Project Zoran: CelesteBot

An application of NEAT machine learning

www.projectzoran.com

Introduction

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. There are 14,865 games on speedrun.com, a site dedicated to keeping track of times in leaderboards (www.speedrun.com 2019). Each game has a community of runners improving the knowledge base about the run and practicing 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. Awesome Games Done Quick 2018 raised $2,295,190.66 and averaged a viewer base of over 100,000 (Games Done Quick 2019).

Awesome Games Done Quick 2019 Super Mario World race between 4 runners.
https://www.rockpapershotgun.com/2019/01/06/charity-speedrunning-marathon-awesome-games-done-quick-2019-starts-today/

Being human, 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 are impressive to watch, but they 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.

Physical representation of TASbot at Games Done Quick events.
https://arstechnica.com/gaming/2016/01/how-a-game-playing-robot-coded-super-mario-maker-onto-an-snes-live-on-stage/

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. More detailed information about NEAT can be found in the original paper. (Evolving Neural Networks through Augmenting Topologies, nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf)

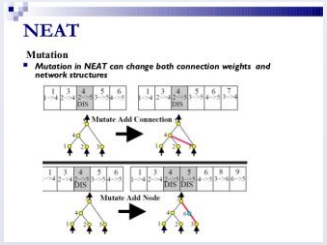
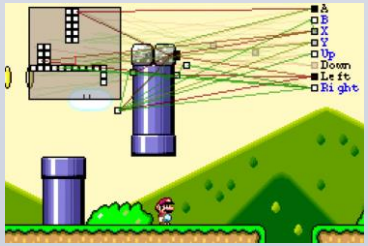


Illustration demonstrating how NEAT mutates.
https://www.slideshare.net/theavs/evolving-neural-networkds-through-augmenting-topologies-neat

MarI/O

MarI/O is a well known existing application of NEAT for playing games developed by popular Youtuber SethBling. Specifically, 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: the 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. This is possible because MarI/O is simply a lua program designed to be run in the BizHawk emulator. Initially, the bot knows nothing about the game. After many repetitions, it becomes competent and could be confused for a human player. For more information on MarI/O, watch SethBling's video. (MarI/O – Machine Learning for Video Games, www.youtube.com/watch?v=qv6UVOQ0F44)



MarI/O operating. Machine vision is shown in the upper left, the neural net is represented by red and green lines spanning input and output nodes.
https://www.engadget.com/2015/06/17/super-mario-world-self-learning-ai/

CelesteBot

CelesteBot is a Celeste mod, it is an application of NEAT machine learning for playing Celeste developed by Github user sc2ad. 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 some other 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 in game controls: up, down, left, right, jump, dash, and grab. These controls are then executed in game. This whole process occurs once every 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. The source code is public and

CelesteBot operating. Neural net is still early in training. Machine vision is represented by the white, green, and blue squares. White is air, green is solid tile, and blue is spike. Neural net is represented by red and green lines spanning input and output nodes.

Features

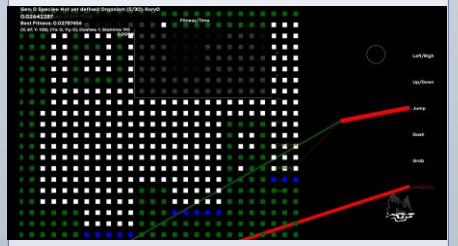
This mod contained an excellent NEAT baseline, including all the essentials for running machine learning training. However, it was missing some key features that would improve functionality significantly. My time was spent implementing these features. Among them were:

- 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 more training done.

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

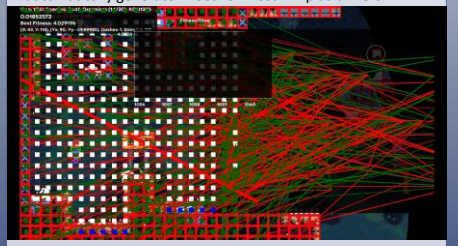
Findings

After making these improvements, I ran the bot for about 1,200 generations. I used 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. The bot waits 4 seconds in real time before resetting, so a generation always takes at least 2 minutes real time to complete. 1,200 generations takes at least 40 hours to complete in real time, and probably much more, but at 10x speed it takes much less. With gameplay time, reset time, and processing delays, it ends up taking about 20 hours to complete 1,200 generations at 10x speed.



CelesteBot operating. Neural net is still early in training. Background is dark so the machine vision is more clearly visible.

In this time, the farthest the bot made it was to the 3rd room. This is a little disappointing, but Celeste is a much harder game than Mario, so it makes sense that the bot would need more training. It is possible that running the training with different NEAT settings, such as the chance to add a connection or a node, would yield more successful results. In any case, this kind of testing will be easier in the future due to the new features I implemented. CelesteBot still needs work, but it demonstrated some success and is worth investigating and developing further. Perfecting a Celeste speed running AI 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 operating. Neural net is quite late in training, and is highly developed.