Project Zoran: CelesteBot

An application of NEAT machine learning for testing video games

Matoska Waltz – EECS 499 Section 210: Video Game Independent Research

www.projectzoran.com

Introduction

Awesome Games Done Quick 2019 Super Mario World race between 4 runners. (Image: rockpapershotgun.com)

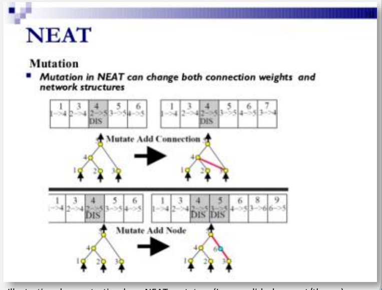
Physical representation of TASbot at Games Done Quick events. (Image: arstechnica.com)

Project Zoran is a project based on the concept of applying artificial intelligence to speed running. Speed running is a hobby in which players attempt to complete games as quickly as possible. Runners practice to improve their personal best time and compete for the world record. This practice pays off at Games Done Quick, a biannual speed running marathon hosted to raise money for cancer prevention.

One problem runners face is the inability to perform certain maneuvers in their speed game without lots of practice. This inability might lead them to believe such a maneuver is impossible. It can take a long time for the community around a game to discover all the tricks possible to optimize times. The TAS (Tool Assisted Speedrun) community helps with this by developing bots that can speed run games better than humans. These runs can take a long time to create, as each input must be painstakingly scripted in advance of the run. Project Zoran is an attempt to develop a tool that can create these TAS bots automatically by leveraging the power of machine learning.

NEAT

NeuroEvolution of Augmented Topologies is a method for generating neural networks. Each NEAT organism consists of nodes and connections mapping a set of inputs to a set of outputs. Organisms mutate in a genetic format, sometimes randomly and sometimes by combining multiple successful organisms. Success is determined by a fitness metric, species with low fitness eventually die out and are replaced by the high fitness species.



NEAT

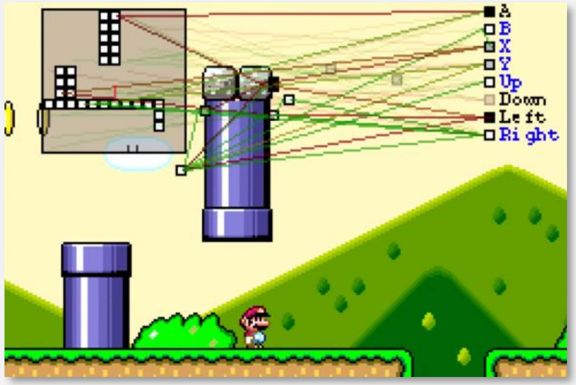
Mutation

- *Mutation in NEAT can change both connection weights and network structures*

Illustration demonstrating how NEAT mutates. (Image: slideshare.net/theavs)

MarI/O

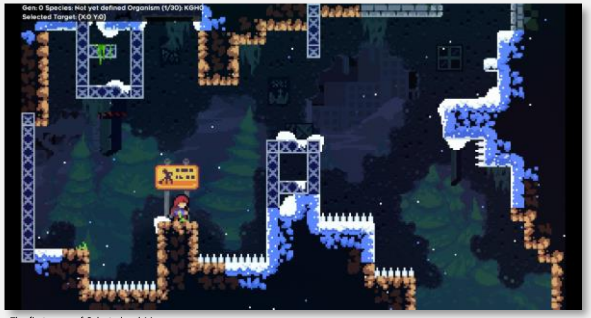
MarI/O is a well known existing application of NEAT for playing games developed by popular Youtuber SethBling. It can play Super Mario World (USA) and Super Mario Bros. It uses a simplified view of the game state as the input to NEAT, and uses the game controller buttons as the output. For Mario, the fitness model is very simple: moving right is always good. If Mario dies or gets stuck, MarI/O moves on to the next NEAT genome and resets the game to a save state at the start of the level. Initially, the bot knows nothing about the game. After many repetitions, it becomes competent and could be confused for a human player.



Machine vision is shown in the upper left, the neural net is represented by red and green lines spanning input and output nodes. (Image: engadget.com)

CelesteBot

CelesteBot is a Celeste mod built using the C# Everest framework, it is an application of NEAT machine learning for playing Celest. It is vision based, it simplifies the region near the player into a grid of colliding tiles and entities. It uses this as input, along with parameters like player position, velocity, player stamina, and a Boolean representing whether the player can dash. These inputs are mapped to outputs on the game pad representing game controls: up, down, left, right, jump, dash, and grab. These controls are then executed in game. This process occurs once per frame. The bot's effectiveness is determined through a performance metric, it is based on distance toward the next checkpoint. Multiple checkpoints can be tracked sequentially, thus a route through a map can be pre-planned for the bot.


Gen: 0 Species: Not yet defined Organism (1/30): KGHO
Selected Target: (X:0 Y:0)

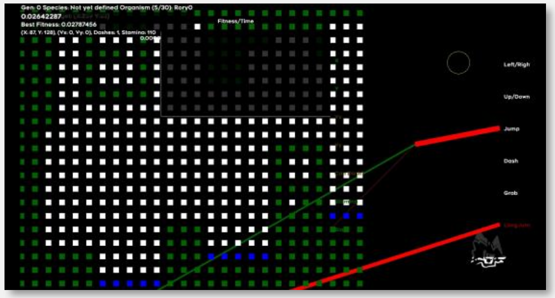
The first room of Celeste level 1A.

Features

This mod contained a NEAT implementation, including all the essentials for running machine learning training. However, it was missing some key features that would improve functionality significantly. My time was spent implementing these features:

- Entity and tile caching – Previously, tile and entity positions weren't being cached and were instead being located through collision detection every frame. By implementing caching, the bot's vision could be expanded from 10x10 to 30x30 without a significant performance loss.
- Data serialization – Previously, all NEAT population data was held in memory. If the game was ever closed, all this training data was lost forever. Serialization allows for the saving and loading of population data between training sessions.
- Fast mode – The game can be run at around 10x speed consistently to get training done more quickly.

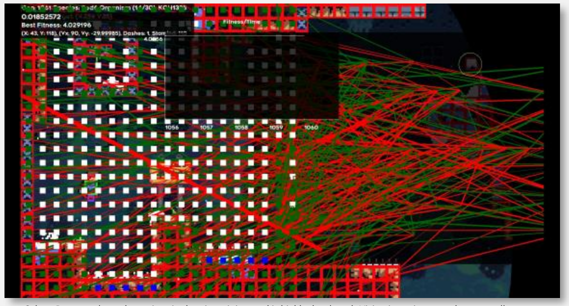
Aside from these major features, I also fixed several bugs, improved the bot's vision for distinct entities, and optimized the machine brain rendering.



Machine vision is shown by the white, green, and blue squares. Neural net is represented by red and green lines and nodes.

Findings

After making these improvements, I ran the bot for about 1,200 generations on Celeste level 1A, with checkpoints present in each room to guide the bot. It was run with a 30x30 vision grid at 10x speed. Each generation tests 30 organisms before evolving, resulting in 30 attempts by the bot to complete the level. With gameplay time, reset time, and processing delays, it took up taking about 20 hours to complete 1,200 generations at 10x speed. In this time, the farthest the bot made it was to the 3rd room. It is possible that running the training with different NEAT settings, such as a higher or lower chance to add a connection or a node, would yield more successful results. CelesteBot still needs work, but it demonstrated some success and is worth investigating and developing further. Perfecting CelesteBot would have significant implications for the rest of the platforming genre, as it would then be possible to automatically generate TASes for most 2D platformers.



CelesteBot neural net shown is quite late in training, and is highly developed. Vision input is mapped to controller output.