Implementation

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

Implementation Overview

We implemented each of the three bots separately: explanations of each of these implementations is included below.

Mimic Bot

The game state is taken every 50 frames, unless the player takes an action (changes direction, jumps, etc.). If two states are taken within 15 frames of each other, the earlier of the two is deleted; this prevents too-similar states from being taken with separate actions and prevents confusion. States consist of information on the player's position and the position of all other elements in the level, as well as the current player action (direction and whether or not they're jumping. When determining actions under the bot's control, features are extracted from each state and used to train a Naive Bayes classifier from the NLTK library. The extracted features are simple: for each type of element in the level (pipes, Goombas, etc.), the distance from the player to the nearest instance of that type is saved as a feature.

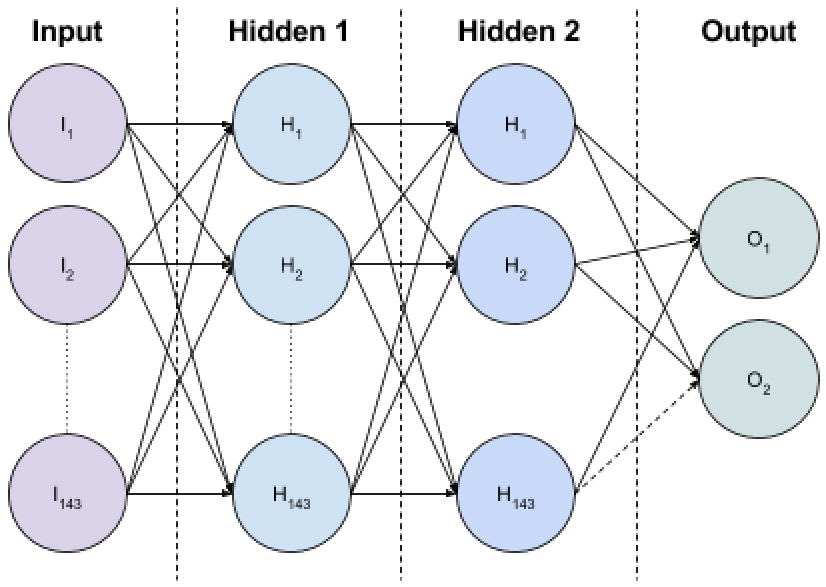
Reactive Bot

The bot receives the game state from the game, and as output returns a tuple containing its decision. As part of the game state the bot receives a list of entities on the screen. Fortunately enough, pipes and blocks are counted as entities which simplies the job of navigating these obstacles. One of the shortcoming of the reactive bot as compared to the more advanced neural network bot is that it struggles in multi faceted situations, such as when an enemy is approaching and there is a low ceiling that prevents jumping. In order to combat this, the bot will retreat for a fixed number of frames in order to draw the enemy into a position in which Mario can safely jump over it. Another common obstacle are pits, which Mario needs to jump over. In order to do this, it is necessary to check the position in front and below Mario to see if there is a gap. This is done by getting a RGB color value from a pygame image file that serves as the level's map.

Neural Network Bot

The neural network for this bot consists of an input layer of 143 nodes, 142 of these receive information about the player and world state while 1 is for bias. The hidden layers hold an equal number of nodes. All layers use the tanh activation function and the first layer also makes use of Gaussian initialization, allowing greater randomness and the potential for faster learning.

Because of the slow speed of evaluation for the neural network, a frame skip of 3 frames is used to prevent it from running excessively. In order to direct the learning of the neural network a heuristic score was used that increases as the bot advances through the level, increases their game score by killing enemies or collecting coins, and decreases with the amount of time taken. The heuristic score also decreases dramatically when the bot is killed by an enemy or through stalling.



Structure of the neural network underlying the neural network bot.

Changes in Design and Implementation

There were a few minor changes in our implementation from how we had planned. First, we had originally planned for a fourth bot, combining the mimicking bot and the neural network bot. The purpose of this bot would have been to examine how the neural network performance, including how quickly it learned, when it was bootstrapped with human-player data. However, we did not have enough time to combine the two bots. 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.