“Defense Bot” Final Project Report

Ryan Brewer ◆ Colleen Venhuis ◆ Alan Zhao

When first discussing our plan for DefBot, we had yet to establish most of our implementation goals. We knew, however, that we wanted to create a strategy which required the least amount of variation between opposing races and strategies. StarCraft is a game of extreme optimization; changing your strategy, and your tech as a result, will put the player at a disadvantage, as their opponent hasn’t had to make such concessions. The only time which you can safely switch strategies is before you’ve actually committed to one, hence the importance of early game scouting.

However, we knew our fellow students could not be relied upon to stick to the conventional openings and hyper-optimized build orders that experienced StarCraft players have become accustomed to. Subtle hints our scout provides us may not be valid against the logic of there other bots. Combined with the amount of game knowledge required for accurately acting on the early game timings, buildings, and unit production of our opponent, we decided it would be more beneficial to focus on a single build order/strategy.

An additional benefit to focusing on a single unit composition is that it limits our implementation to very specific unit types. As we quickly discovered, the base UAlbertaBot has very little logic for controlling unique unit types. Very few units have any specific micromanagement strategies, nor are they capable of using their “special abilities”. By reducing the number of unit types we have to account for, we could focus on implementing advanced logic specifically for our chosen unit composition.

To this end, we chose to use a well known Terran strategy; the so called “Siege Expand”. It is infamous for simply not caring what your opponent is doing. By building the Command Center intended for your “natural” expansion at the chokepoint into your main base (most often the top of a ramp), you can create an effective, high health “wall”. This prevents any early game aggression from gaining access, and often even vision, of your main base. By backing this up with a few sieged tanks, it becomes almost impossible to break. As a result, this allows you to begin building your second Command Center much earlier than would otherwise be safe, and although you do lose a few seconds of potential mining in the time it takes to float this Command Center to your natural expansion, it’s a small price to pay for almost guaranteed safety.

Following this, we create a strong, mechanical based, unit composition, and quickly move towards taking a second expansion. Siege Tanks are a staple of almost any Terran army, and are notorious for being effective against almost any opposing unit composition. The combat micromanagement of Vultures already implemented in UAlbertaBot was also extremely powerful. Combined with Goliaths to deal with enemy air units, as well as a few Science Vessels for detection, these units give us a strong generic army composition going into the mid-game.

As we're not attempting any kind of tricky, or "cheesy", behaviour, we have to win through standard play. Our goal was to spend minimal resources in the early game on units to hyper inflate our economy and simply out produce, and thereby muscle out, our opponent. By maintaining, and hopefully increasing our resource advantage throughout the game, we can wear down our opponent through semi-frequent, strong attacks. The intention is to never let them gain momentum, keeping them locked in their base while we slowly dismantle them.

UAlbertaBot: Advantages and Disadvantages

We were initially unsure of the capabilities of the current UAlbertaBot when we sat down to write our project proposal. The only successful Terran strategies it implemented involved early game rushes, and the closest premade strategy to our own, "Tank Push", was very weak in the early game and nonoptimal towards the mid-late game. This made it difficult to tell just from watching the bot how well it could perform in our context. As it really does require working with a code base as large as UAlbertaBot's before we could properly understand the capabilities of the bot, we overestimated what we could accomplish in our initial project proposal.

As we came to work more with UAlbertaBot, we were disappointed to find just how barebones it was. Many of our more complex tasks were put on the backburner as we worked to add much of the core implementation required for any Terran, player or bot, to be effective. To give a quick list of some of these missing implementations:

- Units would attack as soon as they were able, leading to them marching alone into the enemy base early in the game. No concept of defending established positions.
- Very few units were capable of using special abilities, such as a Vulture's mines or a Science Vessel's EMP.
- SCV's were only capable of repairing buildings, leaving mechanical units no way of regaining health.
- Furthermore, if an SCV was killed while constructing a building, it would never be sent a second SCV to complete it, nor would it be destroyed by our bot. This could have game changing consequences if this building was our first barracks, factory, or expansion.
- Before attacking, units would group in the middle of the main base. This made defense sloppy, as they lacked a positional advantage and were slow to respond to any attackers.
- Absolutely no logic for the intelligent placement of detector buildings, or for the movement of detector units.
- Wasteful anti-scouting, infinitely chasing the enemy scout with an SCV.
- Extremely basic, linear expansion logic. Nonoptimal SCV production.
- No way to place buildings at specific locations, such as chokepoints. No way to lift off, move, and set down buildings once built.

In addition to implementing these changes, we also had to have our core strategy and initial build order working successfully so we could test our changes in a more useful context.

Implementations

We successfully managed to implement some, but not all, of these essential features. In general, we prioritized early game survivability with mid-game greediness. The central philosophy behind our prioritizations was to avoid wasting resources unnecessarily.

Surviving the Early Game

As discussed above, our original plan was to wall off our base with our second Command Centre. Actually creating a successful wall-off by having it build close chokepoints proved difficult, and the time investment involved in creating individual build positions for every map pushed this goal back until more essential functions were completed. We were also unsure how to go about lifting the Command Center and moving it to the expansion.

We considered creating a bunker for our marines as a temporary workaround. However, when we ran into issues with intelligent bunker placement, this was scrapped, especially considering it required more resources, marines, and supply than our strategy could account for, slowing down our early game production significantly.

Eventually, we settled on delaying our initial expansion by a few minutes and optimizing our build order to focus on the early production of mechanical units. While this does slow down our early game somewhat, these extra units allow us to safely take a second expansion quickly after our natural. We also ensured our units defended the high ground and the ramp rather than the center of the base, giving them a significant positional advantage. They will then move down the ramp to protect the natural expansion once our second Command Center is placed. Avideo ofthisinactioncanbefoundhere.

To accomplish this, we used the chokeDefense flag in CombatCommander to keep track of if we had made special changes to the main base defense. To keep our units from blocking the ramp, we placed them a fifth of the way towards the closest base. This ensured our own units could pass through our defenders while enemy units would still be forced to engage them.

Greedy Expansion Timing

The other core element to our strategy was to ensure we had a resource advantage throughout the game. The basic expansion timing build into UAlbertaBot simply expanded linearly, once every 5 minutes. Changing this was simple, however the game rarely reacted as expected when given specific times to expand. It took significant trial and error testing to find timings which worked well with our build order and production. Our final expansion timings, in minutes were:

{ 3, 8, 15, 24, 35, 48, 63, 80 }

Note that the bot appears to ignore the first number, as it references the first Command Center, already built when we begin the game. In practice, this resulted in us reliably expanding to our natural at almost exactly 9 minutes, and gaining our second expansion shortly thereafter. Our third comes at just past the 20 minute mark, but after this point, we normally only expand if we had an abundance of minerals (>4000). This prevents us from expanding in close games

where unit production is essential, while expanding rapidly in games where we have a clear advantage.
To avoid bloating our supply with SCV's when expanding rapidly like this, we ensure we have no more than 60 SCV's at any given time. This includes "combat" SCV's, discussed below. It is also crucial as we bumped up the number of SCV's built when expanding. The original UAlbertaBot only built 10 new SCV's when creating a new expansion. Optimally, you want 2 SCV's per mineral patch. Most mineral lines have ~8 patches, and we also need 3 SCV's on gas. In the main base, we also have to include additional SCV's for constructing buildings. This brings the optimal number of SCV's per expansion to ~15-20. Therefore, we bumped the number of SCV's built when expanding up to 15. Thisvideoshowsourexpansiontimingoverthecourseofa game.

Anti-Scouting Tactics

As we attempt to gather as many minerals as possible early in the game, the default anti-scouting behaviour of infinitely chasing enemy scouts with a worker unit was unacceptable. We lose critical mining time on our SCV, and achieve very little other than to keep the enemy scout moving; our SCV will never actually kill the scout.
As enemy scouts will generally keep moving without us chasing them, we only send a worker to attack them if they come in proximity with our mineral line. This prevents their scout from attacking and killing our workers before we have combat units available. We then build two early Marines to eliminate this scout before it can ascertain any additional information, as well as to deal with a potential gas steal.

To do this, the CombatCommander::updateScoutDefenseSquad() method saves a set of regions near where our mineral patches are located. Only when an enemy unit enters these regions will our SCV's will stop mining to attack. We ensure this is an enemy worker unit (aka scout) before we attack, but our SCV's will also attack any Zerglings who may reach our base before we can get Marines out. AvideoshowingasuccessfulZerglingdefensecanbefoundhere.

Note that even though we give priority to resource collection, if no unit has yet to be assigned to defending against enemy scouts, i.e. the current scoutDefenseSquad is empty, a while loop is utilized to guarantee that the worker closest to the scout is grabbed from WorkerManager and put into the scoutDefenseSquad. The next time the program runs updateScoutDefenseSquad(), if the enemy's scout is already away from our mineral patches, the SCV defending against the scout will return to the mineral line to continue its resource collection. Therefore, despite its relative complexity compared to the original scout defense strategy, this revised anti-scouting strategy assists other parts of the program in maximizing the amount of resources collected.

Defending Expansions

As discussed, our army composition moving into the mid-game consists of a mixture of mechanical units. Eventually, we transition into building Battlecruisers, as their high health and damage, as well as their ability to attack any unit, provides superb protection against other late-game units.

As expanding is so central to our strategy, we also wished to protect these expansions, rather than leaving them unguarded like before. We had to use UpdateDefenseSquads to do this. First, we removed the code that erased defense squads if the region wasn't under attack anymore. This permanently established a new defense squad for each expansion built. Then we total the number of each available unit type and send each defense squad an equal distribution of those units, up to a calculated maximum amount, using updateStandbyDefenseUnits and findStandbyDefender. This prevents our main base from being assigned all of the units first while other bases have fewer or none.

While it is better to prioritize main base defense, a frontal attack will go through all our other bases before getting to the main base. Thus we also move our choke defense squad closer to the first expansion at the bottom of our ramp to provide twice the protection in this area. The "disadvantage" to limiting the maximum amount of units per base was that some units would end up moving back and forth between bases as the amounts increased. However, this actually provides extra scouting through the center of the map, alerting us of incoming attacks. As such, we decided to leave it as is. Additionally, base defenses will call for backup if overwhelmed using the previous implementation in updateDefenseSquadUnits. Upon being attacked, up to 3 units per enemy unit will be sent from other bases in less immediate need of their defenses.

Attacking the Enemy

While rapidly expanding around the map may give us a resource advantage, our opponent will be free to do the same if we do not keep them on the defensive. To implement our attack strategy, we created different squads for each unit type in CombatCommander, with corresponding update functions for each of them. We also enabled several variables that would dictate the various stages of our attacks. To initialize an attack, we first wait until we have a minimum number of each desired unit type. When we have enough of each, we set a flag to true. Once all flags are true, all defending units we have are sent to attack the enemy base.

We will also attack if at any point we reach >4000 minerals, even if we feel we may not win. At this point, we are confident that we have enough minerals to replenish our army, and it keeps pressure on our opponent. If we are successfully out-producing them, we should be able to afford the loss of units, whereas they may not be able to. At >4000 minerals, we also expand behind this attack, allowing us to safely increase our foothold on the map even if the attack fails.

Sometimes, our first attack is not successful. To improve our chances we initialized a multiplier equal to 1. If at any point we fall below half the required amounts to initialize the attack, we retreat and increase the multiplier, building up a larger force before attempting to attack again. This minimizes losses, allowing us to keep defending our bases without losing too many units. Even if our bases are attacked while on the offensive, the bases will request backup and a sufficient amount of units will be sent to help. Thisvideoshowsasuccessfulretreatand re-engagementinaction.

Repairing Units in and out of Combat

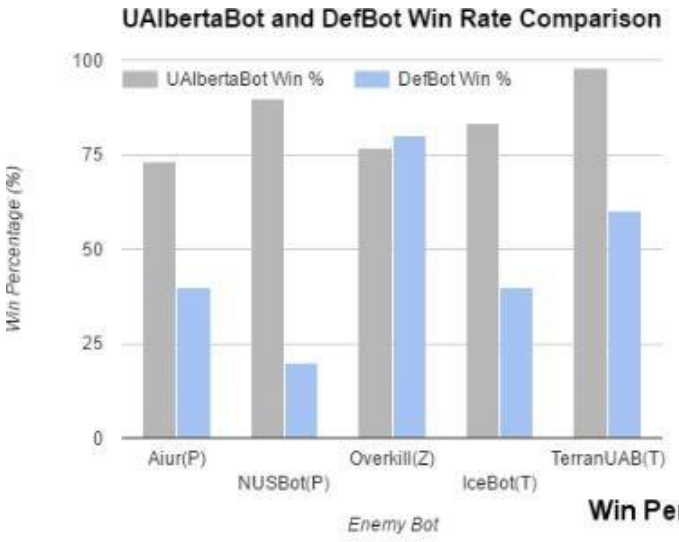
Being a mechanical based army, we felt should have some way to repair damaged units. As UAlbertaBot could not originally do this, we also created an scvSquad to join in with attacks. We ensure we have no more than 5 SCV's in this squad, as not to deplete our mineral lines.

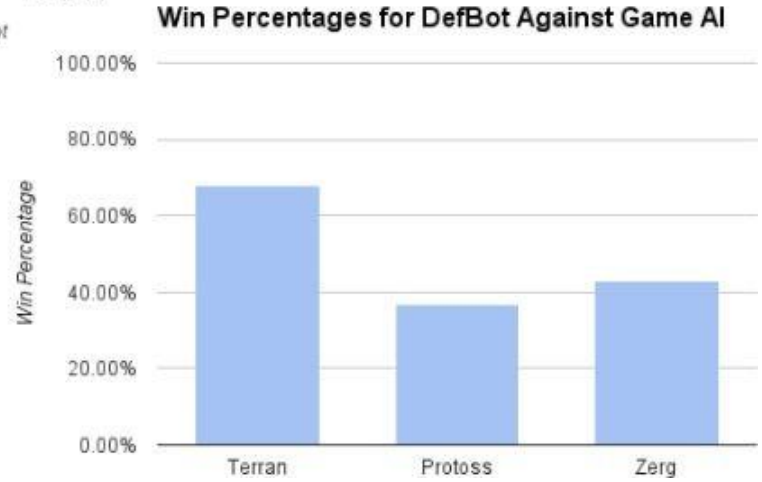
Being a mechanical based army, we felt should have some way to repair damaged units. As UAlbertaBot could not originally do this, we also created an scvSquad to join in with attacks. We ensure we have no more than 5 SCV's in this squad, as not to deplete our mineral lines.

To repair units, we compare each unit's current hit point to its maximum. We then loop through the scvSquad and assign available SCVs to repair damaged mechanical units. Each time CombatCommander::updateSCVSquads() is called, Unitsets of damaged mechanical units are re-determined so that SCVs are guaranteed to be reset to combat workers if all previously damaged units have been repaired (i.e. no unit needs repair). Note that we prioritize Siege Tanks over other mechanical units; that is, SCVs will only repair Vultures after confirming all tanks are at full health.

Unfortunately, SCVs still believe they are part of an attack. While they prioritize repairing units, if everything is at full health they will engage the enemy, and we often quickly lose them. During attacks earlier in the game, this can actually be advantageous, as they disrupt the enemy for our combat units. It is not, however, ideal behaviour.

Testing and Statistics





Some things to keep in mind to make this data more meaningful:

- NUSBot employed strong early game rush strategies every game.
- TerranUAB used a very similar strategy and unit composition to our own.
- Aiur won the 2015 StarCraft AI competition we took these bots from.

Overall, we believed we performed fairly well. Our results against NUSBot shows our weakness in the early game, especially against Protoss, but the fact that we were able to win at all without any defensive structures validates our build order, and shows our ability to transition successfully out of the early game. We did extraordinarily well against the best Zerg bot the competition had to offer, and had moderate success against the the best Terran and Protoss. Finally, we showed our superiority to a bot employing a similar strategy to our own. While there is definitely room for improvement, we certainly managed to hold our own with 48% win rate overall.

Future Goals

While our bot shows significant improvement over the default UAlbertaBot “TankPush” strategy, we did not succeed in meeting all of our implementation goals (although all of them

were attempted). As a result, we also did not have the opportunity to begin work on any of our more complex ideas we had for DefBot.

If we continue to work on our bot in the future, we would like to first and foremost get our early game wall-off working. Alongside this, we feel it's important to get SCVs to finish unfinished buildings without an assigned worker. This would go a long way to minimizing our early weaknesses, and will allow us to start snowballing on economy even sooner.

We would also like to improve the micromanagement of our units in combat. This is an area we unfortunately did very little work in. When attacking, sieged tanks form solid walls, and forward units tend to block chokepoints, not allowing more units through. We'd like to separate units more and have them move further into the base to avoid blocking chokepoints; especially when on ramps. We would also like to introduce logic for using the Vulture's Spider Mines and the Science Vessel's EMPs. Logic for when to when to queue upgrades could also be improved. Finally, we'd like to have siege tanks move forwards in a "leapfrog" formation when in combat, half un-sieging and moving forward while the others cover them, and then re-sieging and repeating the process.

Finally, we'd like improve our detection of invisible units and our defense against drops. Primarily, we've had significant difficulty getting a Science Vessel to follow along with an attack, and Missile Turrets build only in the main base and with no real intelligence. Additionally, we'd also like to have our units group up more before initializing an attack, and to prioritize attacking expansions if they exist, rather than the main base where more defenders are likely to be stationed.

Contributions

Ryan Brewer:

I did most of the research and planning involved for our bot, and acted as a consultant for other group members regarding strategy and unit behavior. I also developed and refined our strategy and build order. This included refining our expansion strategy detailed in the report.

I also made an unsuccessful attempt to implement a wall off of our main ramp using our second Command Center, and started, but didn't finish the code to have a new SCV continue constructing a building after it's original builder SCV had died. I also implemented unused, non-optimized code to send Marines into bunkers.

Finally, I found specific game instances, then saved, recorded, and edited them to create the animated gifs / videos shown in our presentation and project report.

[Colleen Venhuis : AGREE]
[Alan Zhao : AGREE]

Colleen Venhuis:

My contributions included the defense squads and attack squads. I originally started by having the attackSquad wait until there were at least 10 or so units and then attack. Eventually it evolved into creating a squad for each unit, with the intention of having them act differently and communicate with one another. We added battlecruisers towards the end and changed 'goliathSquads' to 'airAttackSquads' to have them act the same way. For the defense, it was suggested that instead of having all the units wait in the main base, we should have them positioned between all the bases instead, which was successful. Later we implemented the chokepoint defense as well. After this, I went back to the attack squads and implemented the multiplier and the retreat so that we would retreat to increase our attack forces after losing half of them. The attack squads and defense squads worked closely together because we could not be attacking and defending at the same time. All in all, all of the work I did was in CombatCommander with the exception of updateSCVSquads and updateScoutDefenseSquads.

[Ryan Brewer : AGREE]
[Alan Zhao : AGREE]

Alan Zhao:

I modified the micro for defending enemy's scouting units. Rather than using WorkerManager to deal with everything, I optimized the updateScoutDefenseSquad() method. Once an enemy scout comes in proximity with our mineral patches, we only send one mineral worker to chase/attack it. The next time this method is invoked, if no enemy's worker unit is detected, i.e. it has been chased out of our mineral region, then the scoutDefenseSquad will be cleared by resetting the combat worker back to mineral worker.

The majority of our attack squads (of mechanical units) are in a severe situation when they are under massive attack. Thus, I implemented the updateSCVSquads() method so that some SCVs will repair damaged mechanical units in and out of combat. The scvSquad follows all the attack squads all along. Once a damaged unit is detected, a SCV will repair it until its hit point reaches its max hit point.

Aside from the 2 methods mentioned above, I also implemented some helper methods across many modules. Also, I ran the Tournament Manager to collect stats from games against other bots from 2015 AIIDE StarCraft competition.

[Ryan Brewer : AGREE]
[Colleen Venhuis: AGREE]