Flappy AI:
Side-scroller Runner Genre Game Tester AI Using NEAT

Submitted by:

Chavez, Miguel Antonio

Encabo, Johndel

Lamadrid, Christian Kalki M.

Viñegas, Edward Raphael

Vitug, Maria Cassandra

Submitted to:

Ms. Sheryl Kamantigue

Purpose and Description

Gaming is a big industry, it racks up numerous consumers, and the demand for new video games goes up over time with the video game market expected to grow by three times the market value in the next 10 years (Saha, 2023). With more demand for video games, more will try to develop them. Before games can be published for public use, they must first be assessed for any issues that might hinder the enjoyment of their players. Testing helps with quality assurance which helps create a positive impression among players and attract more people to play it. Consumer satisfaction also goes a long way for games because it can help retain consumers to the game and future releases (Syed, 2019).

While testing is important, smaller developers will not have the resources to hire game testers to check every part of their game, they need to find numerous things to fix such as game balance, game physics, and other entertainment related aspects to ensure the quality of their game (Santos et al., 2018). This is where AI can help fix this issue, wherein an AI can assess videogames, different aspects can be tested such as balancing, exploration, and bugs (Politowski et al., 2022). Politowski et al also found that building complex specific AI testing models is more expensive than hiring a game tester and suggested that employing an open-source general video game testing tool would be the practical solution to overcome this problem.

The goal for this project is to develop an AI that will be able to play a specific genre of games, this is a step above having an AI that can only play one game, the proposed AI will be able to play the side-scroller runner genre, this genre is where a character will move in one specific direction and survive as long as possible, an example would be games like Flappy Bird, wherein the character must navigate across obstacles to survive for as long as possible. The developed AI would then be able to be exported and used for games within its trained genre without much complexity.

Scope and Limitations

This project will focus on the side-scroller runner genre, specifically games like the mobile game Flappy Bird. There are numerous variations of the side-scroller games such as Jetpack Joyride, wherein you can collect powerups and the ability to destroy obstacles, or zombie tsunami wherein there is effectively an hp meter that you must keep up a certain level.

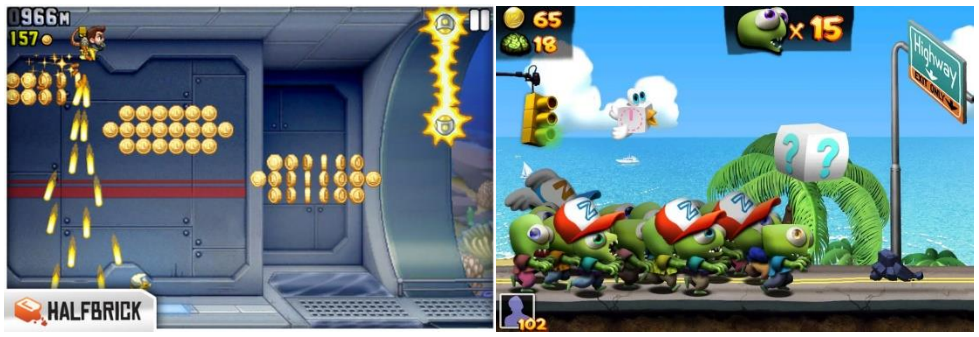


Figure 1. Jetpack Joyride and Zombie Tsunami

Source: arstechnica.com; zombie-tsunami.fandom.com

These games have different mechanics that make them unique, however they all still abide by the general mechanic of avoiding obstacles. These obstacles can be pipes like in Flappy Bird, lasers in Jetpack Joyride, city objects in Zombie Tsunami. To this end, this paper will focus on the ability of the AI to avoid obstacles, instead of accommodating each unique mechanic of every side-scroller runner game. For simplicity's sake, simple games like Flappy Bird will be developed to train the model, which will help focus on obstacle avoidance instead of other gameplay mechanics.

There are numerous ways to give game data to the AI model, the simplest one and closest to the real world is by inputting raw image data to the model, similar to what happens in the real world where humans can only know what they see in front of their screen. However, this is more difficult for computer models, because by sending image data every single frame of a videogame would be computationally expensive, thus causing performing issues for the game as well as receiving less accurate data from what is is inside the game. To this end, image data will not be used and game data from within the game will be used as input for the AI.

Target Audience

The project's target audience comprises a diverse array of stakeholders within the gaming industry and the artificial intelligence field. Game developers, particularly small or indie studios facing resource constraints, stand to benefit from the implementation of AI-driven testing solutions, offering a cost-effective alternative to manual testing processes. Additionally, AI researchers and developers interested in exploring AI applications in gaming would find this project compelling, as it delves into the technical intricacies of developing AI models tailored to specific game genres. Professionals within the gaming industry, including designers, producers, and quality assurance teams, would also be interested in learning about innovative approaches to game testing and quality assurance. Moreover, educational institutions could leverage this project as a valuable resource for educational purposes or as a case study in computer science, AI, and game development programs. Tech enthusiasts and hobbyist game developers intrigued by the intersection of technology and gaming would also likely be interested in the potential of AI-powered game testing tools and their implications for the future of the industry.

Methodology

Pygame will be used to create the game, Pygame is a free open-source Python library that is used to develop video games. It uses the Simple DirectMedia Layer library to help make writing games easier. Pygame was chosen due to its simplicity and because it uses Python, this is important because we will be using another Python library called NEAT-Python.

NEAT-Python is a Python library that implements the NEAT algorithm which helps create optimal neural networks for tasks. This library has a very easy setup process, where you only must put a character class which acts as the agent for everyone in a population and evolves them using reward functions to be able to improve over time. This lets the AI learn the game over time. This library also provides numerous functions that help monitor the process of training the AI. NEAT-Python also allows the exporting of the trained neural network so that the AI developed can then be used for other games.

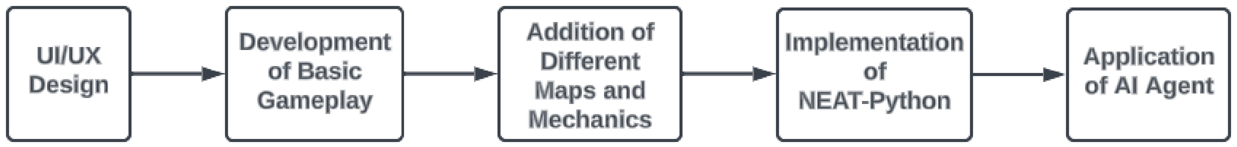

UI/UX Design
Development of Basic Gameplay
Addition of Different Maps and Mechanics
Implementation of NEAT-Python
Application of AI Agent

Figure 2. Methodological Framework

Our application will have three main features, the training of the neural network model using NEAT-Python, the main gameplay of Flappy Bird, and the application of the trained neural network model for testing the developed games. Six maps with different mechanics will be developed to train the model in different scenarios, showing the trained model’s ability to adapt to different gameplay mechanics. To do this, we would have to do it iteratively, as shown in Figure 2, a step-by-step process is planned to achieve this, since each is dependent on the step before it.

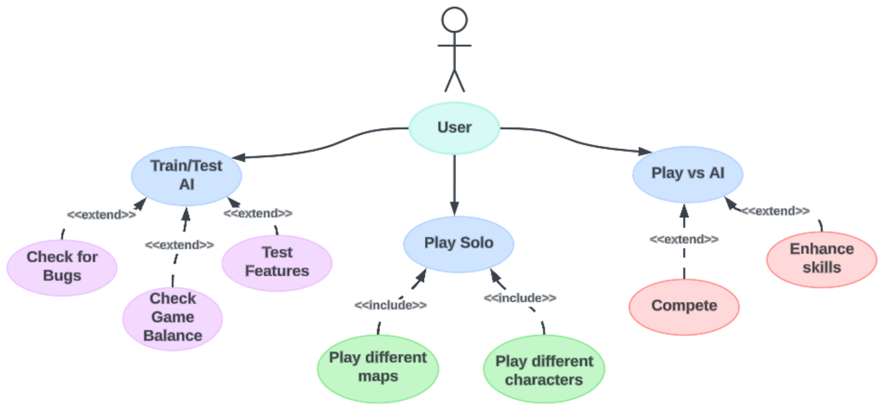


Figure 3. Use Case Diagram

As shown in Figure 3 is the use case diagram of our application. The user will be able to train/test the AI so that they can check for bugs, game balance, and test game features, the user can also play the game for themselves, we have different maps and characters available in the game for them to try out, lastly, they can play against the trained AI so that they can try to compete or enhance their skills.

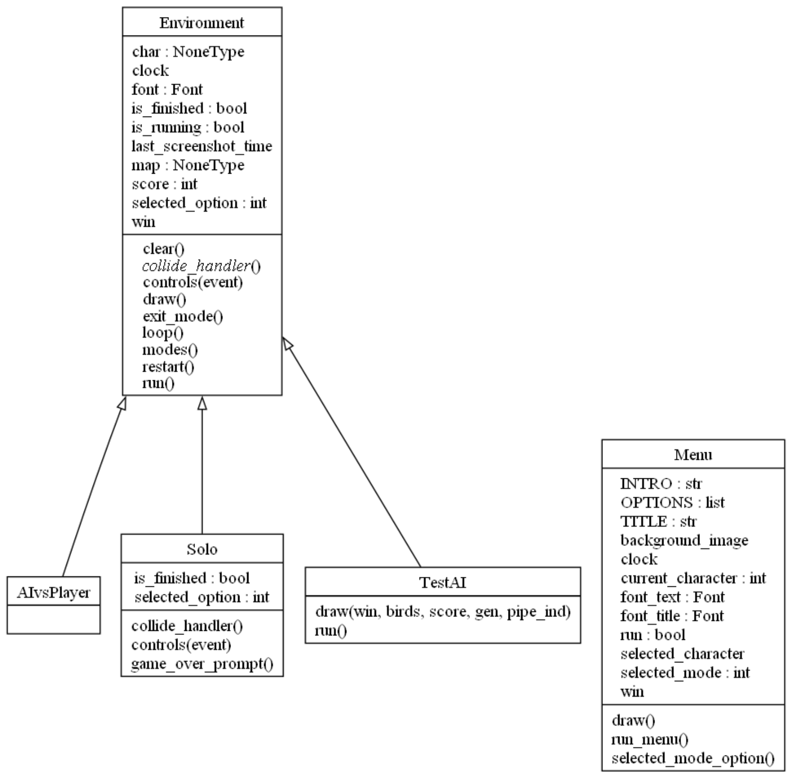


Figure 4. Game Class Diagram

Figure 4 shows the class diagram for the game. Two base classes make up most of the game which are the Environment and Menu class. On startup of the python script, the Menu class runs, it has different attributes that define its state and appearance, but the most important parts of the class are the draw() and run_menu() methods, the draw() method handles the visuals of the menu such as drawing each individual object on the screen and it works hand on hand with the run_menu() method, this method listens for user inputs every clock tick and decides on the next course of action. When the user selects a game mode they want to do, it proceeds to the Environment class, it is the class that handles all the game mechanics and is like the playground for the other objects in the game. Upon selection of a desired game mode, the run() method is ran which tells the game to loop() and handles the different mechanics present in the game. Other methods such as loop(), restart(), draw(), and clear() are all similar for all the game modes and are necessary for the basic functionalities of the game. The Environment class has three subclasses, namely the AIvsPlayer, Solo, and TestAI class, these are the basic game mode provided by the game and are all reliant on the Environment class, these subclasses differ ini how players are able to interact with the game, such as the Solo class that has a custom controls() method which lets the player play the game. The Solo mode and TestAI mode are different in the way they run the game, when the player character interacts with an object is then prompts a game_over_prompt() to allow the player to restart or exit, whereas the TestAI mode does not end when one character collides with an obstacle, it then proceeds throughout multiple generations to train the model. TestAI also draws multiple characters and has a different mechanic of character generation than Solo, since NEAT-Python is the one that handles generation of individual characters, it is then handed to it.

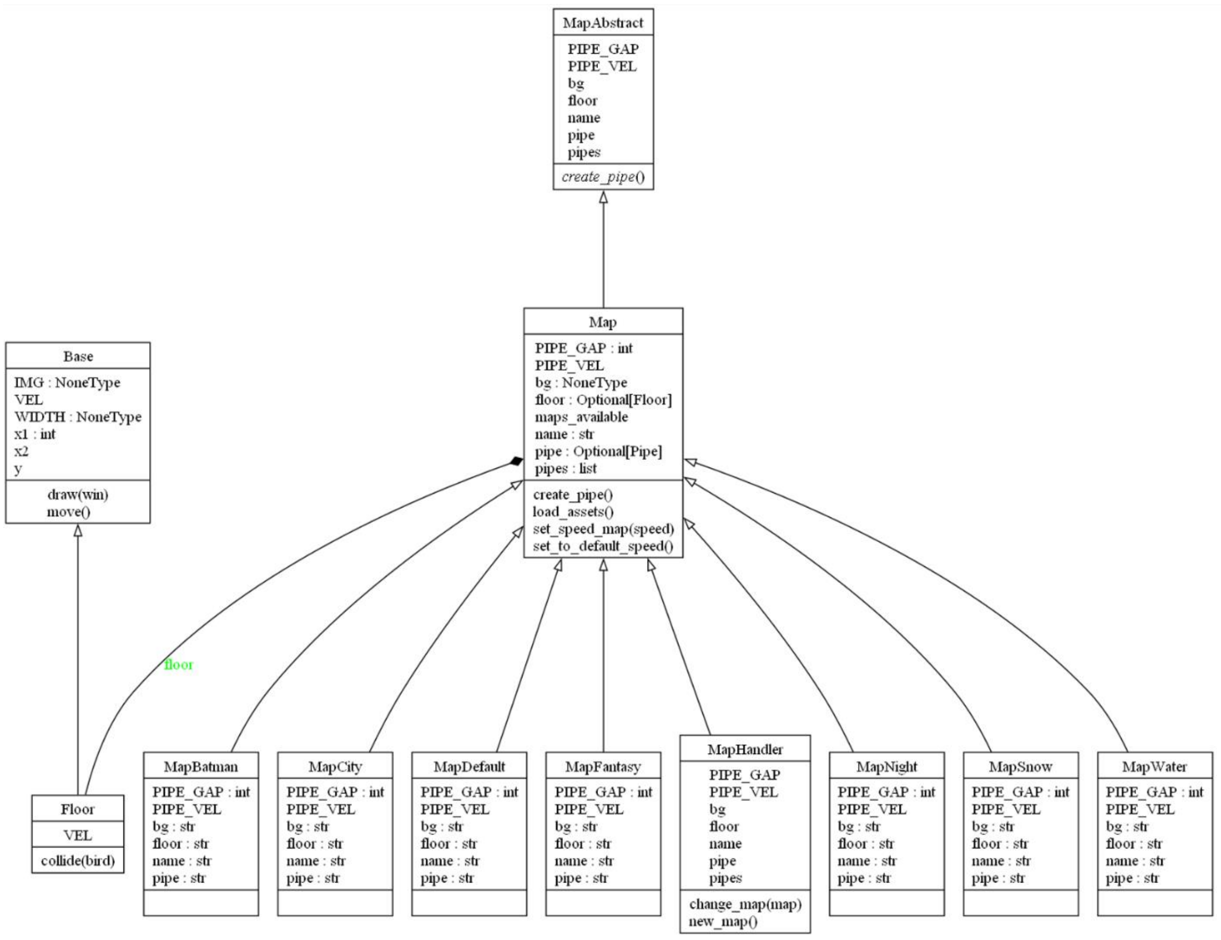


Figure 5. Map Class Diagram

Figure 5 shows the class diagram for the Map object in the game. The map is based upon the MapAbstract class that defines the properties such as pipe gap, pipe velocity, background, floor, name, and pipes that are necessary for a map. The Map class handles creation, loading, drawing, and movement of the pipes and floor of the game. The Map class requires the inclusion of the Floor subclass that is extended from the Base class, this class handles the bottom floor of the game, it handles two tasks, the draw(win) which draws the img on the window with its corresponding width, position, and image, it also handles the movement of the base with the move() method which changes the position based on the x1, and x2 values. The floor subclass interacts with the character using the collide(bird) method, it returns a boolean value whether the bird character overlaps with the floor object.

The Map class has eight subclasses, the MapBatman, MapCity, MapDefault, MapFantasy, MapHandler, MapNight, MapSnow, and MapWater. These different subclasses are all the different levels available in the game, except for the MapHandler which handles the selection of the map for the selected game mode using the change_map(map) and new_map() methods. The map level subclasses are all similar in functionality, but they differ in the attributes such as PIPE_GAP, PIPE_VEL, bg, floor, name, and pipe. These attributes are responsible for the difficulty present in the game and the assets used for the current level.

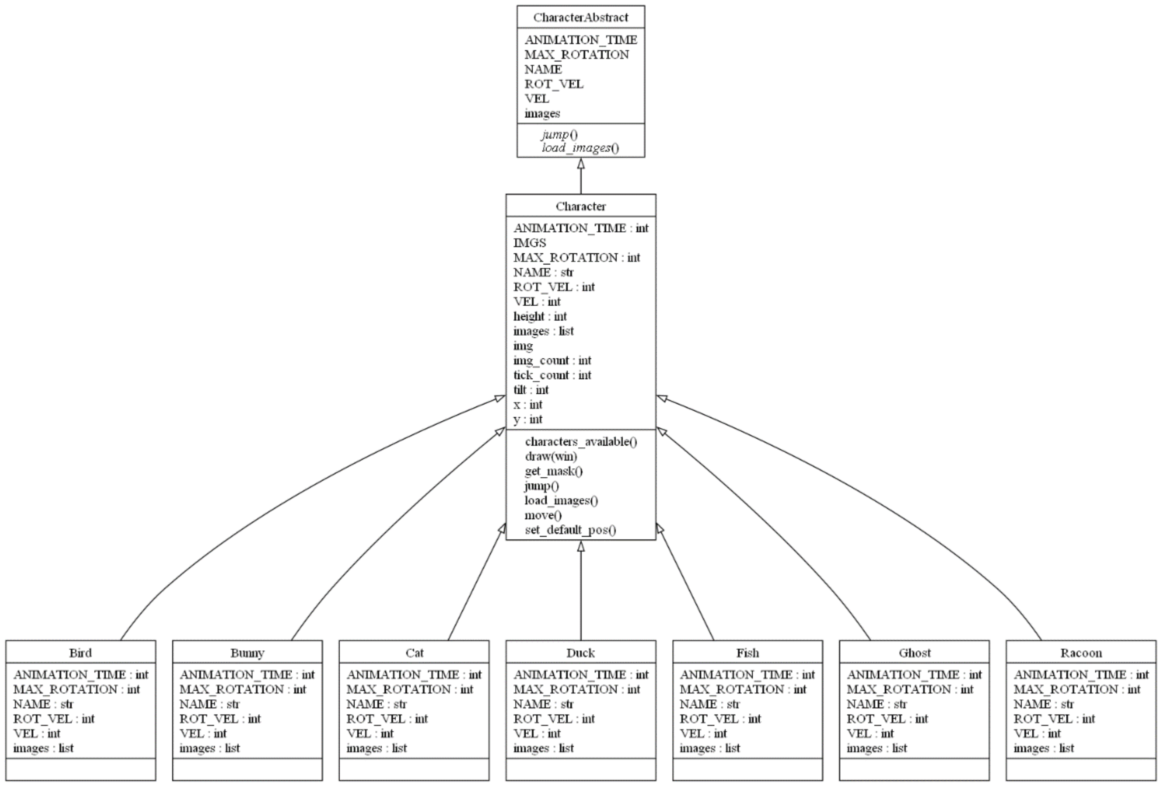


Figure 6. Character Class Diagram

Figure 6 shows the class diagram for the Character object in the game, the character class based upon the CharacterAbstract class that defines the properties that will be used for the character class. The Character class handles creation, loading, drawing, and movement of the character in the game.

The Character class has seven subclasses, the Bird, Bunny, Cat, Duck, Fish, Ghost, and Racoon. These different subclasses are all the different characters available in the game. In terms of functionality, these subclasses are the same, they are animated the same way, and has the functionality of jumping the same way, however where these different characters are the character image that is used for them, as well as their individual speeds, name, and rotation time.

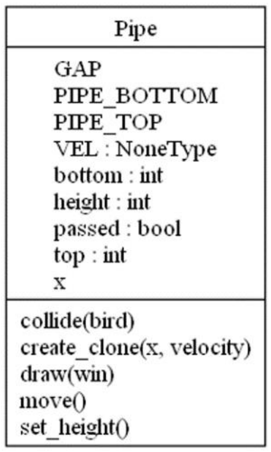


Figure 7. Pipe Class Diagram

Figure 7 shows the class diagram for the Pipe class. The Pipe class is composed of the GAP between each pipe in the map, and the position of the bottom and top parts of the pipe, as well as the velocity of the pipe that is handled by the move() method. When the pipes are generated by the game, the vertical positions of these pipes are controlled by the set_height() method. The environment can check every frame to see if the character has collided with the pipe using the collide(bird) method that returns a boolean if the character overlaps the pipe. The passed attribute can be checked if the character passed the pipe, which then adds a score to the player.

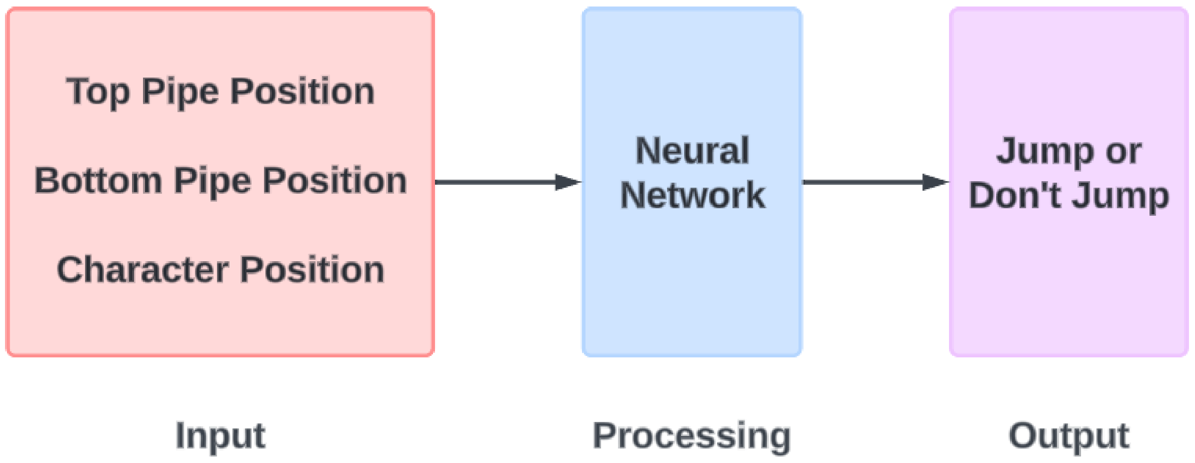


Figure 8. Operational Framework

Figure 8 shows the operational framework for the AI. Each individual AI character has its own neural network that was created and mutated from the NEAT-Python algorithm. Each AI character requires three inputs for it to decide whether to jump or not, it requires the positions of the top pipe, bottom pipe, and the character itself. The output of the neural network is a value between –1 and 1, if the value is higher than 0.5, it will then tell the character to jump, otherwise it will not trigger the jump method.

NeuroEvolution (NE) is the evolution of the weights of a neural network using genetic algorithm, its goal is to improve a neural network through an iterative process of evolution instead of training. NeuroEvolution of Augmenting Topologies (NEAT) is a method of Neuro Evolution that performs on reinforcement tasks, instead of just focusing on finding the optimal weights for a neural network, it will also find the best structure/topology for a certain task (Stanely & Miikkulainen, 2002). NEAT is like a genetic algorithm of different neural networks wherein only the best performing neural networks will pass on to the next generation, and it will mutate and crossover to create better performing neural networks.

NEAT aims to solve three problems in evolving effective artificial neural networks. Firstly, it needs to represent a network's structure and connections (its genome) in a way that allows for meaningful information exchange during evolution. This ensures that offspring inherit and potentially combine beneficial traits from their parent networks. Secondly, NEAT must safeguard valuable structural innovations that arise during evolution. These innovations can lead to entirely new network architectures with improved performance, and NEAT prevents them from being lost through random mutations. Finally, NEAT encourages the development of simpler, more efficient network structures. While complex networks can achieve satisfactory results, they can be computationally expensive. NEAT discourages overly intricate solutions by favoring networks that perform well with fewer connections and neurons.

The first part of NEAT is genetic encoding, allowing the structure of a neural network to complexify, meaning adding nodes and connections. In NEAT, there are two parts, the node genes, and the connect genes. In Figure 9 is an example genome of a population, it contains the list of nodes available in a network and their purpose, a sensor(input), output, or hidden. The connect genes contain all the connections that are in the network for a genome, it contains the source and target node of those connections as well as the weight and whether a connection is active or not.

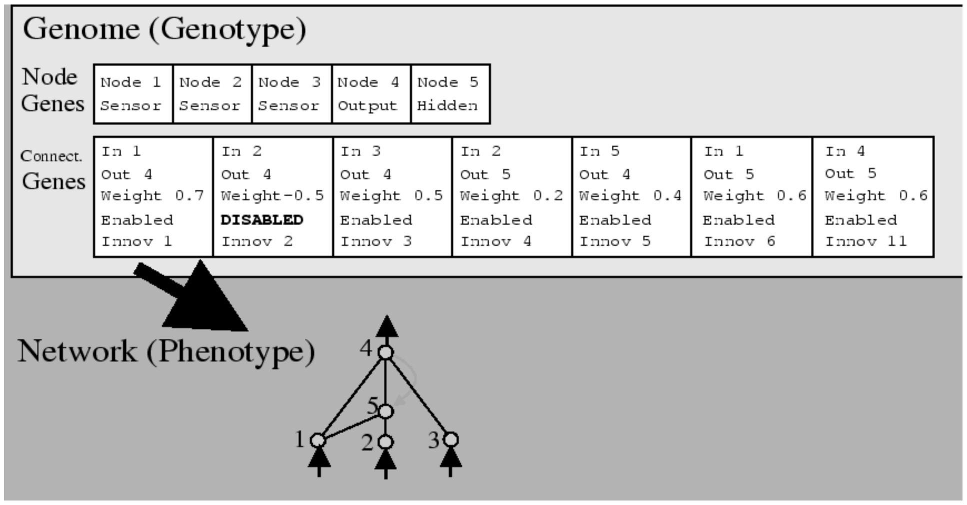

Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Output
Node 5 Hidden
Connect. Genes
In 1 Out 4 Weight 0.7 Enabled Innov 1
In 2 Out 4 Weight-0.5 DISABLED Innov 2
In 3 Out 4 Weight 0.5 Enabled Innov 3
In 2 Out 5 Weight 0.2 Enabled Innov 4
In 5 Out 4 Weight 0.4 Enabled Innov 5
In 1 Out 5 Weight 0.6 Enabled Innov 6
In 4 Out 5 Weight 0.6 Enabled Innov 11
Network (Phenotype)
4
5
1
2
3

Figure 9. Genetic Encoding

Source: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/stanley04a-html/node3.html

There are two ways in which a genome can mutate, either by adding a node or a connection. In Figure 3 it is shown that when you add a connection, you increase the size of the genome by adding a gene, in this case either a connection or a node. When you add a connection, a gene is added at the end of the genome, specifying the source and target of a connection, when you add a node, it disables the connection of the previously connected nodes, and instead connects it as an intermediary between those nodes.

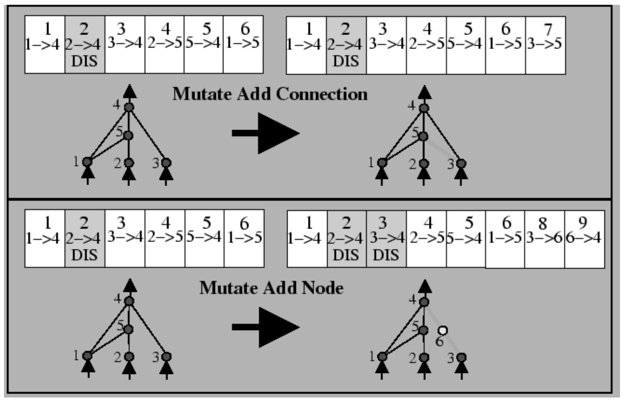


Figure 10. Mutations in NEAT

Source: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/stanley04a-html/node3.html

When you keep mutating this way, the genome will increase so much that connections in the same positions will exist, difficulty then arises on how to crossover genes to combine networks with different topologies. To solve this, you need to track the ancestry of the genes through historical markings. The way the genomes are built in this algorithm is that for every gene added in a genome, there is a global index/ global innovation number that is assigned for that specific gene, so all genes are unique and can represent chronology of every gene in the system. An example is shown in Figure 4, where if you look at the genes the two genomes has, they both have similar genes, without the use of the innovation numbers, combining them both would lead to duplication of genes, but with the ability to track genes with the same innovation numbers, you don't run into the problem of duplication. When doing a crossover of two genomes, their similar genes will be copied, and the connection and weights of those genes will be inherited from the more fit parent, if they are equal, they are selected randomly. Disabled genes have a 25% chance to be re-enabled on crossover. Different genes will be inherited to the offspring.

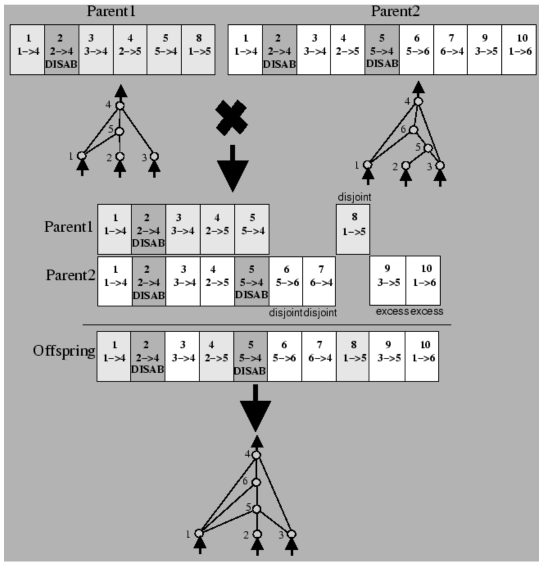


Figure 11. Matching Genomes for Different Networks

Source: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/stanley04a-html/node3.html

When you overtake the genes of newly mutated topologies, you run the risk of stopping long term innovations from happening, meaning you stop potentially performant networks from existing. This is solved through speciation, speciation is when you group similar genes with one another, instead of including it with the entirety of the population. So, newly evolved genes can develop on their own instead of being overtaken by better performing genes before they can develop, which lets innovation develop within different species.

The historical markings are useful in this front because it helps in segmenting similar genes with each other. The similarity between each genome is measured with this formula, wherein the delta (δ) or distance between to network encodings is the linear combination of the number of excess (E) and disjoint (D) genes, as well as the average difference in weight of the matching genes.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}$$

C_1, C_2, and C_3, are multipliers for the importance of the three factors of excess, disjoint, and weight. N measures the number of genes in the larger genome, and is normalized for genome size. Each genome is tested one at time to properly speciate all genomes, other properties exists to help make this process more optimal.

The purpose of this process is to help create a fitness score that is more relative so that structural innovation is protected, and genomes are given time to grow to discover better solutions. f_i^j is the adjusted fitness of an organism i that is calculated based on the distance from every organism j in the population.

$$f_i^j = \frac{f_i}{\sum_{j=1}^{n} \square = sh\big(\delta(i,j)\big)}$$

Now that structural innovation is protected, the final goal is to make the whole process efficient. It is made efficient by only letting new structure to be created incrementally, and only structures that are found to be useful will be able to survive, this way NEAT does not search through every weight dimension but instead adds complexity incrementally, this reduces the number of generations needed to find an optimal solution. This puts a soft limit on the scale of complexity the search looks for and will accept structures that are the right complexity for the given task, so no unnecessary search will be done for overly complex structures.

Application Design and Features

In terms of design and features, the project aims to emulate the mechanics of the popular game Flappy Bird while introducing unique elements to enhance gameplay diversity. Players will encounter a familiar gameplay experience reminiscent of Flappy Bird, where they navigate a character through obstacles with simple tap controls. However, the project distinguishes itself by offering a variety of maps that are randomized for each game session, ensuring that players face new designs every time they play to heighten their engagement. Additionally, players can choose from a selection of characters, adding a layer of personalization to the gaming experience. This approach not only pays homage to the simplicity and addictiveness of Flappy Bird but also expands upon its core mechanics to offer a fresh and engaging gameplay experience for players.

Moreover, incorporating player options, the main menu presents three distinct gameplay modes to cater to different preferences. Firstly, players can opt to "Test the AI," providing an opportunity to observe the AI in action and evaluate its performance. This mode serves as a platform for developers and enthusiasts alike to scrutinize the AI's capabilities and fine-tune its behavior if necessary. Secondly, players may choose to "Play Against the AI," engaging in competitive gameplay where they challenge the AI to see who can navigate the obstacles more effectively. This mode offers a dynamic experience, testing players' skills against an intelligent opponent. Lastly, for those seeking a solitary gaming experience, the option "Solo" is available. In this mode, players can immerse themselves in the game without external influences, focusing solely on their performance and progress. These diverse gameplay modes ensure that players can enjoy the game in a manner that best suits their preferences, whether it be for testing, competition, or personal enjoyment.

Conclusion

In conclusion, the "Flappy AI" project illustrates the substantial role that artificial intelligence can play in refining the development and testing processes within the gaming industry. By leveraging NEAT-Python, the project not only enables an AI to effectively navigate and test side-scroller runner games like Flappy Bird but also offers a scalable solution for enhancing game quality. This is particularly beneficial for smaller game studios, which often struggle with resource limitations. The AI's capability to autonomously detect and resolve gameplay issues ensures that developers can uphold high standards of quality effortlessly, an essential factor for achieving success and ensuring player satisfaction in the competitive gaming market.

Moreover, the integration of AI into game testing extends beyond improving efficiency and cost-effectiveness. The "Flappy AI" project facilitates a more creative and experimental approach to game design, allowing developers to explore varied gameplay elements without the financial burden associated with extensive manual testing. This accelerates the development cycle and aids in refining game mechanics to boost player engagement. Additionally, the project serves as a valuable educational and research tool, offering insights into the practical applications of AI in real-world settings, thereby enriching both the academic and practical landscapes of game development and artificial intelligence.

Therefore, the "Flappy AI" project not only underscores the immediate advantages of using AI for game testing but also highlights the broader implications for future technological advancements in gaming. As AI continues to evolve, its integration into game development and testing is expected to deepen, leading to more innovative, engaging, and high-quality gaming experiences. This development heralds a promising future for the gaming industry, where AI enhances both the creative and operational aspects of game production.

Recommendations

To further enhance the use of artificial intelligence in game development and testing, it is recommended that the application of AI testing methodologies be expanded across various game genres. Currently focused on side-scroller runner games, this AI framework holds potential for adaptation to other genres such as platformers, puzzles, and adventure games. Such expansion would enable a broader range of developers to benefit from AI-driven testing, improving game quality and player experience across diverse gaming formats.

Additionally, integrating this AI testing technology into popular game development platforms like Unity and Unreal Engine would make advanced testing methodologies more accessible, particularly to small and mid-sized studios. By developing plugins or modules that can be seamlessly incorporated into existing workflows, developers can more easily adopt AI testing tools, enhancing efficiency and effectiveness in game development processes. This integration would democratize access to cutting-edge technology, fostering innovation and quality improvement in game design and testing.

References

Saha, S. (2023) Video game market, Future Market Insights. Available at: https://www.futuremarketinsights.com/reports/video-game-market (Accessed: 19 March 2024).

Syed, M. (2019). Quality control in games development.

Santos, R. E., Magalhães, C. V., Capretz, L. F., Correia-Neto, J. S., da Silva, F. Q., & Saher, A. (2018, October). Computer games are serious business and so is their quality: particularities of software testing in game development from the perspective of practitioners. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 1-10).

Politowski, C., Guéhéneuc, Y. G., & Petrillo, F. (2022, May). Towards automated video game testing: Still a long way to go. In Proceedings of the 6th international ICSE workshop on games and software engineering: engineering fun, inspiration, and motivation (pp. 37-43).

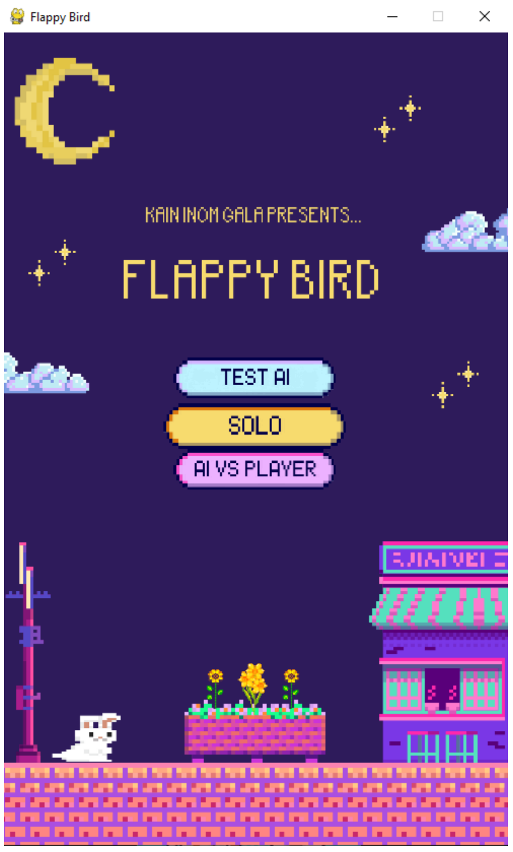
Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2), 99-127.

APPENDIX

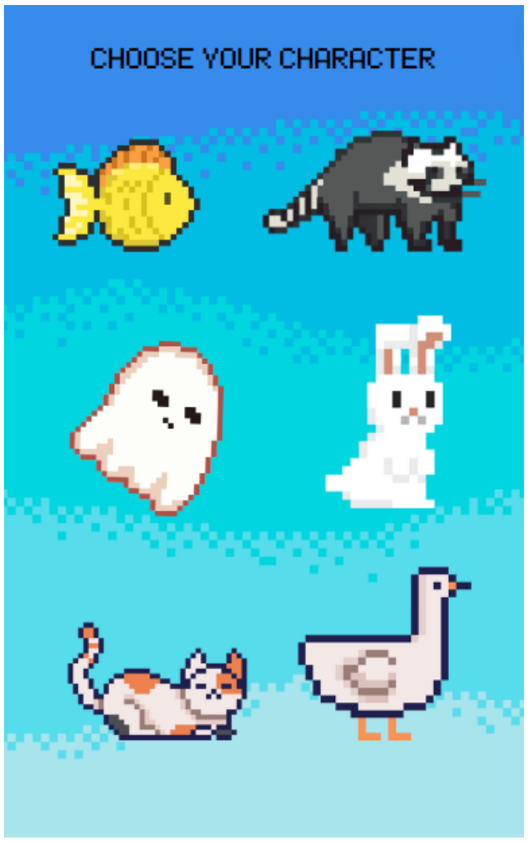
APPENDIX A

DESIGN GALLERY

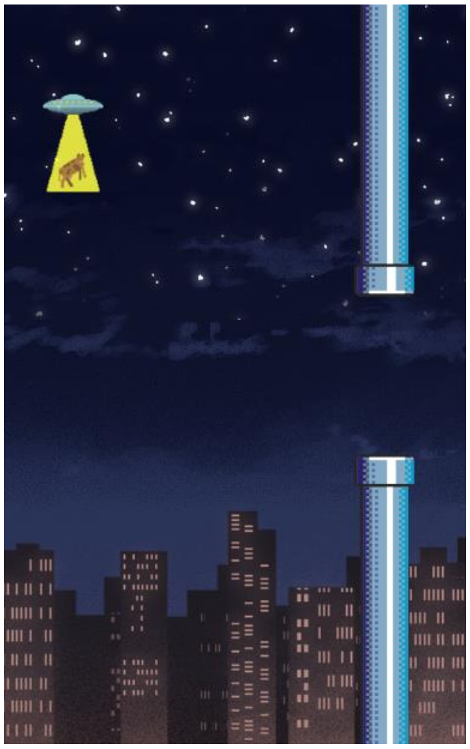
Main Menu

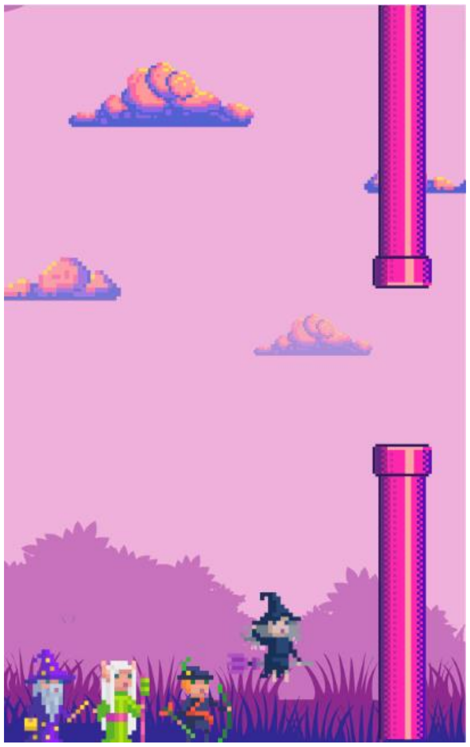
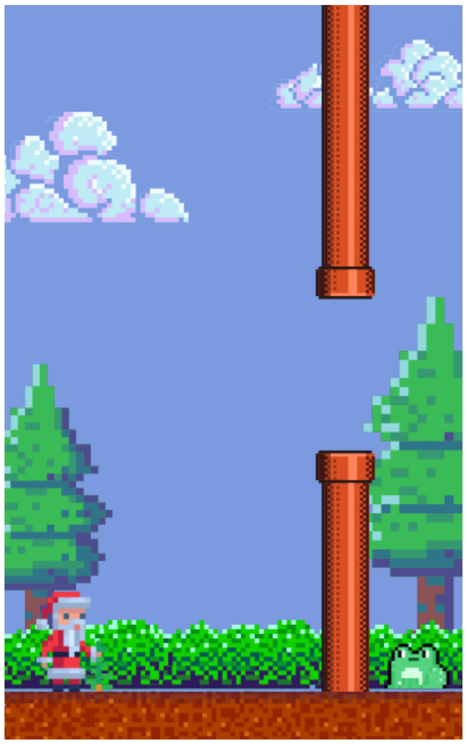


Characters



Sample Maps



APPENDIX B

Manual

INSTALLATION

Requirements:

- Python 3.6 or higher
- Pygame 2.0.1 or higher
- NEAT-Python 0.92 or higher
- Tensorflow 2.16

Execute the following command to install the required packages:

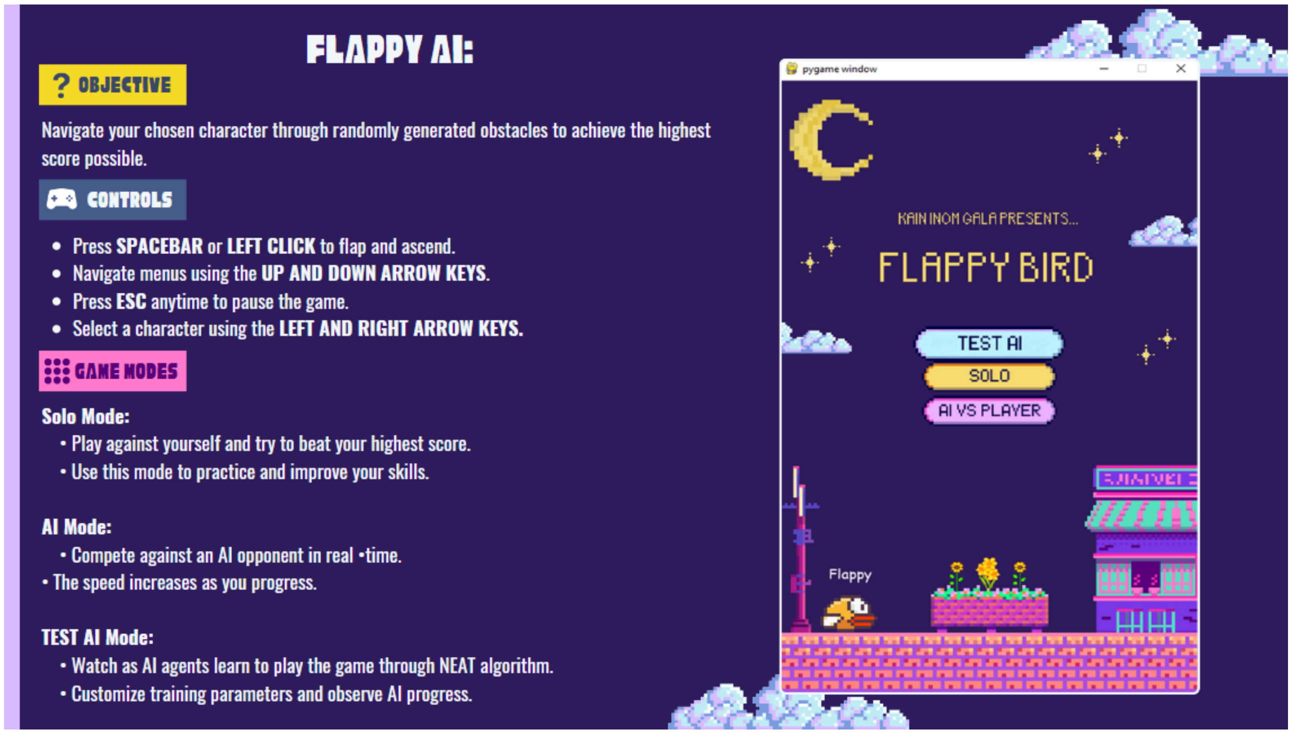
```
pip install -r requirements.txt
```

Usage:

To run the game, execute the following command:

```
python main.py
```

PLAYING THE GAME



FLAPPY AI:

? OBJECTIVE

Navigate your chosen character through randomly generated obstacles to achieve the highest score possible.

CONTROLS

- Press **SPACEBAR** or **LEFT CLICK** to flap and ascend.
- Navigate menus using the **UP AND DOWN ARROW KEYS**.
- Press **ESC** anytime to pause the game.
- Select a character using the **LEFT AND RIGHT ARROW KEYS.**

GAME MODES

Solo Mode:

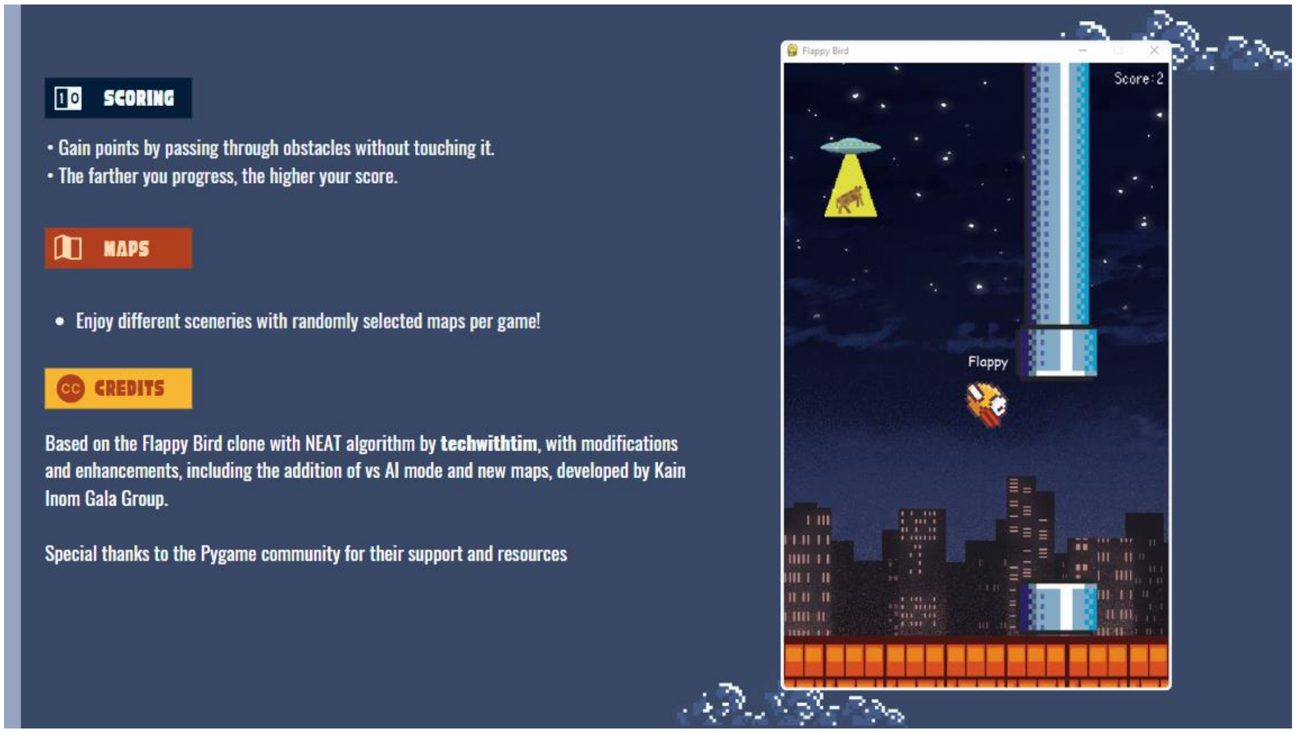
• Play against yourself and try to beat your highest score.
• Use this mode to practice and improve your skills.

AI Mode:

• Compete against an AI opponent in real •time.
• The speed increases as you progress.

TEST AI Mode:

• Watch as AI agents learn to play the game through NEAT algorithm.
• Customize training parameters and observe AI progress.



SCORING

• Gain points by passing through obstacles without touching it.
• The farther you progress, the higher your score.

MAPS

- Enjoy different sceneries with randomly selected maps per game!

CREDITS

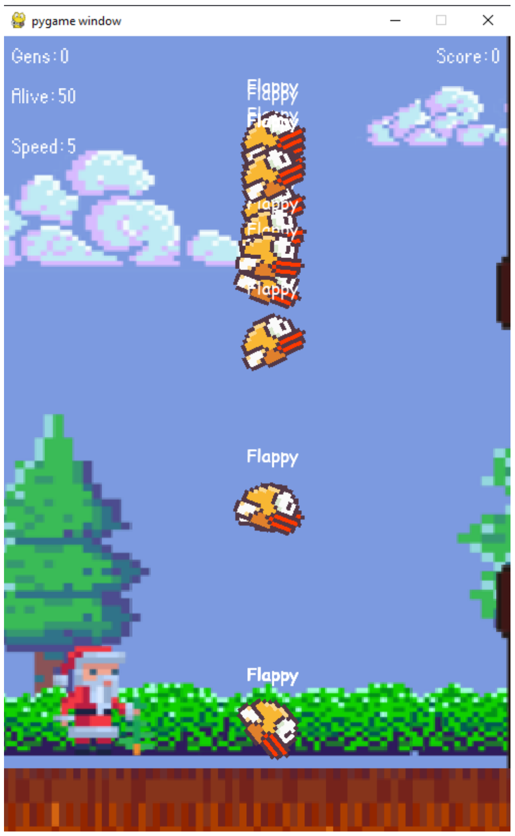
Based on the Flappy Bird clone with NEAT algorithm by **techwithtim**, with modifications and enhancements, including the addition of vs AI mode and new maps, developed by Kain Inom Gala Group.

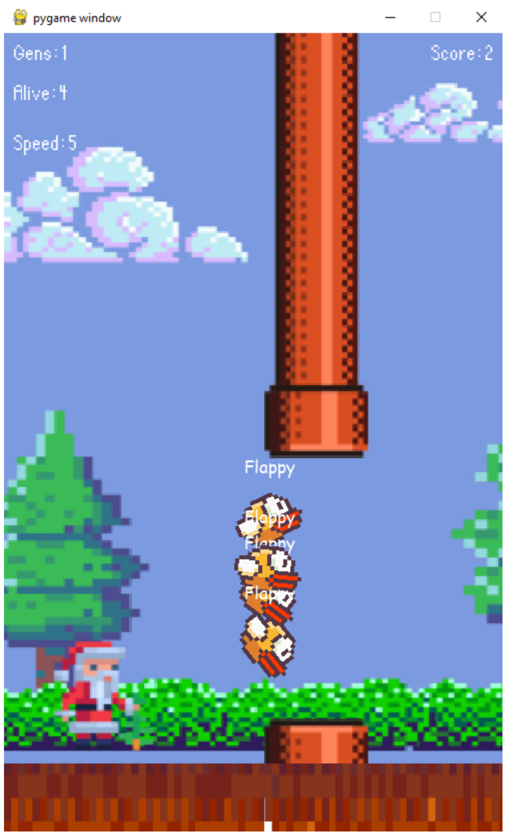
Special thanks to the Pygame community for their support and resources

APPENDIX C

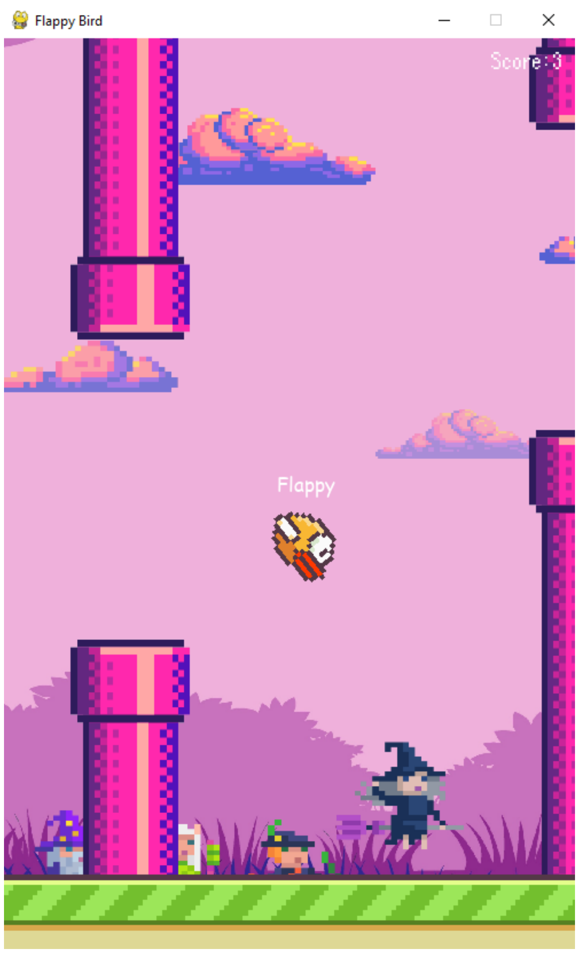
Sample Outputs

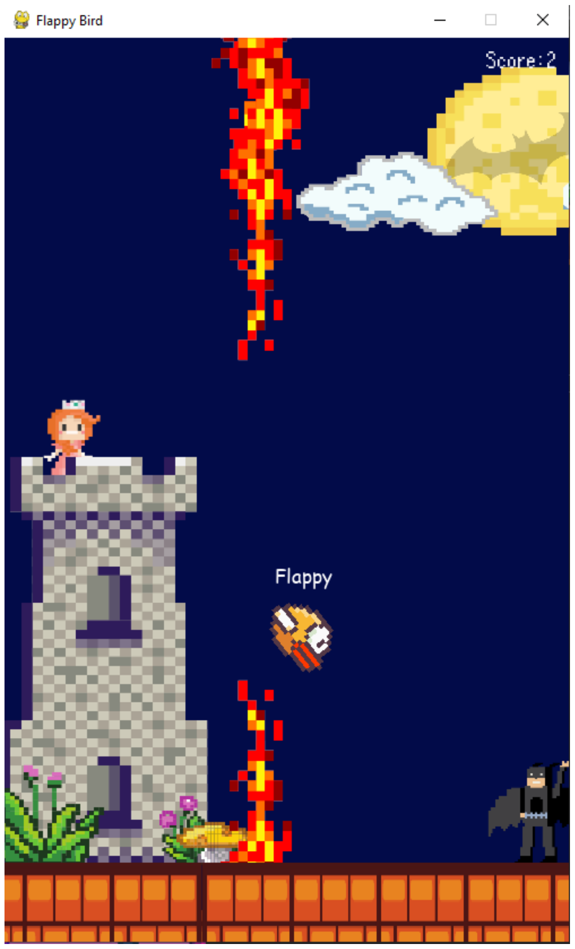
NEAT Training (Test AI Mode)



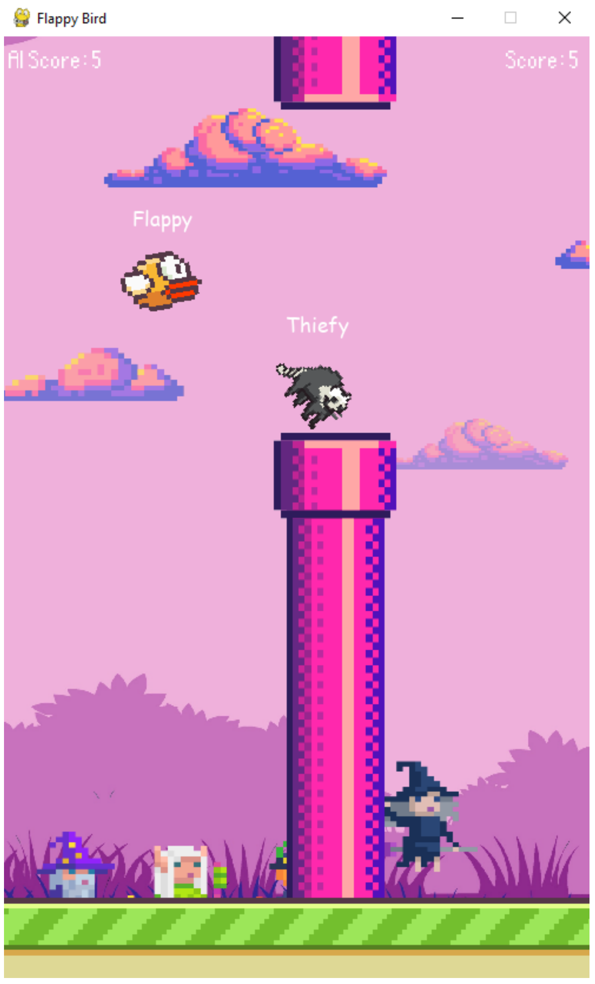


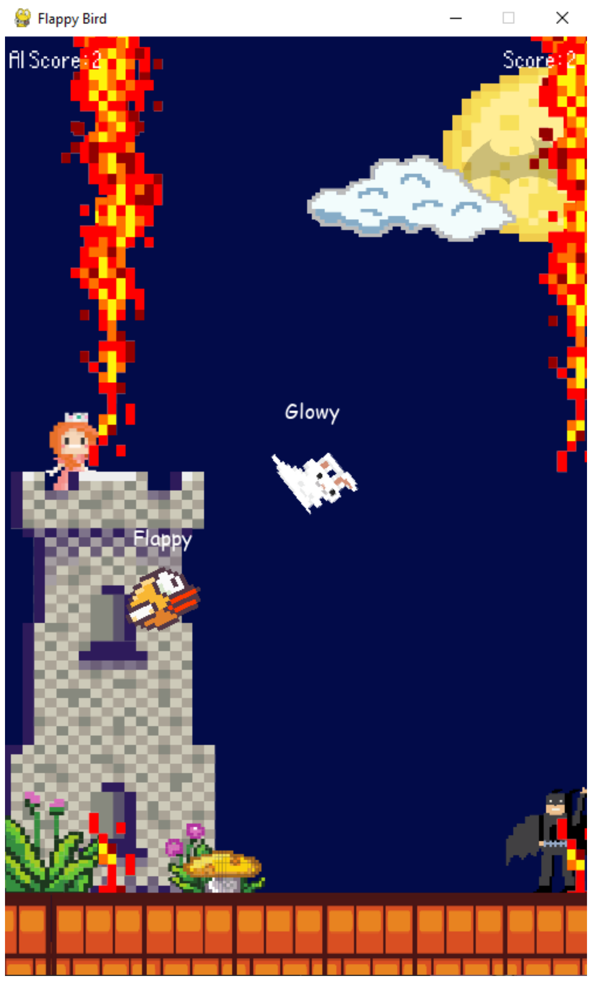
Solo Mode



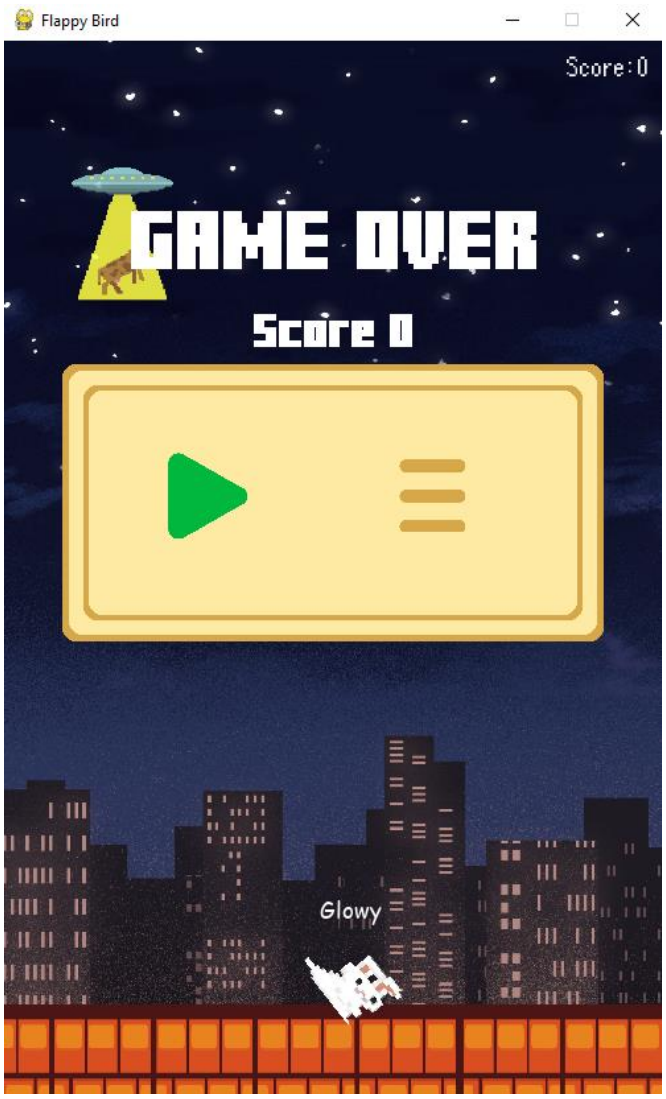


AI vs PLAYER Mode





Game Over Screen


Flappy Bird
Score : 0
GAME OVER
Score 0
Glowy