Conclusions

***Team Name**: The Super Mario Bros.*
***Topic**: Side-scroller Solving With Intelligent Agents*
***Team Members**: Kevin Patel, Gavin Scott, Jason Swanson, Spencer Mulligan*

Overview

The utilization of neural network and supervised learning agents for problem solving is being researched in a number of fields. In order to better our understanding of both of these types of agents we developed our own and tasked them with playing an implementation of Super Mario Bros., a classic two-dimensional platforming game. By exploring how these agents learn and develop behaviors we hope to develop strategies for our future endeavors into machine learning.

Features and Requirements

In this section, we will discuss our original Features and Requirements and how they've changed, including some aspects of the original specs that caused us difficulties.

Our original project called for 4 agents to be delivered: a reactive agent, a mimic (supervised learning) agent, a neural network agent, and a hybrid agent that utilized the decision making power of both the mimic and neural network agents. Over the course of the project as our design implementation drifted from our original plan it became more and more difficult to implement the hybrid agent. The end result was that while we did implement the first 3 agents, the hybrid agent was not completed under the scope of this project.

The first agent, the reactive agent, was a hard-coded approach to solving levels in Super Mario Bros. It had basic logic that allowed it to recognize enemies and terrain in front of it and jump over these objects as it approached them. This allowed it to solve levels reasonably well so long as they did not require changes in direction as the agent was programmed to always head forward in the direction of the goal.

The mimic agent was the second we developed. It utilized supervised learning in order to learn how to solve levels through "watching" a human player. It performed very well when we were able to feed it data of several human made runs through levels that were similar in structure to what it was encountering. It did occasionally perform extraneous actions however, such as jumping when there was no obstacle.

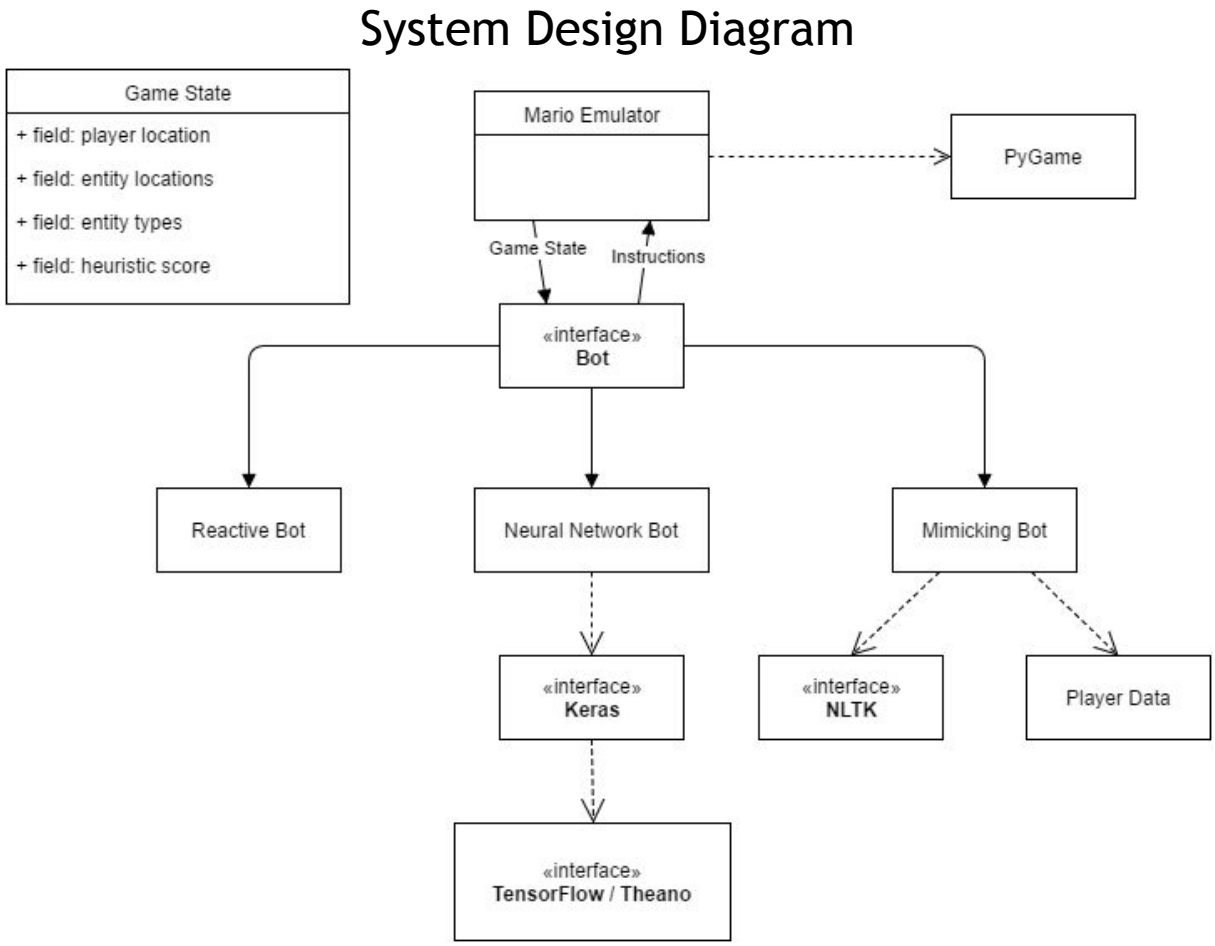
The neural network agent was the final agent we developed. By utilizing data from the game state and a heuristic score based on that game state it was able how to learn how to bypass obstacles completely without human interaction. The main difficulty with this agent,

aside from the setup, was creating a method of heuristic scoring that could definitively tell the agent whether its actions were correct or not.

System Design and Implementation

This section will discuss our resulting system design and implementation. See the System Architecture for our previous designs.

The system design diagram below follows very closely with our original one. It differs in only two areas. One is in that it now lacks the hybrid agent due to our inability to finish it on time. Secondly, the mimicking bot was initially going to use SciKitLearn to power its supervised learning but it was implemented using NLTK's Naive Bayes instead. This was done purely for convenience, the underlying algorithm (Naive Bayes) was the same as we'd planned.As can be seen in the diagram our agents all utilize a bot interface that we created in Python. This interface allows the various bots to control the game character as if they were a human player, permitting them all the same in game actions. The game itself was coded in Python with a heavy reliance on the PyGame modules.



While the reactive bot did not require any additional libraries or data in order to function aside from the game state which it received from the bot interface, the other two agents did. The mimic bot used the Natural Language Toolkit platform in order to transform collections of player actions into decisions in real time. The neural network bot relied on an interface with Keras, which is a high level neural networks library written in Python. Keras in turn utilizes either Theano or TensorFlow (in our case Theano) in order to perform the formation of neural networks and training of said network. These actions happen in the C language and are capable of utilizing the GPU for increased performance.

More information on the details of individual bot implementations can be found in the Implementation document.

Lessons Learned and Future Work

Over the course of this project we utilized a number of tools that going in we knew next to nothing about. The bots we produced, while a good proof of concept, do have areas where they can be improved.

The neural network bot in particular has tremendous room for improvement before it. This improvement would be found in several areas, the first being the heuristic score used to train the bot. Currently this score is based on player position, whether they have been hit by an enemy, the game timer, and the game's player score. Having this heuristic score refined in order to help the neural network distinguish between good and bad actions more readily would immensely help its learning process. The second area where it could be improved would be in the structure of its layers and activations, having more nodes and different types of activation functions for its nodes could allow the neural network bot to become better at learning how to distinguish between situations in game.

The mimic bot also has some area for improvement. While it is good at learning how to best a particular type of obstacle, collections of obstacles still give it trouble. Feeding this bot more player data and perhaps changing the format of the data so that the bot could form models for the different types of obstacle groups would be of benefit.

The last area in which this project would benefit from continuation would be in the completion of the hybrid bot. This bot could be used to showcase how both neural networks and supervised learning could work together to form even smarter decisions.