Validation & Evaluation

Team 04: Side-Scroller Solving with Neural Networks
Gavin Scott, Kevin Patel, Jason Swanson, Spencer Mulligan

Evaluation Plan

Our evaluation plan was designed in our "Features, Requirements, and Evaluation Criteria" document. Below are tables summarizing that information, matching each requirement to an overall feature. The rightmost column contains an evaluation of how well our final product met that particular feature-requirement pair.

Evaluation

#	Feature
1	An emulated Super Mario game environment that can be played either manually or through an autonomous agent
2	The game passes a game state to the selected agent each frame and responds to its movement decision
3	Three different autonomous agents in addition to a manual player
4	A reactive agent
5	A player-mimicking agent
6	A neural-network agent
7	An automated procedural level generator for the game based on user-provided constraints

Feature	Requirement	Evaluation (based on the Evaluation Criteria)
1	The user can specify which bot (or manual control) from the command line at startup	This was completely implemented, along with other optional command line inputs (level generation parameters, etc.)
1	When under AI control, the player will not be able to affect the gameplay in any way, and the AI will attempt to finish the level on its own	The user can never influence the actions of a bot once the bot has been given control of the character. This includes taking back control of the character.
1	If an agent finishes the level it will restart the level and continue	This has been fully implemented & tested

1, 7	If an agent finished a generated level a new one will be generated with the same constraints.	This has been fully implemented & tested
1	If an agent fails to complete the level (dies or times out), it will restart the level and try again	This has been fully implemented & tested
1	The game-speed will be able to be increased to speed-up testing of agents	This was not implemented, and was not deemed necessary. Particularly in the case of the neural-network agent, where the decision-making process of the bot was the limiting factor in terms of speed, not the game.
6	The neural network agent will be “killed” if it stands still or fails to increase their score for too long	The neural-network agent is given a small window of time after it stops moving or makes negative progress (moves backwards) before it is terminated. This allows it to make small adjustments to its direction but prevents it from getting stuck or moving back and forth. This was not implemented for the other agents, as neither are expected to improve their performance over time (learning from their mistakes and adapting their strategy).
2	The game state is taken at each frame and passed to the selected bot (if not under manual control)	The game state is passed to a bot interface every frame and requests a decision, leaving the currently active bot up to decide what action to take.
2	Game state includes the position of all objects in frame, including the player control) to receive its movement decision	The game state includes all of this information, although no individual bot makes use of all of it; it is left up to the implementation of each bot to determine which features to implement.
2, 5	Game state is cached frequently to be used as training data for the mimicking bot	The state is saved to a python pickle file every time it is taken while in manual mode, to get human data for the mimic bot
7	The user will be able to provide constraints to the level generator in terms of what game elements they want included in the level, as well as how many of each element	The level generator allows the user to specify which objects (pipes, bad guys, etc.) that they want in the level, and adds them accordingly.
7	The requested elements will be randomly placed throughout the level, ensuring that there is a valid path and no elements overlap with one another	To ensure that every level is playable, the generator sets a maximum number of objects that can be added based on the width of the level, and ensures that no objects overlap.
4	The reactive agent, given a set of rules, will be able to perform well and be used as a baseline to compare to the other bots	The reactive agent was fully implemented.
6	An agent using a neural network will learn how to play based on the provided game states and a heuristic	The reactive agent was fully implemented. It suffered some performance issues based on the complexity of the calculations, but it behaved as expected.
6	The performance of the neural network agent will improve as it plays and learns how best to improve its score	The neural network agent plays noticeably better when it has been left to train for a while

5	A mimicking agent will use data from a human player to learn how to play a level on its own.	The mimicking agent was fully implemented, with limited success. It successfully mimics player behavior and acts in the same style as the player that supplied it with data. However, it often did not perform as well as we expected when given complicated levels that required changes of direction. We expect that this was due to either a lack of enough data, or to how often game states were cached (moving to the right was so frequent that the bot would almost never decide to go left, etc.)
5	A human player will be able to decide at any time whether they want to cede control to the mimicking agent	This has been fully implemented & tested
5, 7	The level(s) the mimicking agent plays will be different than the level(s) the human played when using the level generator	This has been fully implemented & tested

Overall Assessment

Overall, we were pleased with how our project met our initial expectations. We were intentionally vague on the performance goals of each agent because we had no previous experience with the technologies and especially their integration with a video game. To that end we focused on evaluating the learning abilities of each agent, such as requiring that the neural network bot progressively improves as it learns, but not requiring that it learns enough to perform based on some benchmark. Each bot was successfully and performed as expected; the neural network bot noticeably improves as it plays, the mimicking bot acts noticeably similarly to the player controlling it, etc. The area we would most like to improve if given more time would be how states are taken, hopefully allowing the mimicking bot to perform better when player data has left movement as well; we believe the state taking algorithm was at fault for the poor performance here, not the mimicking bot's algorithm. Other than those, and potentially generating more complex levels and continuing to play with the neural net design, we were very happy with how our final product matched up with our initial hopes and the evaluation criteria we designed at the start of the project.