Universitat de Lleida
Escola Politècnica Superior

02/06/2021

NEAT - Jetpack

Grau en Enginyeria Informàtica

Joel Farré Cortés (78103400T)
Joel Aumedes Serrano (48051307Y)
ESCOLA POLITÈCNICA SUPERIOR – UNIVERSITAT DE LLEIDA

Index

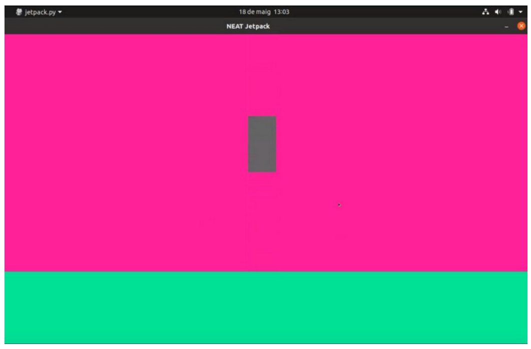
Introduction

The main idea for this free subject machine learning project was to research and learn more about neural networks since it is an unknown field for both of us so we though that we could learn a lot while doing this project. We decided to replicate a well-known game from our childhood called "Jetpack Joyride" and then add neural networks using NEAT to make the AI play the game by itself.

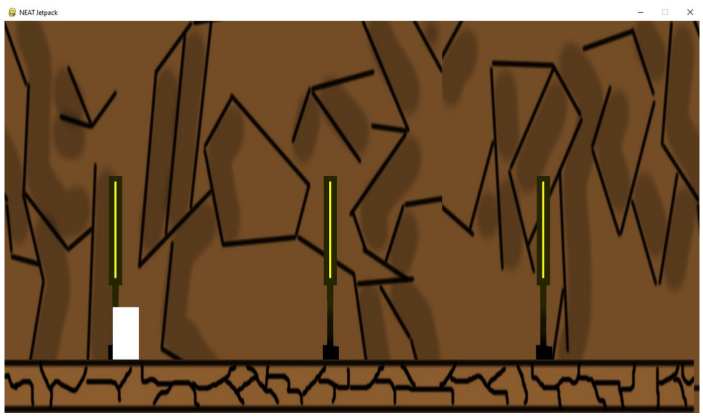
The game

Since we want the game to be easy to modify and we want to avoid problems when implementing the AI, the game will have only one state, a single GameState. When the player dies, the game simply restarts or closes.

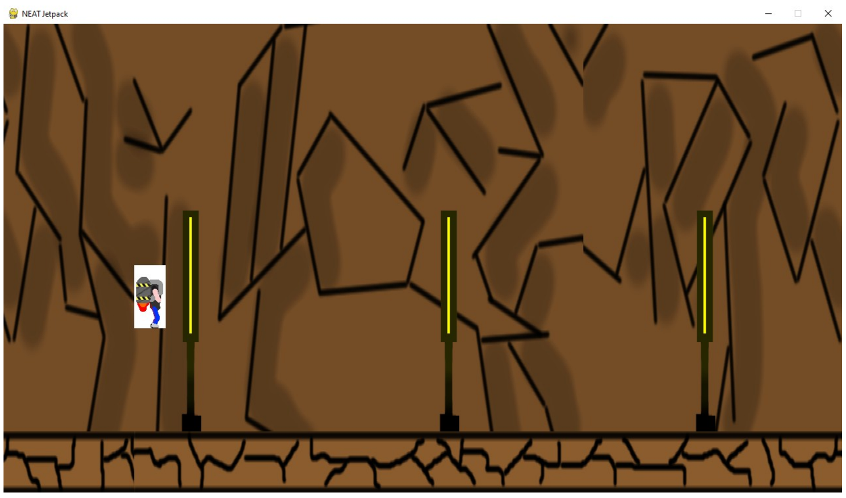
First, we created some rectangles, which we will later transform into the ground, player and background of the game. We then coded the physics for the player, which were simply gravity and the jetpack's force, which allowed us to move our player up and down the screen with an acceleration, making it harder to lift up the player the faster it is falling.



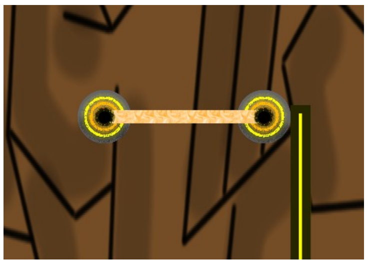
Then, we started creating our sprites. Using a free image editing software, we designed them by ourselves and, in the case of the player, we added some faces of our classmates to make it funnier. Once we had the player we created the ground and background, which are images that give an illusion of depth and movement using parallax scrolling.



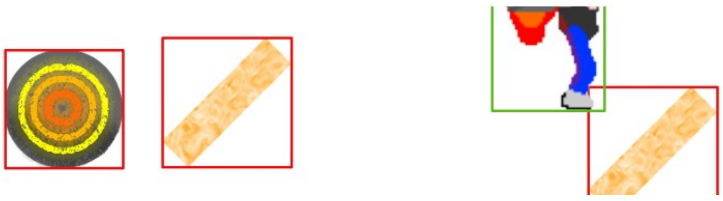
We added animated sprites to our player, to make it look like it's running. Keep in mind that even though it looks like it, the player is not moving along the x axis. The background and ground are.



The obstacles have 3 parts; 2 electric balls, referred to as coils, and the lasers that connect them. These obstacles, referred to as CoilPairs, are generated randomly every execution. They can have 5 possible angles and 4 possible lengths.



However, by just creating them, the player phases through without a reaction. In order to have a collision registered, we need to code it. The most used method for collisions in 2D games is called AABB collision, short for Axis Aligned Bounding Box. This method surrounds the objects in rectangles aligned to the x and y axis of the screen, and simply performs a logic with a linear cost to check if one corner of one rectangle is inside the other rectangle.



However, this method is inaccurate with non-square sprites like ours, as we can see in the image above. To fix these inaccuracies we use mask collision. This other type of collision is pixel-perfect, but very inefficient. It works by creating masks of both sprites, which are just arrays the size of the rectangle signalling in which pixels is there sprite and in which there isn't. Then, it compares these arrays to check for common points. If there are there is a collision. If not, there isn't. To compensate for this inefficiency, in our code we check for AABB collision every frame, since they are very fast, and if there is a collision, we use a mask collision to check if it is indeed a collision, or if otherwise it's a false positive.

The AI

The NEAT algorithm uses neural networks as the brain that processes inputs into outputs, reacting to the events in the game, and genetic algorithms to evaluate the performance of these neural networks, and evolve them into better networks.

Neural networks

During our project we tested different neural network initial configurations. The one that gave the best results used 9 inputs and 1 output.

The inputs are:

- The distance of the player to the floor.
- The horizontal distance of the player to the first laser node.
- The horizontal distance of the player to the second laser node.
- The vertical distance of the player to the first laser node.
- The vertical distance of the player to the second laser node.
- The horizontal distance of the player to the first node of the second laser.
- The horizontal distance of the player to the second node of the second laser.
- The vertical distance of the player to the first node of the second laser.
- The vertical distance of the player to the second node of the second laser.

The output is:

- Activate the jetpack.

Genetic algorithms

In order to give the genetic algorithm a way to know which AI performs better, we implemented a fitness function, which gives scores to the players so that the better the player, the better the score. The function increases a player’s score every frame it survives, giving a bonus if it passes an obstacle, and giving less points if it crashes.

The algorithms generate a population of networks and test them in the game. This population is called a generation, and, once all members of it are dead, the genetic algorithms create a new generation from them using these procedures:

- Inheritance: To transfer the characteristics from parents to descendant.
- Mutation: To randomly change some features to maintain the genetic diversity.
- Crossover: Combining the characteristics of 2 individuals to create a new one.
- Elitism: Preserve the best individuals untouched on the next generation.

Evaluation of configurations

We tested different possible configurations of the initial networks and the population size to check its differences.

5 inputs and population size 20

Due to the small population size, improvement is very rare. It has taken more than 20 generations to start getting past more than 150 fitness, which has been done purely out of luck. Still by generation 30 almost no visible improvement is seen. By generation 36, the total record is 172 fitness.

5 inputs and population size 100

A vast improvement is seen just by increasing the population size to 100, which reaches 600 fitness in just gen 2. This happens because the amount of population generates a lot more random possibilities, and thus getting us closer to a good AI in less time. On generation 12, the players have realized how to avoid obstacles and that they get bonus fitness for staying off the ground and away from the ceiling, reaching 1200 fitness.

5 inputs and population size 200

By doubling the population, as expected, we get the same results in half the time.

9 inputs and population size 20

If we use 9 inputs, we can also give to the player information about the next 2 lasers in order to have more anticipation. However, with just 20 members, we face the same problems as before, evolution is slow and high scores are purely based on luck. The most notable example is when a species gets a really high score by staying on the ceiling, and then when a laser appears on the ceiling nothing is done to dodge it. No real improvement except staying on the ceiling has been made by gen 24.

9 inputs and population size 100

With 9 inputs and a population of 100, it takes approximately 15 generations to realize the importance of not staying always in the ceiling, finally obtaining players actively dodging obstacles. However, it also takes some luck to get past 1000 by gen 10, because of the possible over-fitting caused by obstacles not appearing on the ceiling or ground.

9 inputs and population size 200

With 200 players per gen, we can see a player reach a fitness of 900 by gen 9. The large amount of players makes learning faster and more resistant to over-fitting. A few generations later, at gen 21, it has reached 100000.

Conclusions

To sum up we enjoyed a lot developing this project and also watching the AI playing alone and viewing how was it improving his records so we had a great time.

Due to our previous knowledge of game design and development, it was easy and fun developing the game from scratch, from coding the physics and logic to drawing our own sprites.

On the other hand, it was also interesting to do research on a topic we had never worked on before, such as the NEAT algorithm. We did not struggle in any concrete section of the project thanks to the vast amount of content on the Internet, and we also understood how do the neural networks work and how can we implement them in a project.