Project: RunAwAI

ECS170, Spring 2018

Group Members

Kyle Chickering, krchicke@ucdavis.edu, 914976291
Xin Jin, jin@ucdavis.edu, 912702315
Alexander Mirov, armirov@ucdavis.edu , 999848357
Taaha Chaudhry, tbchaudhry@ucdavis.edu, 915025073
Simon Wu, simwu@ucdavis.edu, 999227965

Problem and Contribution Statement

We are going to implement an AI that runs away from an enemy player and tries to stay alive from a hostile unit as long as possible. We plan to use the DeepMind PySC2 library to customize an environment train the AI. We intend to use an evolutionary algorithm to learn how to avoid opponents and obstacles.

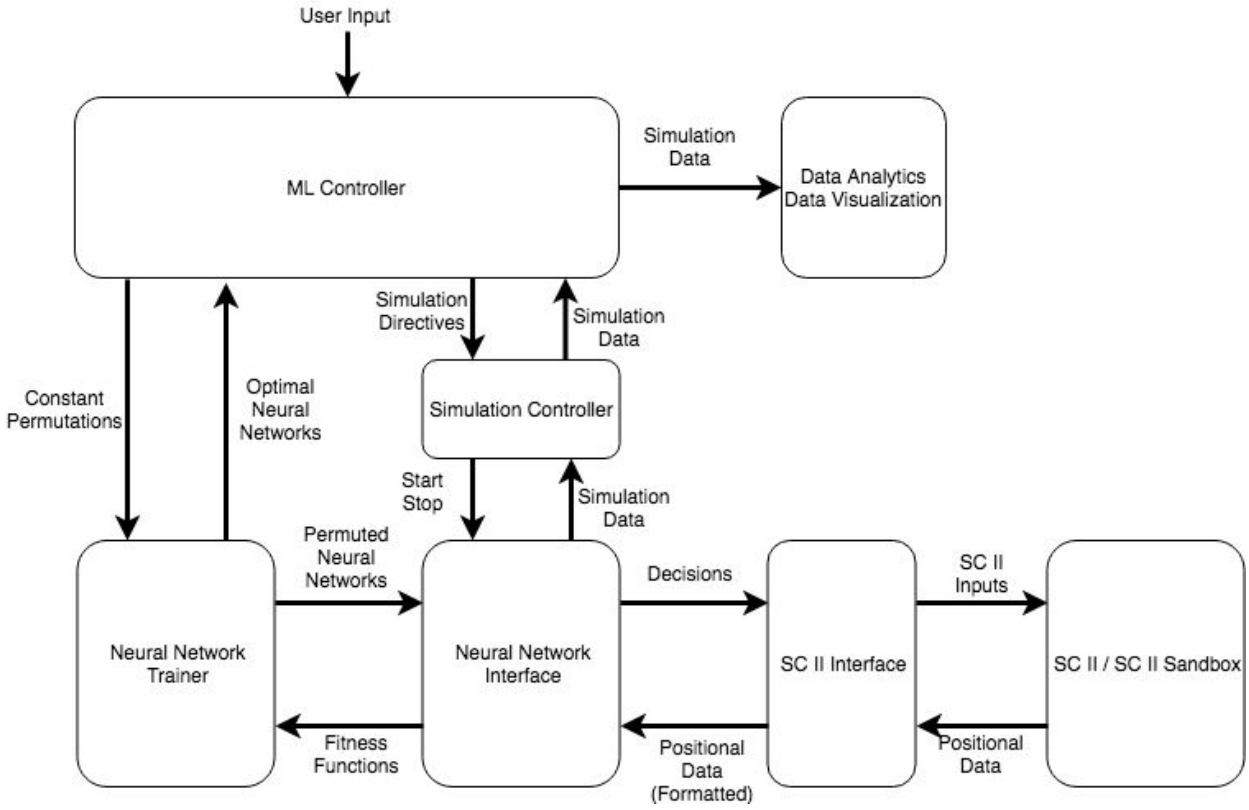
- Why is this problem important to solve?
    - To implement enemy avoidance algorithms. This algorithm could later be implemented as part of a larger AI implementation to develop escape strategy from adversaries and be used in contexts beyond games.
- Does there already exist a solution to this problem?
    - Not explicitly, but this is an inherent behavior to any SCII AI.
- Why is this problem technically interesting?
    - It involves sensing enemy players and learning how to best escape from the enemy.
    - It will demonstrate that capabilities of ML from go from a rudimentary state to a (hopefully) exhibit a complex behavior.
- What possible AI approaches are there to this problem?
    - Genetic Algorithms and Evolutionary Algorithms to train a neural network. Neuroevolutionary algorithms. Gradient descent training.
    - Current existing approaches are very similar to this and use Evolutionary Algorithms. We plan to use the paper *Evolving Neural Networks Through Augmenting Topologies* by Stanley, Miikkulainen as a reference. This was the paper that was used as a guide for the implementation of MarI/O, which had a similar goal and methodology to our proposal.
    - Other existing approaches: PySC2 DefeatRoaches Minigame
- What is your solution and contribution to the solution of this problem?
    - Combination of neural networks and genetic algorithms to avoid enemy players (Neuroevolution).
- Why did you choose your solution given the possible alternatives?
    - The idea that the AI will be able to learn on its own in the environment is more preferable than hard coding the rules.

Design and Technical Approach

- What AI techniques are you proposing?
    - Neural networks and genetic algorithms
- How do the techniques address the problem? Connect the levels of abstraction between problems

space and technical solution. Architectural diagrams help.
    - Evolutionary algorithms work well for our particular problem statement because an optimal solution for running away from a hostile target is generally unknown. By implementing an algorithm that selects the most “fit” solution on its own without considering optimality would be favorable.
    - We will run simulations to train the neural net on a remote Linux machine.
- What is your technology stack?
    - Python with SciKit Learn for the the learning. Python for wrapping the C++ SCII API.
    - DeepMind PySC2 library.
    - Custom Python evolution algorithms tailored to our problem space.
- Why is your technology stack the appropriate choice?
    - Our stack is appropriate because SciKit learn is a popular library in the AI field, and Python is a friendly language for AI.
    - The DeepMind library will help provide the learning environment to train and test our algorithms
    - Python as much as possible, C++ when needed for interfacing with StarcraftII
- What programming environments are you using?
    - Python
    - C++
- How will you use GitHub?
    - We create a private GitHub repository and add everyone as collaborator.
    - Use pull requests and code review to modify the codebase.
    - Use extensively documented commit messages to document changes and implementations.
    - Use the projects tab on Github to track tasks.
    - Convert tasks from projects tabs into issues with corresponding pull requests.
- What code quality assurance tools are you using?
    - Thorough peer code review.
    - Running test simulations using the Linux Starcraft II headless API.
    - Use pylint for python code assurance.

Design Diagram:



Scope and Timeline

Scope:

Develop an AI agent that attempts to maximize time alive (fitness score) from an adversary attempting to drop the health of our agent to 0. Initially, our input space will be any location on a given map. The AI will be able to set random waypoints on the map. The AI will be completely passive/defensive and survive only by avoiding conflict. Based on our progress over the next several weeks, we can expand the input space to include offensive measures and unit/building creation if time allows.

Timeline:

Week 1: Investigate Python API wrapper for s2client-api, ideas for structure of codebase, look into Starcraft 2 learning environment interface.

Week 2: Implement simple genetic algorithm, begin running evolutionary iterations.

Week 3: Run many iterations to train the algorithm. Analyze strategy that algorithm has developed and make appropriate changes if necessary. Start documentation, begin to implement testing suites.

Week 4: Continue running trials, begin writing up results and gathering demos

Week 5: Scale up implementation to handle more enemy units, obstacles, test cases, etc.

Week 6: Testing overall project, complete write up, and buffer time.

Documentation and Access

We'll mainly use Github for code sharing and a readme for documentation. A Github page or project website will further illustrate and demo our project.

Evaluation

Our criteria for success is an AI capable of avoiding being killed by an enemy unit for as long as possible. We will have time as our fitness score and check back with iterations of the trained network to see if subsequent generations are improving the fitness metric. We will gather data by running many trials on a dedicated Linux machine and saving the time survived as well as information about the genetic variations.

Plan for Deliverables

We will have a GitHub Repo and Readme explaining our methods and implementation. We will make some demos and write-ups to show off our AI and the techniques we used to implement it.

Separation of Tasks for Team

Python API Wrapper:	Kyle
SCII API:	Kyle, Simon, Taaha
Data representation:	Taaha
SciKitLearn interface:	Xin, Alex
Documentation:	Xin
Testing:	Alex
Simple genetic evolution algorithm:	Kyle, ALex
Write ups, demos, presentations:	Taaha