Project: Live Prediction of Winning a Game and Q-Learning for Building Marines

ECS170, Spring 2018

Group Members (Bot the Builder)

Max Nedorezov, maxned@ucdavis.edu, 913151428
Matthew Quesada, mtquesada@ucdavis.edu, 914953175
Josue Aleman, jaleman@ucdavis.edu 914982971
Ivan Timofeyev, itimofeyev@ucdavis.edu, 998671541
Suhayb Abunijem, sabunijem@ucdavis.edu 914956143

Problem and Contribution Statement

In this section, you will describe the problem you want to solve. You should answer the following questions:

- Why is this problem important to solve?
    - Starcraft II is a very difficult game to play using AI due to the huge search space. There are very many possible actions and possible states and training AI to play is difficult because the result of a game only occurs at the end. Using simple machine learning algorithms, it is virtually impossible to reward a certain action because its effect may only be witnessed near the end of a game.
    - Being able to gauge progress of a game may help identify certain patterns that often lead to success, which may aid in training machine learning algorithms to become master players. Being able to provide a heuristic to an algorithm may allow it to learn much faster.
    - Also, while playing a game, there are certain things that are often essential but take a lot of micromanagement such as building marines. A bot that can help micromanage these mundane tasks may make it easier for a player to focus on more strategic tasks.
- Does there already exist a solution to this problem?
    - Blizzard and Deepmind are trying to build AI to play the game without heuristics or prior knowledge. Our solution would oppose this by providing the machine with prior knowledge, which would currently allow a bot to be very good.
    - Even though our solution would use prior knowledge, it may help in the discovery of algorithms that would work without prior knowledge.
    - The built-in scripted AI already plays a game on its own, but we want our bot to teach itself the fastest way to build certain units such as marines.
- Why is this problem technically interesting?
    - It is technically interesting because using our solution, we can get a real time indication of progress independent of who the player is, a real human or AI.
    - Also, the implementation allows us to use real AI techniques to score our own playing as well as our bot's playing.
    - Building a reinforcement learning algorithm will also allow us to discover new techniques/paths of efficiently building certain unit types.

- What possible AI approaches are there to this problem?
    - K-Nearest Neighbors
    - Q-Learning
- What is your solution and contribution to the solution of this problem?
    - Our solution is to look at thousands of replays and classify all of the replays based on whether they are a win or not. We want to find a pattern of units/buildings built in a game that leads to an optimal result, a win. Then, we will be able to score a game's progress based on the number of units built so far and whether that is close to an optimal path.
    - Furthermore, our Q-Learning algorithm will be able to find the optimal path of building marines which will make it possible to have it help a human player in the opening steps of the game to focus more on strategic tasks.
- Why did you choose your solution given the possible alternatives?
    - Finding patterns in large data sets is often difficult but due to the nature of the game, using a classification algorithm will allow us to find a simple pattern much easier. Classifying the number of units built throughout a game will give an easier measure of whether a player is on the right path to win.
    - We will use the Q-Learning algorithm to build marines because they are relatively easy to build using only a few steps. This will make our action and state space much smaller than that of the whole game, but will still allow us to learn practice a machine learning algorithm.

Design and Technical Approach

- What AI techniques are you proposing?
    - KNN Classifier
    - Q-Learning
- How do the techniques address the problem? Connect the levels of abstraction between problem space and technical solution. Architectural diagrams help.
    - Using KNN, we will be able to classify winning games based on the number of units of each type that are built. Since there are many different types of units, the graph has many dimensions and it would be impossible to classify the replays by hand. By marking games as wins or losses, we will be able to then compare a current game against the classification and figure out if we are on the right path. The classifier will return a probability of a win depending on which path we are on.
    - More specifically, we will have different classifications depending on the race being played which will allow us to compare against the classification since different races require different strategies.
    - For the Q-Learning, we will run many iterations of the game to figure out the optimal way of building a certain number of marines. The algorithm will teach itself the appropriate order of building marines, such as at what point to build barracks.
    - Using the trained Q-table, we will be able to have a human play alongside the AI while it manages building marines while the human is able to focus on more strategic tasks.

- What is your technology stack?
  Development

Unix/Linux
Google style guide
Sublime, Visual Studio Code
Python 3.6
Github

Technology Stack

StarCraft 2
PySC2, StarCraft 2 API
Python
Linux
Google Cloud

- Why is your technology stack the appropriate choice?
  - Python is our first choice for this project because it is versatile and has very readable syntax.
  - Additionally, python has the pysc2 environment and access to Scikit learn, Tkinter, Matplotlib as well as many other packages that will make development fast and easy, giving us more time to work with the gameplay mechanics.
  - PySC2 is already well built for machine learning so it is the appropriate choice for the Q-learner.
- What programming environments are you using?
  - Various text editors and IDEs based on each individual member's preferences (Sublime, Visual Studio Code, Atom, Xcode).
  - Will be running python scripts through the terminal.
- How will you use GitHub?
  - To view all changes to the code we make
  - The master branch will always contain a working version of the AI
  - Branches then pull requests will be created for any changes to the AI as well as different team members separating their tasks from the main branch

- What code quality assurance tools are you using?
    - Github with forks and code reviews before commiting to master
    - We will all follow the [Python Google Style Guide]

Timeline

Week 1 (4/30)	~~Setup environment~~ ~~Make a bot follow a build order~~ ~~Research dependency tree techniques~~
Week 2	~~Make bot smarter by immediately rebuilding buildings as they are destroyed~~ ~~Work on the dependency tree technique to make bot even smarter~~
Week 3	Parse the thousands of replays provided by PYSC2 for labels we are interested in Load the parsed win/lose files to test KNN from Scikit learn (using various k values) Build simple GUI to view probability of winning at each time step of a replay Apply Q- Learning for simple test case (build 2 marines) and scale up Figure out how to train the Q-Learner quickly and efficiently
Week 4	Interface GUI with live gameplay to view plot of probability Apply Q-learning algorithm to help player build 20 marines
Week 5	Present to class.

Feasibility

KNN and visualization of KNN outputs

- Planning on implementing a Tkinter GUI with Matplotlib embedded into the application which will display a bot's chances of winning as the game progresses.
- Pre-parse the replay files using a python script which implements the sc2reader API to write out game events into a file. Suhayb has already written such a file which can successfully parse replay files.
- Sc2reader events will be parsed by K-nearest-neighbors algorithm whose outputs can be run alongside the actual replay to visualize the bot's chances of winning as time goes on, in order to quantify good and bad game decisions.
- It is generally easy to get a Tkinter GUI up and running with a small amount of boilerplate code, and there are many resources available for wrapping Matplotlib and other graphing utilities into the interface.
- Matt has experience wrapping Matplotlib into Tkinter GUI's and live-updating datasets, so implementing the live-action graphics should be feasible in the scope of the project.

- The K-nearest algorithm used from Scikit learn has many resources and examples available that allows our group to test various k values and multiple unit type count.

Q-Learning

- Pysc2 is already very well-suited for machine learning applications.
- There are examples of how to build a Q-learner which we will use to better understand the algorithm.
- The hardest part is figuring out how to quickly train the Q-learner with many iterations.
- The state space will be the number of units relevant to the Q-learner such as number of marines, barracks, SCVs, and command centers.
- The action space consists of build marine, SCV, barracks, command center, or do nothing (meaning wait for more mining to happen)
- Very feasible because we have reduced the state and action spaces to be much more manageable for the algorithm requiring much less iteration to converge.
- The Q-Learning algorithm itself is very simple which makes implementation less of a hassle.

Documentation and Access

How are you sharing code? What about a readme? A website is recommended. A Github page or project is recommended.

- We will be using Github and creating forks and making pull requests with the latest changes. The master branch will always have the most recent working version of the AI.
- The Github page will have a Readme which will be consistently updated as time goes on.
- A PDF of this page will be on the Github repo.

Evaluation

Our implementation will be completely successful if we can have the Q-learning algorithm build a determined amount of marines, assisting the player. We will be able to specify the number of marines we need and measure the algorithm's progress based on how fast it builds the marines using the KNN classifier.

Along with this, the KNN classifier will be successful if we have one of the team members play while a window is able to plot their probability of winning the game based on their current units built in the game so far. If we are unable to do this on the fly, we can save a replay played by one of us, then score it after the game is played.

Plan for Deliverables

We plan to showcase it with one of our team members playing live. It will have the help in building for example, 20 marines using the Q-learning algorithm. As the game advances, the live plot will display the probability that the player will win using the KNN classifier.

Separation of Tasks for Team

Q-learning implementation to build marines:	Max
Q-learning training setup:	Ivan
Implementation of KNN and packaging output data:	Josué
Using the KNN data, give the probability of a win during a live game:	Matt
Replay parsing for KNN algorithm:	Suhayb