Super MeatBoy AI vs. Human Participants Report

Research Written and Performed by
Doriana Othman

Problem Description:

As the video game industry continues to grow and develop, so do those who are consuming its media. Those playing games are constantly seeking new challenges and immersive experiences to improve their excitement and enjoyment while indulging in games. A large part of enhancing this technology is through the use of Artificial Intelligence. NPCs (Non playable characters), pathfinding, decision making, procedural modeling and player experience modeling are all aspects of games that can be improved through the use of AI to advance player experiences while playing video games. In this report, I will be focusing on the problem of trying to create a suitable AI for the video game *Super Meat Boy*. The goal of this AI will be to get to its target as quickly as possible in as few deaths as possible.

Background Information:

Video Game Description- *Super Meat Boy*:

Super Meat Boy is a tough as nails platformer where you play as an animated cube of meat who's trying to save his girlfriend (who happens to be made of bandages) from an evil fetus in a jar wearing a tux. Our meaty hero will leap from walls, over seas of buzz saws, through crumbling caves and pools of old needles. Sacrificing his own well being to save his damsel in distress. Super Meat Boy brings the old school difficulty of classic NES titles like Mega Man 2, Ghost and Goblins and Super Mario Bros. 2 (The Japanese one) and streamlines them down to the essential no BS straight forward twitch reflex platforming.Ramping up in difficulty from hard to soul crushing SMB will drag Meat boy though haunted hospitals, salt factories and even hell itself. And if 300+ single player levels weren't enough SMB also throws in epic boss fights, a level editor and tons of unlockable secrets, warp zones and hidden characters.

Light World Levels:

In *Super Meat Boy*, “Light World” levels are the standard levels included in the base game. These levels are considered to be the normal difficulty in the game.

Dark World Levels:

In *Super Meat Boy*, “Dark World” levels are the harder levels included in the base game, which can normally only be accessed after getting an A+ rating in its “Light World” level version; this is achieved by reaching or beating that “Light World” level’s time limit. Each “Dark World” level will have a similar looking environment to its “Light World” counterpart. Increases in difficulty usually include more obstacles that will cause harm to Meat Boy or hinder pathways towards his goal.

<u>Methods Used:</u>

Pygame:

To make the implementation of AI easier, Pygame was used to recreate the game *Super Meat Boy*. Key features of the character movement (Meat Boy) were kept; this includes walking, running, and wall jumping. Character animations were kept to a minimum and were only included to differentiate between movements during play. All graphics used are identical to *Super Meat Boy*; this was done to create familiarity between the pygame version and the actual version.

Level Selection:

For this report, The first three Light World and Dark World level designs were used from World 1 of *Super Meat Boy*. These levels were chosen for their simplicity of recreation, ease of introduction to game mechanics, and time scope of this project.

Level Creation:

To make implementation easier, each level was implemented as a list of strings with characters that represent different behaviors. Every level had 46 rows of 70 characters each; P represented Player, T represented Target, X represented Wall, D represented Death. Where these characters are located match where its represented behavior is on the screen; this style of implementation helps pygame determine how Meat Boy (the player) should react in the game. If the programmer or user wants to see this visually represented, the “render_blocks” option can be set to True in “settings.py”. All of the behavior related to how the player reacts in the game can be found in “level.py”.

AI Methods:

To create an AI, machine learning was used with a combination of Genetic Algorithms and Neural Networks during training. The general process of training functions as follows: a predetermined number of Meat Boys are created per generation. In each generation, each Meat Boy has its own Neural Network that determines the pathway it takes and when Meatboy should jump. For each generation, the fitness of each Meat Boy is determined by the length of its pathway and if that Meat Boy is alive; the average and best fitness of each generation is then reported. The next generation is then created based on the predetermined percentage of “good” and “bad” Meat Boys to keep, with the remainder being newly generated Meat Boys. Training happens for as long as the user desires; the weights produced can be saved and/or loaded if wanted through “settings.py”.

Pathway Algorithm:

To determine the best pathway distance for each Meat Boy, BFS was used; this was specifically used as a fitness measure to rank the Meat Boys in each generation, however, if the actual path taken from the Neural Network was better it will use that as its fitness value. Although A* would be the better algorithmic choice, due to the amount of time that would be needed to generate weights for each level, it was decided that BFS would be the best approach for this project iteration.

Neural Networks:
The Neural Networks in this implementation represent the “brain” of a Meat Boy; this determines the path that a Meat Boy takes and also determines if it should jump. The weight values of each Meat Boy are only changed based on the predetermined percentage chance set by the programmer. The Neural Networks initial input layer, hidden layer, and outer layer values are also initialized by the programmer.

Genetic Algorithms:
Genetic Algorithms is the method used to improve each generation of Meat Boys. This is done by using an initial cut off percentage to determine the number of “good” and “bad” Meat Boys of each generation based on their ranking; the number of “good” and “bad” Meat Boys kept for the new generation is decided by initial predetermined percentages set by the programmer. The remaining Meatboys needed to fill the new generation are then determined by picking random “good” Meat Boy parents and creating new children based on a combination of their parent’s weights from their Neural Networks; the percentage taken from each parent is based on a predetermined percentage value. Another predetermined percentage also decides if these newly created children will have their weights modified. For each new generation, all of the “bad” Meatboys kept will have their weights modified.

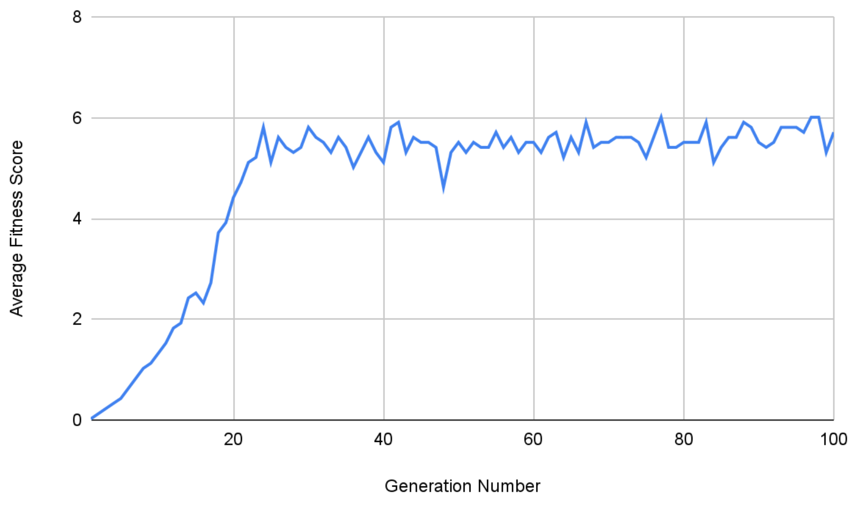
<u>Results:</u>

For this report, I will determine success based upon if after training for 100 generations per level that a successful test run can be achieved for each one. The overall goal is that Meat Boy can safely reach its target, bandage girl, without failure. Although many different variations of settings can be used, I have decided to use the settings below based on experimentation. For each level tested, both Dark World and Light World, the Meat Boy with the best weights was used for its Neural Network. Since all of the obstacles in these levels were static, it was unnecessary to conduct multiple tests as the same results would be achieved. The max average fitness level that can be achieved for a given level is a score of 10.

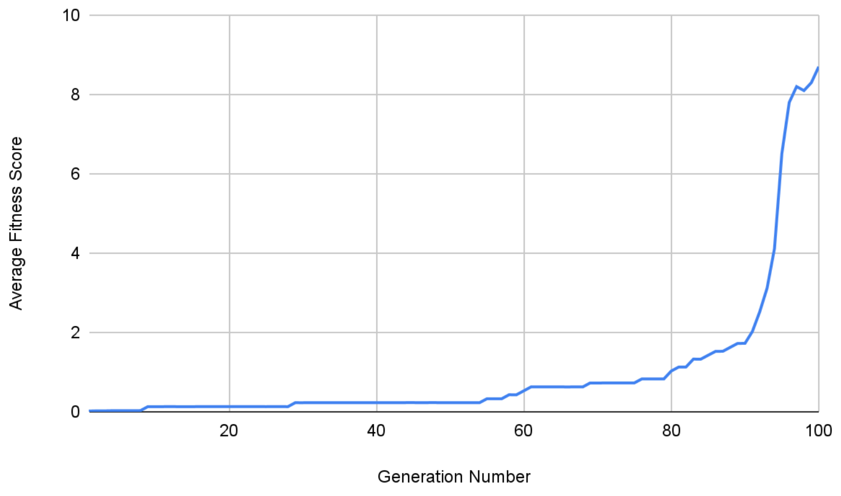
Observations:
For each level, regardless of difficulty, the target was reached in its corresponding test run. All levels also experienced a notable increase in average fitness score once a Meat Boy found its target in a given generation. Light World levels were able to reach its target fairly quickly in early generations while Dark World levels typically did not reach it until much later; this is unsurprising as Dark World levels had higher difficulties. Dark World level three had 2 notable spike increases, once when it found the target around generation 90, and another around generation 175. During the training process, Meat Boys were able to learn running, jumping, and walking tasks fairly easily, however, had a much more difficult time learning how to wall jump or use a combination of different motions. Meat Boys would also occasionally get stuck jumping between two walls; while amusing to watch, this proved to be an ineffective strategy when trying to reach the target.

Settings:
```
get_nn_inputs = True
max_alive_time = 7
render_blocks = True
save_img = True
tile_size_x = 10
tile_size_y = 10
screen_width = 700
screen_height = 468
scale_factor = 0.35 if lvl in [1, 2, 3, 6, 7, 8] else 0.2
jump_speed = -9 if lvl in [5, 7] else -10
GEN_SIZE = 100
generation_size = GEN_SIZE if train else 1
IN_INPUT, N_HIDDEN, N_OUTPUT = 3, 10, 2
MUTATION_WEIGHT_MODIFY_CHANCE = 0.1
MUTATION_ARRAY_MIX_PERC = 0.5
MUTATION_CUT_OFF = 0.5
MUTATION_BAD_TO_KEEP = 0.1
MUTATION_MODIFY_CHANCE_LIMIT = 0.1
```

Training Light World Level 1:



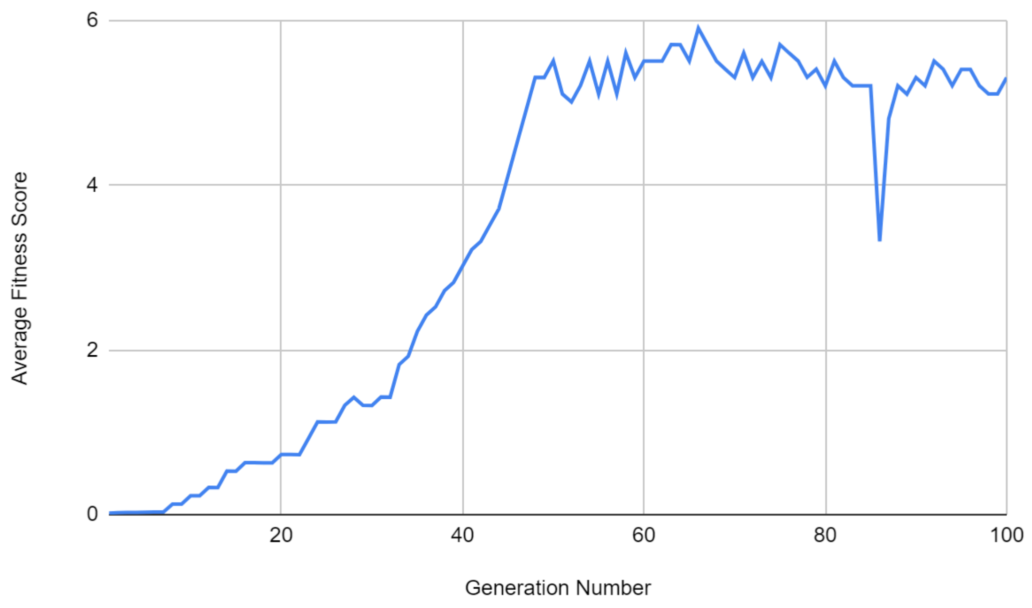
Training Dark World Level 1:



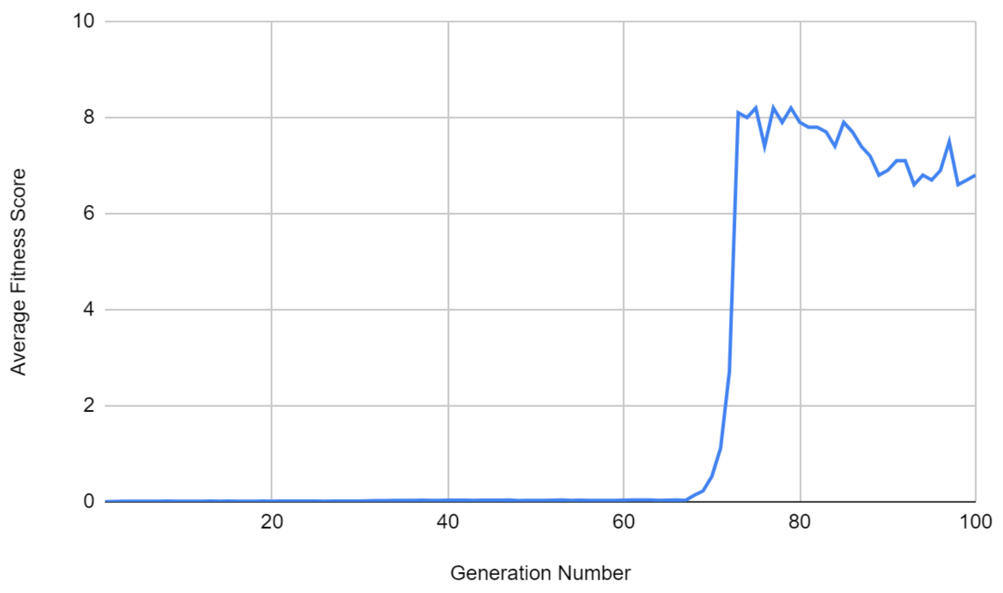
Testing Level 1:

World	Target Reached
Light World	TRUE
Dark World	TRUE

Training Light World Level 2:



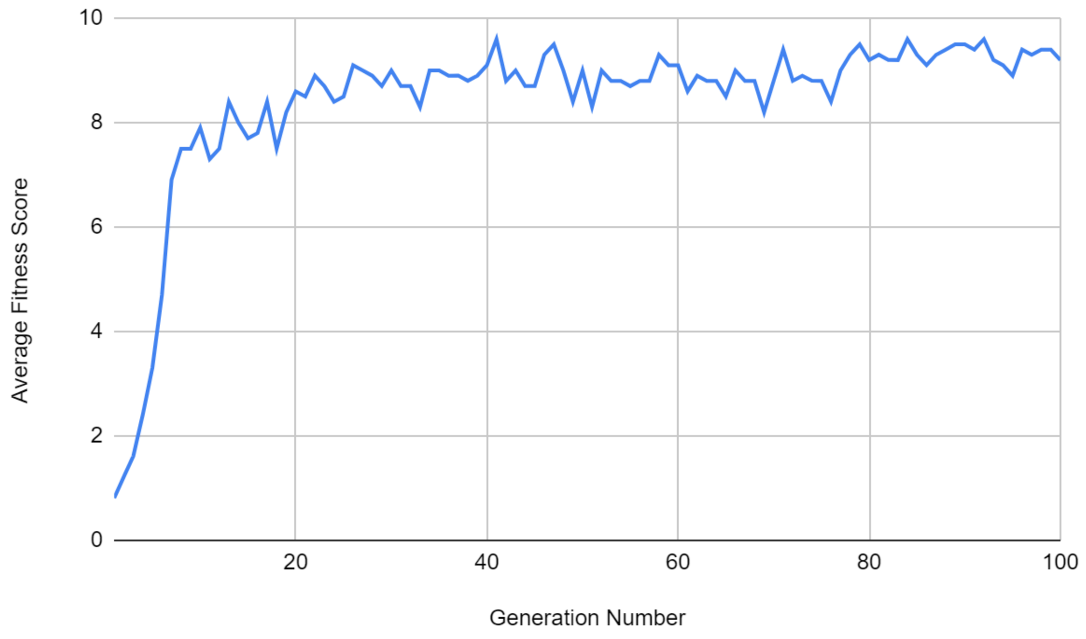
Training Dark World Level 2:



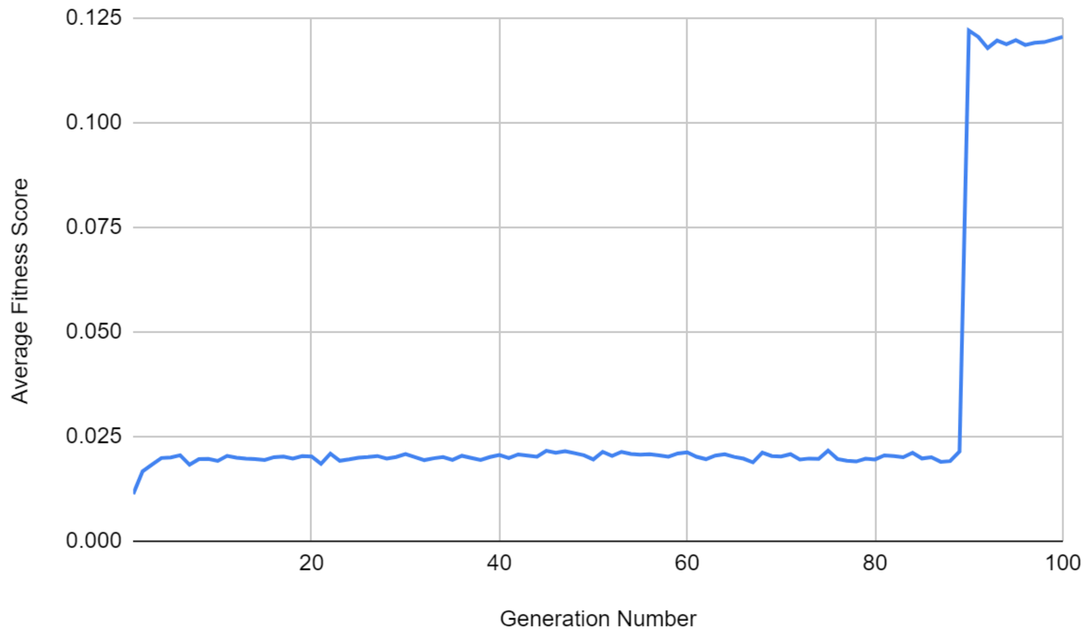
Testing Level 2:

World	Target Reached
Light World	TRUE
Dark World	TRUE

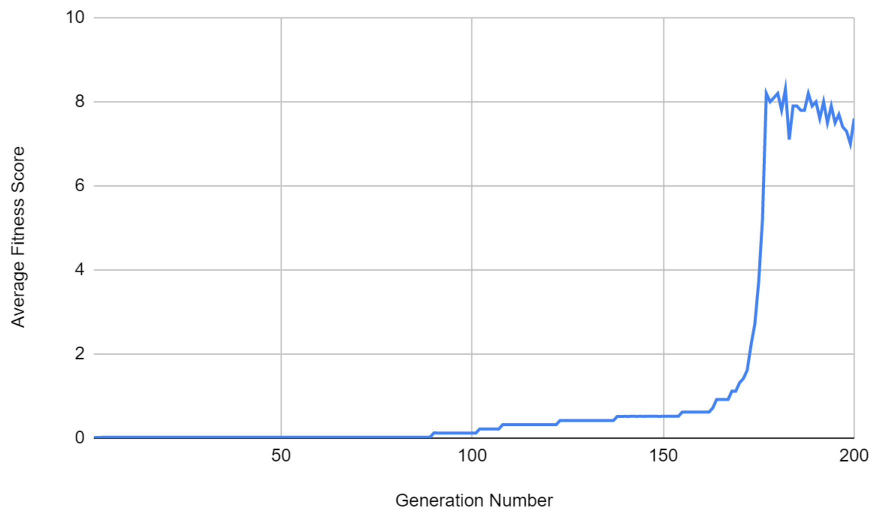
Light World Level 3:



Dark World Level 3 (100 Generations):



Dark World Level 3 (200 Generations):



Testing Level 3:

World	Target Reached
Light World	TRUE
Dark World	TRUE

Conclusion:

From the gathered results, the generated AI was a complete success! I found it particularly interesting how the AI may perform poorly for the first several generations in training, but improve performance dramatically once it has found the right combination of weights that allows it to reach its target safely. Although this project installment was a success, there are still a number of adjustments that can be made to improve this AI further. As mentioned in the methods section of this report, it would have been better to implement an A* path searching method for the fitness calculation; once implemented this would improve the measurement of what is considered a “good” Meat Boy. For further installments I would include time measurements of successful Meat Boys in the fitness calculation to encourage better efficiency among Meat Boys. More difficult levels in *Super Meat Boy* require implementing moving obstacles, which can produce some interesting results from future AI installments. Lastly, it is worth noting that other implementations of AI tactics may curate better, or more interesting, results that would be worth investigating and comparing in future works.

Literature Review:

One of the earliest examples of AI in a computerized game was from a game called Nim in 1952. Although advanced for its time, Nim took the form of a small box that was able to win games against fairly skilled and experienced players. Following this, the Ferranti Mark 1 machine from University of Manchester ran a checkers program that also was able to outsmart several players. Surprisingly, these were some of the earliest computer programs ever written! IBM's Deep Blue computer would eventually be the next impressive installment in AI to defeat chess grandmaster Garry Kasparov.

After these AI installments, the popularity of single player game modes started appearing as a result because of these technological discoveries. The type of AI experienced by these games were limited to enemy movement with stored patterns. As computational power increased, more random elements would be able to be included in these enemy movements. One of the most notorious early uses of early single player video game AI was used in the game *Space Invaders.* The increase of difficulty between levels and distinct enemy movement patterns based on player input put a new edge into video game AI that had not been seen before.

This use of AI exploded into many different game genres afterwards. *Pacman* in 1980 was the first to use AI patterns in maze games, and *Karate Champ* in 1984 would be the first for fighting games. We would even see AI begin to appear in sports games like *Madden Football*, *Earl Weaver Baseball*, and *Tony La Russa Baseball*. Eventually these new AI game genres would inspire the creation of formal AI tools like finite state machines. Roleplaying and Real Time Strategy games using these AI tools would often be littered with issues due to using incomplete information that would be needed for these algorithms; thankfully games developed later in this genre recovered and developed more sophisticated AI methods.

AI in modern games usually adopt a bottom-up approach when it comes to creating new AI; bottom-up AI behavior is categorized as using information from its environment to piece together new information. This means that newer AI development in games will model their behavior based on player behaviors and data to determine actions and analysis.

Resources

A brief history of computing. AlanTuring.net A Brief History of Computing. (n.d.). Retrieved November 28, 2022, from http://www.alanturing.net/turing_archive/pages/Reference%20Articles/BriefHistofComp.html

Eugene F. Grant, R. L. (1952, July 26). *It*. The New Yorker. Retrieved November 28, 2022, from https://www.newyorker.com/magazine/1952/08/02/it

Game ai revisited - yannakakis.net. (n.d.). Retrieved November 28, 2022, from http://yannakakis.net/wp-content/uploads/2012/03/gameAI.pdf

Guardian News and Media. (2021, July 19). *Think, fight, feel: How video game artificial intelligence is evolving*. The Guardian. Retrieved November 28, 2022, from https://www.theguardian.com/games/2021/jul/19/video-gaming-artificial-intelligence-ai-is-evolving