# Project Report

TerranFormer
Carter Sabadash, Gary Ng Kwong Sang, Waseem Yusuf, Yuqian Cao

## Motivation

The following is contains quotes from the researchers at Northeastern University [1]:

StarCraft 2 is a real-time strategy (RTS) computer game created by Blizzard Entertainment. Released in July 2010, StarCraft 2 sold over 3 million copies worldwide after only one month on the market, making it one of the fastest-selling RTS games of all time.

StarCraft 2, like many other titles in the RTS genre, is a game for two or more players. Each player chooses from one of three different races (Protoss, Zerg, or Human) and builds an army of units to defeat his or her enemies on a specific map: a 2-dimensional grid of finite size with various terrain artifacts such as resources and obstructions. To build their armies, players must mine resources, construct buildings, research technology, and train units, all in real time. Additionally, players must balance their time and efforts between creating a strong economy, which enables faster production of units as well as the production of more advanced units, and attacking their opponents. Players win a game by destroying all of their opponents' buildings. Although there is no limit on the length of a game, many games last between 5 and 30 minutes.

Each of the three races in StarCraft 2 has its own unique behaviors, buildings, units, and technologies that allow for many different styles of play. Each map in StarCraft 2 is also unique, allowing for different strategies that take advantage of the terrain and positioning of resources. Because of the open-ended nature of the game, the computational complexity involved in creating artificially intelligent (AI) opponents is very high. AI opponents are typically built to simply brute-force efficiency, issuing many hundreds of orders per minute with the goal of overwhelming human opponents who cannot physically or mentally match the micromanagement capabilities of a computer.

In this project, we want to build a Starcraft Bot who can play the game better than we can. And we also want to apply knowledge we gain in CMPUT350 to our bot.

## Goals

- Effective and efficient scout

- Optimal troop training
- Optimize resource allocation
- React dynamically to enemy actions
- Bug free code, runtime efficient code

## Description of our approach

We analyzed the game and decided to split the bot up into 4 different agents based on the 4 different aspects of the game: ATTACK, RESOURCE, DEFENSE, SCOUT.

All 4 agents share an observation and action interface and are capable of communicating with each other through sending tasks. Each task has a priority and is handled by a priority queue. Tasks with higher priority will be executed first. For example, if the ATTACK Agent wants to train more Marines, it will send a task to the RESOURCE Agent, whose job is to allocate resources. The RESOURCE Agent will receive the task, and the task will eventually be executed. How soon the task will be executed depends on the priority and current resources.

ATTACK Agent is in charge of leading troops to attack enemy bases. It uses information gathered by the SCOUT Agent to determine enemy locations and enemy bases. It will also use abilities against enemies when attacking and track remaining enemies.

RESOURCE Agent is in charge of gathering resources and distributing the gathered resources according to need. The need is determined by the tasks in RESOURCE Agent's task queue. It also manages command centers, scvs, and where to place each building. Buildings are placed in rings surrounding the command center to provide more protection during an attack.

DEFENCE Agent is in charge of base defence, it has the highest priority of all the agents. Due to its high priority, it is also in charge of all upgrades. The upgrades follow a fixed progression, and we would like to replace it with a behavior tree in the future which will allow for a more dynamic upgrade structure. Defence can take control of all units to ensure that our bases are protected

SCOUT Agent provides crucial information for the attack and defence agent. It controls 8 scvs to scout out all potential base locations and record all enemy units seen. It also has an interface that allows the other agents to query enemy units seen at a certain location within a certain time.

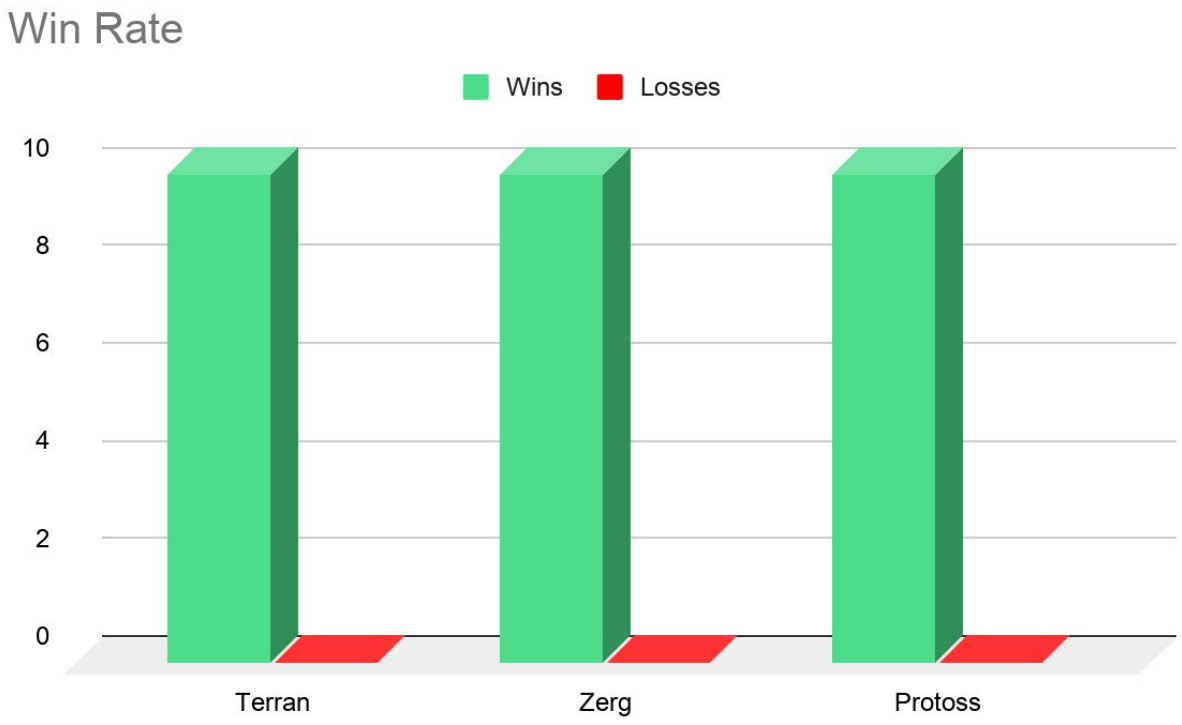
## Advantage and Disadvantage

The biggest advantage with our approach is modular code, therefore, allowing all 4 of us working together without the fear of merge conflicts. Saving time and energy. However, our approach also comes with some downside.

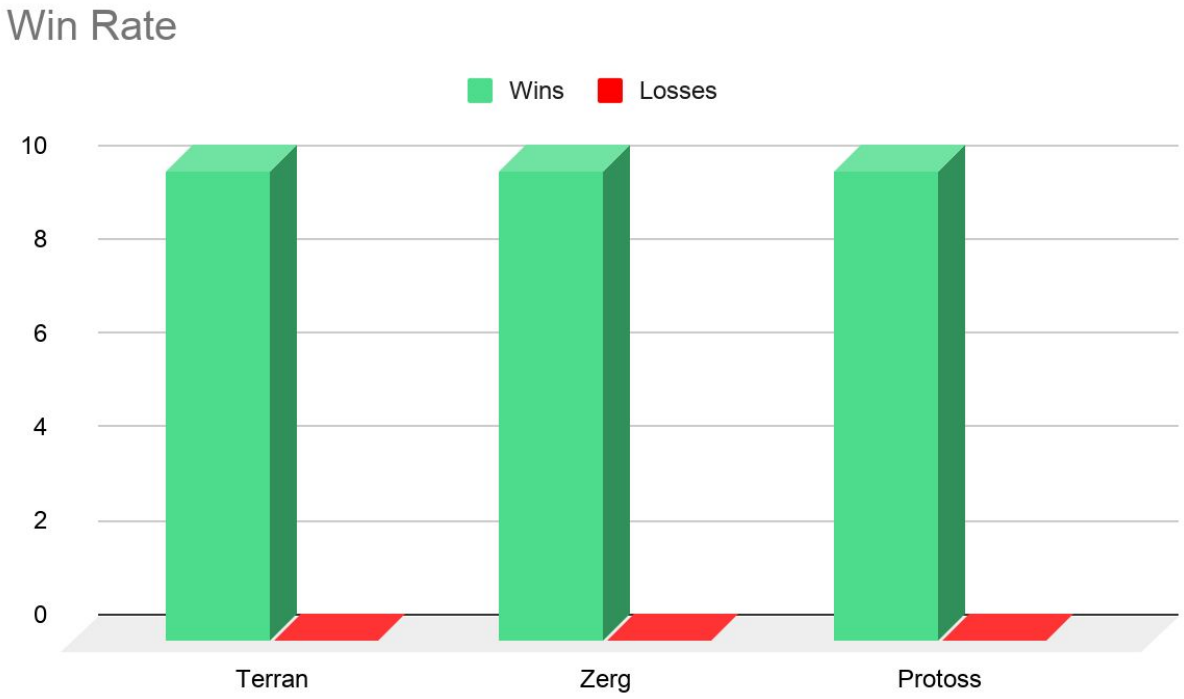
First one is performance, because the 4 agents are separate, a lot of the computation is redone, wasting precious computation resources. Another disadvantage is the competition for resources. The 4 agents are constantly competing with each other for resources. We had to run many trials to figure out the perfect priority structure. It took a lot of time and energy and it is still not perfect.

## Statistical evidence

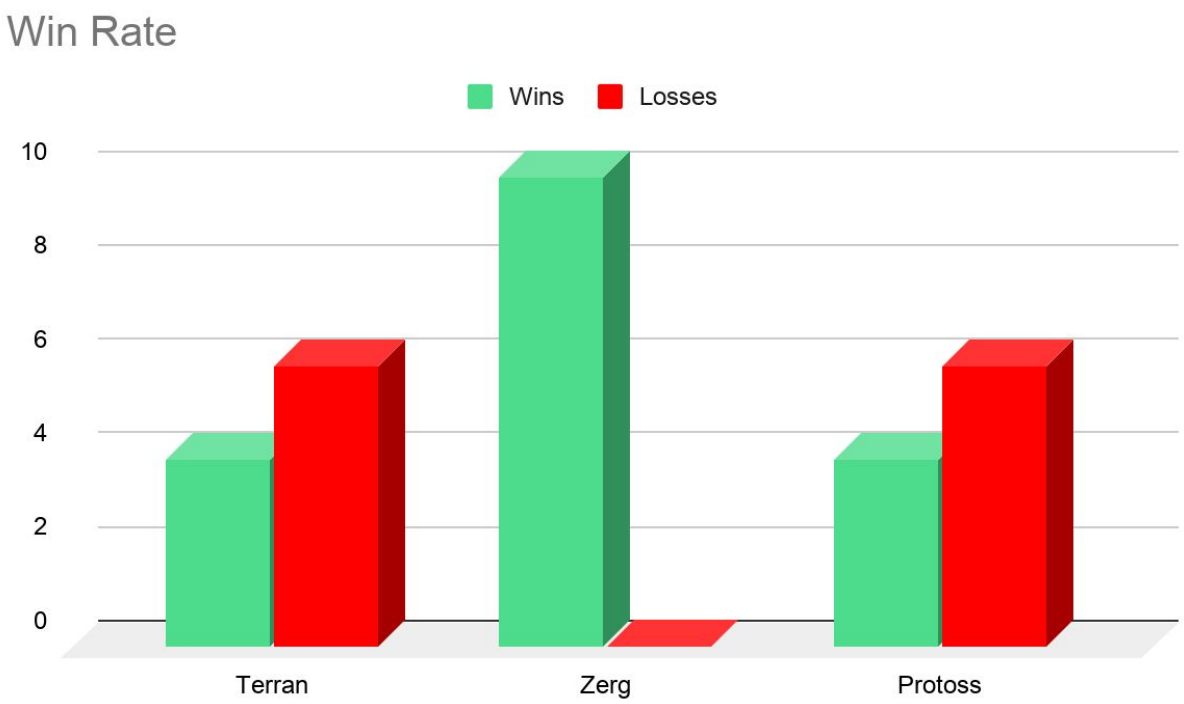
- Against Easy AI




- Against Medium AI




- Against Hard AI



## Software Modules

cpp-sc2 - https://github.com/cpp-sc2/cpp-sc2

- Provides an API to the StarCraft2 game

sqlite3 - https://github.com/sqlite/sqlite

- Allow us to interact with the SQLite database

SQLitePlus - https://github.com/yuqian5/SQLitePlus

- A cpp wrapping library for SQLite3, allow easy access to the SQLite database

## Evaluation

Our bot does pretty well in the early stages of the game; it almost never loses in the first 10 minutes. This is by design as we get several early marines, and the scvs will repair damaged buildings, and continue building buildings when the worker building them is destroyed. The result of this is that it's very hard for the enemy to disrupt our build order or destroy buildings. This ensures that we will progress to mid game and start aggressively expanding to create a resource advantage.

Currently, our bot is really strong in mid game. We have a tech advantage with cloaked banshees and stimpacks, and we mass produce units to overwhelm the opponent. We also make sure to build ravens so that we can deal with cloaked units.

However, due to the lack of advanced units, we are at a significant disadvantage during late game. In the future, more heavy mech units such as Thor and Battlecruiser should be built.

Another downfall of our bot is the inability to use raven's auto turret ability, we believe this is caused by an API issue so there's nothing we can do at the moment. There are other abilities that we haven't figured out how to use, such the Thor and Battlecruisers' abilities, and we only use a few units.

Overall, we think we did pretty well as our bot wins almost every game against the Medium AI, and Zerg Hard. However we still usually lose against Protoss and Terran. While we did very well with the units and features we chose to use, perhaps expanding the units we use and changing our late game strategy would have better results. Additionally, there is room to improve our attack policy, such as by making sure all the units remain closely clumped together.

## Future Work

First and foremost, we would like to use a behavior tree to replace many of the logics of our game. Also it would allow us to build a significantly more complex progression system that hard coding simply just can't or have a difficult time to scale up.

Secondly, we would like to use some form of reinforcement learning in our project. More specifically, to use RL in combat policy, such that a RL agent can determine when and where our bot should attack. We believe that this shouldn't be too difficult if we apply coarse coding and function approximation while using an Expected SARSA agent. While this is no AlphaStar, we believe that it will increase our chances of winning significantly.

Lastly, in future versions of this bot, we would like to implement some of the more advanced combat techniques such as kiting, which would allow our siege tank to fight at the significant advantage. We had this in our original goal, however, due to time constraints, we were not able to research methods to accomplish this goal.

## Source:

[1] Helmke, Ian, et al. "Approximation Models of Combat in StarCraft 2." ArXiv:1403.1521 [Cs], Mar. 2014. arXiv.org, http://arxiv.org/abs/1403.1521.