Doodle Jump'NN

By: Kyle Chan, Ryan Lu, Tarmu Hu, Enrique González

1. Problem Statement:

In Doodle Jump, the users control "Doodler", a character that continually jumps upward across platforms without falling. The platforms include:

- Green: Standard platform that doesn't move and provides normal upward boost
- Brown: Break upon contact
- Blue: Move horizontally but provide normal upward boost.
- Green with springs: Provides an extra high boost for greater upward progress.

Our goal: develop an AI player that uses neural networks with genetic algorithms to learn game mechanics and get high scores.

2. Approach:

We began with a three-layered neural network architecture that consisted of an input, hidden, and output layers. While the model worked as intended, the performance fell short of our expectations. A frequent issue we ran into was that the "doodlers" would struggle to make decisive choices between adjacent platforms. We suspected that our initial model's complexity might be too complex for the problem.

Our second iteration simplified the neural network to a two-layered architecture, which consisted of only input and output layers. This simpler architecture performed a bit better, but introduced new issues. One example, the "doodlers" would develop repetitive patterns – either would jump in one spot or would move in a single direction. We suspected that these issues came from the network's reduced ability to process non-linear relationships within the game environment.

Finally, in our last iteration, we improved the original model by adding a second hidden layer, which creates a four-layered neural network that consists of an input, two hidden and output layers. These changes increased the network's capacity to learn and process complex game patterns. We found that adding depth to the network, rather than making it simpler, provided a better balance of efficiency and capability.

3. Description of the software

Programming Language and Libraries:

- Written in Python
- Key libraries:
    - Pygame: Game engine
    - Random: Random # generation for game elements
    - Time: Time tracking and game loop management
    - CSV: data collection
    - Numpy: AI input processing & neural network computations

Main Components:

- DoodleJump.py: Controls game loop and main mechanics, handles platform generation, tracks score/game state and screen scroll
- Platform.py: Generation of platforms(green,blue, and brown), handles platform movement/collision detection and handles platform generation probability depending on score.
- Player.py: Handles character movement, “doodler” decision making, sprite management, collision detection with platforms and “vision” for neural network input.
- ga.py: Manages population of “doodlers”, handles selection and mutation of best performers, and creates new generation based on fitness.
- neuralnet.py: Neural network consisting of 5 input neurons and 3 output neurons along with an activation function that helps the “doodler” decision making.

We used an existing open source AI Doodle Jump implementation for our base foundation. This allowed us to focus on developing our own neural network. We also added new features to the base game including a time trial mode with timer. Also, we implemented screen scroll to follow the “doodler” movement.

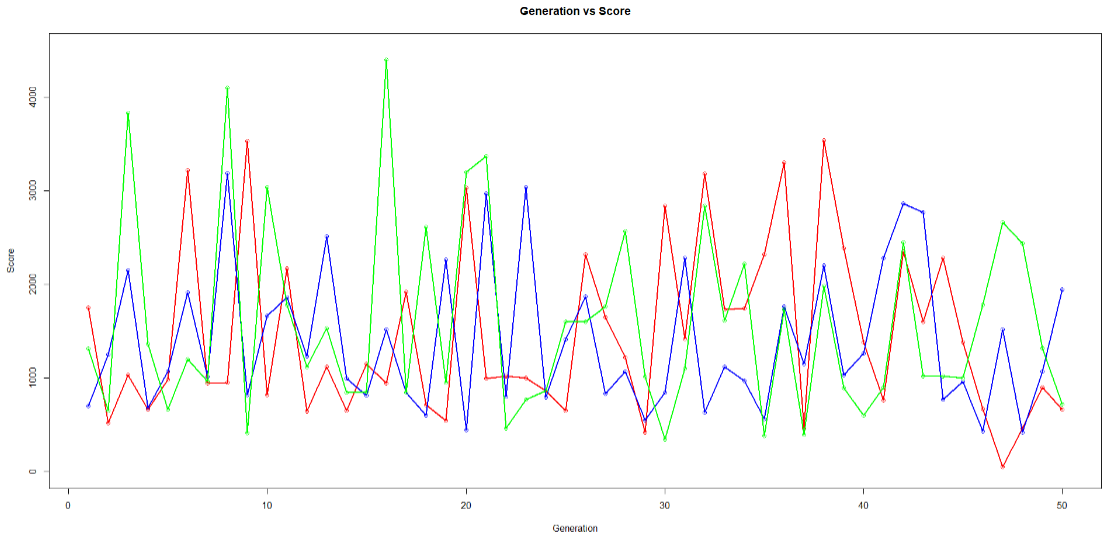
Game Modes:

- Highscore mode: “Doodler” tries to maximize its score without time limit
- Time trial mode: “Doodle” aims to maximize its score within 30 secs.

Data Management:

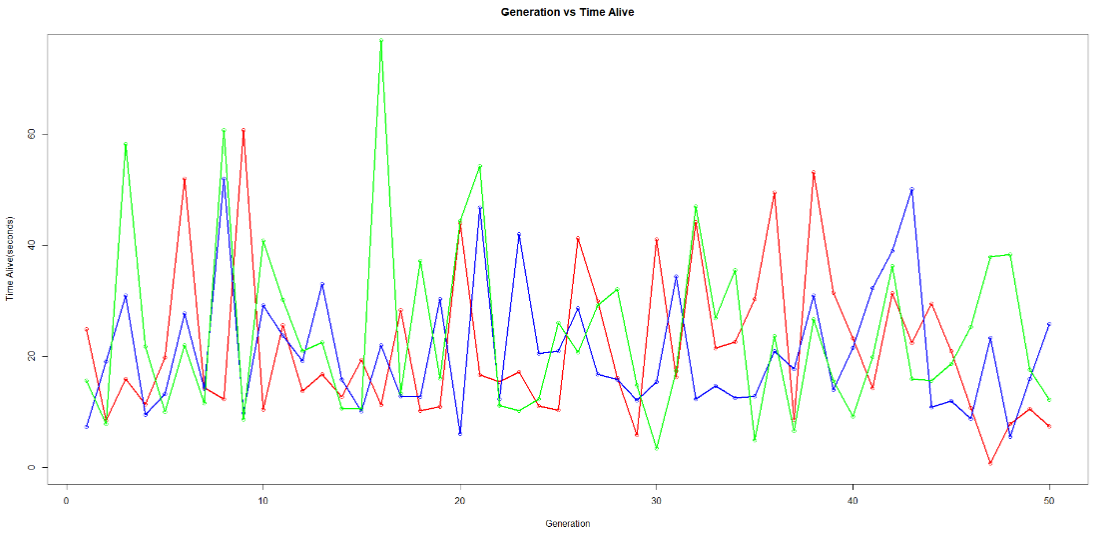
- Results are saved to csv file “v3_results”, which tracks generation number, time alive, and score. Used for analyzing “doodler” performance over each generation.

4. Evaluation



Highscore Evaluation:
Generation vs Score

Blue - v1
Red - v2
Green - v3

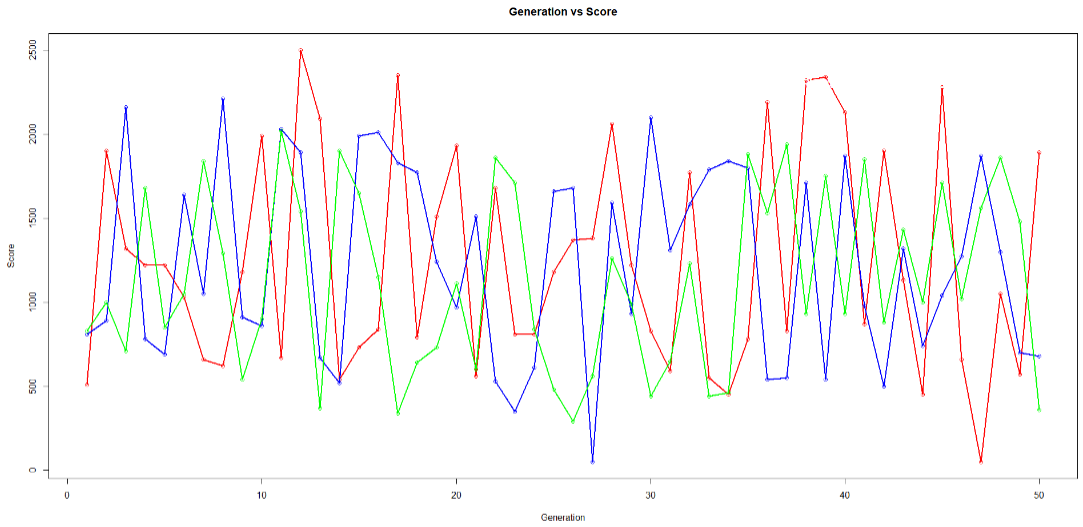


Highscore Evaluation:
Generation vs Time Alive

Blue - v1
Red - v2
Green - v3

V1: survived 20.891 seconds and scored 1,393 points per run
V2: survived 21.671 seconds and scored 1,456 points per run
V3: survived 24.093 seconds and scored 1,508 points per run

Version 1 of our program performed the worst overall. Version 2 produced the most consistent outcomes. Version 3 survived the longest and achieved the highest average score in 'Highscore' mode. However, while Version 3 reached very high scores, it also experienced frequent low scores.



Time Trials Evaluation:
Generation vs Score

Blue - v1
Red - v2
Green - v3

V1: survived 20.305 seconds and scored 1,237 points per run
V2: survived 17.386 seconds and scored 1,246 points per run
V3: survived 19.696 seconds and scored 1,121 points per run

Our Version 1 of the program achieved the highest score in the 'Time Trials' mode and maintained consistent performance without the significant lows observed in Versions 2 and 3. Version 2 performed the worst in terms of time but excelled in scoring the highest points. Version 3, surprisingly, performed poorly compared to the others, losing to Version 1 in time and recording the lowest score.

5. Conclusions:

Lessons learned:

- Building on existing code saved us time. We got to focus on building the neural network instead of building doodle jump from scratch.
- Training insights, for example genetic algorithms can be used to effectively learn mechanics of doodle jump or game mechanics in general.
- Project Management

Ideas for improvement:

- Improve neural network architecture, maybe adding more inputs, neurons, or hidden layers.
- Improving the genetic algorithm parameters and fitness function.

6. References

https://github.com/EthanBautista/Doodle-Jump-AI