Hal9001: A Bot Odyssey

Motivations and Prior Work

With the time constraints of the course and our team's unfamiliarity with Starcraft II as motivations, our team aimed to create a rule-based AI Bot that is capable of performing at a high level through the emulation of an effective build order for a Starcraft beginner. We found a simple yet powerful build order presented by HuShang for the Terran race to serve as the basis of our bot. We also looked to the behaviour of the elite-level built in AI to supplement some of the gaps in our strategy

Emmett had some experience in C as well as puzzle solving AI from previous courses such as CMPUT296 and 455. However he had never played starcraft 2, or any similar RTS titles so there was a lot of learning to be done in that area.

Isaias has experience in game development and puzzle solving algorithms via CMPUT 250 and 355. He has not played starcraft 2 but he plays a lot of MOBA games.

Strategy

The build order informed most of our strategy regarding base building and economy, although we modified it a bit to suit our needs. We start with building two supply depots and a barracks to form a wall-in at the top of our main ramp. Then we build a refinery, upgrade our command center to an orbital command center, and expand to the nearest expansion spot. We also build a bunker near our base facing towards the center of the map and place a widow mine near the bunker for defense. Then we build the structures needed for building our army: a factory, two starports and two more barracks. When a unit producing building has finished being built, we immediately build the add-ons that we wish to have before training any units. This includes a reactor on one barracks (allowing us to produce two marines at a time), techlabs on the other barracks (allowing us to produce marauders), reactors on the starports (allowing us to produce medivacs and vikings), and a techlab on the factory (allowing us to produce tanks). Finally, we build an engineering bay, out of which we will research the infantry weapons level 1 upgrade. Using the barracks techlabs, we also research the stimpack and combat shield upgrades. Throughout the build order, we build supply depots whenever we are close to reaching the supply cap. We also build refineries at various stages of the build order until we have 2 at each base. Once the build order is over, if we ever deplete our mineral lines we expand to a new expansion.

Our attack strategy relies on having a large, well structured army. We define three stages of unit ratios that determine the minimum number of army units that we need in order to attack. We observed the elite level bot to define these unit ratios, which are defined as follows:

{marines, marauders, vikings, tanks, medivacs}
Stage 1 = {10, 3, 2, 3, 4}
Stage 2 = {15, 5, 5, 5, 6}
Stage 3 = {15, 10, 8, 8, 10}

We start with the first stage, and increase the stage each time we need to retreat after an attack. Our condition for retreating is if we have less than half of our army remaining and the

enemy has more army units than us. Once we reach the desired unit ratio, we send our army out to rush the enemy base (which we have found by sending a scout at the start of the game). Since we continuously train units up to a maximum of double the current unit ratio, it's possible that we'll have more of certain units than specified in the unit ratio, but this doesn't have any negative effect on our attack. Also with the amount of time that it takes to build our army, it's guaranteed that we'll have our upgrades before we attack. Further, while we are attacking we continue to train units and send them as reinforcements for our attacking army. We originally tried keeping these reinforcements at the staging area, to build another army while the first one is out on the rush, but found that our bot performs better if we send them immediately as reinforcements.

Our defense strategy is tied to the attack strategy. As we are building our army, we send the army units to a staging area near our base so that they intercept any enemy units who are trying to attack our base. On top of this, we have the previously mentioned bunker and widow mine that remain at our base for defense while our army is attacking. The fact that we continue to train army units while our army is out attacking also serves as a defense, since these units will run into any enemy units that may be attacking our base while they are on the way to join the rest of the army.

We also have an endgame strategy, which comes into play once we destroy the enemy's main base. Once this happens, we use the scanner sweep effect from our orbital command center on unvisited expansion locations to expose any remaining enemy units that are hidden by the "fog of war". We send our army to attack these units until there are none remaining.

Implementation

For our implementation the Blizzard example bot is what we used as a reference, since there were not many other places to see the API used.

Build order:

The build order is managed through if statements. We use supply, mineral, and vespene counts as well as the existence of certain buildings as conditions to execute a given step of the build order. This allows us to execute the build order in a sequence that suits our needs, and it allows for buildings to be rebuilt if they are destroyed since all of the if statements are evaluated on each step. We build structures in positions relative to other structures to ensure that there is enough space for the structure and any add ons it will have in the future. This also allows us to strategically place structures, such as the wall-in or the bunker. The only time we build structures in random locations is when we build additional supply depots, and we ensure that the critical structures and add ons for our build order have been built before building at random locations.

Army management:

There are 3 separate components to the army management system. The first component is the production manager which handles which units to produce and how many we need. It is a fairly simple implementation which takes predetermined ratios of units and produces until we have double the amount required for a full army.

The second component is a movement handler which is at the beginning of the ManageArmy function. By default our army rallies at the staging location (unless our bunker is not made yet, in which case they wait at the building they were built at.) Once we get to the number of units defined in the unit_ratios list, we switch into attack mode. In attack mode, the flying units follow the marines in order to prevent them from pathing up and over terrain and seperating from the main army. All other units go to the closest base to launch an attack on it.

Each unit type has a different behaviour for the final component, which is the unit manager. This is a large switch case which provides the following behaviour to the units. Marines will activate their stim buff if within attacking range of an enemy, as well as move away from them if they are too close (for kiting). Marauders have the same stim behaviour as marines, except with a longer range, and no kiting behaviour. Widow mines will simply burrow themselves if within attacking range. Siege tanks will enter siege mode if within attacking range, as well as disengage siege mode if there are no enemy units nearby. Medivacs will heal any healable units in our army that are missing health, otherwise they just follow the bio-units around. Vikings will enter ground mode if there are no flying units in the enemy army, and back into flying mode if there are either no units or flying units to attack.

Important Helper Functions

buildNextTo() - This function allows for the bot to build structures in reference to another structure. This queries the ability id to find the footprint radius of the structure to be built. Then it takes the direction where the structure would be placed (FRONT, LEFT, RIGHT, etc). It checks if it can be placed on the current direction. If not, it goes to the other qualifying direction. Overall, this provides the means to follow the build order more accurately.

Expand() - This function will build a command center at the closest viable expansion location. First, it gets all the possible expansion points in the map. Then it evaluates if a comm center is positioned in the vicinity. The closest of which will be returned and will be the site for expansion.

GetRandomUnits() - This function returns an SCV that will be assigned a build order. It iterates over units and evaluates if they are idle, only mining, and in range. This allows for proper delegation of tasks to SCV’s in order to fulfill the build order quickly.

Performance (Win Percentage)

Each difficulty and map was the average of 10 games, as well, we had problems getting the proper difficulty, so our tiers are easy/medium, harder, and elite. We assume based on our win rate on Harder that we likely win all the games on hard as well.

Vs Terran:

	Easy + Medium	Harder	Elite
Cactus Valley	100%	50%	0%
Proxima Station	100%	60%	0%
Bel'Shir Vestige	100%	60%	0%

Vs Zerg:

	Easy + Medium	Harder	Elite
Cactus Valley	100%	100%	0%
Proxima Station	100%	100%	0%
Bel'Shir Vestige	100%	70%	20%

Vs Protoss

	Easy + Medium	Harder	Elite
Cactus Valley	100%	70%	0%
Proxima Station	100%	90%	0%
Bel'Shir Vestige	100%	80%	0%

Challenges

Debugging in a real-time program is very difficult as you need to make guesses as to where the problem may be and change them to test. On top of this, testing takes much longer because some issues

may not show up until a couple minutes into the simulation, and thus every change requires a few minutes of waiting for the bot to reach the desired state.

More specifically, optimizing unit behaviour is very challenging because of how the orders work, you don't want to send them the same command every single tick, however you want to make sure that their orders reflect the current state of the game. On top of this the movement and attack orders are seperate, not to mention other abilities the unit may have.

Building placement and building timings were another problem area due to the nature of having multiple maps and starting locations. Bugs were encountered sometimes exclusively on one starting location in one map, making detecting them hard and fixing them annoying because of the random nature of the starts. As well, there is no easy way to tell distances just by looking at the map so we had to estimate and test multiple times before we had consistency in our buildings.

Future Changes

The wall-in at our main base currently serves no defensive purposes. We had to adjust the placement of the wall-in because our tanks weren't able to move down the ramp, even when the supply depots were lowered, and because sometimes the orientation of the barracks would prevent the reactor add-on from being built. In the future we would place the wall-in so that it correctly blocks enemy units during an incoming attack.

Overhauling the unit attack manager to handle both movement and attacking at the same time would be very beneficial in removing unnecessary ability calls and greatly improving performance. Having some sort of army formation on top of this would make our army much more effective by having a row of marines, then a row of marauders and tanks in the back.

Having a more dynamic army rather than our 3 stage model is something we would implement given more time. Right now having only 3 steps really limits our versatility and it would be much better if we could recognize that we are far ahead and want to launch an attack early on. Having some sort of dynamic allocation of units based on our total supply was something proposed but not implemented in time.

Our army lags very badly later on. A change we wanted to make but never figured out how to do effectively is limit the number of UnitCommand calls. We tried filtering through ability tags and locations, to some success, but especially later on the bot slows down significantly as a result of all of these calls. Fortunately, if we are able to win within our attempted rush, the army never gets large enough for the game to slow down to slower than real time.