraft AI, we are still overall pleased with the results we attained. Our team had no one with prior Starcraft experience, and while we had no reference point for if our strategy was effective during the planning stages we are pleased with the results we ended up with. Our bot is able to survive for a long enough period for us to safely say it is successful in defense, and that we were able to accomplish the strategy we set out for, so we believe we achieved our goals.