Computer Science NEA

By Darren Bouleke

Analysis 4
The Problem 4
The solution 4
Examples of similar Ideas 4
A.I. Research 7
Computational Approaches 9
Stakeholders 11
Limitations 18
Success Criteria 19
Design 21
Class Diagrams 21
Algorithms 25
Functional Decomposition of the program 30
Mock up GUIs 32
Key Variables 35
Usability features 36
Testing 38
Data post development 38
Iterative Development 39
Creating NeuralNetwork.js 39
Creating Utils.js 53
Creating Main Menu.html 60
Creating Flappy Bird AI.html 69
Updating the main menu #1 93
Creating Flappy Bird Game.html 101
Creating Cube Runner AI.html 106
Updating the main menu #2 127
Creating Background.js 129
Updating the main menu #3 133
Creating Cube Runner Game.html 144
Final Changes 154
Testing 160
Evaluation 162

Point of Evaluation 162
Initial Objectives 162
Limitations 163
Possible Improvements 163
Final Code 166
NeuralNetwork.js 166
Utils.js 171
Main Menu.html 192
Background.js 218
ColorScheme.json 222
Flappy Bird AI.html 224
Flappy Bird Game.html 233
Flappy Bird BG.png 239
Flappy Bird Icon.png 239
Flappy Bird Pipe.png 240
LogoFlappy.png 240
Cube Runner AI.html 241
Cube Runner Game.html 250
LogoRunner.png 255

Analysis

The Problem

Recently there has been a growing interest in artificial intelligence and its applications in the gaming industry according to Lousia Keight who thinks "Video games could be the answer" in how we can use artificial intelligence for good. A.I. in games has usually been always used to "determine behaviour for computer-controlled entities within a game" examples of this being the ghosts in Pac-man, the bots in chess and the hostile mobs in Minecraft. Most of these A.I.s are usually a set of algorithms which tell the entity what to do and when to do it. However, the issue with this approach is that it may tend to get too easy as the players will learn and discover ways to exploit these algorithms. This could be solved by just changing the algorithm to make the game harder for players but then that will have the risk of making the game too difficult and frustrating for people to play. There needs to be a way in which players themselves can figure out the difficulty of the games they play, which will allow them to have more freedom in choosing how difficult a game should be to play.

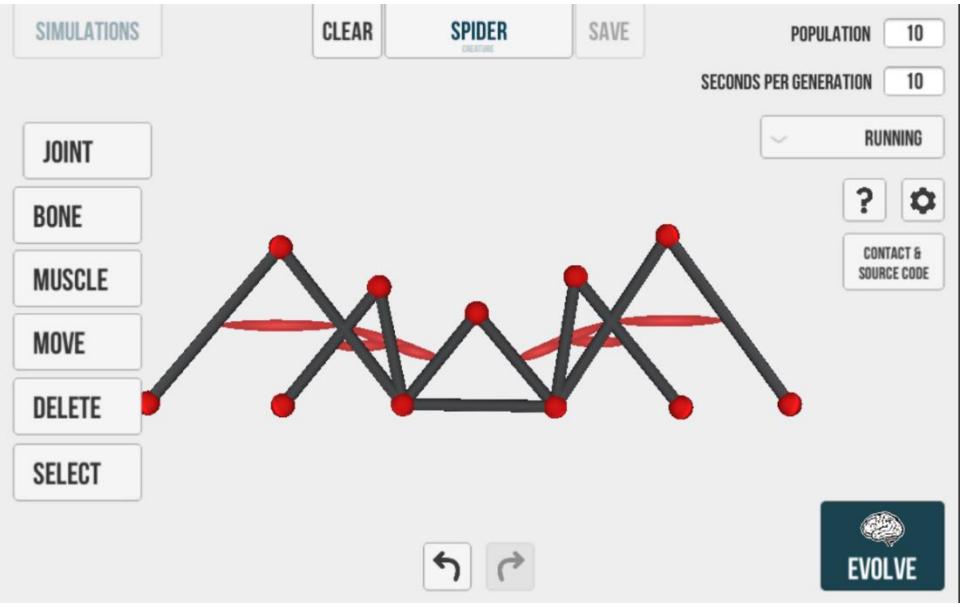
The solution

For my computer science NEA project, I plan to create a web application in which the user can train A.I. to play a variety of games. Also, it aims to provide users with a unique and engaging experience by letting them actively take part in the development and training of A.I. models. This project is valuable and important to stakeholders, including developers and potential users, for several reasons. Firstly, it offers an educational aspect, allowing users to learn about machine learning through enjoyable games. Secondly, it provides a platform for users to personalize their A.I. training experience, promoting ownership and creativity. Additionally, the project can cater to gamers seeking challenging opponents by training A.I. models to excel in various games, enhancing the gaming experience, even when playing alone. Further, the project has significance in computer science as a practical demonstration of A.I. training techniques and real-world applications. Some of these examples I have researched and presented below. In summary, this project aims to create a web application that entertains and educates users about A.I., offering a unique gaming experience, fostering creativity and personalization, and contributing to a broader understanding and appreciation of artificial intelligence in gaming.

Examples of similar Ideas

I decided to look at other examples of what I plan to do so that I can have a better idea of how I should go about creating this project. Below is a list of examples of similar ideas with information about them along with their pros and cons.

Evolution by Keiwan



Source: keiwando.com

About

"Evolution" is a game created by Keiwan that offers users the opportunity to design their own creatures using joints, bones, and muscles. These creatures are then cloned and placed in a unique environment where they must accomplish tasks such as running, jumping, or climbing within a specific time limit. After each round, a fitness function evaluates and determines the best-performing creature for the given role. This process repeats, with slight mutations introduced during cloning, creating variety. The game features a user-friendly interface and extensive customization options, allowing players to personalize their gaming experience by adjusting parameters like task objectives, generation duration, and population size.

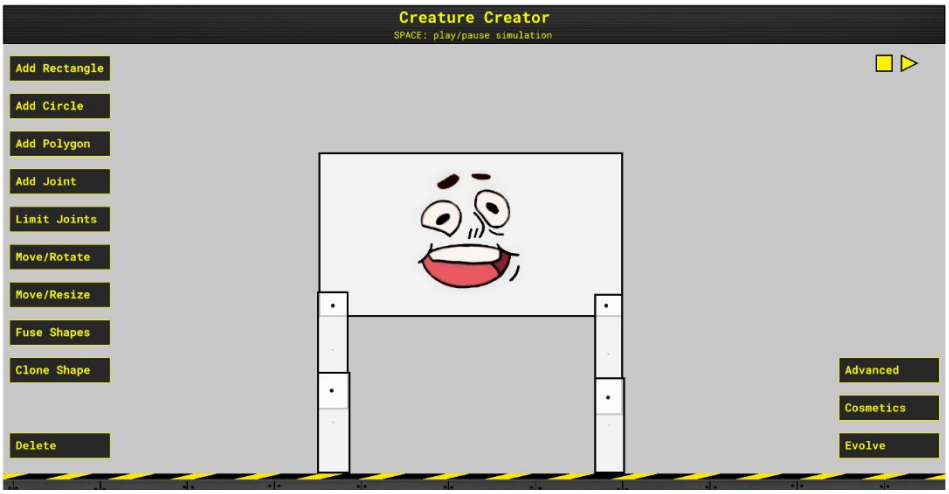
Pros:

- **Simplified Learning:** "Evolution" provides an accessible and engaging introduction to concepts like evolution and machine learning, making these complex ideas more approachable for players.
- **Customization:** The game's wide array of customizable features empowers users to tailor their experience, fostering creativity and personalization.
- **Creative Freedom:** Players can exercise creative freedom in shaping their creatures and experimenting with different evolutionary paths, adding depth and uniqueness to gameplay.

Cons:

- **Repetitive Gameplay:** Some players may find the game repetitive, as much of the gameplay involves observing creatures evolve. This lack of variety can lead to tedium.
- **Time Consumption:** The evolutionary process can be time-consuming and potentially boring for players - *In my project, I will create a speed option to accelerate evolution and adding more interactive elements might alleviate this issu*e.
- **Limited Interactivity:** While the game focuses on evolution, it may lack sufficient interactivity for some players. *I will find more ways for players to engage with the evolutionary process, such as modifying the environment or A.I. properties, which could enhance player involvement and enjoyment*.

Creature Creator by Code Bullet



Source: thebigcb.com

About:

Creature Creator is like Evolution in which the user creates a creature, and it is placed in an environment where it must learn to move and evolve before it is killed by a laser which moves from the left to the right. After all the creatures have died, the creature who has gotten the furthest is then selected and cloned with slight mutations and then the simulation starts again. There are many similarities between this and Evolution by Keiwan such as the ability to create creatures and the process of training them via a neuro-evolution approach, however, they do differ quite a bit in certain areas for example, in creature creator, the end of a generation is defined by when all of the creatures remaining have been killed whereas in evolution, the end of a generation is defined by when a time limit has been reached. Another key difference is that while evolution use muscles, joints, and bones to create creatures, Creature Creator utilizes shapes such as rectangles, circles, and a line tool to create unique shapes which then can all be connected via joints which have motors that rotate the shapes around to simulate movement.

Pros:

- **Customization**- Like Evolution, Creature Creator has a wide variety of options available such as population size, number of batches and speed of the laser. It goes a step further by adding the ability to add cosmetics to the creatures and change the colours of certain shapes which may

not affect the actual gameplay, but it helps give more creative freedom to the user to personalize their own experience.

- **Help and Tutorials**- Throughout Creature Creator there are many useful notes scattered throughout the game which give the player information about what options do what. There is also a video which can further aid the player in using the application. https://www.youtube.com/watch?v=wpHVlN1BVxM
- **Humorous style-** The Creature Creator program incorporates humour into its style, evident in features like the "how the f@#k do I use this?" button, which directs users to a tutorial video. Additionally, the program includes humorous tooltips and allows players to add custom faces to their creatures. This humorous approach can enhance the overall experience for users of the program, making it more engaging and enjoyable.

Cons:

- **Lack of Useability –** Creature Creator has a choice to scale the size of some of the polygons which can be used to make them larger or smaller. The problem is that it requires a mouse, which makes this feature obsolete for mobile users and users who do not own a mouse (users that use a laptop). This automatically reduces the number of users who have access to this program as they may not have the required hardware to use it. *In my project, I will make sure that my program is accessible to as many users as possible by adding functionality which will not require the use of hardware which only a select part of the player base will have.*
- **May not be appropriate for younger audiences-** There are many instances where Creature Creator has demonstrated that it is not appropriate for younger users. Examples of this are the "How the f@#k do I use this?" button which features profanity which may not be suitable for some people. Another example is the video, which is supposed to aid users in using the program. It contains a lot of profanity and inappropriate sexual references which is also not appropriate. *I will make sure that in my program that there are no inappropriate references and that it appropriate for people of all ages*
- **Bland Visuals -** One noticeable drawback of Creature Creator is its simplistic and bland visual design. The use of basic shapes like rectangles and circles, while functional, may not be visually appealing to some players. The lack of detailed graphics and animations can make the game feel less immersive and engaging. *To address this issue, I will focus on improving the visual aesthetics of my project by incorporating more vibrant and visually pleasing elements, such as more detailed creature designs, background environments, and animation effects. Enhancing the overall visual appeal can make the game more attractive to a wider audience and improve the overall user experience.*

A.I. Research

For this project, I plan to create and use Artificial Intelligence (A.I.). For this reason, I will need to do research into what different types of A.I. I can use. This section will be about the different approaches I can take to create the best suited A.I for my project.

NeuroEvolution:

NeuroEvolution constitutes an artificial intelligence paradigm inspired by evolutionary principles, leveraging genetic algorithms to iteratively refine neural network architectures and weights over successive generations. This approach is particularly advantageous for addressing complex problem domains where conventional methods may encounter challenges, as it facilitates exploration across a broader solution space.

Pros:

- **Adaptability** - NeuroEvolution excels in addressing complex and dynamic problem domains, highlighting a high degree of adaptability.
- **Global Optimization** - Utilizing genetic algorithms allows for exploration across a broader solution space, increasing the likelihood of finding optimal solutions.

Cons:

- **Computational Expense** - The iterative nature of evolution and genetic algorithms can be computationally expensive, potentially limiting its applicability in resource-constrained environments.
- **Lack of Guarantee** - There's no guarantee that NeuroEvolution will converge to an optimal solution, as the exploration process may not cover the entire solution space.

Reinforcement Learning

Reinforcement Learning (RL) represents a distinctive facet of machine learning, wherein an agent refines its decision-making capabilities by interacting with an environment. The agent receives feedback in the form of rewards or penalties based on its actions, enabling it to iteratively adapt and optimize its behaviour over time. RL has proven highly effective in scenarios involving sequential decision-making, such as gaming, robotics, and autonomous systems. However, it necessitates careful parameter tuning and may exhibit sensitivity to environmental dynamics.

Pros:

- **Sequential Decision Making** - RL excels in scenarios involving sequential decision-making, making it well-suited for applications such as gaming, robotics, and autonomous systems.
- **Autonomous Learning** - The ability of agents to learn autonomously through interactions with the environment is a key strength, requiring minimal human intervention.

Cons:

- **Sensitivity to Parameters** - RL algorithms often require meticulous tuning of parameters, and their performance can be sensitive to changes in the environment, leading to potential challenges in real-world applications. Exploration-Exploitation
- **Trade off** - Balancing the exploration of new actions with the exploitation of known strategies is a critical challenge in RL, influencing the efficiency of learning.

Deep Learning

Deep Learning, a subset of machine learning, involves the utilization of neural networks with multiple layers, commonly referred to as deep neural networks. These networks possess the capability to autonomously learn hierarchical representations of data, thereby enabling the capture of intricate patterns and features. Deep Learning has engendered transformative advancements across various domains, including image and speech recognition, natural language processing, and medical diagnostics. Its inherent strength lies in its capacity to automatically discern pertinent features from raw data. Nonetheless, the efficacy of Deep Learning is contingent upon the availability of substantial datasets and computational resources, and its lack of interpretability may pose challenges.

Pros:

- **Automatic Feature Extraction** - Deep neural networks can automatically learn hierarchical representations of data, enabling the extraction of intricate patterns and features without explicit feature engineering.
- **Versatility** - Deep learning has demonstrated remarkable success across diverse domains, including image recognition, natural language processing, and medical diagnostics.

Cons:

- **Data and Resource Intensive** - Deep learning models often require enormous amounts of labelled data for training and significant computational resources, potentially limiting their applicability in resource-constrained settings.
- **Lack of Interpretability** - The complex nature of deep neural networks can result in models that lack interpretability, making it challenging to understand the rationale behind specific predictions, which is crucial in certain applications like healthcare.

After researching and looking at what different approaches I could use, I have decided to take the NeuroEvolution way. I have chosen this approach as I believe it will yield the best results for what I aim to do. The similar examples I researched also use this approach so using it too will make my project like theirs. To implement a NeuroEvolution approach to my project, I will need to learn how to create a neural network. I decided to create one by myself instead of using a library such as TensorFlow.js as I believe creating one will enhance my machine learning knowledge. It will also allow me to make specific changes which I could not do if I were to use a library which can make this neural network more tailored to what I intend to do. After researching about how I could create one, I came across a video by 3blue1brown which explains what a neural network and how I could create one.

Computational Approaches

Algorithmic Thinking:

- Implementation of NeuroEvolution: A systematic algorithm will be devised for the NeuroEvolution process. This algorithm will delineate the various stages, including initialization, selection, crossover, mutation, and evaluation. The objective is to render the training process transparent and accessible for users. Intelligent
- Difficulty Adjustment Algorithm: The development of an algorithm for automatic adjustment of game difficulty based on AI performance. This algorithm will scrutinize user performance and strategically modify game parameters to maintain an optimal balance between challenge and engagement.

Abstraction:

- Refinement of Game Mechanics: The intention is to distil game mechanics into a generic interface, affording users the capability to train AI models across diverse games without being encumbered by specific rules. This initiative aims to establish a versatile and adaptable training platform.

- Streamlined NeuroEvolution Framework: An additional layer of user-friendliness will be incorporated into NeuroEvolution, concealing the intricacies of genetic algorithms and neural networks. This ensures the training process is presented in an easily comprehensible manner while retaining the freedom for customization.

Automation:

- Facilitation of Training Cycles: The creation of an automated system wherein users can initiate AI training cycles effortlessly. This system will allow users to either adhere to default settings or customize parameters, thereby streamlining the training process.
- Automated Data Collection Mechanism: Inclusion of an automated data collection mechanism during the training process. This feature will systematically compile information regarding AI performance and user interactions, thereby enhancing the learning experience and providing valuable insights for future iterations.

Decomposition:

- Modularization of Game Components: A strategy involving the decomposition of game environments into modular components. This modular approach enables users to mix and match different elements, fostering a high degree of customization in crafting challenges for their AI models.
- Structured NeuroEvolution Components: The NeuroEvolution process will be deconstructed into distinct components, such as neural network architecture, mutation strategies, and fitness evaluation functions. This modular approach facilitates user focus on specific facets of the training process for nuanced adjustments.

Logical Reasoning:

- Systematic Representation of Game Logic: A commitment to represent game logic in a structured and systematic manner, ensuring that AI entities can comprehend and adapt to diverse game rules. Logical reasoning will be a pivotal element in the decision-making process of the AI.
- User-Defined Logical Framework: The provision for users to define custom game rules, thereby necessitating the AI to adapt its logical reasoning in accordance with user-specified parameters. This feature introduces complexity, encouraging users to engage in logical thinking while formulating game scenarios.

Problem Solving:

- Integration of Dynamic Problem Scenarios: The introduction of dynamic problem scenarios within games, requiring AI entities to adapt and solve challenges dynamically. This may involve changing environments, unpredictable obstacles, or evolving opponents, thereby fostering robust problem-solving capabilities in the AI.
- User-Crafted Challenges: The empowerment of users to craft custom challenges within games, prompting AI entities to engage in problem-solving endeavours as users experiment with diverse

scenarios. This feature aims to cultivate an environment where users observe how their AI models strategize to overcome obstacles.

Pattern Recognition:

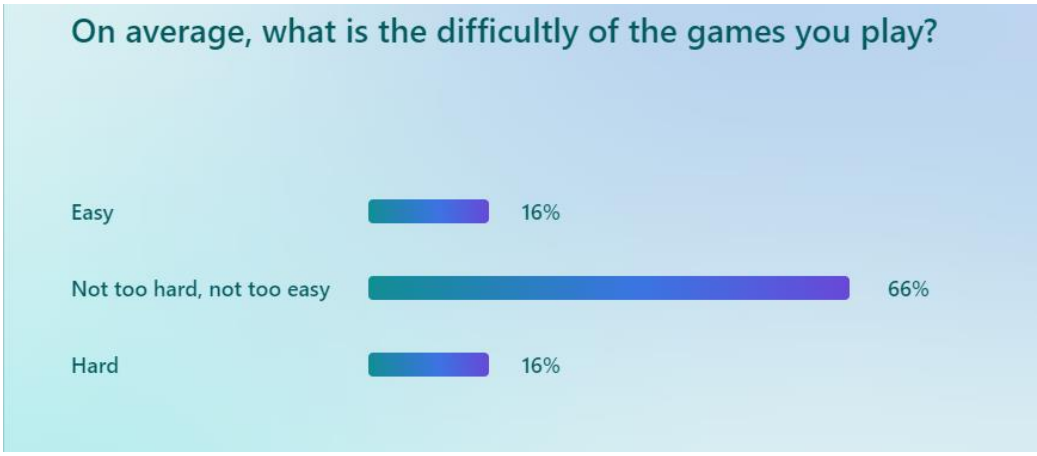
- Incorporation of Pattern Learning Mechanisms: The development of mechanisms within the AI entities to recognize and learn patterns within the game environment. This strategic approach enables AI entities to anticipate events, make informed decisions, and adapt their strategies based on recurring patterns.
- Analytical Pattern Recognition: Implementation of pattern recognition in the analysis of user feedback and AI performance data. This analytical approach aims to identify trends and patterns, informing future updates and customization options based on discerned user preferences.

Stakeholders

The primary audience for my project consists of gamers aged 16 and above but possibly not for the older generation, 16-45. This is due to artificial intelligence (A.I.) which may pose challenges for younger and older individuals to grasp fully. However, I am committed to making my program user-friendly for all age groups. My stakeholders are expected to include gamers with an interest in machine learning and its application in the gaming industry. Competitive gamers can also benefit from my program, as it involves training A.I. to a specific skill level, allowing users to compete against it. Additionally, game developers looking to incorporate A.I. into their games—such as using A.I. as enemies or companions for players—could find value in my project. To gather insights and inform my program's features and design, I conducted a questionnaire. I ensured the questions were easy to read and understand for my stakeholders, even though this limited my ability to delve deeper into A.I.-related inquiries. Below, you will find the results of my questionnaire.

The first question was asking about the devices people used to play games. According to the results, 50% of my census use PC the most when playing games followed by others (33%) which mostly just consisted of consoles such as PlayStation and Xbox. The reason for this could be because PC has a wider range of games to play compared to the other platforms such as mobile or console devices. With the results of this question, making this game accessible for PC users is a necessity. I will also investigate adding controller support to this program as well due to the fact it is the second most popular option. Despite mobile being one of the lowest portions of my census with only 8% choosing the option, it

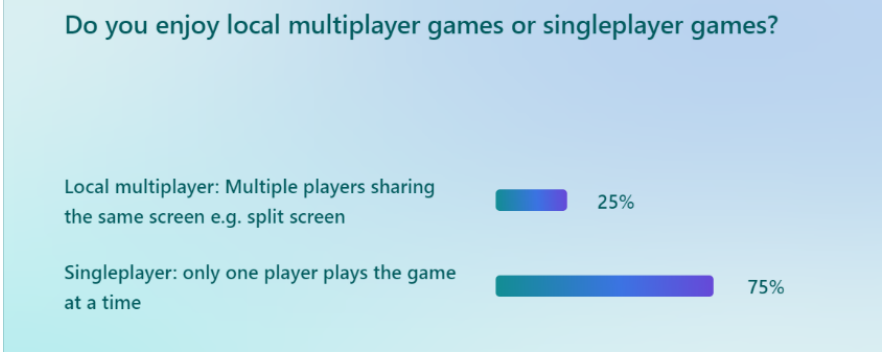
claims in this article that "92% of gamers are exclusively using phones". As a result, I will also make sure that this program is suitable for mobile users.



This question was based around the difficulty of games that people tend to play. Based on the results, most people tend to play games that are neither too hard nor too easy. As shown later, this is most likely because players usually play games for entertainment and if games were too challenging, the player may get too frustrated and give up, however if the game were to be too easy, the player may get bored and do something else. Having it so the games are not too hard or easy will be a challenging task which I will have to overcome.

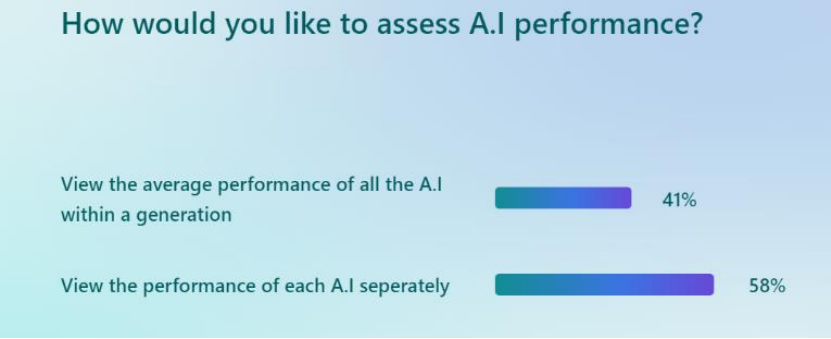
This question was asking about the main motivation for playing games. These results were very varied as the 2 biggest reasons were entertainment with 32% choosing it and challenge with 29% of them choosing it. The results from this question clearly show how making my program entertaining for users should be the priority. I should also consider adding a wide variety of challenges within my games as 29% of my census seem to enjoy it. Completion has around 19% of users choosing it which shows there is a prominent interest in having features which users can complete such as a storyline or achievements which I will consider adding into my program.

Results from this question show that most of my census prefer competitive games rather than cooperative games. This means in my program; I will need to make sure the games I include are competitive. However, this does not mean that I will neglect the cooperative features of my game as 41% of them still voted that they preferred cooperative over competitive. As a result, I will make sure that I include both cooperative and competitive. But I will prioritize completive due to time constraints.

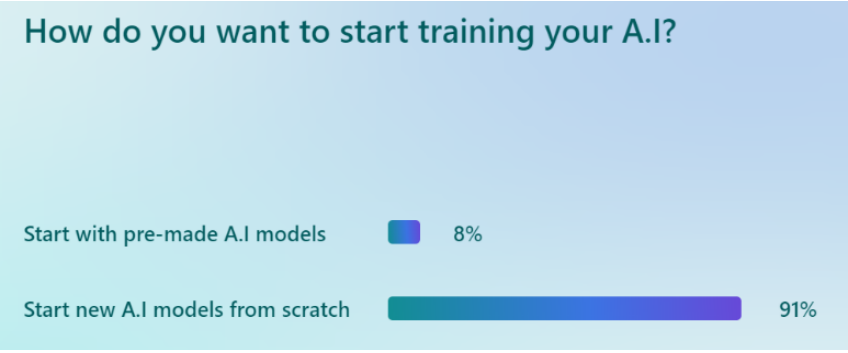


According to the results from this, a significant majority of respondents, accounting for 75% of the total, prefer singleplayer games, where they can enjoy gaming experiences individually. However, an interesting finding is that 25% of respondents expressed an interest in local multiplayer games, which involve multiple players sharing the same screen. These results provide valuable insights for my project. To cater to these preferences effectively, it is crucial to incorporate engaging singleplayer experiences into my project, given the larger portion of my audience leans in that direction. Simultaneously, for the 25% who enjoy local multiplayer, offering multiplayer features or modes could enhance the versatility of my web application. Striking this balance between singleplayer and local multiplayer experiences can help ensure my project appeals to a wide range of gamers and aligns with the user preferences identified in the questionnaire.

Based on the results from this, it is apparent that gamers value various aspects of their gaming experiences. A huge portion of respondents, accounting for 37%, place a high priority on the story element in games. This suggests that incorporating engaging narratives or storylines into my games is crucial to capture the interest of a substantial part of my audience. Additionally, the importance of leaderboards, highlighted by 24% of respondents, indicates a desire for competitive features and opportunities to measure performance against others. Integrating leaderboards can foster engagement and competition within my web application. Graphics and multiplayer options, each at 13%, should not be overlooked, emphasizing the need for visually appealing games and social interaction opportunities. Furthermore, the 10% who mentioned "other" values, mostly focusing on gameplay mechanics, underscore the importance of delivering enjoyable and well-designed gameplay experiences. Balancing these preferences will be key to creating a web application that caters to a diverse gaming audience and aligns with the values identified in the questionnaire.

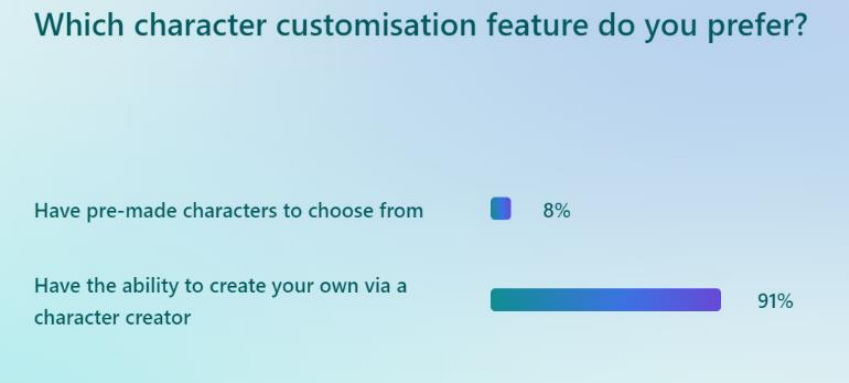


Based on the survey findings presented in this image, users have distinct preferences when it comes to assessing A.I. performance within my project. A significant majority, accounting for 59% of respondents, favours the option to view the performance of each A.I. separately. This preference highlights the importance of providing users with detailed insights into how individual A.I. models are performing, enabling them to make precise adjustments and decisions during the training process. However, it is also noteworthy that 41% of respondents express an interest in viewing the average performance of all A.I.s within a generation. To cater to these varied preferences effectively, my project should offer both options, allowing users to assess A.I. performance on an individual level or gain a broader overview of A.I. performance trends. Striking this balance will ensure that my web application aligns with the diverse assessment needs of users, enhancing their A.I. training experience.

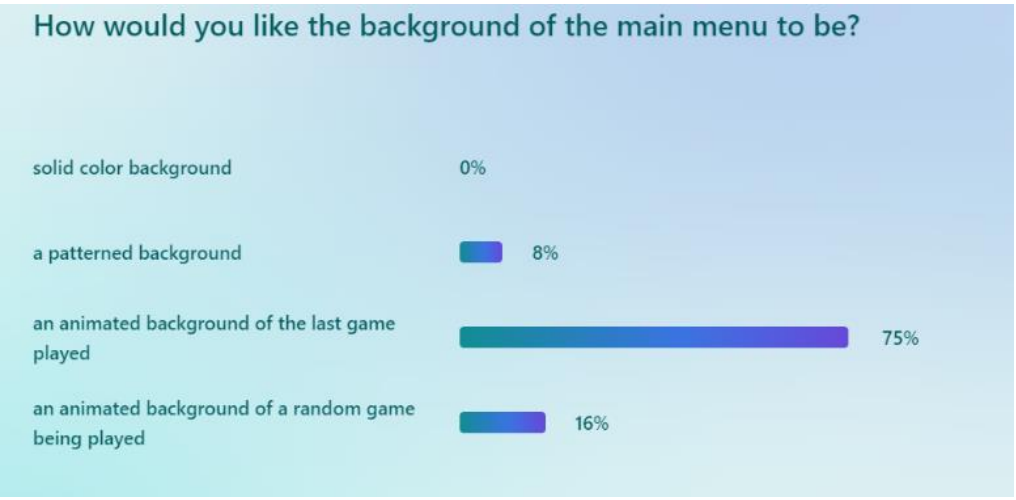


Based on the insightful feedback from the survey, it is evident that most respondents, a substantial 92%, are inclined to begin their A.I. training journey by creating new A.I. models from scratch. This strong preference underscores the importance of offering users the tools and resources needed to craft A.I. models according to their specific needs and objectives. Users clearly value customization and having

full control over the A.I. training process, which should be a central focus of my project's development. However, it is also essential to consider the 8% of respondents who expressed interest in starting with pre-made A.I. models. Providing this option can be valuable, especially for those who may be new to A.I. training or seek to experiment with existing models. Striking a balance between customization and accessibility will ensure that my web application caters to a diverse audience, ranging from A.I. experts to beginners, while delivering a comprehensive and user-friendly A.I. training experience.

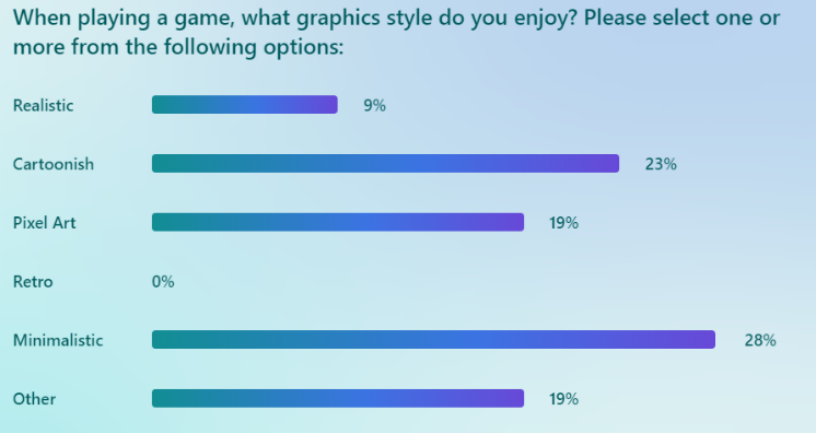


The results from the survey clearly highlight users' strong inclination towards character customization within my project. An overwhelming 92% of respondents express a strong preference for the ability to create their own characters via a character creator feature. This resounding preference underscores the significance of providing users with the means to craft unique and individualized characters, enhancing their overall gaming experience. To cater to this demand, my project should prioritize the development of a robust character creator tool, allowing users to unleash their creativity and tailor their characters to their liking. However, it is also important to consider the 8% who prefer pre-made characters, and offering a limited selection of such characters can provide convenience for users who may want quick customization options or as a starting point for their character designs. Striking a balance between customization and convenience ensures that my web application caters to the diverse needs and preferences of users, leading to a more engaging and user-centric gaming experience.



The feedback from the survey regarding the main menu background preference is quite illuminating. An overwhelming 75% of respondents express preference for an animated background featuring the last game played. This finding underscores the significance of delivering a dynamic and personalized main menu experience, allowing users to relive the essence of their previous gaming sessions. This feature can enhance user engagement and create a memorable and immersive entrance to the gaming world

within my project. Additionally, the 16% of respondents who favour an animated background of a random game being played should not be overlooked, as this option can provide variety and surprises to the main menu presentation. Although the preference for patterned or solid colour backgrounds is minimal, prioritizing the development of an animated background highlighting the last game played is the key to crafting a captivating and user-centric main menu experience that aligns with the preferences identified in the survey.



The survey results concerning preferred graphics styles in gaming shed light on the diversity of user preferences. Notably, the "Minimalistic" graphics style emerged as the top choice, capturing the preference of 28% of respondents. This preference suggests a strong interest in clean, uncluttered visuals that emphasize gameplay and mechanics, aligning with the trend of minimalist design in modern gaming. "Cartoonish" graphics, with a 23% preference rate, also garnered substantial support, reflecting an appreciation for visually distinctive and lighthearted art styles that bring a unique charm to gaming experiences. The enduring appeal of pixel art was evident, as "Pixel Art" graphics and the "Other" category, mostly consisting of pixel art preferences, both received 19% support. This indicates that pixel-based graphics continue to hold a special place in the hearts of gamers, highlighting the importance of including this style in game development. Interestingly, "Realistic" graphics style, with only a 9% preference rate, appeared to be the least favoured. This suggests that, for a sizable portion of users, hyper-realism may not be a top priority in the gaming experience. Considering this diversity of preferences, my project should aim to provide a variety of graphics styles, with an emphasis on "Minimalistic" and "Cartoonish" options, while also offering choices that cater to pixel art enthusiasts. Striking a balance between these styles ensures that the project appeals to a broad and varied gaming audience, offering an engaging visual experience for different player tastes.

Responses
should be able to go from the training straight to a game
examples of pre built AI
have the ability to change the environment of each game
Can we have co-op games with the a.i?
give a unique to the a.i using the character creator
can we have the ability to work together with the a.i
with the background feature, can it be our current a.i playing the game in the background. And can we also have the ability to see the background by itself to see the a.i play?
have the ability to save an a.i regardless of how good it is. Also achievements for example, training an a.i for 100 generations

responding to the question" Do you have any other features/design choices that you would like in this program?"

I appreciate the valuable suggestions provided by respondents for additional features and design choices in my program. These insights reflect a profound understanding of user preferences and desires within the A.I. training and gaming platform. The concept of a seamless transition from training directly to the game is a commendable feature. It enhances the user experience by streamlining the process and optimizing engagement, which aligns with the primary objectives of the project. The proposition of including examples of pre-built A.I. is noteworthy, as it can serve as an invaluable educational resource and practical starting point for users commencing their A.I. training journey. The option to change the environment of each game presents an opportunity to introduce dynamism and personalization, enriching the overall gaming experience to cater to a broader range of user preferences. Cooperative gameplay with A.I. companions is a promising concept that can foster collaboration and introduce unique gaming dynamics, diversifying the gameplay experience. The idea of imbuing A.I. characters with distinctive attributes through the character creator adds depth and personality to A.I. opponents or companions, enhancing the immersive aspect of the platform. The notion of collaborative play with A.I. introduces intriguing possibilities, potentially reshaping gameplay scenarios and further enhancing the cooperative aspect of my project. The dynamic background feature, where the current A.I. plays the game, offers an innovative connection between training and gameplay. Additionally, allowing users to view the background separately provides flexibility and increased engagement. Lastly, the ability for users to save A.I. progress regardless of performance is a user-centric approach that encourages experimentation and continuous learning. Incorporating achievements, such as training an A.I. for a specified number of generations, contributes to user motivation and a sense of achievement within the platform. These suggestions align effectively with the objective of creating a comprehensive, customizable, and engaging A.I. training and gaming platform in my project. I will duly consider these

ideas as I continue the development and refinement of my program to meet the diverse needs and desires of my users.

Limitations

The choice of programming language- I have chosen JavaScript as the primary programming language for my project, but this decision comes with a potential limitation in terms of performance. JavaScript is versatile but known to be slower compared to natively compiled languages like C++ or Java. I identified this limitation during my project planning phase when selecting the technology stack. Slower performance could affect the responsiveness and speed of my program, particularly when handling complex A.I. algorithms and gameplay mechanics. This limitation is noteworthy because it has the potential to impact the user experience, especially as the application grows and tackles more complex tasks. While JavaScript suits web-based applications well, it may not be the most performant choice for certain A.I. and gaming requirements.

All the games will be in 2D- I have decided that all games within my platform will be exclusively in 2D due to constraints related to my experience, knowledge, and project timeline. I recognize the challenges associated with transitioning to 3D game development, including the need for additional time and expertise. Consequently, I have opted to prioritize 2D games. While 2D games have their unique charm, this limitation restricts the diversity and visual complexity of the games I can offer on my project, potentially limiting user experiences.

Limited game variety- Another limitation I have acknowledged is the potential constraint on game variety due to time constraints. During the project planning phase, it became apparent that developing a wide variety of games might not be feasible within the available time. Game development, even in 2D, can be time-intensive, and I may choose to create a smaller number of high-quality games rather than a larger quantity. This limitation is significant because it could affect the platform's appeal to a broader audience. Users might quickly exhaust the available games and seek more variety. However, prioritizing quality over quantity is a valid approach to ensure that the games I offer are polished and engaging.

Graphics will have to be limited to a pixel art style- I have decided to limit the graphics to a pixel art style, primarily influenced by my experience and time constraints. I acknowledge that I may have limited experience with other graphic styles, making pixel art a more manageable option within my project's time. While pixel art can be charming, this limitation narrows the visual diversity of the games I can offer on my program. Users who prefer different graphic styles, such as realistic or cartoonish, may not find those options available, potentially limiting the platform's appeal to a broader audience. However, Due to the results from the questionnaire, I will try my best to create a look using a pixel style which fits with my target audience.

Inclusion of a full storyline will be difficult- One significant challenge I foresee is the potential difficulty in including a full storyline within the games due to time constraints. I recognize that developing

comprehensive and engaging narratives for each game can be time intensive. With my limited time available, it may be challenging to create in-depth storylines that enhance the gameplay experience. This limitation is notable because storylines often contribute to immersion and user engagement. The absence of extensive storylines may result in games that lack a crucial element of depth and engagement.

Limited A.I techniques will be used- A key limitation of my project is the potential restriction on the use of advanced A.I. techniques due to my limited knowledge and time constraints. During my project planning phase, I acknowledged that implementing complex A.I. techniques require a deep understanding of the field. However, due to my limited knowledge and the constraints of my project timeline, it may not be feasible to incorporate advanced A.I. techniques. This limitation is significant as it may impact the level of challenge and sophistication of the A.I. opponents in the games. Limited A.I. techniques may result in less dynamic and challenging gameplay, potentially affecting the overall gaming experience.

Success Criteria

- **1. Create a menu for my game:** I handle developing a user-friendly menu interface for the game. My goal is to create a menu that users can navigate intuitively within the first month of the project. To measure its success, I will assess the menu's usability through user testing. The ability level I aim to achieve is that the menu should be intuitive and positively rated by most user testers. This objective should be completed within the first month of the project.
- **2. Create a neural network which will function as the brain for the A.I:** I will develop a functional neural network for A.I. decision-making. My objective is to establish a basic working model of the neural network within the first three months. This model should exhibit rudimentary decision-making capabilities and serve as a foundation for later enhancements. To measure its success, I will assess the neural network's performance in game scenarios and benchmark it against predetermined success criteria. The proficiency level I aim for is that the neural network should consistently make intelligent decisions in most test cases. This objective should be completed within the first three months.
- **3. Create a game for the A.I. to play:** I am responsible for the development of a simplified game tailored to the A.I.'s capabilities. My objective is to create a functional game prototype within six months, allowing for gradual improvement and expansion in subsequent stages of development. To measure its success, I will assess user feedback and the A.I.'s ability to interact with and understand the game mechanics. The proficiency level I aim for is that the game should be engaging and compatible with A.I. behaviour. This objective should be achieved within the first six months.
- **4. Add an ability to compete against the A.I. in the game:** I will integrate a feature allowing users to compete against the A.I. My objective is to implement a competitive mode within the first nine months, ensuring that users can engage with the A.I. in gameplay scenarios. To measure its success, I will gather user feedback and assess user success rates when competing against the A.I. The proficiency level I aim for is that users should find it challenging but feasible to compete against the A.I. This objective should be completed within the first nine months.
- **5. Add a saving system for scores and A.I. neural networks:** I am responsible for integrating a system for saving user scores and A.I. neural networks. My goal is to develop a basic yet

functional saving system. Users should be able to save and retrieve scores and A.I. networks reliably within the first twelve weeks. To measure its success, I will assess the efficiency and reliability of the data storage system. The proficiency level I aim for is that the system should securely store and retrieve user scores and A.I. networks without data loss. This objective should be completed within the first twelve weeks.

- **6. Optional Improvement 1: Implement a player customization feature:** I will introduce a basic player customization feature to enhance user engagement. If pursued, my objective is to implement this feature within the first thirteen weeks. The feature should allow users to personalize their gaming experience, albeit with certain limitations due to time constraints. To measure its success, I will assess user satisfaction and engagement levels before and after the customization feature implementation. The proficiency level I aim for is that users should find the customization feature enjoyable and visually appealing within the given constraints. This objective should be completed within the first thirteen weeks if pursued.
- **7. Optional Improvement 2: Incorporate advanced A.I. techniques for enhanced gameplay:** I will integrate one or two advanced A.I. techniques that noticeably enhance gameplay. If pursued, my goal is to implement these techniques within the first fifteen weeks. To measure their success, I will evaluate A.I. performance and gather user feedback regarding A.I. behaviour. The proficiency level I aim for is that the A.I. should exhibit more dynamic and challenging behaviours, enhancing the overall gaming experience within the given constraints. This objective should be completed within the first fifteen weeks if pursued.
- **8. Optional Improvement 3: Expand game variety with an additional game title:** I am responsible for creating a new game title that complements the existing offerings. If pursued, my objective is to develop a single new game title that is functional and engaging within the first sixteen weeks. To measure its success, I will assess user satisfaction and engagement with the new game title. The proficiency level I aim for is that the new game should be well-received and add to the platform's appeal within the given time. This objective should be completed within the first sixteen weeks if pursued.

Software and Hardware Requirements

For my project, I have outlined the hardware and software requirements to ensure accessibility, efficiency, and functionality. I have prioritized compatibility with various operating systems, including Windows, macOS, and Linux, to ensure that a wide audience can easily engage with the platform. I recognize the importance of a capable processor like the Intel Core i5 or AMD Ryzen 5, which is essential for ensuring smooth software development and efficient multitasking, crucial for coding and project management.

While the primary focus is software development, I also acknowledge the potential benefit of a decent graphics processing unit (GPU) for enhancing visual aspects and supporting game development tasks. In terms of storage, I have considered the need for adequate space, particularly a minimum of 256 GB SSD, which facilitates fast application loading and swift data access. To support seamless updates, collaboration, and online resource access, I have emphasized the importance of a stable internet connection.

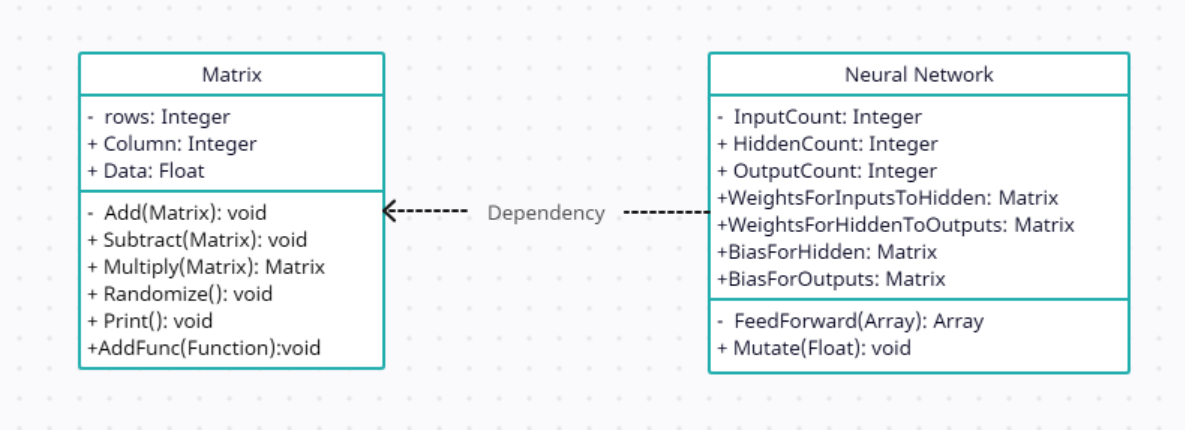
Furthermore, I have prioritized compatibility with modern web browsers, such as Google Chrome, Mozilla Firefox, and Microsoft Edge, to ensure effective web-based development and thorough testing. For coding and project management, I have chosen Visual Studio Code (VSCode) as the primary code editor, given its comprehensive features. Finally, in case of game development needs, I have included a pixel art editor, which allows for the creation and editing of pixel-based graphics and animations. These carefully chosen specifications and justifications collectively create an accessible and efficient environment for both software development and potential game development tasks, catering to a diverse user base and development needs within my project.

Design

This section will show the design features and planning for my project and the tests during and after the iterative development process.

Class Diagrams

The following information below will display and describe a few of the classes that will be part of my project. This will not be all the classes I use in the program as more may be needed to achieve the vision of my project.



Here are the class diagrams that will be included within my Neural Network file. The classes are “Matrix” and “Neural Network”. Below will be a quick explanation of the attributes and methods within each class.

Matrix

The Matrix class will be used within the Neural Network and will perform the linear algebra required for it to function correctly. It will take in 2 parameters:

- Rows: how many rows the matrix has.
- Cols: how many columns the matrix has.

Attributes:

- Rows: how many rows the matrix has.
- Cols: how many columns the matrix has.
- Matrix: the values within the matrix will be held here.

Methods:

- Randomize: goes through the array and gives each entry a random value between –1 and 1
- Add: used to add 2 matrices together.
- Subtract: used to subtract 2 matrices together.
- Multiply: used to multiply 2 matrices together and returns a new matrix.
- Print: outputs the matrix's values in the console.
- AddFunc: Allows functions to be applied to the matrix.

Neural Network

The Neural Network class will function as the "brain" for all the A.I. in this program. It will take in 3 parameters which are:

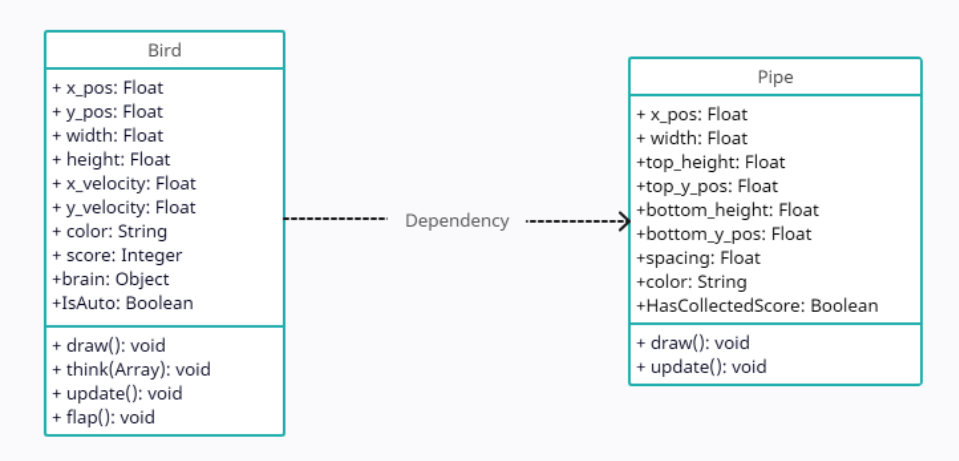
- Input Count – How many inputs the A.I will have e.g.; it will need 2 inputs to know its x and y position in game.
- Hidden Count – How many Hidden nodes the A.I will have.
- Outputs Count – How many outputs the A.I will have e.g., it will need to have an output if it needs to jump, another if it needs to move to the left and another if it needs to go to the right.

Attributes:

- Input Count: How many inputs the A.I will have e.g.; it will need 2 inputs to know its x and y position in game
- Hidden Count: How many Hidden nodes the A.I will have.
- Outputs Count: How many outputs the A.I will have e.g., it will need to have an output if it needs to jump, another if it needs to move to the left and another if it needs to go to the right.
- WeightsForInputsToHidden: A matrix which will store the different weights between the input layer and the hidden layer of the neural network.
- WeightsForHiddenToOutputs: A matrix which will store the different weights between the hidden and the output layer of the neural network.
- BiasForHidden: A column matrix which will store the bias values for the hidden layer within the neural network.
- BiasForOutputs: A column matrix which will store the bias values for the output layer within the neural network.

Methods:

- Feedforward: will take in an array of inputs as a parameter and will perform the feedforward algorithm and will return an array of outputs depending on the number of outputs specified in the parameter.
- Mutate: will take in a float value between 0 and 1 as a parameter and will go through all the weight and bias matrices and slightly by the amount given by the parameter.



These 2 classes will be used specifically for the flappy bird game within my program. The Bird class will Depend on the Pipe class as without it, the Bird class will not be able to function correctly.

Bird

The Bird class will be used by both the player and the A.I. within the game. They will take in 1 Boolean parameter which will be used to decide whether a player or an A.I. is controlling this bird class.

Attributes:

- X_pos: decides where the bird is on its x-axis (left and right).
- Y_pos: decides where the bird is on its y-axis (up and down).
- Width: used for the width of the bird.
- Height: used for the height of the bird.
- X_velocity: the speed of the bird on its x-axis (left and right).
- Y_velocity: the speed of the bird on its y-axis (up and down).
- Colour: will be used for the bird's colour.
- Score: what score the Bird has. It will also be used in the restart algorithm.
- Brain: will be a neural network and will be used for the A.I. to control the bird.
- IsAuto: this will decide whether the A.I. is controlled by the A.I or the player.

Methods:

- Draw: The draw method will be called every frame and will draw the bird onto the screen.
- Think: Will take in the array of pipes as the parameter and will be used to constantly update the inputs for the neural network.
- Update: This will perform all the logic within the game such as updating score and physics for the bird.
- Flap: This will allow the bird to “bounce” up when it is activated.

Pipe

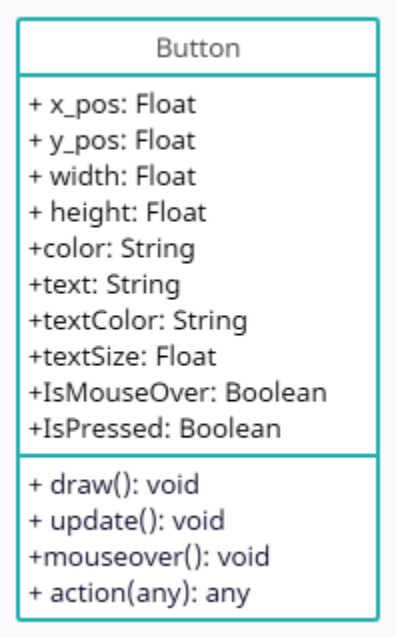
The Pipe class will be an obstacle that spawns depending on a timer the Birds will have to avoid colliding with the pipes by using their flapping ability.

Attributes:

- X_pos: finds where the pipe is on its x-axis (left and right).
- Top_height: used for the height of the top pipe.
- Top_y_pos: figures out where the top pipe is on its y-axis (up and down).
- Bottom_height: used for the height of the bottom pipe.
- Bottom_y_pos: figures out where the bottom pipe is on its y-axis (up and down).
- Colour: will be used for the pipe's colour.
- Spacing: used to decide the spacing between the bottom and top pipe.
- HasCollectedScore: A Boolean value to find out whether the birds have passed the pipe or not.

Methods:

- Draw: The draw method will be called every frame and will draw the bird onto the screen.
- Update: This will perform all the logic within the game such as moving the pipe from one end of the screen to the other.



This is the class diagram for the button within my project. This will be an essential part of my project as it will be used to navigate around the entire program. Most of the attributes listed will be parameters for the button class.

Attributes:

- X_pos: the x position of the button relative to the screen.
- Y_pos: the y position of the button relative to the screen.
- Height: the height of the button.
- Width: the width of the button.
- Colour: the colour of the button.
- Text: the text that will be displayed on the button.

- TextColor: the colour of the text on the button.
- TextSize: The size of the text on the button.
- IsMouseOver: a Boolean value to find whether the mouse is over the button.
- IsPressed: a Boolean value to decide whether the button has been pressed.

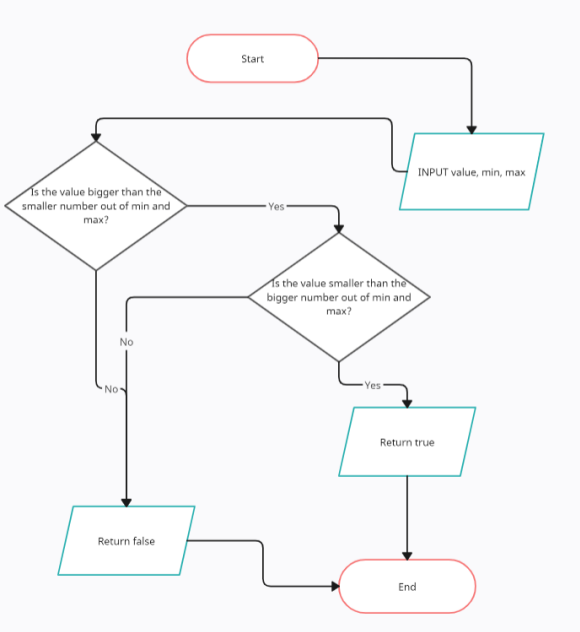
Methods:

- Draw: will draw the button on the screen.
- Update: will be used to check if the button has been pressed or if the mouse is over the button.
- MouseOver: will perform an action if the mouse is hovering over the button e.g., change colour.
- Action: takes in any parameter (usually based on the purpose of the button) and does a function which may or may not return something back. Is only activated when the IsPressed function is true. It will also automatically turn it back to false after the function has been run.

Algorithms

The following below will show a few of the algorithms that I plan to implement either flowchart form or pseudocode form. I will then explain what each of these algorithms do and why they will be added into my program.

```
Function InRange(value, min, max)
    If value >= Min(min, max) AND value <= Max(min, max) Then
        Return True
    Else
        Return False
    End If
End Function
```



This is a function which will find out if a value is in-between 2 values. The function takes in 3 float values. These are value, min, and max. The algorithm works by first choosing the larger value out of the min and max value. This is done so that it does not matter what value is put in as the max or min value. After finding out which value is bigger, it finds out if the value is in-between the min and max values. If it is, then it returns true, if it is not then it returns false. This function will be used within other functions within my project to make some games work.

Test Data:

Type of Data	Value	Expected Outcome	Actual Outcome	Evaluation
Normal	Value = 20 Min = 1 Max = 10	The expected outcome is that the function will return false	The program outputs false	The function works as intended so there are no changes that need to be made
Normal	Value = 6 Min = 1 Max = 10	The expected outcome is that the function will return true	The program outputs true	The function works as intended so there are no changes that need to be made
Extreme	Value = 23862.55 Min = - Infinity Max = Infinity	The expected outcome is that the function will return true	The program outputs true	The function works as intended so there are no changes that need to be made
Erroneous	Value = 5 Min = - 10 Max = “ten”	The expected outcome is that the function will return an error	The program outputs false	Within the function, I will need to make sure I implement a feature which would recognize if the parameters inputted are a string and if so, returns an error

```
Function Interact(x, y, object)
    If InRange(x, object.x, object.x + object.width) AND InRange(y,
object.y, object.y + object.height) Then
        Return True
    Else
        Return False
    End If
End Function
```

This function is called Interact. It takes in 3 parameters. The parameters x and y are float values while the object parameter must be an object with the attributes x, y, width, and height. This function will be used to see if a point (x,y) is within an object. It will be the main function that is used with my button class to find out when the mouse is either over it or is pressing it. It returns a Boolean value depending on the output.

```
Function Touching(item1, item2)
    If (item1.x + item1.width >= item2.x) AND (item1.x <= item2.x +
item2.width) AND (item2.y <= item1.y + item1.height) AND (item1.y <=
item2.y + item2.height) Then
        Return True
    End If
End Function
```

This function is known as Touching and takes 2 parameters, known as item1 and item2. These objects must have attributes x, y, width, and height. This function decides whether 2 objects are colliding and if so, it will return true or false. This function will be used as collision detection within the games in my project such as flappy bird.

```
Function Lerp(a, b, t)
    Return a + (b - a) * t
End Function
```

This function is known as Lerp. It takes in 3 parameters. A and B can be any float value while t must be in between 0 and 1. The function will return a float value which would have been interpolated in between A and B. This function will be used within the mutate method in my neural network which will allow it to slightly change all the weights and biases within the network.

```
Function Sigmoid(num)
    FinalNum = 1 / (1 + Exp(-num))
    Return FinalNum
End Function
```

This function is known as Sigmoid which takes in 1 float parameter. After that, it uses the parameter and places it in the Sigmoid function, a function which turns any value into a value from 0 to 1. This function will be used as an activation function within the Feedforward method of my neural network.

```
Function SquareNum(Num)
    Return Num * Num
End Function
```

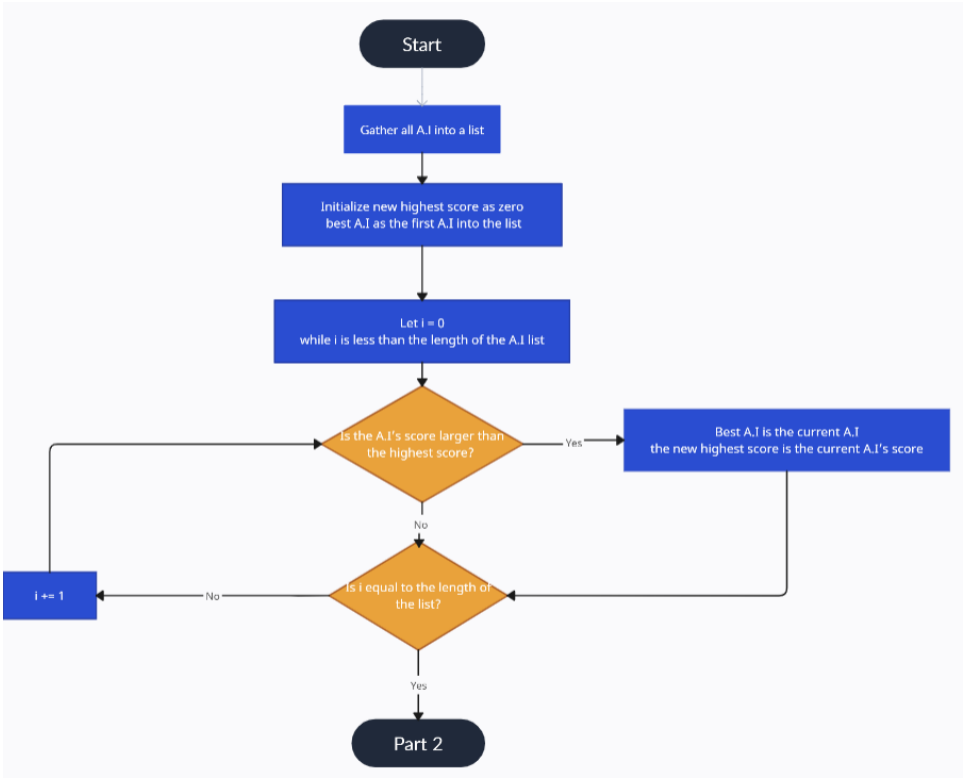
This function is known as SquareNum which will take in any float value. After that it will square the number and return that new value back. This function will be used as a test function to make sure that the AddFunc method in my matrix class works as intended.

```
function TurnArrayToMatrix(array)
    let matrix = new Matrix(array.length, 1)
    for i from 0 to matrix.rows - 1
        for j from 0 to matrix.cols - 1
            matrix.matrix[i][j] = array[i]
    return matrix
```

This function is called TurnArrayToMatrix. It takes in 1 parameter which must be an array and using that array it creates a vector matrix (A matrix with only one column) with the number of rows being the length of array. The function then loops through the matrix and puts the values of the array in the matrix. After that, it returns the matrix. This function will be used within the feedforward method within my neural network class and will be used to turn the array of inputs into a matrix which can be used for matrix operations.

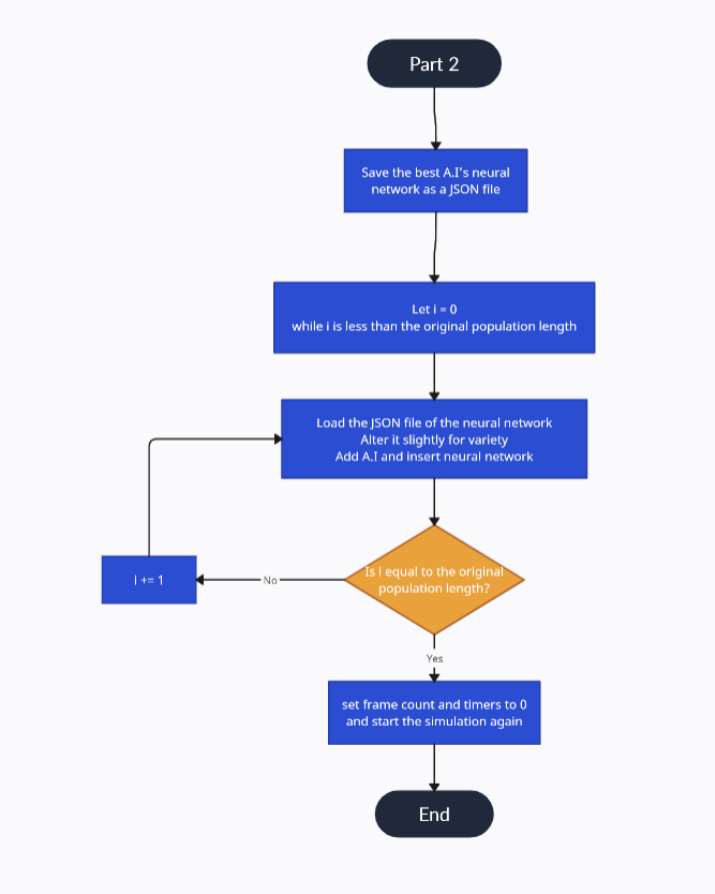
```
function TurnMatrixToArray(Matrix)
    let array = []
    for i from 0 to Matrix.rows - 1
        for j from 0 to Matrix.cols - 1
            array.push(Matrix.matrix[i][j])
    return array
```

This function is called TurnMatrixToArray. It is the opposite of the TurnArrayToMatrix function in which it takes a matrix as a parameter instead of an array. After that it creates a new empty array that will be given the values of the matrix. It then returns the array as a value. This function will also be used in my feedforward method within my neural network as a way of turning the outputs that will be in matrix form into an array of values.



This flowchart displays what the restart function will do within my project. This is one part of the function as the whole thing was impossible to place in one screenshot. The first part will involve getting all the A.I. into a list/array so we can compare all of them at once. We then initialize the highest score as zero and best A.I as the first A.I. in the list. We then loop through the list to find the highest score and

which A.I has that score. The function then moves to part 2.

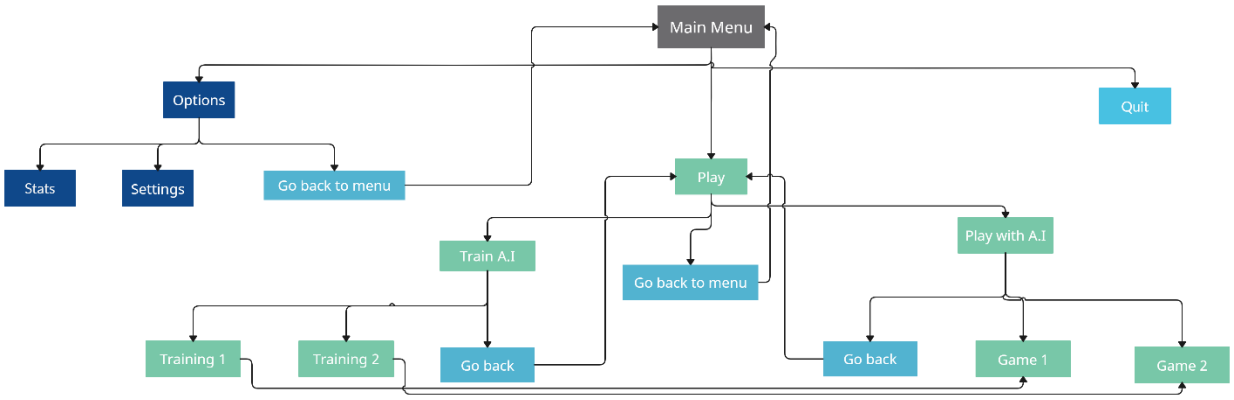


In part 2, we save the best A.I's neural network as a JSON file and then create a new loop, adding new A.Is into the list with slightly altered versions of the best A.I's neural network and once the list is filled with the new A.I, it will restart the simulation again. This algorithm will be used in all the training games within my project and will be called after a certain condition is met such as a timer or if all the A.I have been eliminated.

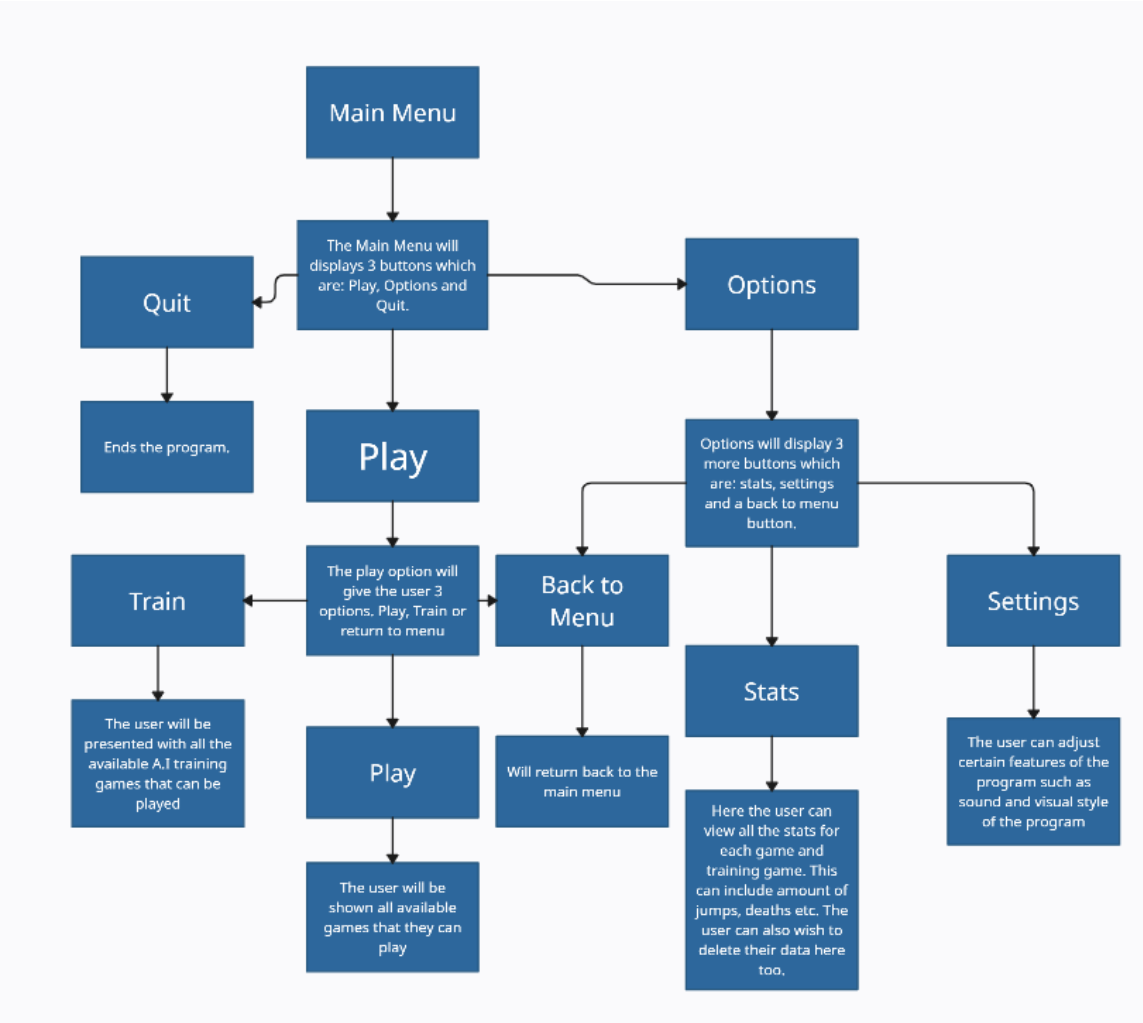
Functional Decomposition of the program

The functional decomposition of my program will show how parts of my project will link together such as the main menu. Currently it shows a draft of what I plan for it to look like, and this may change once I

start developing my program.



This image shows a diagram of the layout of the project and the navigation around the program. The user will first be introduced to the main menu where he will be able to view the 3 options available. Below is an example of how the modular design diagram which displays the features of what each part of the navigation system can do.

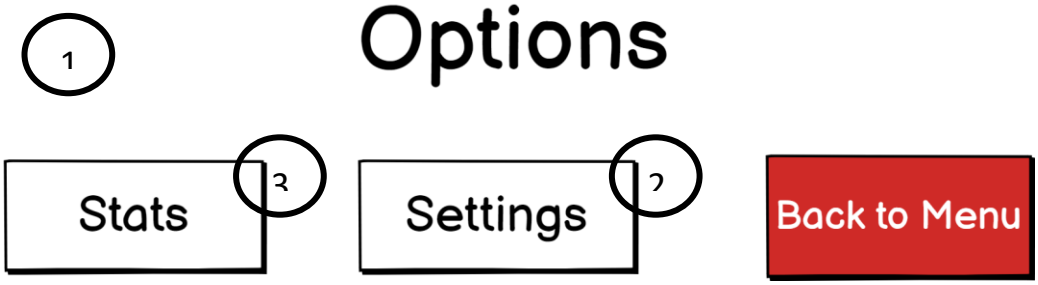


Mock up GUIs

This part of the document will focus on the graphical user interface of the program and how I imagine the project to look while I am developing it. This is currently just a draft and when creating the program, I may change some features of the guis to make it more appealing.

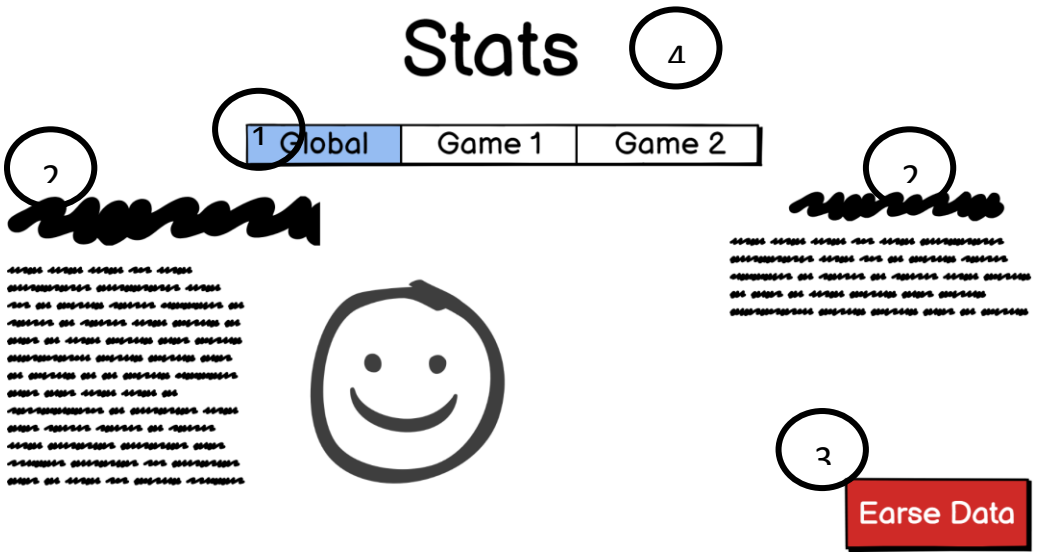
The main menu will look something like this and will also have the following features:

- **Title (2)-**The main menu will say “A.I. Training Simulator” as a way of introducing the user to the program. The colour will be based on the game playing in the background.
- **Gameplay in the background (1)-**The gameplay will be of the last played game with the A.I that the user trained playing it. If there is not a last game played, the program will just display a random game playing in the background.
- **Buttons allowing navigation (3)-**In the main menu there will be 3 buttons. One leads to settings and stats, one leads to the training or playing section and one quits the game. The button’s colour will be adjusted depending on the game in the background.



This is an example of what the options menu will look like within the program. Here are some of the features of this program:

- **Background and text will fit the colour scheme (1)-**The colour of the background will be of a similar colour to what the last game played was like the main menu.
- **Settings option (2)-**Within the options menu, one of the available options will be settings and when it is clicked it will open the settings menu which will allow the user to change various features within the program.
- **Stats Option (3)-**The stats option is another available option from the options menu and will allow the user to view information about the program such as the highest score achieved by the A.I and how many generations it has been ran for. In here the user can also delete the saved data from the program if they wish to. Below is a mock up GUI of how I intend for the stats menu to look like.



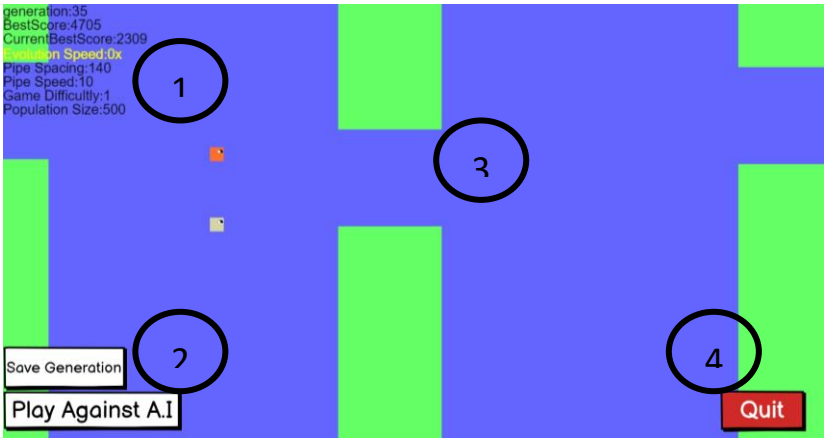
This is the stats menu, and it will display all the information about the games such as the number of times the player has jumped in said game, died in game etc. Here are some features of the menu:

- **Tab Menu UI (1)-**There will be a tab menu just below the title of the menu where the user will be able to navigate between the specific games and training programs to see the individual stats of each program.
- **Display of information (2)-**When the user has decided which game/training game it would like to view, the menu will display a list of information. These can range from the following choices: Number of jumps, Highest score achieved, Highest generation, Number of deaths and many more. It may also display an image of the game to remind the users what game they are currently viewing
- **Colour scheme (4)-**The colour scheme will be based on what game the user is currently viewing. I may decide to change this as it may become irritating to the eyes, but I will double check this while I am developing the program.
- **Button to delete data (3)-**For each game/training game, the user will have the option to delete all the data associated with that game. For training games, this will remove all the A.I's saved within the game and will require the user to train a new A.I from scratch. If they are on the global option (which shows all the users stats combined), then this will completely wipe all the data from the project, and it will be as if they are using the program for the first time. Because

of the extreme nature of this feature, when clicked, it will create a popup box to make sure the user wants this and will remind them that the data cannot be restored.

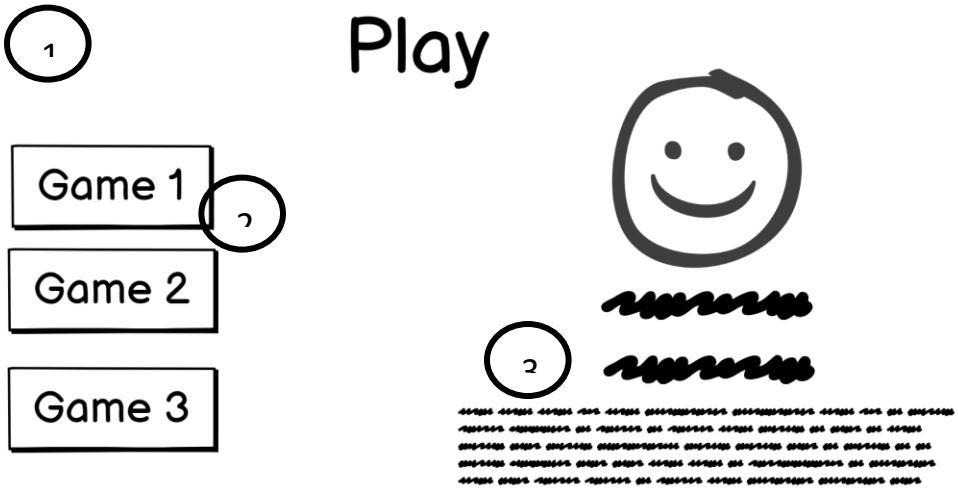
If the user pressed play and then train, this is the menu that they would be introduced to. Here the user can choose what training program they would wish to use.

- **Background (1)-**When the user hovers over one of the buttons, the background will display an image of the following program e.g., if the user hovers over the flappy bird game, the background will display an image of flappy bird. If the user does not hover any of the options, then the background will just continue displaying the last played game just like in the main menu.
- **Text relating to the game (2)-**Again, whenever the user hovers over the buttons, text about the program will display at the bottom of the screen.



Once the user has clicked on a training game, they will be able to start training the A.I. and this is an example of how I imagine the training UI to look like for the flappy bird game.

- **Game Information (1)-**In each training game, I will display information about it such as the generation, the current highest score and the highest score achieved. Depending on the game, there may be more information displayed.
- **Buttons for interaction with the A.I. (2)-**For each game, there will be a choice to either save the current generation or play against the current generation. The colour of these buttons will depend on the current game.
- **Gameplay (3)-**The game will take up the full screen and will be playing the instant the program loads. For some, there may be a count down before the start of the game
- **Button to quit (4)-**There will be a button at the bottom corner of the program so the user can quit the current game and return to the menu. The A.I. will automatically be saved so if the user were to go back to the game, the data would not be lost



This is a GUI of how the play option will look within my project. This menu will display after the second play option has been clicked on.

- **Background and colour theme (1)-**The colour theme will be like the main menu theme and the background will change depending on what the user is hovering on.
- **Buttons on the side (2)-** The buttons will be linked to each game and will be on the left side of the screen.
- **Information and picture (3)-**Whenever the user hover overs one of the buttons to one of the games, an icon of the game will appear along with extra information such as what the game is about, total minutes on the game and the current level A.I. for this game.

Key Variables

The following table displays information about a few of the variables I will use within my program. This table does not display all of them, however it does contain the ones I currently view as vital to the project.

Name	Data type	How will it be used

Buttons	Array	Will be used to store all the buttons used within the program, so I will not have to create multiple variables for each individual button.
Brain	An attribute for every A.I class (object)	Will be where the neural network is given to the A.I so it can perform decisions with a given number of inputs.
FrameCount	Integer	Will increase every frame and will be used for spawning items in certain games e.g., spawning pipes in flappy bird.
Birds	Array	Will store all the A.I. and player for the flappy bird games so they can be updated all at once by use of a for loop.
Pipes	Array	Like the birds array, this will store all the pipes within the flappy bird games so they can be updated all at once via a for loop.
Score	An attribute for every A.I class (Float)	Will decide how well the A.I did at the game. Depending on what the aim is, score will be given to the A.I differently e.g., in the flappy bird game, score will be given on how long they survive while if I decided to add a game such as a shooting game, their score will be given based on accuracy.
Generation	integer	The generation variable will be used to keep track of what generation the A.I. is on. This will be needed so the user can see the growth of the A.I.

Usability features

- **Settings Options**: Within this project, a "Settings" menu has been seamlessly integrated, affording the user the ability to tailor their gaming experience. Options encompass the choice to display frames per second (FPS) for performance tracking or the discretion to disable animated backgrounds, thereby conserving system resources. Additionally, a "Lower Power Mode" is made available, extending gaming sessions while saving energy. This provision underscores the commitment to providing players with a heightened degree of flexibility and autonomy over their gaming milieu. These settings are artfully designed to allow users to personalize their experiences to align with their proclivities and hardware capabilities, thereby ensuring that a diverse user base can derive enjoyment from the platform.

- **Tips on Button Hover**: Whenever the user positions their mouse cursor over a button or any interactive element within the platform, a salient and instructive tooltip materializes, imparting swift and pertinent insights about the specific feature or action in question. This feature has been meticulously crafted to address the pivotal element of lucidity in platform navigation. These tooltips function as personal guides, adeptly illuminating the user's path through the platform's multifarious functionalities. Their purpose is to minimize ambiguity, thereby endowing the user with a seamless and user-friendly experience.

- **Readable Text**: The text incorporated into the fabric of this project is thoughtfully designed with an unwavering emphasis on legibility. This meticulous approach extends to the deployment of fonts that are unequivocally clear and legible, complemented by judiciously adjusted text size and contrast, culminating in a user interface wherein every textual element is conspicuously legible. Such an unwavering commitment to readability is rooted in an unswerving dedication to inclusivity, with the overarching aim of ensuring that the platform stays accessible and user-friendly to a broad spectrum of users, irrespective of their visual acuity.

- **Pause Menu**: During gameplay, the user may avail themselves of the "Pause" button to gain access to a menu that serves as the quintessential control centre. This menu offers a comprehensive array of functions, including the ability to temporarily interrupt gameplay, make real-time adjustments to settings, safeguard progress, and seamlessly resume gameplay from the precise point at which it was paused. The architecture of the pause menu is underpinned by an unwavering commitment to accommodating the user's unique preferences and requirements. By conferring enhanced convenience and flexibility during gameplay, it effectively places the reins of control in the hands of the user, thus epitomizing a commitment to the provision of a seamless and enjoyable user experience.

- **Ability to Delete Data**: Should the user ever harbour the inclination to start afresh, they may do so without reservation. The ability to expunge saved game data, AI models, or any other user-generated content has been thoughtfully integrated, akin to a virtual reset mechanism. This feature epitomizes a commitment to endowing the user with complete control over their data and the trajectory of their experience. The option to delete data encapsulates an earnest commitment to respecting user privacy and their prerogative to embark on new beginnings. It reinforces the idea that the user stays the ultimate arbiter of what they choose to keep and what they choose to relinquish.

- **Menus**: A cornerstone of this project is the presence of meticulously structured menus that offer streamlined and intuitive navigation. The main menu marks the point of embarkation on the user's journey, serving as the gateway to the entire experience. These encompass the game selection menu, which helps the user's choice of gaming adventures, the AI model customization menu, enabling users to sculpt AI behaviour, and the settings menu, which gives the means to calibrate the user experience. The architecture of these menus closely mirrors a well-ordered cartographic experience, whereby clarity and user-friendliness are paramount. They have been diligently crafted to empower the user to effortlessly find their desired gaming experiences and AI model customization options, thereby underscoring a commitment to delivering a straightforward and gratifying user experience.

Testing

The following table displays the tests I will do while I am developing the project to make sure everything works as it should. It contains the item tested, the applied test and the expected outcomes of each test if it were to go smoothly.

Item Tested	Applied Test	Data	Expected outcome
Hover over buttons	Move the mouse until it hovers over the button. And do the same when the mouse is not over the button.	N/A	The button should perform the mouseover method) which could be anything that it is set to be e.g., the colour changes when the mouse is over the button, and it should not do anything while the mouse is not over the button.
Pressing the buttons	Left and right click over and outside the button.	N/A	The button should perform the action method which again can be whatever the button's task is e.g., loading up a new menu. When the mouse has been clicked outside of the button, the button should not do anything.
Flappy bird flap ability	Pressing the space bar whilst the flappy bird game is playing normally and again when the game is paused or when the game is not playing.	N/A	While on the flappy bird game, the player bird should bounce upwards in a flapping motion. Anywhere else and nothing should happen.
Parameters for the matrix class	When initialising a matrix, I should be able to only put integers as it is impossible to have a float as a parameter. E.g. it is not possible to have 1.5 rows	Normal integer data such as 5 and 2. Extreme integer data such as 100 and 50 and other data types such as floats and strings.	It should create a matrix with the given parameters if both parameters are integers. If one or both parameters are non-integers, it should return an error.

Data post development

Like the testing table, this table will also display the tests I will perform, however these tests will be done once the entire program has been finished. This is to ensure that everything works together just as it should and to make sure it does not break once it is all put together.

Item Tested	Applied Test	Data	Expected outcome
Hover over buttons	Move the mouse until it hovers over the button. And	N/A	The button should perform the mouseover method which could be anything that it is set to be e.g., the

	do the same when the mouse is not over the button.		colour changes when the mouse is over the button, and it should not do anything while the mouse is not over the button.
Pressing the buttons	Left and right click over and outside the button.	N/A	The button should perform the action method which again can be whatever the button's task is e.g., loading up a new menu. When the mouse has been clicked outside of the button, the button should not do anything.
Flappy bird flap ability	Pressing the space bar whilst the flappy bird game is playing normally and again when the game is paused or when the game is not playing.	N/A	While on the flappy bird game, the player bird should bounce upwards in a flapping motion. Anywhere else and nothing should happen.
Deleting data	Save some data to the program then go to the settings to delete it	The A.I. from one of the games e.g. flappy bird	The data should be erased and should not be able to be recovered.
Neural network works	Use the neural network to play one game and sees if it manages to learn to play it.	A neural network	The neural network should be able to eventually play the game in a successful way and within reasonable time.
Menu navigation.	Press all the buttons in the main menu to see if the program will break. E.g. press the play button, press the train A.I. button then press the go back to menu button.	N/A	The program should not break at all, and it should be able to move through the menus in reasonable time
Settings	Change some of the settings and see if the program will apply those settings	N/A	The program should apply these settings without any errors.
Play without training	Attempt to play one of the games without first training an A.I	N/A	The program should not allow me to play any of the games.

Iterative Development

My project will be split into multiple files. One for the neural network, one for the menu and one file per game and training game.

Creating NeuralNetwork.js

The code shown in this section will be part of the NeuralNetwork.js file within my project. This file will contain the neural network and matrix class needed for this project. The user will never directly interact with this file which means the design and layout of this will not matter as much as they will in later files.

```
//Matrix class
class Matrix{
    constructor(rows,cols){
        this.rows = rows
        this.cols = cols
        this.matrix = []
        for(let i = 0; i < this.rows; i++){
            this.matrix[i] = []
            for(let j = 0; j < this.cols; j++){
            this.matrix[i][j] = 0
            }
        }
    }
```

Here, I have created a new class called Matrix that will take in the number of rows and columns as parameters. It then creates a 2D array with the given rows and columns. This will be used for linear algebra within my neural network. To create this, I utilized JavaScript classes and constructors. The class constructor accepts parameters for specifying the matrix dimensions. Additionally, nested for loops are used to set up the initial matrix structure.

Below is a table which displays the test data I will use to make sure that creating a new matrix works as intended. Due to the nature of using JavaScript files, I created a html file which will allow me to use the classes and functions within the NeuralNetwork.js file. Here is what it contains:

```
School Project > Javascript tests.html > script
1   <script src="NeuralNetwork.js"></script>
```

All it does is link the file so I can use it within the browser.

Type of Data	Data	Expected Result	Actual Result

Normal	Rows = 2, cols = 2	It will create a matrix with 2 rows and 2 columns	`let m1 = new Matrix(2,2)`
			`undefined`
			`console.table(m1.matrix)`
			VM163:1
			(index) | 0 | 1
			0 | 0 | 0
			1 | 0 | 0
			The result is a matrix with 2 rows and 2 columns. This was exactly what I predicted so I will not have to change anything

Normal	Rows = 5, cols = 2	It will create a matrix with 5 rows and 2 columns	`let m1 = new Matrix(5,2)`
			`undefined`
			`table(m1.matrix)`
			VM374:1
			(index) 0 1
			0 0 0
			1 0 0
			2 0 0
			3 0 0
			4 0 0
			The result is a matrix with 5 rows and 2 columns. This was exactly what I predicted so I will not have to change anything

Extreme	Rows = 100 cols = 1	It will create a matrix with 100 rows and 2 columns	`let m1 = new Matrix(100,2)`
			`undefined`
			`table(m1.matrix)`
			VM534:1
			(index) 0 1
			0 0 0
			1 0 0
			2 0 0
			3 0 0
			4 0 0
			5 0 0
			6 0 0
			7 0 0
			8 0 0
			9 0 0
			10 0 0
			The result is a matrix with 100 rows and 2 columns. Due to the length of the rows, I will not be displaying all 100 rows, however, it has created 100 rows. This was exactly what I predicted so I will not have to change anything.

Erroneous	Rows = “one” cols = 5	It will throw an error as “one” is not an integer value.	`let m1 = new Matrix("one",5)`
			`undefined`
			`table(m1.matrix)`
			The result is a matrix with no rows or columns. This is not what I expected to happen so I will add a condition which will throw an error if the data type is incorrect.

Erroneous	Rows = “”, cols = “”	Based on the previous test data submitted, I expect it to create a matrix class with no columns and no rows.	`let m1 = new Matrix()`
			`undefined`
			`table(m1.matrix)`
			
			The result is a matrix with no rows or columns. This is what I expected and to fix this error, I will need to redefine some things within my matrix class.

Erroneous	Rows = 0.5, cols = 5.5	I expect it to throw an error due to the 0.5 rows.	`let m1 = new Matrix(0.5,5.5)`
			`undefined`
			`table(m1.matrix)`
			VM794:1
			(inde… 0 1 2 3 4 5
			0 0 0 0 0 0 0
			The output is a 1x5 matrix which is not what I expected at all. I will try to change this when I go back to editing the code of my matrix class.

Errors and Fixes

There were a few logical errors within my implementation of my matrix class. I think I can fix all of them with some simple modifications. The following code below displays how I changed the matrix class.

```
//Matrix class
class Matrix{
    constructor(rows,cols){
        this.rows = rows || 2
        this.cols = cols || 2
        if(!Number.isInteger(rows) || !Number.isInteger(cols)){
            throw new Error("Please enter an integer when defining a matrix")
        }else{
            this.matrix = []
            for(let i = 0; i < this.rows; i++){
                this.matrix[i] = []
                for(let j = 0; j < this.cols; j++){
                this.matrix[i][j] = 0
                }
            }
        }
    }
```

Here, I added some error catching which will display the message “Please enter an integer when defining a matrix” if the parameters defining the rows and columns of the matrix, are not integers. This input validation will ensure that the program will not break upon the incorrect data being inputted. I have also added a feature where if the number of rows or columns are not defined, then the default will be a 2x2 matrix. I did this to further prevent the chance of errors occurring further within my program.

```
//randomizes the values of the matrix
randomize(){
    for(let i = 0; i < this.rows; i++){
        for(let j = 0; j < this.cols; j++){
        this.matrix[i][j] = Math.floor(Math.random()*10)
        }
    }
}
```

I have added the randomise method that I said I would add in the design section of my project. Here it loops through the matrix and generates random values for each element.

```
m1 = new Matrix()
▶ Matrix {rows: 2, cols: 2, matrix: Array(2)}
m1.randomize()
undefined
table(m1.matrix)
                                                    VM1080:1
```

(index)	0	1
0	0	6
1	3	9

```
▶ Array(2)
```

Here is an example of the code working. As shown here, the values in the matrix are now filled with integers from 1-10.

```
print(){
    console.table(this.matrix)
}
```

Again, here is another of the methods I intended to add. This is the print function which allows me to print the contents of the matrix in an easier way than the previous version of constantly table logging the matrix.

```
let m1 = new Matrix(); m1.randomize()
undefined
table(m1.matrix)
```

VM145:1

(index)	0	1
0	1	3
1	3	4

▶ Array(2)

```
undefined
m1.print()
```

NeuralNetwork.js:93

(index)	0	1
0	1	3
1	3	4

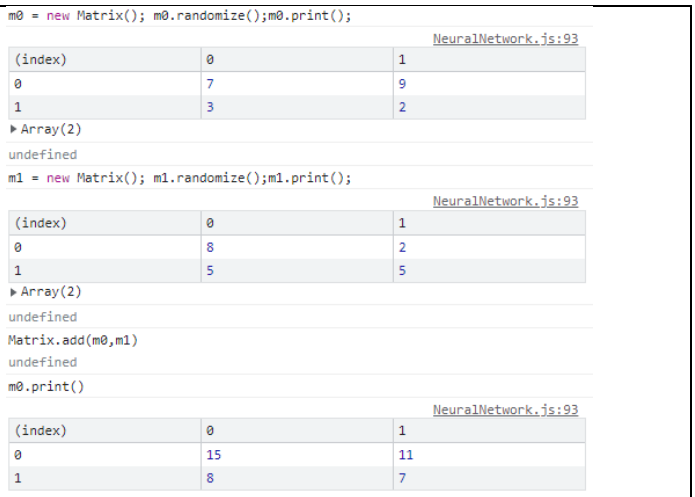
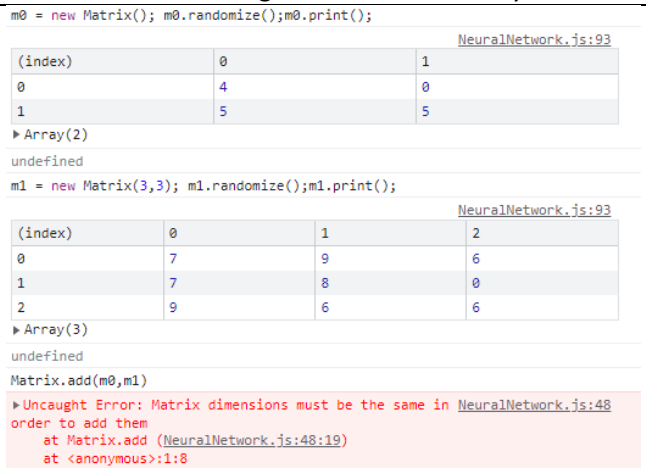
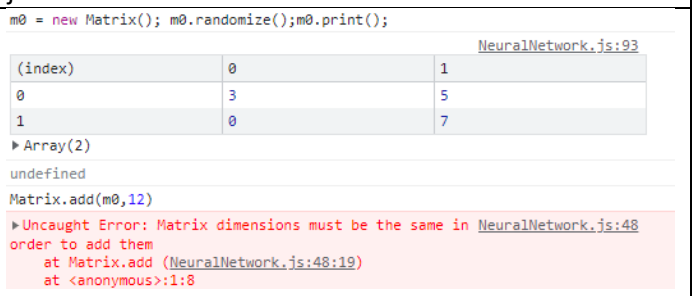
▶ Array(2)

Here is the output. As shown in the image, all that is required to display the output is just MatrixName.print, which will make debugging the matrices within the program a lot easier and quicker which will lead to quicker bug fixes in the future.

```
static add(m0,m1){
    //adding matrices
    if(m0.rows == m1.rows && m0.cols == m1.cols){
        for(let i = 0; i < m0.rows; i++){
            for(let j = 0; j < m0.cols; j++){
                m0.matrix[i][j] += m1.matrix[i][j]
            }
        }
    }else{
        throw new Error("Matrix dimensions must be the same in order to add them")
    }
}
```

This image displays the add method of the matrix which was mentioned in my class diagrams of the function. I have used a static method over a regular method as it will be more efficient within the long run because I will not need to create another instance of a matrix to use it. The method takes 2 parameters which will be matrices. It then checks whether the matrices have the same dimensions, and if they do, it adds them element by element. If the dimensions do not match, it raises an error.

Type of Data	Data	Expected Result	Actual Result

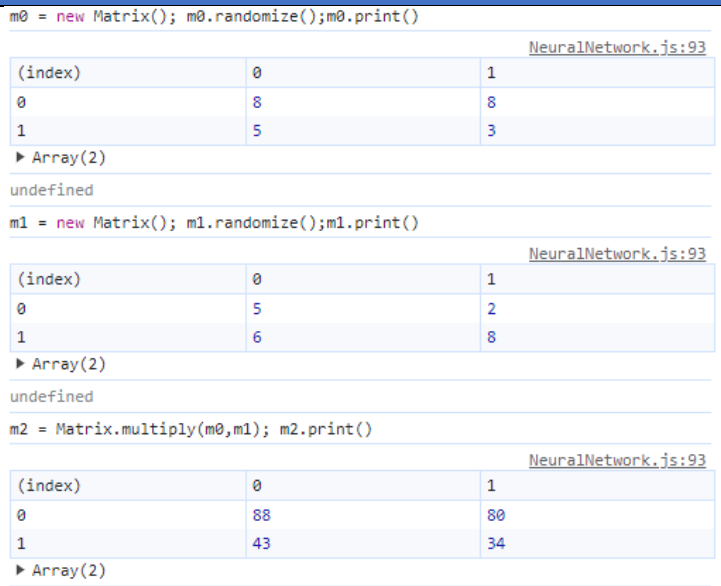
Normal	2 2x2 matrices added together	It will add the corresponding elements of the matrices together	 Here, I have created 2 matrices m0 and m1, I then used the add method and added m1 to m0. The result is when I print m0. We can see that the elements within the matrix have changed in the correct way.
Erroneous	1 2x2 matrix and one 3x3 matrix added together	It will throw the error message “Matrix dimensions must be the same in order to add them”	 Here, I have created 2 matrices m0 and m1, m0 is a 2x2 matrix whereas m1 is a 3x3 matrix. When I try to add them using the add method. It throws an error message just as intended
Erroneous	1 2x2 Matrix and an integer	It will throw an error	 I have created a 2x2 matrix m0 and I have tried to add the integer 12 to it. It threw the same error message as if the matrix dimensions were not the same. An error was expected; however, I did not expect this message.

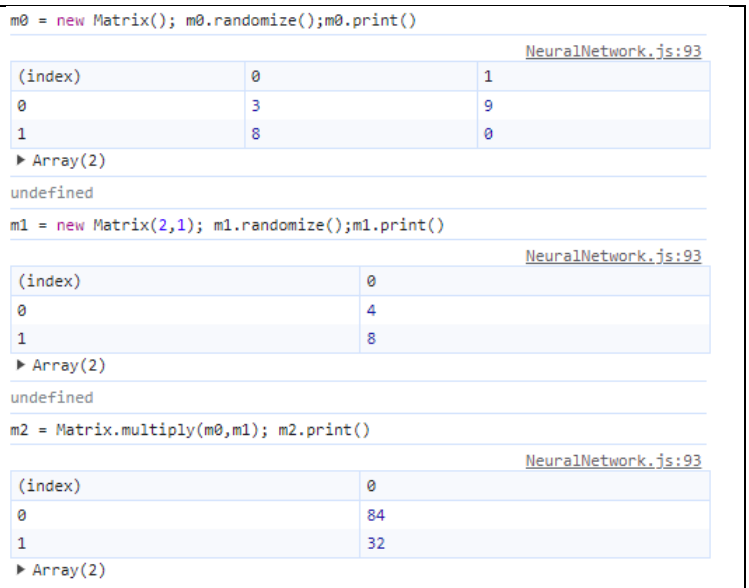
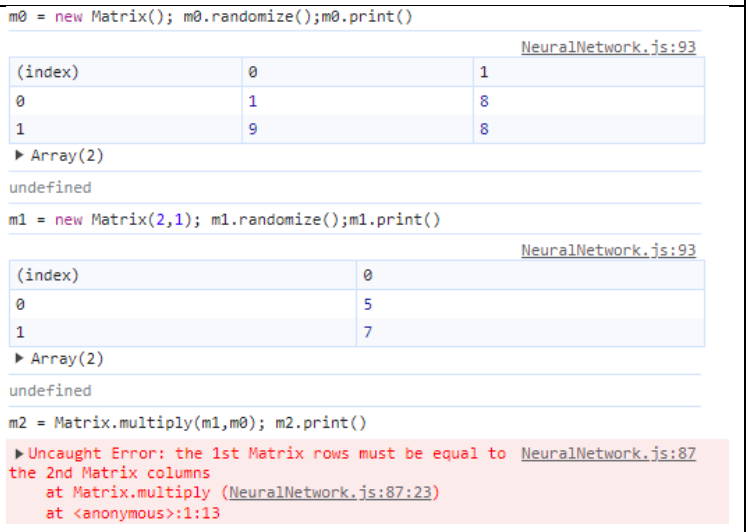
Errors and Fixes

Based on the test data, the method seems to function as intended which means that I will not have to make any modifications to improve it. If I have leftover time, I will improve the efficiency of this program in any way I can.

```
//Matrix multiplication
static multiply(m0,m1){
    //rows n cols must be equal
    if(m0.cols == m1.rows){
        var rows = m0.rows
        var cols = m1.cols
        var newMatrix = new Matrix(m0.rows,m1.cols)
        for(let i = 0; i < rows; i++){
             for(let j = 0; j < cols; j++){
                var sum = 0
                for(let k = 0; k < m0.cols; k++){
                    sum += m0.matrix[i][k] * m1.matrix[k][j]
                }
                newMatrix.matrix[i][j] = sum
            }
        }
        return newMatrix
    }else{
        throw new Error("the 1st Matrix rows must be equal to the 2nd Matrix columns")
    }
}
```

This method is used to multiply 2 matrices together. It takes in 2 parameters which again must be matrices. After that the first matrix's columns are compared with the second matrix's rows. If they are the same, then we start the multiplication process. If not, they an error is thrown to the console which currently has been written wrong but will be corrected in the updated version of the code. A new matrix is created with the first matrix's row amount and the second matrix's columns amount. The matrices are then multiplied, and the new matrix is constructed and returned to the user.

Type of Data	Data	Expected Result	Actual Result
Normal	2 2x2 matrices multiplied together	It will multiply the 2 matrices together and return the result 2x2 matrix	 I have created 2 2x2 matrices m0 and m1. I then used the multiply method to multiply the 2 matrices to create a new matrix m2 which worked as intended.

Normal	1 2x2 matrix and one 2x1 matrix multiplied together	It will multiply the 2 matrices together and return the result 2x1 matrix	 In this test, I created a 2x2 matrix m0 and a 2x1 matrix m1. I then multiplied them using the multiply method to create a new 2x1 matrix m2 which values are exactly as they should be.
Erroneous	1 2x1 matrix and one 2x2 matrix multiplied together	It will throw the error message “the 1st Matrix rows must be equal to the 2nd Matrix columns ”	 Here, I did the same thing as the previous test however I flipped the order of which the matrices are multiplied. This should throw an error as you cannot multiply a 2x1 and a 2x2 and in the result it threw an error just as intended.

Errors and Fixes

Based on the test data, the method seems to function as intended which means that I will not have to make any modifications to improve it. If I have leftover time, I will improve the efficiency of this program in any way I can.

```
    //add functions onto the matrix
    static AddFunc(m0,Func){
    for(let i = 0; i < m0.rows; i++){
        for(let j = 0; j < m0.cols; j++){
            m0.matrix[i][j] = Func(m0.matrix[i][j])
            }
        }
    }
}
```

This is the AddFunc method, A static method which allows functions to be used on the elements of the matrix. It takes 2 parameters, the first one being a matrix and the second one being a function. It then loops through the matrix applying the function to each element until all of them have been changed by it. This will be used to allow the sigmoid function to be used later within the neural network.

For the test data to make sure this function works, I have made a square function which will work as a "test function". It will never need to be used within the program itself as JavaScript has its own functions to square numbers which are more optimised and faster to use. Below is the function.

```
//square the numbers
function SquareNum(Num){
    return Num*Num
}
```

The function is self-explanatory. It takes in 1 float value as a parameter and returns the square output.

```
m0 = new Matrix(); m0.randomize();m0.print()
```

NeuralNetwork.js:93

(index)	0	1
0	7	8
1	4	3

▶ Array(2)

undefined

```
Matrix.AddFunc(m0,SquareNum);m0.print()
```

NeuralNetwork.js:93

(index)	0	1
0	49	64
1	16	9

▶ Array(2)

As shown in the image, I have applied the SquareNum function to m0 which has squared every element within the matrix.

```
//Sigmoid function
function Sigmoid(num){
    FinalNum = 1/(1+Math.pow(Math.E,-(num)))
    return FinalNum
}
```

This is the sigmoid function which will be used as an activation function for the neural network. It takes in a float value and outputs a value between 0 and 1

```
class NeuralNetwork{
    constructor(I,H,O){
        //get the neuron counts
        this.inputsCount = I
        this.HiddenCount = H
        this.OutputsCount = O
        //initalise the matrices
        this.weights_IH = new Matrix(this.HiddenCount,this.inputsCount)
        this.bias_H = new Matrix(this.HiddenCount,1)
        this.weights_HO = new Matrix(this.OutputsCount,this.HiddenCount)
        this.bias_O = new Matrix(this.OutputsCount,1)
        this.weights_IH.randomize()
        this.bias_H.randomize()
        this.weights_HO.randomize()
        this.bias_O.randomize()
    }
```

This is the neural network class which takes in 3 integer values: Input, Hidden and output. These all correlate to the number of nodes each layer will have for example, if we put in the values 3,5,2 we would create a neural network which takes in 3 inputs, has 5 hidden nodes, and produces 2 outputs. The neural network uses these parameters and creates the required matrices needed to compute a feedforward algorithm. It then randomizes each matrix to generate weights and biases. Due to the nature of neural networks, the weights and biases are usually between the values of –1 and 1 which means I will have to change my randomize method to make sure that it only produces values between that range. Below is the updated code:

```
    //randomizes the values of the matrix
    randomize(){
        for(let i = 0; i < this.rows; i++){
            for(let j = 0; j < this.cols; j++){
            this.matrix[i][j] = Math.random()*2-1
            }
        }
    }
```

In the updated version, I have changed the range from 1-10 to –1 to 1

```
m0 = new Matrix(); m0.randomize();m0.print()
```

NeuralNetwork.js:93

(index)	0	1
0	0.1970838652848781	0.034609630358098276
1	-0.39684979582639857	-0.19061453755594693

▶ Array(2)

The output can be seen here. Now every element of the array is between the values of –1 and 1

I intend for the inputs to be given in as an array of values as I stated in the design section so that means I will need to have a function which can turn an array into a column matrix (a nx1 matrix). Below is the created function.

```
function TurnArrayToMatrix(array){
    let matrix = new Matrix(array.length,1)
    for(let i = 0; i < matrix.rows; i++){
        for(let j = 0; j < matrix.cols; j++){
        matrix.matrix[i][j] = array[i]
        }
    }
    return matrix
}
```

This function takes in an array as a parameter and converts it into a new matrix with the number of rows being the same as the number of items within the array and the number of columns being 1. As I mentioned in my feedforward algorithm back in the design section. The new matrix will be used for the linear algebra later within the class.

```
a1 = [1,4,5,6,25,7,2,9]
```

▶ *(8) [1, 4, 5, 6, 25, 7, 2, 9]*

```
m0 = TurnArrayToMatrix(a1)
```

▶ *Matrix {rows: 8, cols: 1, matrix: Array(8)}*

```
m0.print()
```

NeuralNetwork.js:93

(index)	0
0	1
1	4
2	5
3	6
4	25
5	7
6	2
7	9

Here is an example of it working. We first created an array a1 with some values in. We then applied the function and assigned it to m0 which when we print it, it prints out a column matrix with the values from the matrix. As the function will never be used by the user, I will not need to add input validation as I will always make sure that an array is placed in the function.

Similarly, at the end of the feedforward algorithm, I will need to turn the outputs matrix into an array again and for that I will need to create a function which does the inverse of the previous function.

```
function TurnMatrixToArray(Matrix){
    let array = []
    for(let i = 0; i < Matrix.rows; i++){
        for(let j = 0; j < Matrix.cols; j++){
        array.push(Matrix.matrix[i][j])
        }
    }
    return array
}
```

The image above displays the function which takes in a matrix as a parameter and converts it into an array with the number of values being equal to the total elements within the array.

```
m0 = new Matrix(); m0.randomize();m0.print()
```

NeuralNetwork.js:93

(index)	0	1
0	-0.32580687319466994	-0.8118604970856125
1	-0.44729732639893305	0.4755329810593323

▶ Array(2)

undefined

```
a1 = TurnMatrixToArray(m0)
```

▶ (4) [-0.32580687319466994, -0.8118604970856125, -0.44729732639893305, 0.4755329810593323]

Here is an example of the function working as intended. I first created a matrix m0 with randomised values then I applied the function to the matrix and assigned it to a new array a1. The output where the values from that array have now been.

I now have all the tools required to create the feedforward algorithm which will be one of the main functions within the entire program. Below is the function.

```
    static FeedForward(brain,inputs){
        //turn the array of inputs into a matrix
        let Inputs = TurnArrayToMatrix(inputs)
        let Hidden = Matrix.multiply(brain.weights_IH,Inputs)
        Matrix.add(Hidden,brain.bias_H)
        Matrix.AddFunc(Hidden,Sigmoid)
        let outputs = Matrix.multiply(brain.weights_HO,Hidden)
        Matrix.add(outputs,brain.bias_O)
        Matrix.AddFunc(outputs,Sigmoid)
        return TurnMatrixToArray(outputs)

    }
```

This is a static method which takes in a neural network as 1 parameter and an array of inputs as another. First, it turns the inputs into a matrix which can then be multiplied along with the first set of weights to create the hidden layer matrix. The first bias is then also added to the hidden matrix and then it is run through an activation function which in this case is the sigmoid function. The process repeats using the hidden matrix to create the outputs matrix which at the end is turned into an array and returned to the user.

```
brain = new NeuralNetwork(5,3,2)
▶ NeuralNetwork {inputsCount: 5, HiddenCount: 3, OutputsCount: 2, weights_I
  H: Matrix, bias_H: Matrix, …}
inputs = [5,0.2,1,56,3]
▶ (5) [5, 0.2, 1, 56, 3]
outputs = NeuralNetwork.FeedForward(brain,inputs)
▶ (2) [0.5904671663734129, 0.12120556318350378]
```

This is of an example of the feedforward algorithm working. Here we created a brain with 5 inputs, 3 hidden nodes and 2 outputs. I then created an array of random inputs and finally applied the feedforward algorithm and assigned the outputs to an array called outputs. The output should be an array with length 2 and every value should be between 0 and 1. In our example., we got exactly that which means it works just as intended.

```
//lerp function
function lerp(a,b,t){
    return a+(b-a)*t
}
```

Another method I will add in my neural network will be called mutate. This will be used to slightly alter the neural network to have variety within the A.I. that play the game as I mentioned in my class diagram. One key function that will be needed to create this method will be an interpolation formula. The code is showed above. It takes 3 float values as parameters and using these values it finds a value in-between a and b based on the parameter t. I called it lerp which is short for linear interpolation.

```
static mutate(network,amount){
    for(let i = 0; i < network.weights_IH.rows; i++){
        for(let j = 0; j < network.weights_IH.cols; j++){
            network.weights_IH.matrix[i][j] = lerp(network.weights_IH.matrix[i][j],Math.random()*2-1,amount)
        }
    }
    for(let i = 0; i < network.bias_H.rows; i++){
        for(let j = 0; j < network.bias_H.cols; j++){
            network.bias_H.matrix[i][j] = lerp(network.bias_H.matrix[i][j],Math.random()*2-1,amount)
        }
    }
    for(let i = 0; i < network.weights_HO.rows; i++){
        for(let j = 0; j < network.weights_HO.cols; j++){
            network.weights_HO.matrix[i][j] = lerp(network.weights_HO.matrix[i][j],Math.random()*2-1,amount)
        }
    }
    for(let i = 0; i < network.bias_O.rows; i++){
        for(let j = 0; j < network.bias_O.cols; j++){
            network.bias_O.matrix[i][j] = lerp(network.bias_O.matrix[i][j],Math.random()*2-1,amount)
        }
    }
}
```

Once creating the lerp function, I finally created the mutate method. I again made this into a static function. The method takes in 2 parameters. The first parameter will be an instance of a neural network which will be mutated, and the second will be a float value between 0 and 1 which will decide the amount that the neural network is changed by. The function works by going through the network, using the lerp function to slightly alter the weight and bias values. This function will be used within my project at the end of each generation to add variation into the A.Is playing the game.

Creating Utils.js

The following code will mostly be inside the Utils.js file. This file will contain essential functions and objects that will be used throughout the program such as the button class.

```
class Button {
  constructor(x, y, width, height, color, text, textsize, textcolor, MouseOver, click) {
    this.x = x || Math.random() * 500;
    this.y = y || Math.random() * 500;
    this.width = width || 100;
    this.height = height || 40;
    this.color = color || "red";
    this.Ocolor = this.color
    this.text = text || "Button";
    this.textsize = textsize || 20;
    this.textcolor = textcolor || "black";
    this.textsize = String(this.textsize) + "px"
    this.MouseIsOver = false
    this.clicked = false
    this.MouseOver = MouseOver || function() { this.MouseIsOver ? this.color = "orange" : this.color = "blue" };
    this.click = click || function() { console.log("clicked") };
  }
```

This code displays the Button class. As mentioned in my class diagram back in the design phase, it would take in the following parameters: x, y, width, height, color, text, textsize, textcolor, MouseOver and click. If any of these parameters are not included, then class results to a default button object which will avoid the program breaking.

```
    pen.fillStyle = this.color;
    pen.fillRect(this.x, this.y, this.width, this.height);
    pen.font = this.textsize + " Arial";
    pen.fillStyle = this.textcolor;
    pen.textAlign = "center";
    pen.textBaseline = "middle";
    pen.fillText(this.text, this.x + this.width / 2, this.y + this.height / 2);
```

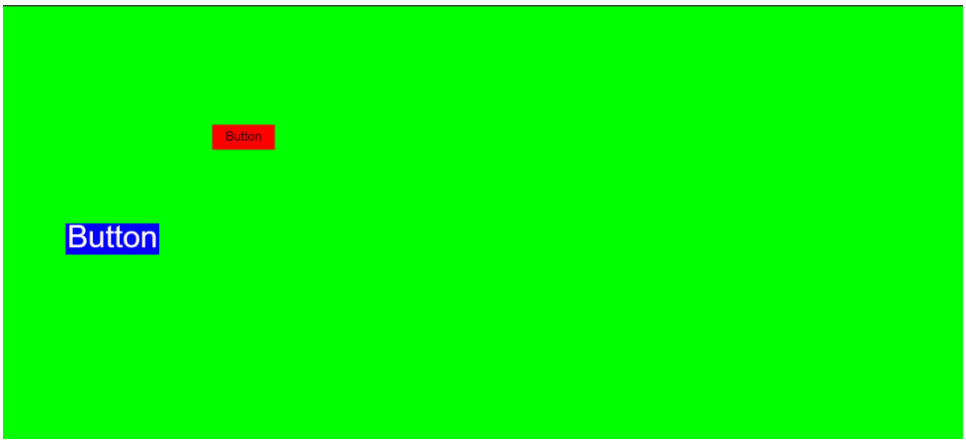
The first method for this class is the draw class. All it does right now is simply draw a rectangle with the parameters given. It also colours it in and writes the text in the centre of the button.

```
<script src="NeuralNetwork.js"></script>
<script src="Utils.js"></script>
<canvas id="box" width="500" height="500" style="border:0px solid ☐black"></canvas>
</canvas>
<style>
  body {
    margin: 0;
    overflow: hidden;
  }
</style>
```

To add the buttons onto the screen, I will need a canvas element. So, I added a canvas element using HTML and I also added the utils file into the test file so I can access the button class. I also removed the margin using simple CSS, so it did not appear at all during the test.

```
<script>
  var box = document.getElementById("box")
  box.width = innerWidth
  box.height = innerHeight
  pen = box.getContext("2d")
  pen.fillStyle = "rgba(0,255,0,1)"
  pen.fillRect(0, 0, box.width, box.height)
  button = new Button(100, box.height / 2, 150, 50, "blue", "Button", 50, "white")
  button2 = new Button()
  button.draw()
  button2.draw()
  let pos = {
    x: 0,
    y: 0
  }
  document.addEventListener("mousemove", function (e) {
    pos.x = e.clientX
    pos.y = e.clientY
  })
```

Here is the JavaScript code within this file. First, I get the canvas element and assign it to a variable called box. After, I made the canvas equal to the width and height of the screen the user will be using. I then created a new variable called pen which uses a method from the box object which allows us to draw onto the canvas. Once I had done this, I filled in the canvas with a green colour before creating 2 buttons. One button has the required attributes other than the mouseover and click parameters as I have not finished creating them yet. The second button has no attributes and will have the default button look. I then draw them to the screen. I then created a new object called pos which has 2 attributes: x and y. I then add an event listener to the document which tracks the position of the mouse coordinates and assigns them to the pos object. This will be used later to determine whether the mouse is over the button or not.



Here is the result of the code. As shown in the image above, we can see 2 buttons on the webpage. The blue one is the one with the parameters given whereas the red one is the one without the given parameters. I now need to add functionality to these buttons.

```
function InRange(value, min, max) {
  return value >= Math.min(min, max) && value <= Math.max(min, max);
}
function Interact(x, y, object) {
  return InRange(x, object.x, (object.x + object.width)) && InRange(y, object.y, (object.y + object.height))
}
```

The first thing I did was go back to the utils.js file and I created 2 functions. The first one is an InRange function which is one of the algorithms I mentioned within my design section. The function takes in 3 float parameters which are value, min, and max. All the function does is check to see if the value is in-between the min and max value and returns a Boolean value. Its main purpose is to be used within the Interact which is also displayed in the image. The interact function will be used to check if a point with parameters x and y are inside the object. Like the InRange function it returns a Boolean value for true or false.

```
document.addEventListener("click", function (e) {
  if (Interact(pos.x, pos.y, button)) {
    button.clicked = true
  }
})
function Loop() {
  pen.fillStyle = "rgba(0,255,0,1)"
  pen.fillRect(0, 0, box.width, box.height)
  button.draw()
  button2.draw()
  requestAnimationFrame(Loop)
  if (Interact(pos.x, pos.y, button)) {
    button.MouseIsOver = true
  } else {
    button.MouseIsOver = false
  }
}
Loop()
```

After making the interact function, I went back to main html file and updated the code. Now for every mouse click, the program checks to see if the mouse position was inside of the button. If it was, then we say the button was clicked. I have also created a menu loop which constantly colours the screen green and draws the buttons on the screen. It also checks every frame if the mouse position is over the button. If it is, then it triggers the mouseover event within the button class.

```
draw() {
  pen.fillStyle = this.color;
  pen.fillRect(this.x, this.y, this.width, this.height);
  pen.font = this.textsize + " Arial";
  pen.fillStyle = this.textcolor;
  pen.textAlign = "center";
  pen.textBaseline = "middle";
  pen.fillText(this.text, this.x + this.width / 2, this.y + this.height / 2);
  if (this.MouseIsOver) {
    this.MouseOver()
  }
  if (this.clicked) {
    this.click()
    this.clicked = false
  }
}
```

I went back to the button class and updated the draw method which now includes 2 if statements which determine whether the mouse is hovering over the button or if it has been clicked. I decided to activate the mouseover and click function within the draw method instead of making new methods like I said I would back in the design phase. I did this as I realised both ways would lead to the same result, however, this way would require less lines of code which will result in better performance in the long term of the program.

The result was this. With the button I added the attributes on, hovering over it changes the colour and changes the text from button to click? One problem I encountered with this was that after the mouse had been over the button, the state of the button would not change even if the mouse was no longer on the button. This was a change that needed to be fixed immediately.

```
//Original values
this.Ox = this.x
this.Oy = this.y
this.Owidth = this.width
this.Oheight = this.height
this.Ocolor = this.color
this.Otext = this.text
this.Otextsize = this.textsize
this.Otextcolor = this.textcolor
```

The solution I produced was adding original attributes. These attributes would be created instantly after the other attributes were created and would act as a form of storage to keep the original attributes of the button.

```
OriginState() {
  this.x = this.Ox
  this.y = this.Oy
  this.width = this.Owidth
  this.height = this.Oheight
  this.color = this.Ocolor
  this.text = this.Otext
  this.textsize = this.Otextsize
  this.textcolor = this.Otextcolor
}
```

I created a new method known as OriginState which when applied will convert the current state of the button back to its first state when it was first created. Using this, I can now check if the mouse is over the button. If it is, we apply the mouseover function, if not, we apply this function instead.

```
if (this.MouseIsOver) {
  this.MouseOver()
} else {
  this.OriginState()
}
if (this.clicked) {
  this.click()
  this.clicked = false
}
```

Here is the updated version of the code with the modifications I said I would implement. I have also added a line of code with the click condition which sets the clicked attribute to false after the click function has run. I added this to prevent the click function from constantly running when the button was clicked once.

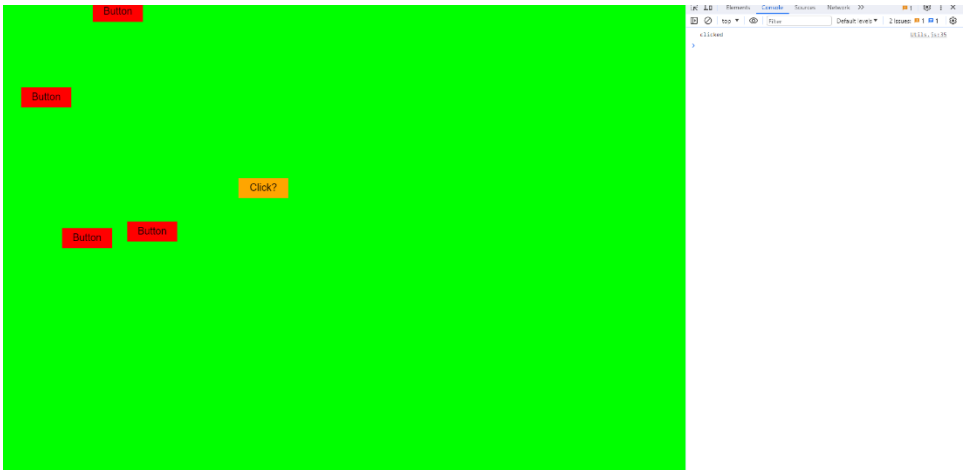
The result is the mouseover function working as intended. Now once you move the mouse off the button, it reverts to its original state.

```
  clicked                                          Utils.js:35
>
```

Here is the result of clicking the button. It runs the click function which in this instance, it outputs to the console the word “clicked”

```
var box = document.getElementById("box")
box.width = innerWidth
box.height = innerHeight
pen = box.getContext("2d")
pen.fillStyle = "rgba(0,255,0,1)"
pen.fillRect(0, 0, box.width, box.height)
let buttons = []
for (let i = 0; i < 5; i++) {
  buttons.push(new Button)
}
let pos = {
  x: 0,
  y: 0
}
document.addEventListener("mousemove", function (e) {
  pos.x = e.clientX
  pos.y = e.clientY
})
document.addEventListener("click", function (e) {
  buttons.forEach(button => {
    if (Interact(pos.x, pos.y, button)) {
      button.clicked = true
    }
  })
})
function Loop() {
  pen.fillStyle = "rgba(0,255,0,1)"
  pen.fillRect(0, 0, box.width, box.height)
  buttons.forEach(button => {
    button.draw()
    if (Interact(pos.x, pos.y, button)) {
      button.MouseIsOver = true
    } else {
      button.MouseIsOver = false
    }
  })
  requestAnimationFrame(Loop)
}
Loop()
```

Back in the main html file, I have removed the 2 buttons and have created a new array called buttons. This will be used to store all the buttons within the main html file of the program. I then created 5 new buttons as an example and rewrote to code to update all the buttons and to check if the mouse is hovering over any of the new buttons.

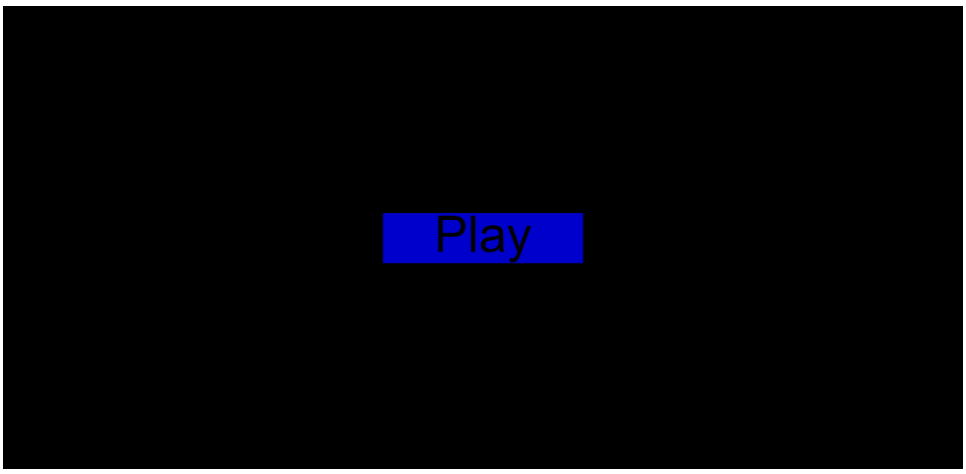


The result is exactly what I expected. 5 buttons were displayed onto the screen and when I place my mouse above one of the buttons, the mouseover event is triggered and when we click one button, the

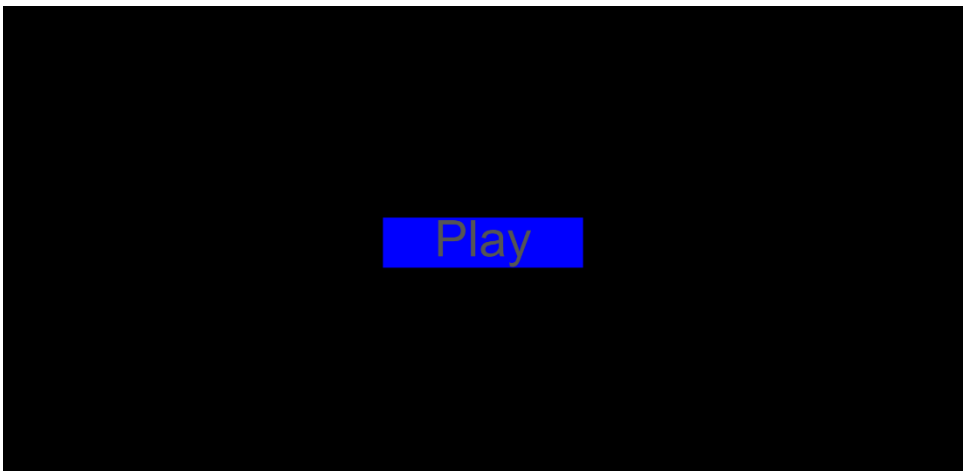
word “clicked” is outputted to the console. This was simply a test to determine whether the program could work if there were multiple buttons on the screen as in the completed version, there may be multiple buttons along with other functions going on at the same time.

Creating Main Menu.html

This will be the main menu of the game. Whenever the user first opens the program, this webpage will be the first to load. This will also be where the user will navigate from training the AI to playing against it.



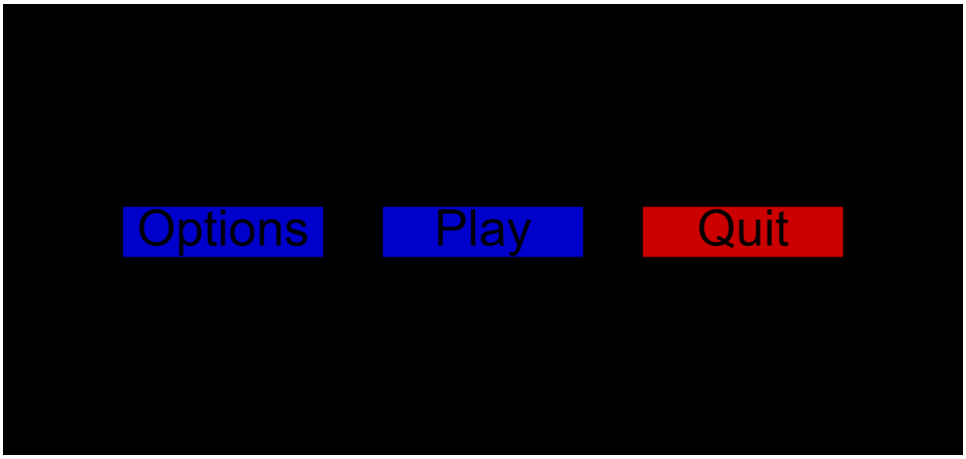
Now that I have completed the main essentials for this project, I can start creating the main menu. I first made the screen black and placed a button in the centre which displays “Play”.



Here is the same thing with the button displaying the mouseover event as the mouse was over the button.

```
buttons.push(new Button(
  (box.width / 2) - 200,//X
  (box.height / 2) - 50,//Y
  400,//Width
  100,//Height
  "#0000CC",//Color
  "Play",//text
  100,//text size
  "black",//text color
  ()=>{
      this.color="#0000FF";
      this.textcolor="#555555"
  }//function when mouse is over button
  ,()=>{
      console.log("Play")
  }//function when mouse clicks button
  ))
```

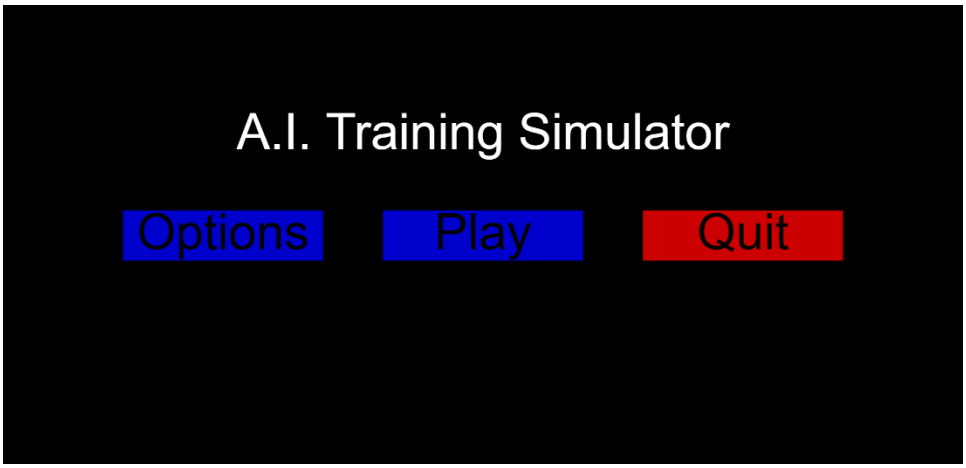
This displays the code of the button. I have used comments to label each parameter so anyone viewing the code and read exactly what parameters are being inputted. From now on, I will be sure to write all my buttons like this for readability.



I have added a few more buttons which are options and quit. Options is the same as the Play button but with different text whereas the quit button is a red colour.

```
//Play Button
buttons.push(new Button(
  (box.width / 2) - 200,//X
  (box.height / 2) - 50,//Y
  400,//Width
  100,//Height
  "#0000CC",//Color
  "Play",//text
  100,//text size
  "black",//text color
  function () {
    this.color = "#0000FF";
    this.textcolor = "#555555";
  },//function when mouse is over button
  () => {
    console.log("Play")
  }//function when mouse clicks button
))
//Options Button
buttons.push(new Button(
  (box.width / 3) - 400,
  (box.height / 2) - 50,
  400,
  100,
  "#0000CC",
  "Options",
  100,
  "black",
  function () {
    this.color = "#0000FF";
    this.textcolor = "#555555";
  },
  () => {
    console.log("Options")
  }
))
//quit Button
buttons.push(new Button(
  (2 * box.width / 3),
  (box.height / 2) - 50,
  400,
  100,
  "#CC0000",
  "Quit",
  100,
  "black",
  function () {
    this.color = "#FF0000";
    this.textcolor = "#555555";
  },
  () => {
     //closes the window
    window.close()
  }
```

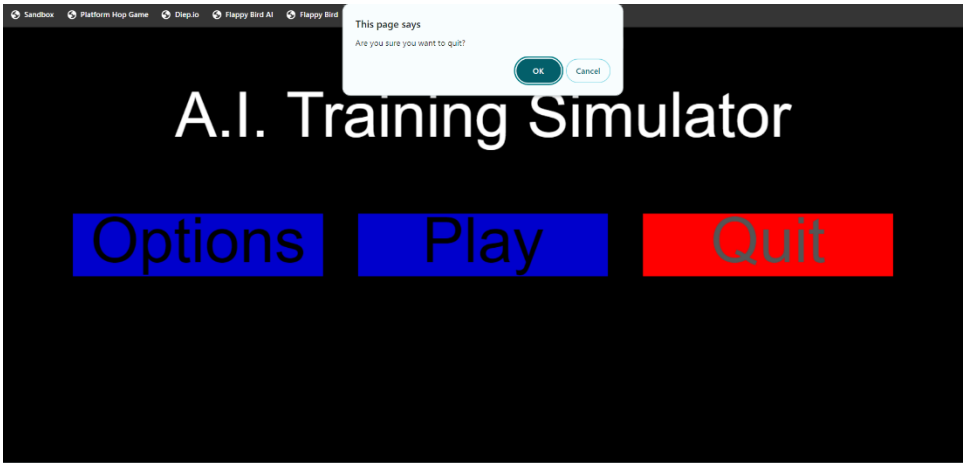
Here is the code for all 3 buttons. As shown the quit button will close the window when the button is clicked however, I may change this later within the program.



Finally, I added the title to the menu and now the main menu is complete. The only feature of the main menu that I have not added is the gameplay in the background. I will be sure to add this once I have one of the A.I games working.

```
() => {
  if(confirm("Are you sure you want to quit?")){
    window.close()
  }
}//function when mouse clicks button
```

I ended up changing the function on the quit button. Now instead of quitting the game straight away, the user will be shown a popup which asks to confirm if they want to quit. This is to prevent accidental closes and to double check if the user really wants to quit.



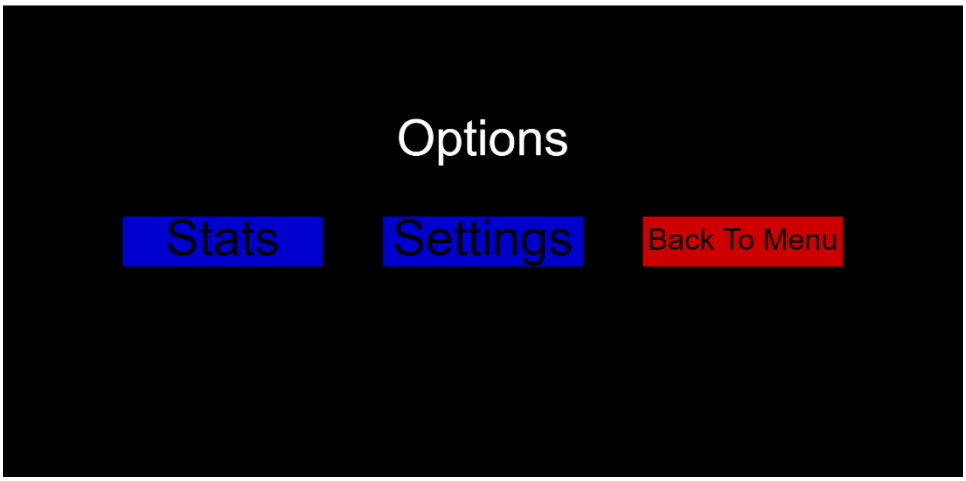
Here is the result. Now when the user presses the quit button, the popup will show up which allows the user to confirm if they really want to quit. If the user presses the ok button, then the window will close.

```
//change menu
function LoadMenu(Menu){
  buttons = []
  switch (Menu) {
    case "Options":···
    case "Play":···
      case "Train":···
      default:···
  }
}
LoadMenu()
```

To make the menu functional, I created a function which would control how the menus would load. It takes a single string parameter which will then be used in a switch statement as shown in the code. Before using the switch statement, I clear the buttons array, removing all the buttons from the screen before adding the new ones in. This is to prevent multiple buttons from overlapping each other and to make sure that I am not constantly adding new buttons to the array which could take up memory and waste time. I then use a switch statement to then decide what menu I will need to load. If no parameter is given, then it loads the main menu shown earlier.

```
case "Options":
  MenuTitle = "Options"
    //Settings Button
  buttons.push(new Button(
    (box.width / 2) - 200,//X
    (box.height / 2) - 50,//Y
    400,//Width
    100,//Height
    "#0000CC",//Color
    "Settings",//text
    100,//text size
    "black",//text color
    function () {
      this.color = "#0000FF";
      this.textcolor = "#555555";
    },//function when mouse is over button
    () => {
      console.log("Settings")
    }//function when mouse clicks button
  ))
  //Stats Button
  buttons.push(new Button(
    (box.width / 3) - 400,//X
    (box.height / 2) - 50,//Y
    400,//Width
    100,//Height
    "#0000CC",//Color
    "Stats",//text
    100,//text size
    "black",//text color
    function () {
      this.color = "#0000FF";
      this.textcolor = "#555555";
    },//function when mouse is over button
    () => {
      console.log("Stats")
    }//function when mouse clicks button
  ))
  //Back To Menu Button
  buttons.push(new Button(
    (2 * box.width / 3),//X
    (box.height / 2) - 50,//Y
    400,//Width
    100,//Height
    "#CC0000",//Color
    "Back To Menu",//text
    60,//text size
    "black",//text color
    function () {
      this.color = "#FF0000";
      this.textcolor = "#555555";
    },//function when mouse is over button
    () => {
      LoadMenu()
    }//function when mouse clicks button
  ))
```

Here is an example of one of the case statements. If the parameter is “Options” then we first change the title to Options and then we add the following buttons: Stats, Settings and Back to menu. The code is mostly just a copied version of the main menu with a few values changed.



Here is the output. Like in the main menu, the only missing feature is the animated background, but I will start working on that once I have implemented one game into my program.

```
case "Train":
MenuTitle = "Training"
  //Flappy Bird Game
buttons.push(new Button(
  (box.width / 2) - 200,//X
  (box.height / 2) - 50,//Y
  400,//Width
  100,//Height
  "#0000CC",//Color
  "Flappy Bird",//text
  70,//text size
  "black",//text color
  function () {
    this.color = "#0000FF";
    this.textcolor = "#555555";
  },//function when mouse is over button
  () => {
    window.open("Flappy Bird AI.html")
  }//function when mouse clicks button
))
//Game 1
buttons.push(new Button(
  (box.width / 3) - 400,//X
  (box.height / 2) - 50,//Y
  400,//Width
  100,//Height
  "#0000CC",//Color
  "Training 1",//text
  80,//text size
  "black",//text color
  function () {
    this.color = "#0000FF";
    this.textcolor = "#555555";
  },//function when mouse is over button
  () => {
    console.log("Train 1")
  }//function when mouse clicks button
))
// game 3
buttons.push(new Button(
  (2 * box.width / 3),//X
  (box.height / 2) - 50,//Y
  400,//Width
  100,//Height
  "#0000CC",//Color
  "Training 3 ",//text
  60,//text size
  "black",//text color
  function () {
    this.color = "#0000FF";
    this.textcolor = "#555555";
  },//function when mouse is over button
  () => {
    console.log("Train 3")
  }//function when mouse clicks button
))
```

Whereas most of the menus are similar in terms of format, the train menu is one of the first menus to be slightly different from the original layout as I mentioned in the design section of the project. It will have an extra button in the bottom corner which will send the user back to the play menu if pressed. In the image, it shows the 3 main buttons which each will be training games the user can enter when the button is clicked. Currently, the only button that works is the flappy bird button which will take the user to the training game which in the files is just called “Flappy Bird AI.html”

```
//Back to Play button
buttons.push(new Button(
  (box.width)-160,//X
  (box.height) - 50,//Y
  140,//Width
  40,//Height
  "#CC0000",//Color
  "Back To Menu",//text
  20,//text size
  "black",//text color
  function () {
    this.color = "#FF0000";
    this.textcolor = "#555555";
  },//function when mouse is over button
  () => {
    LoadMenu("Play")
  }//function when mouse clicks button
))
break;
```

This code shows the smaller button in the corner which will take you back to the play menu if pressed.

This image shows the result of the training menu with all the new buttons.

```
//Description text
pen.font = "20px Arial";
pen.fillStyle = "white";
pen.textAlign = "center";
pen.textBaseline = "middle";
pen.fillText(TextDescription, box.width / 2, (box.height - 50));
```

One feature I mentioned in my design section was the text that relates to the game. This feature would work by hovering over a button and when that happened, there would be text that appears at the bottom of the screen informing the user about the game. I created a new variable called TextDescription which will be used to display the information about the games. In this image, I used the text attributes and methods from the pen object to display the text at the bottom of the screen.

```
//button logic
let InteractedButtons = 0
buttons.forEach(button => {
  button.draw()
  if (Interact(pos.x, pos.y, button)) {
    button.MouseIsOver = true
    InteractedButtons += 1
  } else {
    button.MouseIsOver = false
  }
})
if (InteractedButtons < 1) {
  TextDescription = " "
}
```

I improved the button logic by adding a variable called “interactedButtons” which will keep track of the current number of buttons the user is interacting with on one frame. It works by going through each button in the list and checking to see if the mouse is hovering over it. If it is then the InteractedButtons variable is incremented by 1. At the end of the loop, the program checks to see if the number of InteractedButtons is smaller than 1. If it is then it must mean that the user is not hovering over any buttons meaning that the text description should not have any text. As a result, the text description will display nothing.

```
function () {
  this.color = "#0000FF";
  this.textcolor = "#555555";
  TextDescription = "Teach AI to play the classic Flappy Bird game"
},//function when mouse is over button
```

I updated the mouse over code of the flappy bird button. It will now display the text “Teach AI to play the classic flappy bird game” when the user hovers over the button using the mouse.

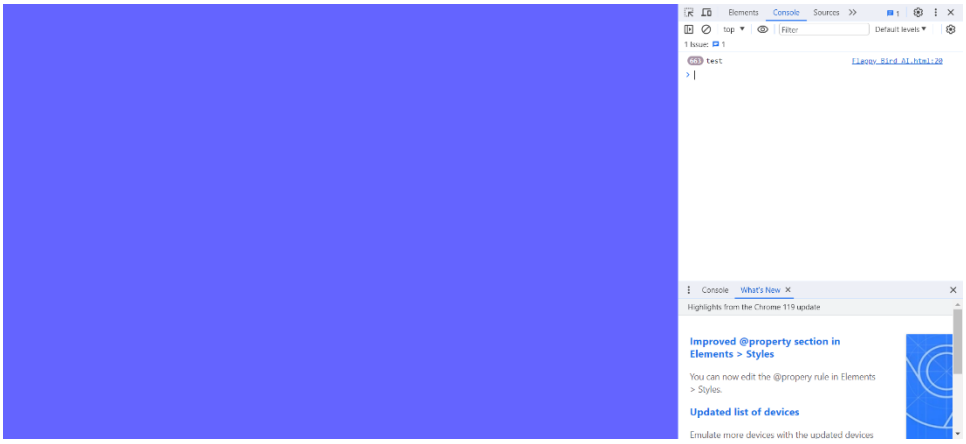
Here is the result of the implemented code. Now when I hover the mouse over the flappy bird button, the text description updates with the following message. It also disappears when I move my mouse off the button.

Creating Flappy Bird AI.html

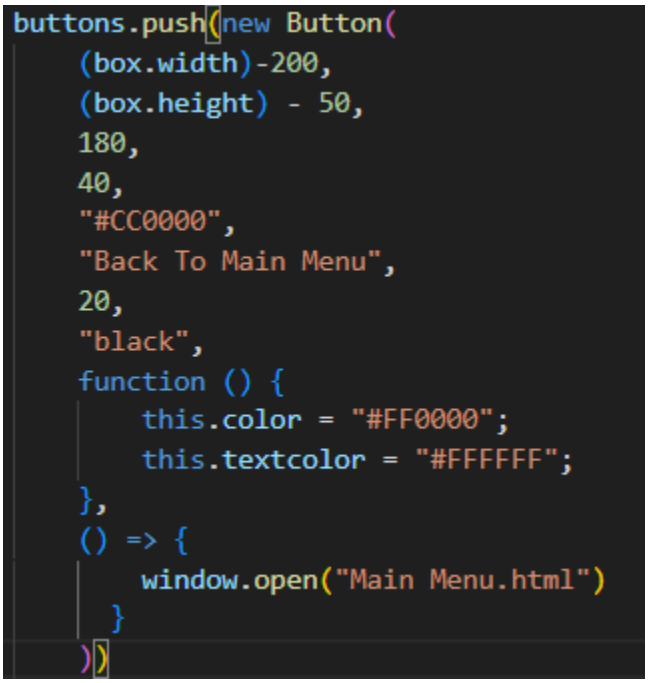
The code in this section will mostly be about the flappy bird AI game. This is where the user can train the AI needed for the flappy bird game.

```
<title>Flappy Bird AI</title>
<script src="NeuralNetwork.js"></script>
<script src="Utils.js"></script>
<canvas id="box" width="500" height="500" style="border:0px solid black"></canvas>
</canvas>
<style>
  body {
    margin: 0;
    overflow: hidden;
  }
</style>
<script>
    var box = document.getElementById("box")
    box.width = innerWidth
    box.height = innerHeight
    var pen = box.getContext("2d")
    function GameLoop(){
        pen.fillStyle = "rgba(100,100,255,1)"
        pen.fillRect(0,0,box.width,box.height)
        console.log("test")
        requestAnimationFrame(GameLoop)
    }
    GameLoop()
</script>
```

I first started by linking the files needed which are the “NeuralNetwork.js” file and the “Utils.js” file. I also added a canvas element, like the main menu part of my project. I then created a function called “GameLoop” which will contain all the game logic such as collision detection and drawing. Here, I filed the background in a blue colour, and I placed a “requestAnimationFrame” line at the end of function which will tell the function to call itself again in a recursive way. To make sure that this code worked, I placed a console log which will output “test” in the console. I then called the function at the end.



As shown in the image, the code works just like I expected to which means I can move on.



```
buttons.push(new Button(
    (box.width)-200,
    (box.height) - 50,
    180,
    40,
    "#CC0000",
    "Back To Main Menu",
    20,
    "black",
    function () {
        this.color = "#FF0000";
        this.textcolor = "#FFFFFF";
    },
    () => {
        window.open("Main Menu.html")
    }
))
```

I added a new button which will sit at the bottom corner of the screen which will take the user back to the main menu if clicked. When I implemented this button, I realised that I would have to rewrite the button logic code in this file if I wanted to interact with this button. I then produced an idea which would involve moving the button logic code from the main menu file to the utils file so I could use it within all my future files without needing to rewrite any code. If this idea worked, it could save lots of time in the development process.

```
//Function for button Interaction
function ButtonInteraction(pos,Arrays,IsMainMenu){
  document.addEventListener("mousemove", function (e) {
    pos.x = e.clientX
    pos.y = e.clientY
  })
  document.addEventListener("click", function (e) {
    Arrays.forEach(array => {
      if (Interact(pos.x, pos.y, array)) {
        array.clicked = true
      }
    })
  })
  Arrays.forEach(array => {
    array.draw()
    if (Interact(pos.x, pos.y, array)) {
      array.MouseIsOver = true
      if(IsMainMenu){
        InteractedButtons += 1
      }
    }else{
      array.MouseIsOver = false
    }
  })
}
```

In the Utils.js file I created a function called ButtonInteraction which takes in 3 parameters. The first parameter is the pos object which contains an x and y value. The second parameter is called arrays and will be used to get the buttons within the given file the function is in. The final parameter is a Boolean value which will be used to see if the function is being used within the main menu file. The function is a copied version of the button logic part of the main menu file with a few changes to make it suitable for all files.

```
//button logic
InteractedButtons = 0
ButtonInteraction(Pos,buttons,true)
if (InteractedButtons < 1) {
  TextDescription = " "
}
```

As a result of this new function, I had to update the button logic code within my main menu file. All I did was remove the code and replace it with the new function I had created. Now the button logic is a shorter and this will help with the efficiency of the program.

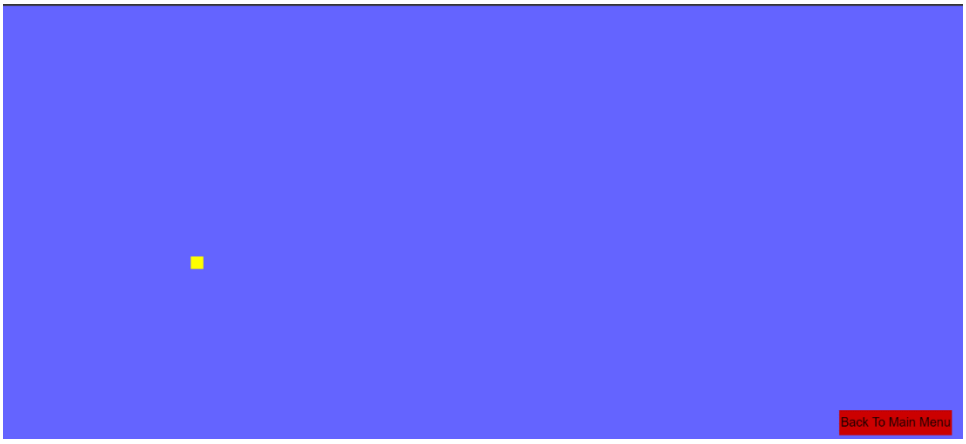
```
function GameLoop(){
    pen.fillStyle = "rgba(100,100,255,1)"
    pen.fillRect(0,0,box.width,box.height)
    ButtonInteraction(pos,buttons,false)
    requestAnimationFrame(GameLoop)
}
```

Back to the flappy bird file, I have updated the GameLoop code by adding the buttonInteraction function inside so I can interact with buttons within this file.

Here is the result, The button is now displayed, and it can be interacted with normally like in the main menu.

```
//bird class
class Bird{
    constructor(){
        this.x = 300
        this.y = 0
        this.width = 20
        this.height = 20
        this.color = "yellow"
        this.y_speed = 0
        this.UpForce = 15
        this.score = 0
        this.isDead = false
    }
    draw(){
        pen.fillStyle = this.color
        pen.fillRect(this.x,this.y,this.width,this.height)
    }
    update(){
        this.y_speed+= gravity
        this.y_speed *=0.9
        this.y += this.y_speed
    }
}
```

After adding the button to work, I created the bird class. This will be used as the AI in this game. As I mentioned in the class diagrams, the bird has and x and y position, width and height, color and a score attribute to keep score. Currently, I have added 2 methods to this class which are the draw and update. The draw method draws the bird onto the screen and the update method updates the bird's position based on the amount of gravity.



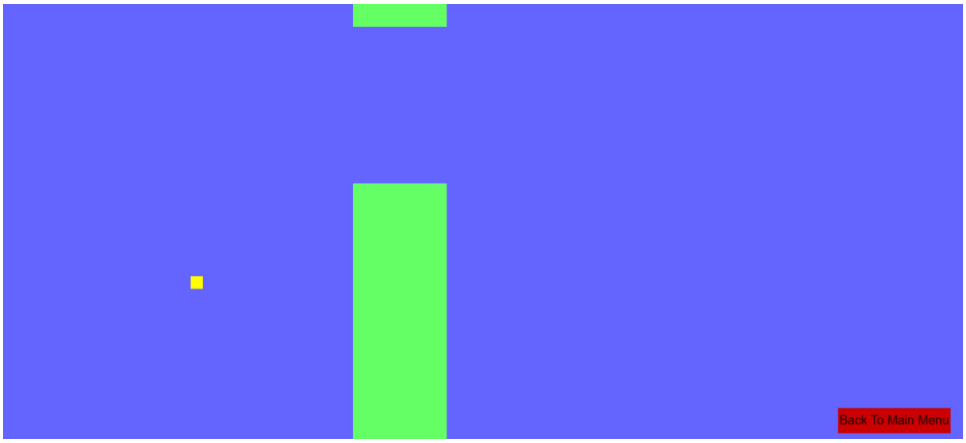
Here is the output. I created a new Bird and assigned it to a variable called testbird. The yellow square represents the bird, and it falls due to the gravity.

```
document.addEventListener("click",()=>{
    testBird.flap()
})
```

I created the flap method which will allow the bird to jump up in the air. It will work by adding the JumpForce attribute to the y speed. This will be the way the bird moves as in the original flappy game it can only bounce up. After adding this, I added an Event Listener. An Event listener is a function in JavaScript which waits for any event to do something. In this case, every time I click the screen, the bird should bounce up. I implemented this and tested it, and it worked just as it should. Now I will be working on the pipes of this game.

```
//Pipe class
class Pipe{
    constructor(){
        this.x = box.width
        this.HasCollectedScore = false
        this.top_height = Math.floor(Math.random() * ((box.height/2) - (box.height/35)+1) + (box.height/35))
        this.top_y = 300
        this.spacing = 250
        this.bottom_y = this.top_height+this.spacing
        this.width = 150
        this.bottom_height = box.height-(this.top_height+this.spacing)
        this.color = "rgb(100,255,100)"
    }
    draw(){
        pen.fillStyle = this.color
        pen.fillRect(this.x,0,this.width,this.top_height)
        pen.fillRect(this.x,this.bottom_y,this.width,this.bottom_height)
    }
    update(){
        this.x -= 1
    }
}
```

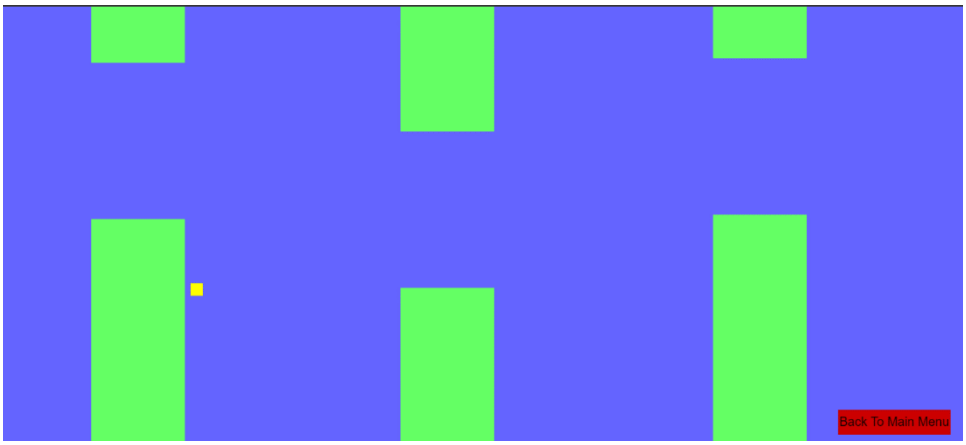
I created the pipe class which has the same 2 methods as the bird class. As I mentioned in the class diagrams, the pipe class has an x position, width and 2 different heights and y position for the top pipe and bottom pipe. The update method currently moves the pipe left by 1, however, I will change this so that the user can choose the speed of the pipe when training the AI.



Here is the result. I have now managed to add a pipe into the game. Now I will need to implement a spawning mechanic which will spawn pipes constantly.

```
let FrameCount = 0
function GameLoop(){
    //updates
    FrameCount++
    testBird.update()
    for(let i = 0; i < pipes.length; i++){
        pipes[i].update()
        if(pipes[i].x < -pipes[i].width){
            pipes.splice(i,1)
        }
    }
    if(FrameCount % 100 == 0){
        pipes.push(new Pipe)
    }
    //drawing
    pen.fillStyle = "rgba(100,100,255,1)"
    pen.fillRect(0,0,box.width,box.height)
    testBird.draw()
    pipes.forEach(pipe=>{
        pipe.draw()
    })
    ButtonInteraction(pos,buttons,false)
    requestAnimationFrame(GameLoop)
}
```

This is the new GameLoop function. As you can see above the gameLoop function I have added a variable known as frameCount. The value initially starts at zero, but it increments by 1 every frame. During this part of the game loop, I update the bird and loop through the pipes array, updating each pipe and deleting them once they are offscreen. To spawn pipes, I preform the modulo operation. The Modulo operation is an operation to find out the remainder after dividing one number by another. In this case, I check to see if the modulo of the frame count by 100. If it is equal to zero. A new pipe is added into the array. After updating all the code, I start drawing everything to the screen. This includes the background, the pipes, and the bird. Finally, I update all the buttons within the program before using the requestAnimationFrame function to call the function again.



The result is shown in the image here. Multiple pipes are spawning with varying positions which is just what will be needed to complete the game. By this point, the flappy bird game is complete and all I must do is add collision detection and add the AI.

```
//collision detection with birds and pipe
pipes.forEach(pipe=>{
    //touches the top pipe
    if(birds[i].x+birds[i].width >= pipe.x && birds[i].x <= pipe.x+pipe.width && 0 <= birds[i].y+birds[i].height && birds[i].y <= pipe.top_height){
        birds[i].isDead = true
    }
    //touching the bottom pipe
    else if(birds[i].x+birds[i].width >= pipe.x && birds[i].x <= pipe.x+pipe.width && pipe.bottom_y <= birds[i].y+birds[i].height&& birds[i].y <= pipe.bottom_y+pipe.bottom_height){
        birds[i].isDead = true
    }
})
```

The first thing I did was remove the testbird variable and replaced it with an array which will contain all the birds. After that, I created the collision detection. All this does is check whether any bird is colliding with any of the pipes and if they are, the “IsDead” attribute on the bird will be turned to true.

```
//check if the birds are out of bounds
if(birds[i].y < 0 || birds[i].y > box.height){
    birds[i].isDead = true
}
```

Another way the birds can die is by going up too high or too low. If this is the case, this code will also turn the “IsDead” attribute to true as well.

```
//checks if a bird is dead
birds.forEach((bird,i)=>{
    if(bird.isDead){
        birds.splice(i,1)
    }
})
```

Now in this part of the gameloop, we check to see if a bird’s “IsDead” attribute is set to true. If it is, that means that the bird must have died, and it will be removed from the birds array.

```
//check if all the birds are dead
if(birds.length < 1){
    restart()
}
```

This if statement checks to see if the birds array is empty. If it is, that must mean that all the birds have died and that means we should run a new generation of birds. So, if that is the case, the restart function is called.

```
function restart(){
    pipes = []
    FrameCount = 0
    for(let i = 0; i < population; i++){
        birds.push(new Bird)
    }
}
```

Here is the restart function. As I mentioned in the design portion of this report, this will be a crucial algorithm that will be implemented within all the training games. Right now, in this unfinished version, all it does is clear the pipes array and set the frame count back to zero. After that it refills the birds array with new birds using a new population variable, I have created which will be used to determine the amount of AI's the user trains.

```
constructor(){
    this.x = 300
    this.y = 0
    this.width = 20
    this.height = 20
    this.color = `rgba(${Math.floor(Math.random() * (255 - 0 + 1) + 0)},${Math.floor(Math.random() * (255 - 0 + 1) + 0)},${Math.floor(Math.random() * (255 - 0 + 1) + 0)},1)`
    this.y_speed = 0
    this.UpForce = 15
    this.score = 0
    this.inputs = []
    this.brain = new NeuralNetwork(8,5,1)
    this.isDead = false
}
```

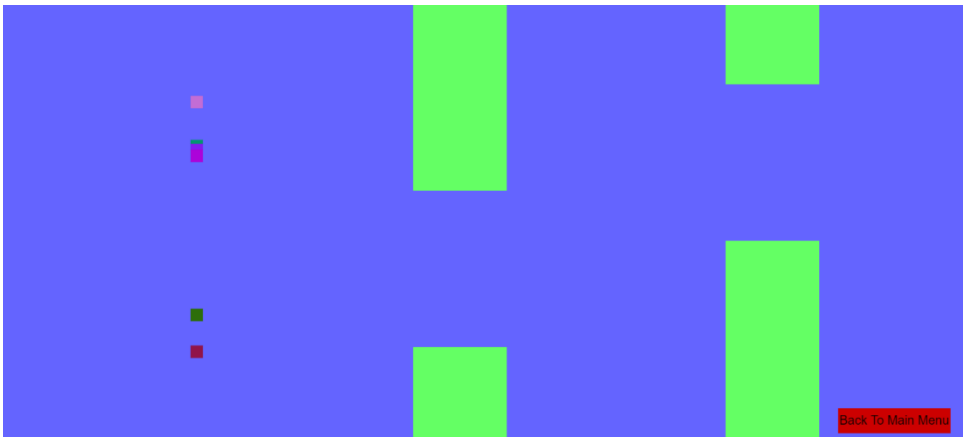
Due to there now being multiple birds, I have updated the attributes of the bird class by first changing the color so now it is randomised upon creation. I have also added 2 new attributes. One being the inputs and one being the brain. The inputs attribute is an array which will hold all the inputs that the AI will use for the feedForward algorithm. The brain attribute will be where the Neural Network will be stored within the class. Currently I have given it 8 inputs and 1 output with 5 hidden layers. This may change throughout the prototyping process of this game.

```
think(pipes){
    let closest = pipes[0]
    let closestD = Infinity
    for(let i = 0; i < pipes.length; i++){
        let d = pipes[i].x - this.x
        if(d < closestD && d > -closest.width){
            closest = pipes[i]
            closestD = d
        }
    }
    this.inputs[0] = this.y
    this.inputs[1] = this.y_speed
    this.inputs[2] = (this.y-closest.top_height)
    this.inputs[3] = (this.y-closest.bottom_y)
    this.inputs[4] = (this.x-closest.x)
    this.inputs[5] = (this.x-closest.x+closest.width)
    this.inputs[6] = this.UpForce
    this.inputs[7] = closest.spacing

}
update(){
    this.think(pipes)
    this.y_speed+= gravity
    this.y_speed *=0.9
    this.y += this.y_speed
    let outputs = NeuralNetwork.FeedForward(this.brain,this.inputs)
    if(outputs[0] > 0.5){
        this.flap()
    }
}
```

As well as updating the attributes of the bird class, I have finally added the “think” method which takes in the pipes array as a parameter. As mentioned in the design section, this will be used to update the inputs for the AI. Within the think method, I loop through the pipes array to try find the closest pipe towards the player. Once the closest pipe has been found, I start giving the inputs to the AI. The basic inputs I have given it are as follows:

- Its y position
- Its speed in the y axis
- The vertical distance between the pipes
- The horizontal distance between the pipes
- The UpForce (how high it bounce up)
- And the spacing of the closest pipe

These are the initial inputs I will give the AI but based on how the performance of the AIs go, I may change or add more inputs to it. Within the update method, I have now incorporated the think method and the Neural Network feedForward algorithm I made. It also checks the value of the outputs and if it is larger than 0.5, the AI will activate the flap method to jump up. Now that the AI has a way of moving on its own, I removed the code which allowed for me to control the bird using the mouse.



This image shows the result. As shown, now all the AIs have their own unique color and neural network which allows them to play the game. Now I need to add an algorithm which selects the best AI out of each generation of AIs. This will be how the AI trains and improves over time to play flappy bird.

```
function restart(){
    //next generation
    generation++
    //reset everything
    pipes = []
    FrameCount = 0
    //determine the best bird
    let BestBird = SavedBirds[0]
    let CurrentBestScore = SavedBirds[0].score
    //get the best bird
    for(let i = 0; i < SavedBirds.length; i++){
        if(SavedBirds[i].score > BestBird.score){
            BestBird = SavedBirds[i]
        }
    }
    //save the brain
        localStorage.setItem("bestFlappyBrain",JSON.stringify(BestBird.brain))
        for(let i = 0; i < population; i++){
            birds.push(new Bird)
        }
        for(let i = 0; i < birds.length; i++){
            birds[i].brain = JSON.parse(localStorage.getItem("bestFlappyBrain"))
            NeuralNetwork.mutate(birds[i].brain,0.25)
        }
        localStorage.removeItem("bestFlappyBrain")
        SavedBirds = []
}
```

I have updated the restart algorithm, and this is the result. When the restart algorithm is first called, the first thing it does is increment the generation by 1. The generation variable was a new variable created to store the current generation of the AI and it will be displayed in the top left corner of the screen. After that, it resets the pipes and frame count, just like before. Now it creates 2 new temporary variables which are the BestBird and CurrentBestScore. By default, they are given the value of the first saved bird. The savedBirds array is a new array created which will store all the dead birds in that generation. The algorithm now starts by going through the saved birds and comparing their score with the current best score. If the bird's score is larger than the current best, that bird becomes the new best

bird, and its score becomes the new current best score. The way the birds gain a higher score is by surviving for a long time. In the update method of the bird class, I made it, so the bird gains score every frame it is alive so the longer it is alive, the more score it gets and the better it is. After the for loop has looped through every bird in the list, I turn the best bird's brain into JSON save it into local storage with the name "bestFlappyBrain". I then repopulate the birds array with new birds and give them the Neural Network from local storage. To add variety, I make use of the "mutate" function within the NeuralNetwork.js file which slightly changes the values of the weights and biases of the Neural Network. Some will improve and some will get worse due to this change. After adding all the birds with the new neural network, I clear the SavedBirds array, and the new generation starts.

```
//checks if a bird is dead
birds.forEach((bird,i)=>{
    if(bird.isDead){
        SavedBirds.push(bird)
        birds.splice(i,1)
    }
})
```

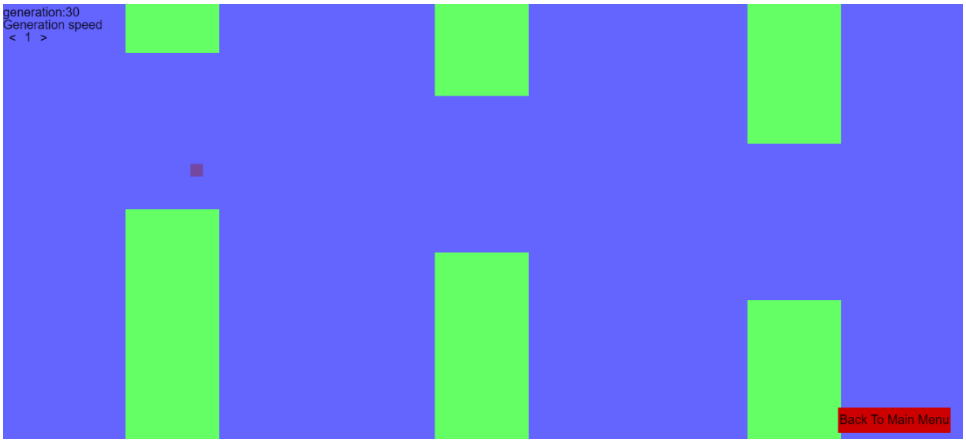
Here is the new code to check for dead birds. Now instead of just removing the bird from the array, I put the bird into the savedBirds array before removing it to save its information so it can be used for the restart algorithm.

```
//generation speed buttons
buttons.push(new Button(
0.1,
40,
30,
30,
"rgba(0,0,0,0)",
"<",
20,
"black",
function () {

    this.textcolor = "#FFFFFF";
},
() => {
    cycles--
  }
))
buttons.push(new Button(
50,
40,
30,
30,
"rgba(0,0,0,0)",
">",
20,
"black",
function () {

    this.textcolor = "#FFFFFF";
},
() => {
    cycles++
  }
))
```

One issue that I knew would be a problem is the waiting for the AI to train. To combat this issue, I have added the ability to speed up the training process by adding 2 new buttons which determine the cycle speed of the training process. The “cycle speed” means speeding up the updating process within the game loop and these 2 buttons will help speed up or slow down the process.

```
//draw and update the buttons
ButtonInteraction(pos,buttons,false)
pen.fillStyle = "black"
pen.textAlign = "start"
pen.textBaseline = "alphabetic"
pen.font= "20px Arial"
pen.fillText("generation:"+generation,0,20)
pen.fillText("Generation speed",0,40)
pen.textBaseline = "alphabetic"
pen.font= "20px Arial"
pen.textAlign = "center"
pen.fillText(cycles,40,60)
requestAnimationFrame(GameLoop)
```

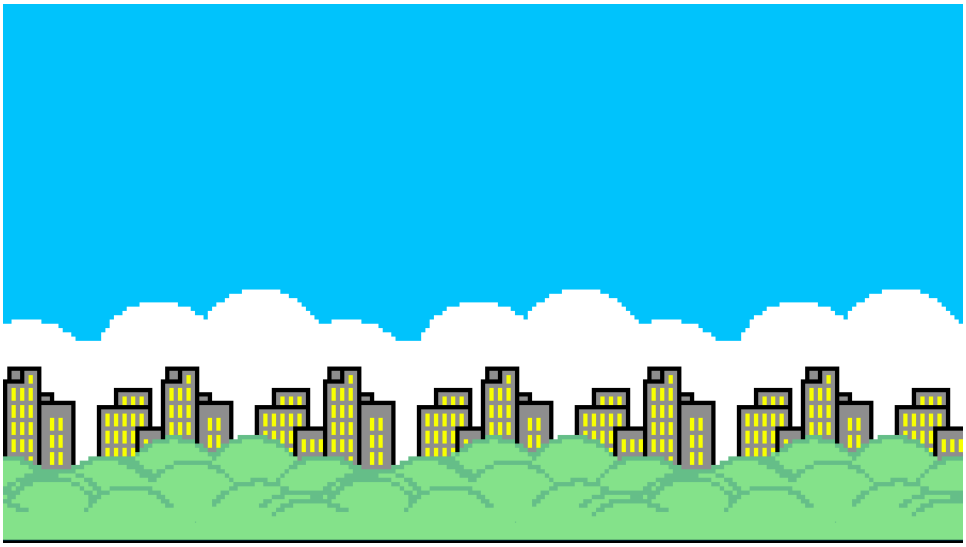
The new drawing code now will also write information onto the screen such as the generation and the generation speed. As I add more customizable features, more information will be displayed on the screen.



Here is the result of all those changes. Now we can adjust how fast the training process happens and we can also keep track of how many generations have passed. The restart algorithm also seems to be working just as intended as the birds are now getting better with each generation that passes.

```
let BG = new Image(box.width,box.height)
BG.src = "Flappy Bird BG.png"
```

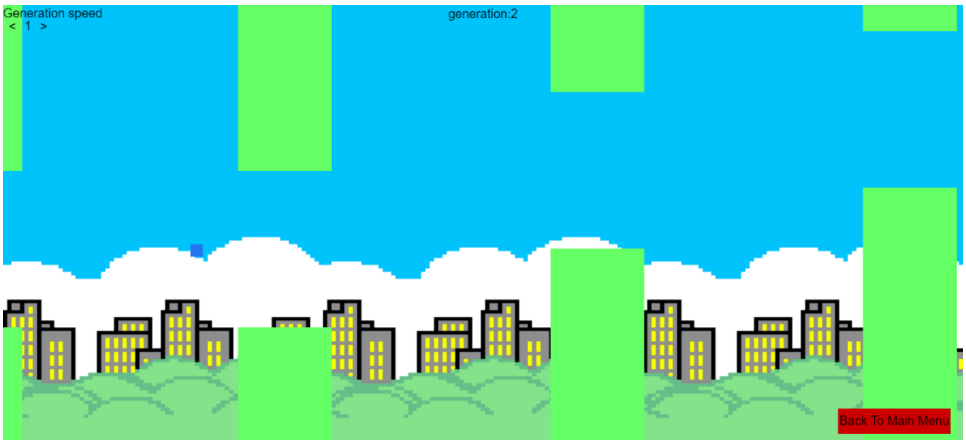
I wanted to add a background to the game to make it more visually appealing for the user, so I added a picture of a background using an Image class. The line “new image()” creates a new image tag in html which I can use to insert images into my project.



Here is the image I will be using as the background for this AI game.

```
//draw the background
pen.fillStyle = "rgba(100,100,255,1)"
pen.fillRect(0,0,box.width,box.height)
pen.drawImage(BG, 0, 0, box.width, box.height+5)
```

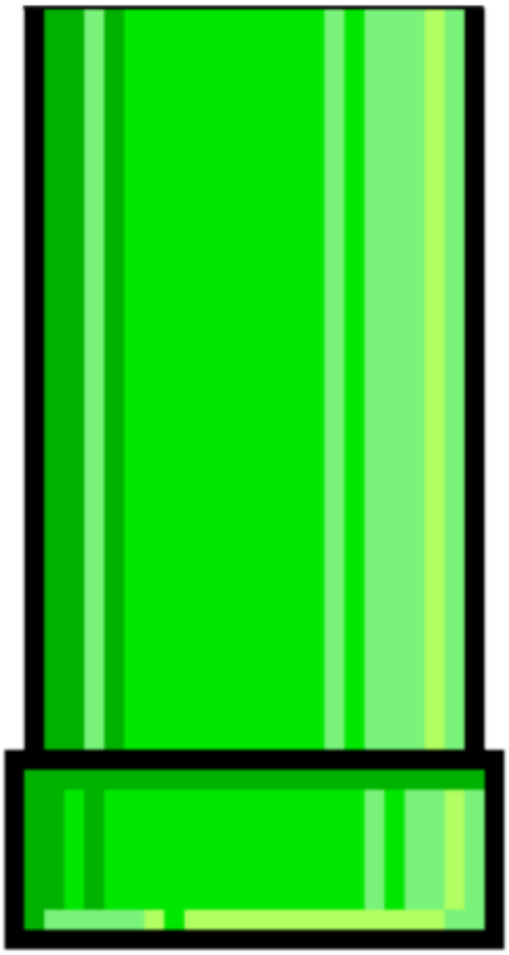
Inserting the image is easy as all I must do is use the “drawImage” function and provide an image, x and y position, width, and height. In this instance, I have made so it fills in the screen. The reason I have added 5 extra pixels for the height is because there is a small black line that is at the bottom of the image I am using. So, to make sure it does not display the black line, I will stretch the height of the image.



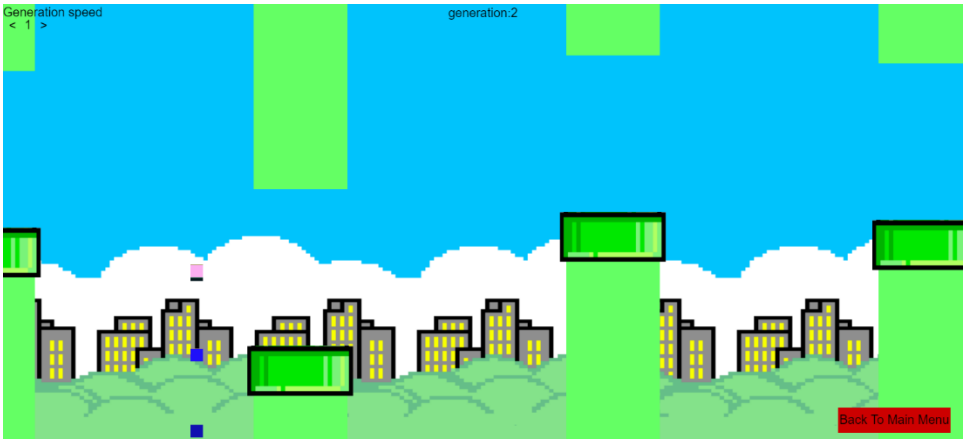
Here is the result of adding the background image. As shown in the screenshot above, the background image is displayed in the background, not obscuring any other items on the screen.

```
this.TopPipeImg = new Image(this.width+2,40)
this.TopPipeImg.src = "Flappy Bird Pipe.png"

w(){
pen.fillStyle = this.color
pen.fillRect(this.x,0,this.width,this.top_height)
pen.fillRect(this.x,this.bottom_y,this.width,this.bottom_height)
pen.drawImage(this.TopPipeImg, 74, 405, 269, 108, this.x-10, this.bottom_y, this.width+20, 80)
```

After inserting the background, I decided to insert the images for the pipes. I first got an image of a pipe and inserted it into the program by the same method I used with the background. Due to there being 2 parts to the pipe (the entrance part and the pipe part itself), I will need to split the image into 2 parts. The screenshot above shows me adding the end part of the pipe to the pipe class.



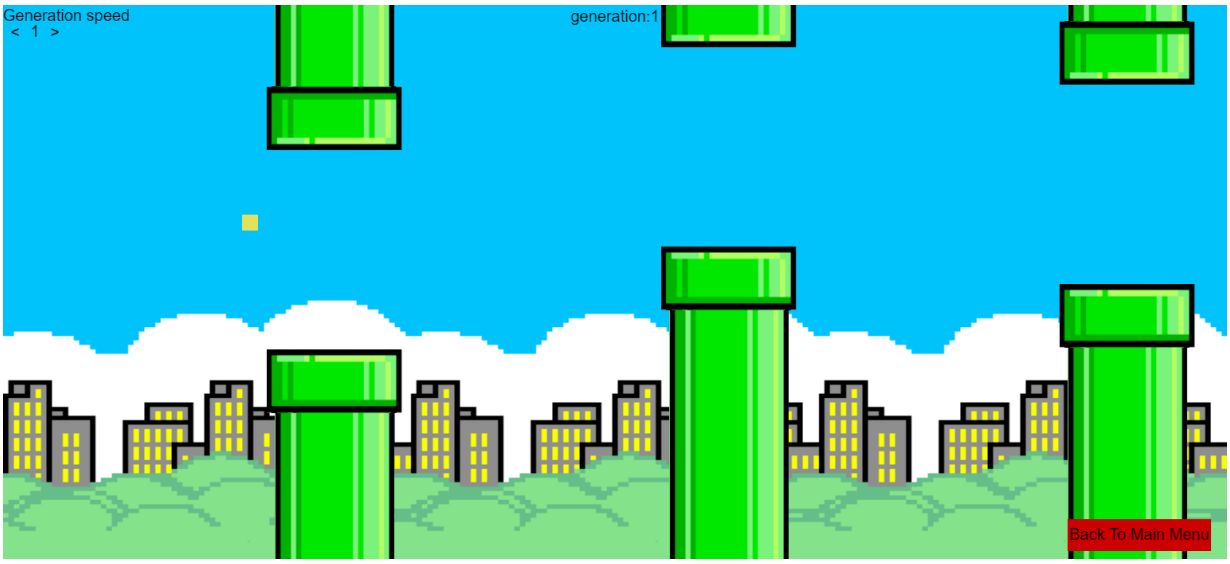
This is the image of the pipe I will be using for flappy bird.



Here is the result. I have managed to add the end part of the pipe to the bottom pipe. In this screenshot the pipe image is flipped and is not the right way around, however, I do manage to fix this mistake when I add the rest of the pipe images into the game.

```
pen.fillStyle = this.color
pen.fillRect(this.x,0,this.width,this.top_height)
pen.fillRect(this.x,this.bottom_y,this.width,this.bottom_height)
//top part
//bottom of pipe
pen.drawImage(this.BottomPipeImg,82,5,249,399,this.x,-10,this.width,this.top_height+10)
//top of pipe
pen.drawImage(this.TopPipeImg, 74, 405, 269, 108,this.x-10,this.top_height-80,this.width+20, 80)
//bottom part
pen.save()
pen.translate(this.x-10, this.bottom_y)
pen.scale(1,-1)
//bottom of pipe
pen.drawImage(this.BottomPipeImg,82,5,249,399,10,-this.bottom_height-10,this.width,this.bottom_height+10)
//top of pipe
pen.drawImage(this.TopPipeImg, 74, 405, 269, 108,0,-80, this.width+20, 80)
pen.restore()
```

Here is the new draw method for the pipe class. In the screenshot above, I had to make use of the "scale" method so I could flip the image to add the pipe images on the bottom pipe



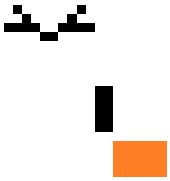
The screenshot above shows the result. The pipes now have images, and the game is starting to look a lot more appealing compared to how it looked before.

```
this.Img = new Image
this.Img.src = "Flappy Bird Icon.png"
```

I then moved on to the bird class. I added a new attribute which would store the bird image.

```
pen.fillStyle = this.color
pen.fillRect(this.x,this.y,this.width,this.height)
pen.drawImage(this.Img,0,0,25,25,this.x,this.y-5,25,25)
```

I also updated the draw method of the bird so that I could include the image of the bird



Above is the image of the bird I will be using. I did not draw a body for the bird as the square that the birds currently use will again be used to define the colour of the bird.

This shows the result of the bird. As shown, the image shows perfectly on the bird. Now I have added all the images onto the flappy bird game.

```
//Edit game stats
class GameStat{
  constructor(x,y,stat,statname,min,max,button,color){
    this.x = x || 0
    this.y = y || 0
    this.stat = stat
    this.statname = statname || "Gen speed"
    this.min = min || 0
    this.max = max || 100
    this.color = color || "black"
    if(!stat){
      console.error("a stat value is required!")
    }
    button.push(new Button(···
      ))
      button.push(new Button(···
      ))
  }
  draw(){
    this.stat = this.stat
    pen.fillStyle = this.color
    pen.font= "20px Arial"
    pen.textAlign = "start"
    pen.textBaseline = "alphabetic"
    pen.fillText(this.statname,this.x,this.y)
    pen.textBaseline = "alphabetic"
    pen.font= "20px Arial"
    pen.textAlign = "center"
    pen.fillText(this.stat,this.x+40,this.y+20)
    return this.stat
  }
}
```

As I mentioned in my design section, I wanted to allow the user to personalise the environment of the game that the AI plays. This means I would have to create many features which can be edited by the player. I decided to create a new class called GameStat within the utils.js file as I will be doing this within every AI game I add. The class takes in 8 parameters which determine the following:

- X and Y- the X and Y position the GameStat appears on the screen. The default will be (0,0)
- Stat- the stat that will be edited e.g. the number of AI's playing.
- Statname- the name of the stat that will be displayed to the user. The default is "Gen speed"
- Min and max- these will be the minimum and maximum values the stat can be changed to. The default is 1 and 100
- Button – this will be the buttons array which will store the buttons
- Color – the colour of the text.

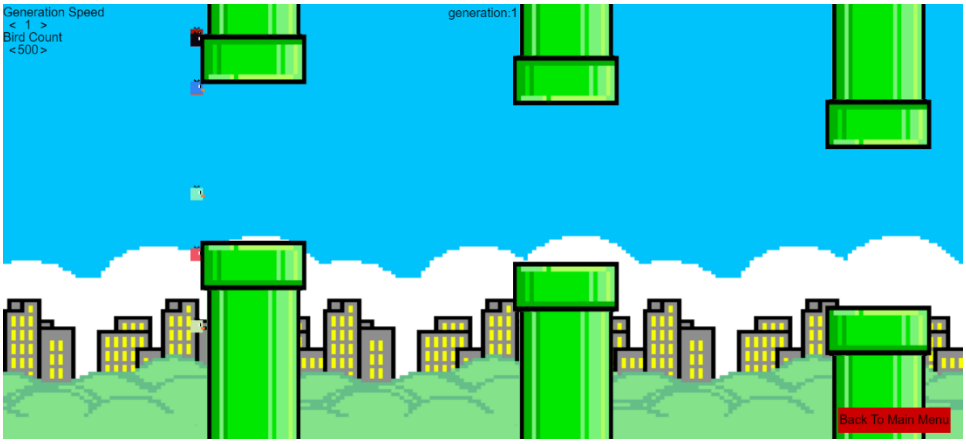
The class has 1 method which will do the same role as the generation speed and output the relevant information.

```
let GenSpeed = new GameStat(0,20,cycles,"Generation Speed",0,100,buttons)
let BirdCount = new GameStat(0,60,population,"Bird Count",1,1000,buttons)
```

Back in the flappy bird file, I have rewritten the generation speed variable and I have also added a new variable which will allow the user to change the number of birds. The implementation of this new class will allow me to create editable features about the game a lot easier.

```
cycles = GenSpeed.draw()
population = BirdCount.draw()
```

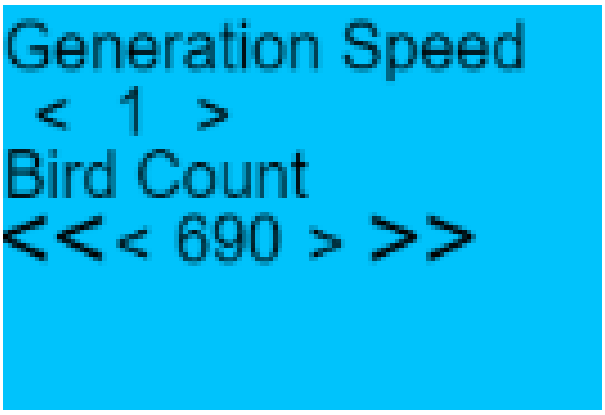
Due to the draw function returning the new updated stat value, I need to assign the features I wish to change with method.



The output shows the 2 game stats generation speed and bird count which I can change using the 2 arrows. This also disables me from changing these stats via the console on the browser which means I will have to use the arrows to change them.

```
constructor(x,y,stat,statname,min,max,button,color,hasExtremes,ExtremeAmt){
  this.x = x || 0
  this.y = y || 0
  this.stat = stat
  this.statname = statname || "Gen speed"
  this.min = min || 0
  this.max = max || 100
  this.color = color || "black"
  this.hasExtremes = hasExtremes
  this.ExtremeAmt = ExtremeAmt || 10
  if(!stat){
    console.error("a stat value is required!")
  }
  if(!this.hasExtremes){···
  }else{
    button.push(new Button(
      this.x+0.1,
      this.y,
      30,
      30,
      "rgba(0,0,0,0)",
      "<<",
      30,
      this.color,
      function () {

          this.textcolor = "#FFFFFF";
      },
      () => {
          if(this.stat <= this.min){
              //do nothing
          }else{
            this.stat -= this.ExtremeAmt
          }
        }
      ))
```

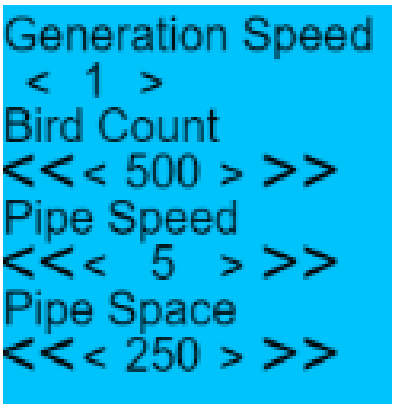
One slight annoyance I had when changing the number of birds was the fact, I had to continuously press the arrows to change them. This was not an issue with the generation speed as I only wanted to change it by a small amount, however this is not the same with the bird count. To combat this, I edited the game stat feature by adding a feature called Extremes. This will allow the user to increase a value by a larger amount than 1. The GameStat now has attributes called hasExtremes and ExtremeAmt. HasExtremes is used to see if they do have extremes and the ExtremeAmt is used to determine how many numbers the value can jump up by. Above shows the implementation of this feature. Now when initialising the gamestat, the feature to have extreme values will be available.



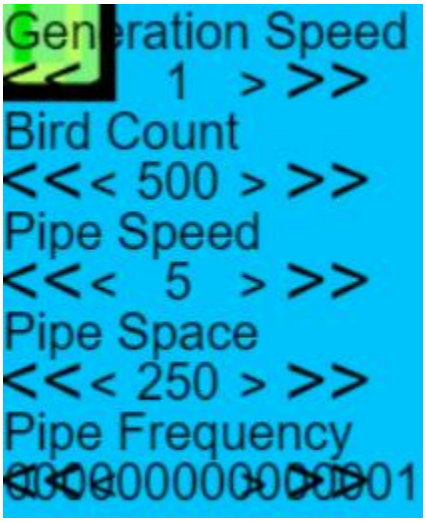
Here is the result of adding the extreme feature to the bird count. Now I can press the arrows to change the values by 1 or 10 depending on what arrow I press.

```
//stats for game that can be edited
let GenSpeed = new GameStat(0,20,cycles,"Generation Speed",0,100,1,buttons)
let BirdCount = new GameStat(0,60,population,"Bird Count",1,1000,1,buttons,"black",true)
let PipeSpeedStat = new GameStat(0,100,PipeSpeed,"Pipe Speed",1,50,1,buttons,"black",true,5)
let PipeSpace = new GameStat(0,140,spacing,"Pipe Space",200,500,1,buttons,"black",true)
```

I have now added new values that can be edited by using the buttons. They are the speed of the pipes and the spacing between the top and bottom pipe.



Here is the result of both attributes being added.



One issue that came up while I was adding values to the game was an error in which the number shown would display a long decimal value like shown above. It covers the buttons which is a problem because the user will not be able to see the buttons properly so I will need to find a way to fix it before moving on.

```
> 0.2+0.1
< 0.30000000000000004
```

The image above shows a clearer version of the error. While doing research on this bug, I learnt that it was a floating-point issue which meant that the issue was due to the programming language I was using. I decided to create a truncate function which would remove all the excess digits and just leave me with the needed information.

```
//truncate a value to 1 d.p
function Truncate(num){
  NewNum = num.toString()
  return NewNum.substring(0, 3)
}
```

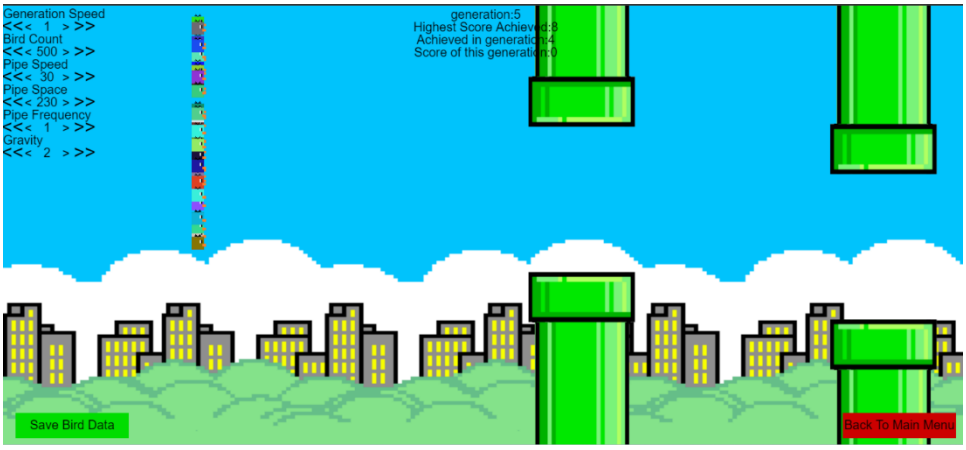
Here is the truncate function which is in the Utils.js file. It takes in a float as a parameter known as num and it converts that number into a string. It then uses a method which is built into all JavaScript strings known as substring. This method extracts characters between 2 positions in the string. I used this method to make the function return the first 3 characters of the number which should include the first zero, followed by a decimal point and ended with the final e.g. 3 if the number is 0.3.

```
if(this.hasExtremes){
  if(Math.abs(this.amt) < 1){
    pen.fillText(Truncate(this.stat),this.x+70,this.y+20)
  }else{
    pen.fillText(this.stat,this.x+70,this.y+20)
  }
}else{
  if(Math.abs(this.amt) < 1){
  pen.fillText(Truncate(this.stat),this.x+40,this.y+20)
  }else{
    pen.fillText(this.stat,this.x+40,this.y+20)
  }
}
```

Here is the implementation of this function within my code. Now, if the absolute value of the number is between 0 and 1, then it will truncate it which will fix the issue of long decimals appearing.

```
//save bird data
buttons.push(new Button(
    20,
    (box.height) - 50,
    180,
    40,
    "#00DD00",
    "Save Bird Data",
    20,
    "black",
    function () {
        this.color = "#00FF00";
        this.textcolor = "#FFFFFF";
    },
    () => {
        let data = {
            brain:JSON.parse(localStorage.getItem("bestFlappyBrain")),
            HighestScore:score,
            PipeSpeed:PipeSpeed,
            Spacing:spacing,
            diff:PipeFreq,
            grav:gravity
        }
        localStorage.setItem("SavedDataFlappyBird",JSON.stringify(data))
        console.log("saved")
        GameRecentlySaved = true
    }
))
```

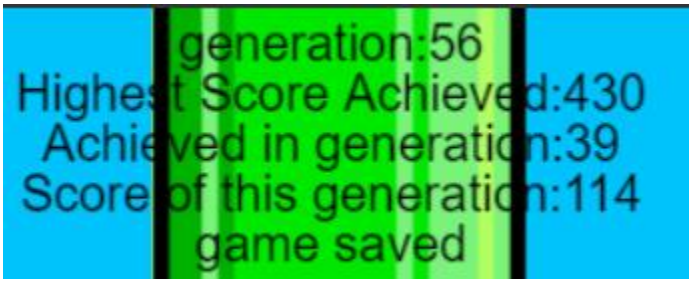
By this point of the program, I have added enough features that can give the player unique control while training the A.I so I decided to add the save feature to the program. This was simple as all I had to do was create a new button which when pressed would save the best A.I of the last generation along with the characteristics of the game such as pipe speed and the gravity. I managed to store all this by first creating an object literal within JavaScript and then I would turn it into a JSON string before saving it to local storage.



Above shows the button within the program. Now whenever I press it, it will save the A.I. It will also change the GameRecentlySaved variable to true which will be used later within the program.

```
//check if game was recently saved
if(GameRecentlySaved){
 DisappearTimer--
}
if(DisappearTimer <= 0 ){
 GameRecentlySaved = false
 DisappearTimer = 500
}
```

I wanted to add some visual feedback to the user so that they would know that their A.I has been saved, so I added a feature which would output a message saying “game saved” for a few seconds after it has been clicked. Above shows the code as to how I did it.



This is the result of this implementation. Now whenever the user presses the save AI button, the “game saved” message will appear. I have now finished the flappy bird AI game and will be working on the game where the user can play against the A.I.

Updating the main menu #1

Now that I have finished creating the A.I game for flappy bird, I can start updating the menu by adding the features I said I would add in the design section.

```
function () {
  this.color = "#0000FF";
  this.textcolor = "#555555";
  TextDescription = "Teach AI to play the classic Flappy Bird game"
  BackGround.src = "Logo.png"
},//function when mouse is over button
```

The first thing I did was update the mouse over feature on the flappy bird AI game so it would now display an image of flappy bird. I did this by creating a new image object called BackGround and when the mouse is over the flappy bird button, the image source will change into the flappy bird image I called “logo.png”

```
if (InteractedButtons < 1) {
  TextDescription = " "
  BackGround.src = "blank.png"
}
```

I also updated the interaction button code which will now also display a blank image if no buttons are being hovered over.

Here is the result of this addition. It does not look as good as I thought it would be so I may end up changing it in the future.

```
async function GetColorScheme(){
  let FilePath = 'ColorScheme.json';
  // Fetch the JSON data
  let response = await fetch(FilePath)
  let data = await response.json()
  return data.ColorScheme.FlappyBird
}
```

As I mentioned in my design section, I wanted to add a colour scheme to the main menu depending on the game that was played last so in my Utils.js file, I made an async function which would fetch the colour scheme for the program. I first declared a file path variable which would hold the file in which the colour scheme data was held. I would then get the colour scheme data by using the fetch Api built into JavaScript. In this screenshot, it specifically fetches the flappy bird colour scheme, however, I will later update this to get the colour scheme of the last game played.

In the ColorScheme.json file, I have this data shown in the image. It contains information on what the colours should be for each instance such as the buttons, title text colour and others. This currently only holds 2 colour schemes which are the default one and flappy bird one.

```
let ColorScheme = GetColorScheme().then(data=>{
  ColorScheme = data
})
```

In the main menu.html file, I added a line of code which would set the colour scheme in the Json file to a variable called ColorScheme. I then went and updated all the buttons so they would use the new colour scheme.

Here is the result of this implementation and now they are using the new flappy bird colour scheme.

```
let LastGamePlayed = localStorage.getItem("LastGamePlayed")
let ColorScheme = GetColorScheme().then(data=>{
  let ColorSchemeToBeUsed;
  switch (LastGamePlayed) {
    case "FlappyBird":
      ColorSchemeToBeUsed = data.FlappyBird
      break;

    default:
    ColorSchemeToBeUsed = data.Default
      break;
  }
  ColorScheme = ColorSchemeToBeUsed
})
```

I now updated the colour scheme code and now it selects the colour scheme based on the last game played. It does this by first going into local storage and getting the last game played. It then uses that value in a switch statement to see what colour scheme to display. If there is no last game played, then the default colour scheme is used.

```
localStorage.setItem("LastGamePlayed","FlappyBird")
```

In the flappy bird AI game, I added this line of code which will set the last game played to flappy bird.

Here is the menu if there is no recently played game. It uses the default colour scheme.

Here is the menu if flappy bird was the last played game played. It used the flappy bird colour scheme.

```
//get the color schemes
async function GetColorScheme(){
  let HasFailed = false
  let FilePath = 'ColorScheme.json';
  // Fetch the JSON data
  let response = await fetch(FilePath).catch(error=>{console.error("Failed to fetch Data. Sorry :("); HasFailed = true})
  if(HasFailed){
    return "Failure"
  }else{
    let data = await response.json()
    return data.ColorScheme
  }
}
```

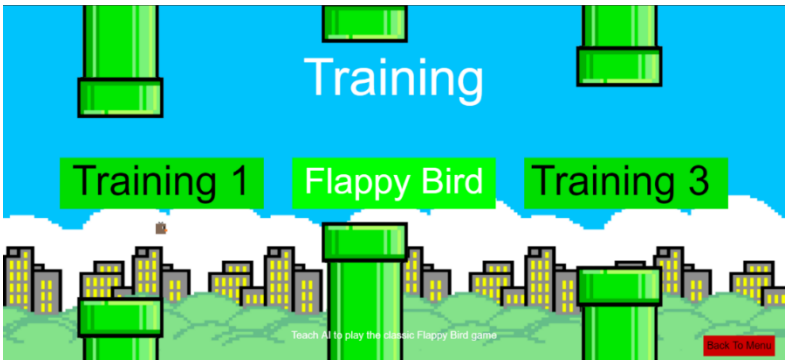
I added a feature in the ColorScheme function if it is unable to fetch the data from the Json file. If this is the case. An error message is displayed in the console and the string “Failure” is returned instead of the colour scheme.

```
let ColorScheme = GetColorScheme().then(data=>{
  if(data == "Failure"){
    ColorScheme = {
    RegularButtonColors:"#0000CC",
    RegularButtonColorsLight:"#0000FF",
    QuitButtonColors:"#CC0000",
    QuitButtonColorsLight:"#FF0000",
    RegularButtonText:"black",
    RegularButtonTextLight:"#555555",
    TitleText:"#FFFFFF",
    BackgroundColor:"black"
    }
  }else{
    let ColorSchemeToBeUsed;
    switch (LastGamePlayed) {
      case "FlappyBird":
        ColorSchemeToBeUsed = data.FlappyBird
      break;

      default:
        ColorSchemeToBeUsed = data.Default
      break;
    }
  ColorScheme = ColorSchemeToBeUsed
  }
})
```

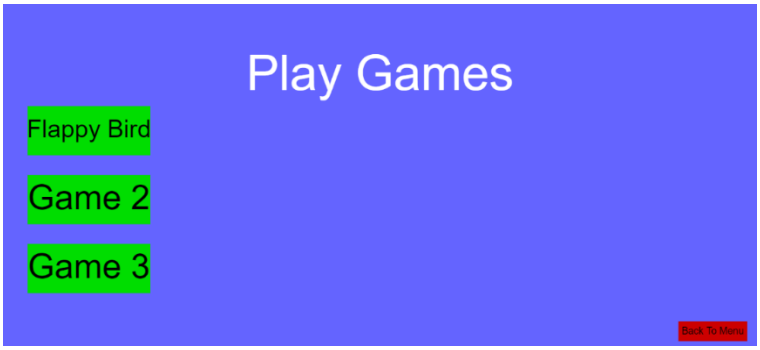
In the main menu.html file, I added a colour scheme which will display if the failure string is returned, and the program will use this colour scheme instead of the others. For now, it is only the default colour scheme but, in the future, I might change this colour scheme to make it unique.

```
//get the background stuff ready
let FrameCount = 0
let FlappyData = JSON.parse(localStorage.getItem("SavedDataFlappyBird"))
let spacing = FlappyData.Spacing
let PipeSpeed = FlappyData.PipeSpeed
let PipeFreq = FlappyData.diff
let gravity = FlappyData.grav
let pipes = []
pipes.push(new Pipe)
let bird = new Bird()
bird.brain = FlappyData.brain
function CoolBackground(){
  FrameCount++
  //Background
  pen.fillStyle = ColorScheme.BackgroundColor
  pen.fillRect(0, 0, box.width, box.height)
  //choose the game that's being hovered over
  switch (ButtonHoveredOver) {
    case "FlappyBird":
      BG.src = "Flappy Bird BG.png"
      //new Background
      pen.drawImage(BG, 0, 0, box.width, box.height+5)
      //update everything
      bird.update()
      for(let i = 0; i < pipes.length; i++){
        pipes[i].update()
        //remove the pipe if it's off the screen
        if(pipes[i].x < -pipes[i].width){
          pipes.splice(i,1)
        }
      }
      //push a new pipe into the list
      if(FrameCount % Math.floor((500 / (PipeSpeed*PipeFreq))) == 0){
        pipes.push(new Pipe)
      }
      //draw everything
      bird.draw()
      pipes.forEach(pipe=>{
        pipe.draw()
      })
      break;

    default:
      break;
  }
  //repeat function
  requestAnimationFrame(CoolBackground)
}
CoolBackground()
Loop()
LoadMenu()
```

Instead of displaying an image of flappy bird when the mouse is hovered over, I decided to make it show the last saved A.I. playing the game. To do this, I had to move the bird class and pipe class to the utils.js file so I could access them in the main menu file. The code shown displays how I created a function called “CoolBackground” (the name will be changed later) which will play flappy bird using the data from the saved flappy bird A.I to create the game setting and place the A.I in the game.



Here is the result of this and I prefer this one over the old static image one I created previously so this will be how it looks from now on.



In this image, I created the Play games menu which displays 3 games on the side like I showed in my design section.

```
let GameDescription = {
  GameImg: new Image(),
  GameName:" ",
  GameDes: " ",
  MinsPlayed:" ",
  AILevel:" ",
}
let AILevels = {
  FlappyBird: CheckLevel("FlappyBird",JSON.parse(localStorage.getItem("SavedDataFlappyBird")))
}
```

I created a new object called GameDescription which will show the information about the game such as an image of the game, the game name, a description of the game, the minutes played and the current A.I level. The A.I level will be determined by a function that I will add into my Utils.js file.

```
//Description for games
pen.font = "50px Arial";
pen.fillStyle = "white";
pen.textAlign = "center";
pen.textBaseline = "middle";
pen.fillText(GameDescription.GameName, (box.width/2)+(box.width/4), (box.height/2)+100);
pen.font = "20px Arial";
pen.fillText(GameDescription.GameDes, (box.width/2)+(box.width/4), (box.height/2)+150);
pen.fillText(GameDescription.AILevel, (box.width/2)+(box.width/4), (box.height/2)+180);
pen.fillText(GameDescription.MinsPlayed, (box.width/2)+(box.width/4), (box.height/2)+210);
pen.drawImage(GameDescription.GameImg,((box.width/2)+(box.width/4))-90,(box.height/2)-90,200,150)
```

The image here shows how the data from the GameDescription will be displayed onto the screen. I have used the text feature to write all the information onto the screen

```
function CheckLevel(GameName,Data){
  let level = 0
  switch(GameName){
    case "FlappyBird":
      level = ((Data.grav*Data.PipeSpeed)/250)+(((1/Data.grav)*Data.diff)/0.2)+((1/Data.Spacing)/(500))
      level = Data.HighestScore/level
      level
      level /=100

    break
  }
  return Math.round(level)
}
```

Here is the CheckLevel function which determines the level of an A.I based on a range of factors. It takes 2 parameters, one being the name of the game and the other being the data of the A.I. After that it uses a switch statement to determine how the A.I should be calculated and then it preforms the necessary

formula before returning the level back. The formula is not a definitive version, and it might change by the definitive version.

After adding all the implementations, here is the result of it all. Now hovering over the game will display the information that is shown in the screenshot above.

This is what the menu looks like once the flappy bird game is clicked. Here the user will be able to choose which A.I. they wish to play against. They will have the option of 2 Prebuilt A.Is and the custom A.I they have trained.

Creating Flappy Bird Game.html

This file will contain the flappy bird game where the player can go against the A.I to compete against it.

```
<title>Flappy Bird Game</title>
<script src="NeuralNetwork.js"></script>
<script src="Utils.js"></script>
<canvas id="box" width="500" height="500" style="border:0px solid black"></canvas>
</canvas>
<style>
  body {
    margin: 0;
    overflow: hidden;
  }
</style>
<script>
    var box = document.getElementById("box")
    box.width = innerWidth
    box.height = innerHeight
    var pen = box.getContext("2d")
    let buttons = []
    let BG = new Image(box.width,box.height)
    BG.src = "Flappy Bird BG.png"
    let pos = {
        x: 0,
        y: 0
    }
    //Return to main menu button
    buttons.push(new Button(
        (box.width)-200,
        (box.height) - 50,
        180,
        40,
        "#CC0000",
        "Back To Main Menu",
        20,
        "black",
        function () {
            this.color = "#FF0000";
            this.textcolor = "#FFFFFF";
        },
        () => {
            if(confirm("Are you sure you want to quit and go back to menu?")){
                window.open("Main Menu.html")
            }
        }
    ))
    function Loop(){
        //draw the background
        pen.fillStyle = "rgba(100,100,255,1)"
        pen.fillRect(0,0,box.width,box.height)
        pen.drawImage(BG, 0, 0, box.width, box.height+5)
        ButtonInteraction(pos,buttons,false)
        requestAnimationFrame(Loop)
    }
    Loop()
</script>
```

I first took some code from the flappy bird A.I game and pasted it into the file as a template for creating the game. All it currently does is display a blue screen with the same back to menu button as the flappy bird AI game.

```
if(!this.HumanControlled){
  this.think(pipes)
  let outputs = NeuralNetwork.FeedForward(this.brain,this.inputs)
  if(outputs[0] > 0.5){
    this.flap()
  }
}
```

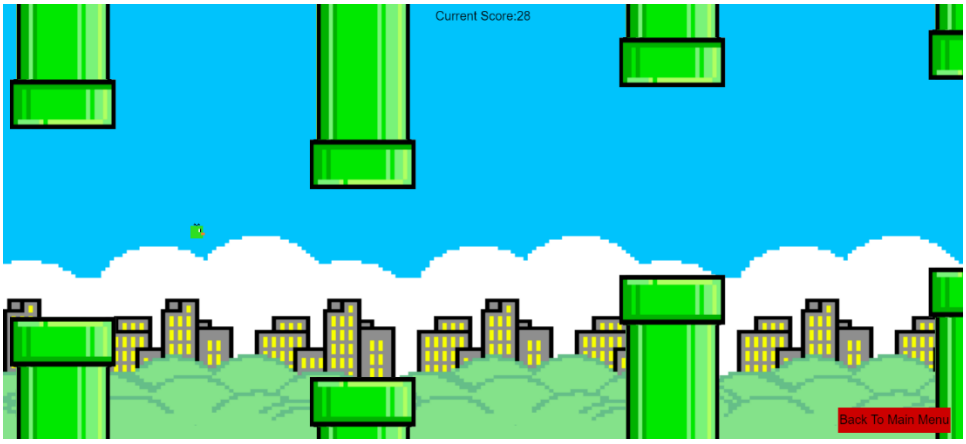
For the user to play the game, I changed the bird class by adding a new attribute which would check to see if a human (the user) were controlling it or not. If not, it would perform the feedforward algorithm and constantly update its inputs to determine whether to flap or not. As a result of this implementation, the bird class now has a parameter which is a boolean value set to true if the user is controlling it and false if the user is not.

```
function Loop(){
    //updates
    FrameCount++
    //update the birds
    for(let i = 0; i < birds.length; i++){
        birds[i].update()
        //check if the birds are out of bounds
        if(birds[i].y < 0 || birds[i].y > box.height){
            birds[i].isDead = true
        }
        //collision detection with birds and pipe
        pipes.forEach(pipe=>{
        //touches the top pipe
        if(birds[i].x+birds[i].width >= pipe.x && birds[i].x <= pipe.x+pipe.width && 0 <= birds[i].y+birds[i].height && birds[i].y <= pipe.top_height){
            birds[i].isDead = true
        }
        //touching the bottom pipe
        else if(birds[i].x+birds[i].width >= pipe.x && birds[i].x <= pipe.x+pipe.width && pipe.bottom_y <= birds[i].y+birds[i].height&& birds[i].y <= pipe.bottom_y+pipe.bottom_height){
            birds[i].isDead = true
        }
        //if the birds have collected the score from the pipe or not
        if(birds[i].x > pipe.x+pipe.width && !pipe.HasCollectedScore){
            score++
            pipe.HasCollectedScore = true
            }
        })
    }
    //update the pipes
    for(let i = 0; i < pipes.length; i++){
        pipes[i].update()
        //remove the pipe if it's off the screen
        if(pipes[i].x < -pipes[i].width){
            pipes.splice(i,1)
        }
    }
    //push a new pipe into the list
    if(FrameCount % Math.floor((500 / (PipeSpeed*PipeFreq))) == 0){
        pipes.push(new Pipe)
    }
    //draw the background
    pen.fillStyle = "rgba(100,100,255,1)"
    pen.fillRect(0,0,box.width,box.height)
    pen.drawImage(BG, 0, 0, box.width, box.height+5)
    //draw everything else
    //draw the birds
    birds.forEach(bird=>{
        bird.draw()
    })
    //draw the pipes
    pipes.forEach(pipe=>{
        pipe.draw()
    })
    //text
    pen.fillStyle = "black"
    pen.font= "20px Arial"
    pen.fillText("Current Score:"+score,box.width/2,20)
    ButtonInteraction(pos,buttons,false)
    requestAnimationFrame(Loop)
}
Loop()
```

Here, I copied the code from the flappy bird A.I game to speed up the process. I have added the pipes, the birds, and the collision detection between the pipes and the birds. I have also added the characteristics of the game from the saved data so the A.I will play in the exact same environment it was trained in. It also displays the score and tracks the current score.

```
//pipes
let pipes = []
pipes.push(new Pipe)
//birds
let birds = []
birds.push(new Bird(true))
birds.push(new Bird)
//give the second bird the A.I
birds[1].brain = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).brain
//control the first bird
document.addEventListener("keydown",()=>{
    birds[0].flap()
})
```

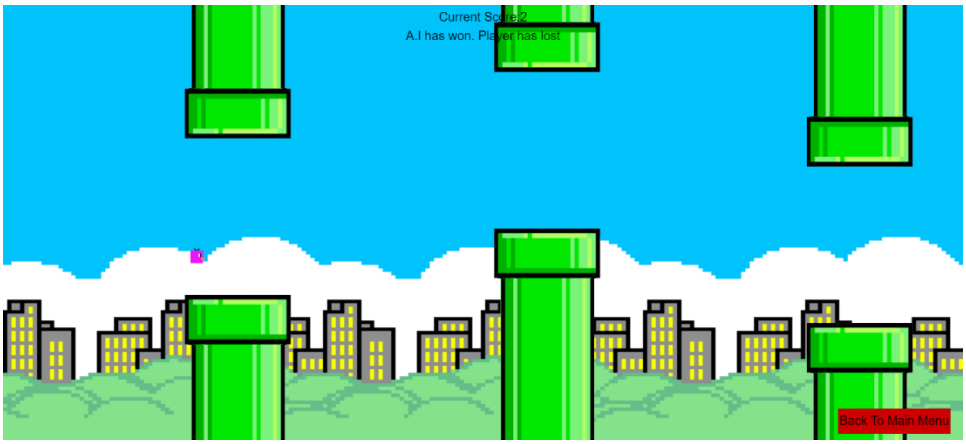
This is the creating the 2 birds. The first bird will be the player and the second bird will be the A.I. I added an event listener so the user can press any key down to make the bird flap. The A.I bird was also given the best A.I brain so it can play the game.



Here is the result of all those additions. Now we have both the user and the A.I playing flappy bird at the same time.

```
//kill the bird and declare a winner
birds.forEach((bird,i)=>{
    if(bird.isDead){
        if(bird.HumanControlled){
            WinnerText = "A.I has won. Player has lost"
        }else{
            WinnerText = "A.I has lost. Player has won"
        }
        birds.splice(i,1)
    }
})
```

After getting everything else in the game to work, I decided to work on the win feature. This was simple to add as all I had to do was check if a bird was dead and the check if it was human controlled or not. If it was, then the player must have died and the A.I is the winner, if not, the A.I must have died, and the Player has won.



Here is the result of adding this in the program. Now when one of the birds die, a winner is declared. I will later improve this feature to provide more feedback to the user.

```
let Count = 11
let CountDownText = "Game Begins in:"+Count
for(let i = 0; i < 1; i++){
        Loop()
    }
function CountDown(){
    Count--
    CountDownText = "Game Begins in:"+Count
    pen.fillStyle = "rgba(100,100,255,1)"
    pen.fillRect((box.width/2)-100,0,200,100)
    pen.fillStyle = "black"
    pen.font= "20px Arial"
    pen.fillText(CountDownText,box.width/2,20)
    if(Count <= 0){
        pen.fillStyle = "rgba(100,100,255,1)"
        CountDownText = " "
        pen.fillRect((box.width/2)-100,0,200,100)
        CanStartNow = true
        Loop()
        clearInterval(CountDownStart)
    }
}
let CountDownStart = setInterval(CountDown,1000)
Loop()
CountDown()
```

For the flappy bird game, I wanted to add a count down from 10 which would prepare the user a bit more time before the game started. I created a function called countdown which would display text showing how many seconds the game is starting in. It runs on set interval timer which means it calls itself every second. Once the timer hits zero, the main game runs.

Here is the result of that implementation. Now a timer appears at the beginning of the game which counts down from 10. One issue is that the textures are not drawing but I will fix this issue later within the development of this project.

Creating Cube Runner AI.html

This section will be about the next AI game I add in my project called Cube Runner. It is another version of the chrome dino game with a few changes.

```
<title>Chrome Dino AI</title>
<script src="NeuralNetwork.js"></script>
<script src="Utils.js"></script>
<canvas id="box" width="500" height="500" style="border:0px solid black"></canvas>
</canvas>
<style>
  body {
    margin: 0;
    overflow: hidden;
  }
</style>
<script>
    var box = document.getElementById("box")
    box.width = innerWidth
    box.height = innerHeight
    var pen = box.getContext("2d")
    //button functionality
    let pos = {
        x: 0,
        y: 0
    }
    buttons = []
    //Return to main menu button
    buttons.push(new Button(
        (box.width)-200,
        (box.height) - 50,
        180,
        40,
        "#CC0000",
        "Back To Main Menu",
        20,
        "black",
        function () {
            this.color = "#FF0000";
            this.textcolor = "#FFFFFF";
        },
        () => {
            if(confirm("Are you sure you want to quit and go back to menu?")){
                window.open("Main Menu.html")
            }
        }
    ))
    function GameLoop(){
        //draw the background
        pen.fillStyle = "rgba(100,100,255,1)"
        pen.fillRect(0,0,box.width,box.height)
        ButtonInteraction(pos,buttons,false)
        requestAnimationFrame(GameLoop)
    }
    GameLoop()
</script>
```

I first started with some basic code which would make the background blue and created a button which would take the user back to the menu. I also created a game loop function which will be used for drawing and updating the game.

Here is the result. As said previously, it turns the background blue and adds the back to menu button to the game.

```
<script>
    var box = document.getElementById("box")
    box.width = innerWidth
    box.height = innerHeight
    var pen = box.getContext("2d")
    //ground
    const GroundHeight = (box.height) - 60
    //button functionality
    let pos = {···
    buttons = []
    //Return to main menu button
    buttons.push(new Button(···
    function GameLoop(){
      //Drawing
      //draw the background
      pen.fillStyle = "rgba(100,100,255,1)"
      pen.fillRect(0,0,box.width,box.height)
      pen.fillStyle = "rgba(9,224,20,1)"
      pen.fillRect(0,GroundHeight,box.width,60)
      ButtonInteraction(pos,buttons,false)
      requestAnimationFrame(GameLoop)
    }
    GameLoop()
</script>
```

I then decided to create a constant called GroundHeight. I made it a constant as it should not change at all throughout the duration of this program. I then wrote some code which would draw the ground on the screen. I made sure that the ground was drawn on the screen before the back to menu button so that the ground would not cover it and the user could still click the button.

Here is the result. The ground now is drawn to the screen every frame as well as the background and the button is on top of the ground so the user can still interact with it if needed.

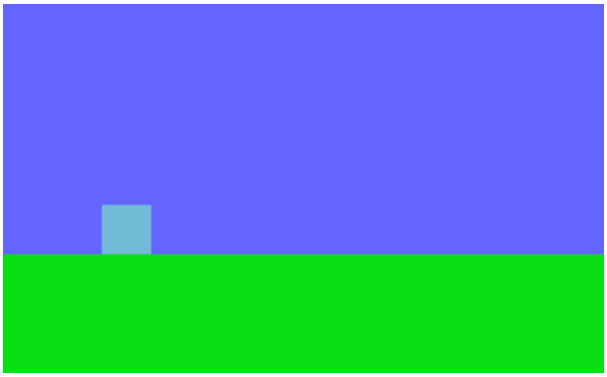
```
class Runner{
  constructor(){
    this.x = 50
    this.y = 0
    this.width = 25
    this.height = 25
    this.color = GenerateRandomColor()
    this.Yspeed = 0
  }
  draw(){
  pen.fillStyle = this.color
  pen.fillRect(this.x,this.y,this.width,this.height)
  }
  update(){
    this.y += this.Yspeed
    this.Yspeed *=0.9
    if(this.y <= GroundHeight - this.height){
      this.Yspeed +=gravity
    }else{
      this.Yspeed = 0
      this.y = GroundHeight - this.height
    }
  }
}
let Runners = []
Runners.push(new Runner)
```

After doing that, I decided to create a new class called Runner. This will be the player of the game and it will be what the A.I control to play the game. The attributes so far are the x and y positions of the player, the width and height, the colour and the Yspeed which is just the speed of the y position. It also has 2 methods which just draw the cube and update it. In the update method, the program will check if the cube is touching the floor. If not, the runner's y position will be affected by the gravity of the program which I have set as a variable with the value 0.5. Just like in the flappy bird class, the y speed is multiplied by a constant smaller than 1. This is to make sure the cube's y speed does not keep increasing until infinity and keeps it limited to a maximum value. For the colour of the runner, I have used a

function called GenerateRandomColor which will be explained below. After creating this class. I made an array called Runners which will hold all the runners within the program. After that. I pushed a new runner into the array and made it so it would draw and update on the screen.

```
function GenerateRandomColor(){
  return `rgba(${Math.floor(Math.random() * (255 - 0 + 1) + 0)},${Math.floor(Math.random() * (255 - 0 + 1) + 0)},${Math.floor(Math.random() * (255 - 0 + 1) + 0)},1)`
}
```

In the Utils.js file, I created a function called GenerateRandomColor which just generates a random colour by using the Math.random() feature in JavaScript. I decided to create a function that does this instead of writing that line as it makes the code a lot more readable. Because of the addition of this function, I also changed the way the color is chosen for the bird class by using this function.



Here is the result of the implementation of the runner. Now we have a cube drawn on the screen which will serve as the player. Refreshing the page will restart the program which in turn will create a new runner with a distinct colour.

```
  Jump(){
    if(!this.IsInAir){
      this.Yspeed -=this.JumpForce
      this.IsInAir = true
    }
  }
}
document.addEventListener("keydown",(e)=>{
  Runners[0].Jump()
})
```

I then added 2 methods to the runner class which are called Jump and Crouch. This screenshot shows the Jump method which will first check if the runner is in the air or not. If it is, then it will not do anything. However, if it is not, it will jump up with a jump force attribute I created, and it will also turn the IsInAir attribute to true.

```
Crouch(){
  this.height = this.crouchHeight
}
```

This is the Crouch method which changes the height of the runner to a crouchHeight attribute which I have set to half the height of the runner.

```
document.addEventListener("keydown",(e)=>{
  switch(e.keyCode){
    case 32:
      Runners[0].Jump()
    break
    default:
      Runners[0].Crouch()
    break
  }
})
document.addEventListener("keyup",(e)=>{
   switch(e.keyCode){
    case 32:
      //nothing
    break
    default:
      Runners[0].height = Runners[0].Oheight
    break
  }
})
```

I then wanted to test the code to make sure that it worked properly so I added controls which would let me control the runner by letting it jump and crouch. The code uses event listeners and switch statements to determine which key I am pressing on my keyboard. If the keycode is 32 (which means I am pressing the space bar) the runner should jump. If I press any other key, then it should crouch. I also added a key up event listener which checks to see if I have let go of any keys. If I have, then the runner Un crouches.

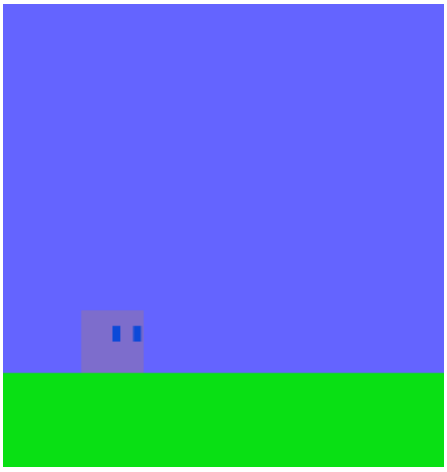
```
//eyes
pen.save()
pen.translate(this.x,this.y)
if(this.IsCrouching){
  pen.fillStyle =  this.eyesColor
  pen.fillRect(this.width/2,this.width/4,this.width/8,this.width/8)
  pen.fillRect((this.width/2)+(this.width/3),this.width/4,this.width/8,this.width/8)
}else{
  pen.fillStyle =  this.eyesColor
  pen.fillRect(this.width/2,this.width/4,this.width/8,this.width/4)
  pen.fillRect((this.width/2)+(this.width/3),this.width/4,this.width/8,this.width/4)
}
pen.restore()
```

I decided as a visual to give the runners eyes. Here is the code that draws the eyes on the cube. The eyes will be different depending on if the runner is crouching or not. This implementation has also made me add a new attribute which is the eye color which is generated from the GenerateRandomColor function.

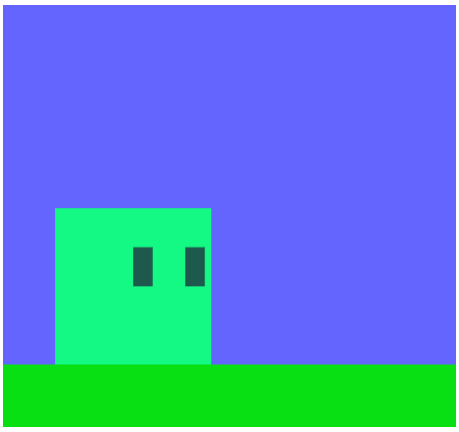
Here is the result of adding the eyes on the cube. I do think that this looks a lot more friendly than before.

```
this.size = 50
this.width = this.size
this.height = this.size
this.Oheight = this.height
this.crouchHeight = this.height/2
```

I then added a new attribute called size which would determine the height and width of the cube. After adding this attribute, I decided to experiment with changing the size attribute.



This is what the cube looks like if I change the size to 20



And here is what it looks like if I change it to 100. Nothing is deformed or not work correctly so I changed it to 40 as the default.

```
Crouch(){
  this.height = this.crouchHeight
  this.IsCrouching = true
}
UnCrouch(){
  this.height = this.Oheight
  this.IsCrouching = false
}
```

Before testing the Jump and Crouch method, I decided to create an Uncrouch method which would turn the height of the runner to the original height. I decided to do this as I wanted to differentiate this game from just being a Chrome dino copy and wanted to change the gameplay slightly.

```
Jump(){
  if(!this.IsInAir){
    if(this.IsCrouching){
      this.Yspeed -=this.JumpForce
    }else{
      this.Yspeed -=this.JumpForce/2
    }
    this.IsInAir = true
  }
}
```

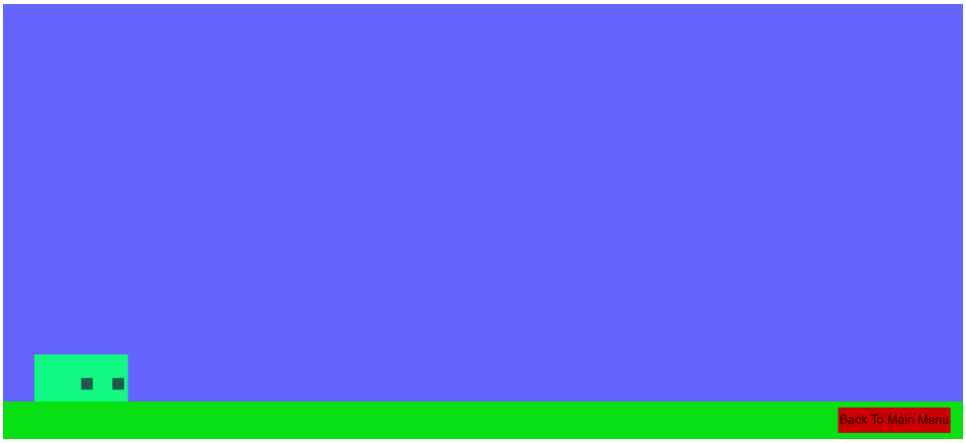
I updated the Jump method and now it checks if the runner is crouching or not. If it is, then it jumps using its full Jumpforce. However, if it is not, it only jumps using half the jumpforce. I decided to add this change as it could create interesting gameplay for this game.

```
document.addEventListener("keydown",(e)=>{
  console.log(e)
  switch(e.keyCode){
    case 32:
      Runners[0].Jump()
    break
    case 87:
      Runners[0].UnCrouch()
    break
    case 83:
      Runners[0].Crouch()
    break
  }
})
```

I updated the controls and now the keys do the following:

- 32 (the spacebar) makes the runner jump
- 87 (W key) Uncrouches the runner
- 83 (S key) Makes the runner crouch

Now I will test these methods to make sure that everything works as it should and if it does, I can move on from the runner and work on something else.



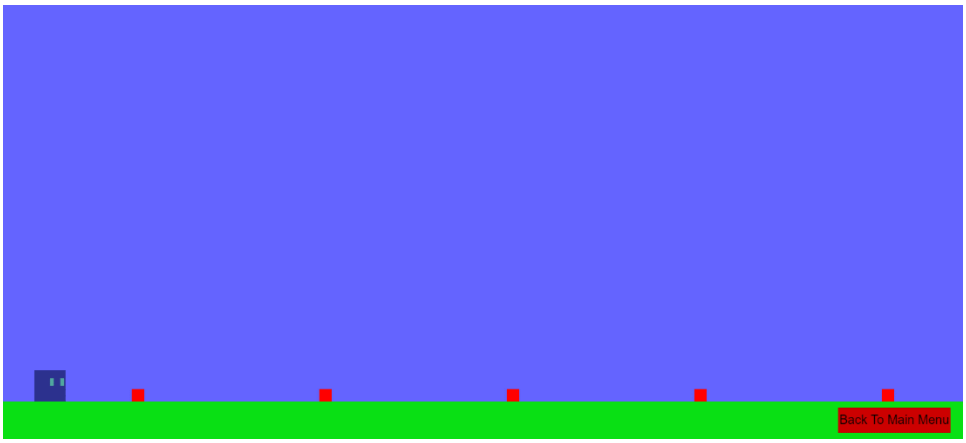
This image displays the runner crouching. The eyes get smaller as expected and the height of the runner reduces. I do not have screenshots of the Jump and Uncrouch method, but they do work just as expected which means I can now move onto another aspect of this game and come back to the runner later.

```
//simple obstacle
class Obstacle{
  constructor(){
    this.x = box.width
    this.y = GroundHeight - 20
    this.width = 20
    this.height = 20
    this.color = "red"
    this.Id = 0
  }
  draw(){
  pen.fillStyle = this.color
  pen.fillRect(this.x,this.y,this.width,this.height)
  }
  update(){
    this.x -=10
  }
}
let obstacles = []
obstacles.push(new Obstacle)
```

For this game, the runner will have to dodge obstacles by either crouching under or jumping over them. So, I created an Obstacle class which is just a simple 20x20 block which moves across the screen. After that, I created an array which will store the obstacles so that I can update them all at once. This Obstacle class is also given an Id which will be used for the A.I. In this instance, it is currently zero.

```
//Obstacles
obstacles.forEach(obstacle=>{
  obstacle.update()
})
obstacles.forEach((obstacle,i)=>{
  if(obstacle.x < -obstacle.width){
    obstacles.splice(i,1)
  }
})
if(FrameCount % 30 == 0){
  obstacles.push(new Obstacle)
}
```

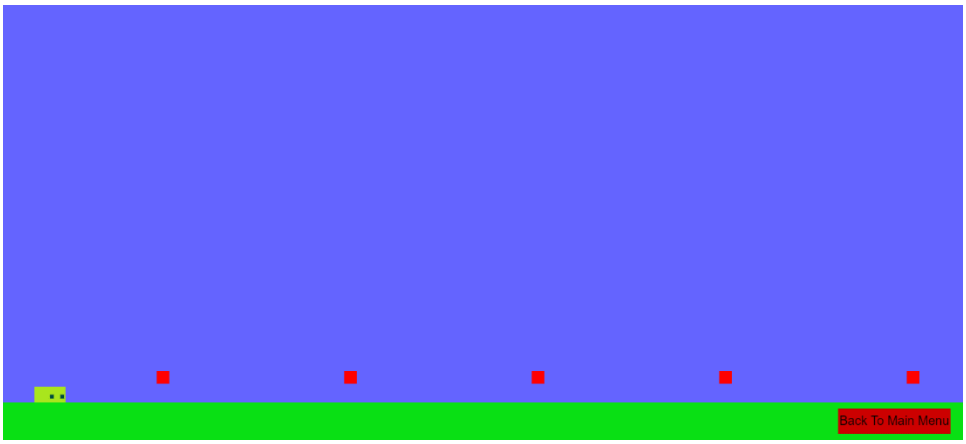
Here is the code which updates and spawn obstacles. It works similarly to the pipe spawning mechanism from the flappy bird game where it checks to see if dividing the FrameCount by a constant gives a remainder. If it does not, then it spawns an obstacle which moves from the right of the screen to the left. The code also removes any obstacles that are offscreen. This is to make sure that memory is not taken up by obstacles that are not being used anymore.



Here is the result of this implementation. Now every 30 frames, an obstacle spawns and once it is offscreen, it is removed to free up space.

```
//Flying Obstacle
class FlyingObstacle extends ObstacleBlock{
  constructor(){
    super()
    this.y = GroundHeight - 50
    this.Id = 1
  }
}
```

To add variety with the obstacles, I decided to create a new obstacle which floats in the air. I did this by using inheritance as it saves time and is a more optimal solution rather than rewriting code. The super() function within the constructor method allows the flying obstacle to inherit attributes and methods from the Obstacle which I renamed to “ObstacleBlock”. I also use polymorphism in this example by changing the Id to 1 for this obstacle and changing the y position of the obstacle.



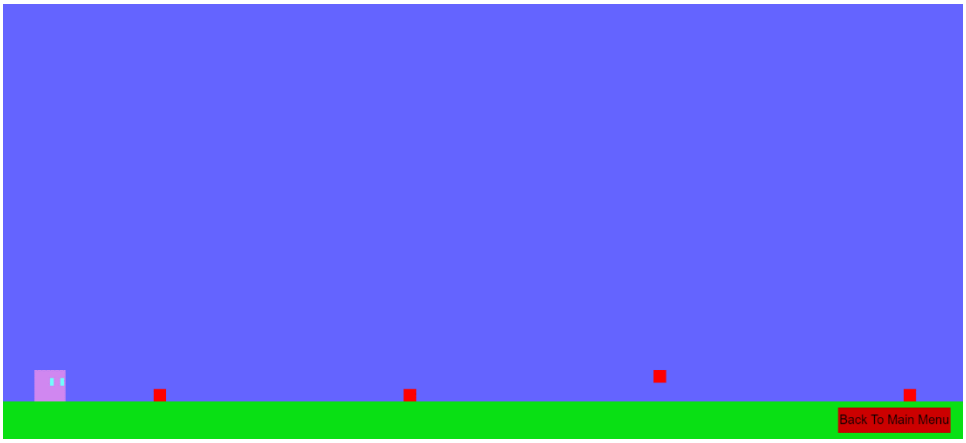
Here is the flying obstacle in the program. As expected, the obstacle is in the air which allows the player to choose whether to either jump over or crouch under the obstacle.

```
if(FrameCount % 80 == 0){
  let NewItem = Math.floor(Math.random()* AllObstacles.length)
  obstacles.push(new AllObstacles[NewItem])
}
```

In this game, I want the different type of obstacles to spawn randomly so I created an array which has a list of all the current obstacles within the game. I do not have a screenshot of it in this stage, but the array looks like this:

Let AllObstacles = [“ObstacleBlock”,”FlyingObstacle”]

Using this array, I updated the spawning mechanic of the program. First, it now spawns an obstacle every 80 frames instead of 30. I made this change so that the player and A.I have enough time in-between obstacles to react to the next one. Next change involves the AllObstacles array. First, I create a new variable called NewItem which is a random number based on the length of the array. I used the Math.floor() method instead of the Math.round() as it will always round down. This is needed to make sure the random number generated is always in-between 0 and the length of the array minus 1. After that, the NewItem number is used as the index to create a new obstacle based on the index of the array.



After implementing that, both obstacles can now spawn randomly as shown in this screenshot. Now if I were to add a new obstacle into this program, all I would have to do is add it to the AllObstacles list.

```
//for simple collision detection between non rotated rectangles or squares
function touching(item1,item2){
  if(item1.x+item1.width >= item2.x && item1.x <= item2.x+item2.width && item2.y <= item1.y+item1.height&& item1.y <= item2.y+item2.height){
    return true
  }
}
```

After getting the obstacle spawning to work, I decided to create a function and add it to the Utils.js file as it is an important function within this project. The function is the ”touching” function I mentioned in the design section of this report. It takes in 2 parameters which must have an x and y position as well as a width and height as it uses these attributes to check if the 2 items are colliding. If they are, then it returns true.

```
  //check if any of the runners have touched the obstacles. If so, rip
  obstacles.forEach(obstacle=>{
    if(touching(runner,obstacle)){
      runner.isDead = true
    }
  })
})
//kill any runners if needed
Runners.forEach((runner,i)=>{
  if(runner.isDead){
    Runners.splice(i,1)
  }
})
```

I used this function to check if the obstacles are colliding with any of the runners. If they are, the IsDead attribute of the Runner is set to true. The program then loops through the Runners list and checks if the IsDead attribute is set to true. If it is, then it is removed from the array.

```
//in this game, the A.I will see 2 obstacles ahead instead of 1
//first one
let closest1 = obstacles[0]
let closestD1 = Infinity
//second
if(obstacles.length > 1){
}
let closest2 = obstacles[1]
let closestD2 = Infinity
for(let i = 0; i < obstacles.length; i++){
  let d = obstacles[i].x - this.x
  // set the first one
  if (d < closestD1 && d > -closest1.width){
      closest1 = obstacles[i]
      closestD1 = d
  }else if(d < closestD2 && d > -closest2.width && closest1 !== obstacles[i]){
      // set the second one assuming its not the same as the first one
      closest2 = obstacles[i]
      closestD2 = d
  }
}
```

After adding the collision detection between the runners and obstacles, I decided to add the A.I into the runner class. Similarly, to the bird class, I added a new attribute called brain which will store the neural network in the runner. I also added a think method which takes in an array as an input and will be used to store and update the inputs every frame. For this game, I wanted the A.I to see not just the closest obstacle, but the first 2 closest ones. This should make it easier for the A.I to play as it has more information to work with. The screenshot above shows the process of selecting the first 2 closest obstacles by checking the distance between the player and the obstacles. I do not have a screenshot of this, but the next part of the think method contains the following code:

this.inputs[0] = this.y

this.inputs[1] = this.Yspeed

If(this.IsCrouching){

this.inputs[2] = 1

}else{

this.inputs[2] = 0

}

If(this.IsInAir){

this.inputs[3] = 1

}else{

this.inputs[3] = 0

}

this.inputs[4] = this.y - (closest1.y)

this.inputs[5] = this.y - (closest1.y+closest1.height)

this.inputs[6] = this.x - (closest1.x)

this.inputs[7] = this.x - (closest1.x+closest1.width)

this.inputs[8] = closest1.Id

this.inputs[9] = this.y - (closest2.y)

this.inputs[10] = this.y - (closest2.y+closest2.height)

this.inputs[11] = this.x - (closest2.x)

this.inputs[12] = this.x - (closest2.x+closest2.width)

this.inputs[13] = closest2.Id

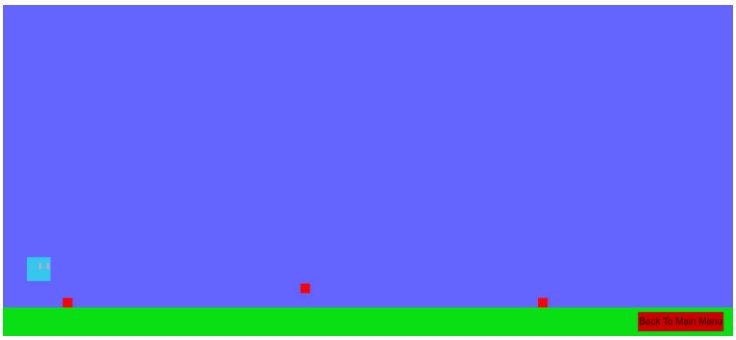
This code sets the inputs for the runner and like in the bird class, it will be called every frame so it can always be accurate.

```
this.think(obstacles)
let outputs = NeuralNetwork.FeedForward(this.brain,this.inputs)
if(outputs[0] > 0.5){
  this.Jump()
}
if(outputs[1] > 0.5){
  this.Crouch()
}
if(outputs[2] > 0.5){
  this.UnCrouch()
}
```

Within the update method, I called the think method every frame as well as add some functionality which allows the A.I to control the runner. Now using the inputs given, the A.I can decide to jump, crouch or uncrouch.

```
let obstacles = []
obstacles.push(new FlyingObstacle)
obstacles.push(new FlyingObstacle)
```

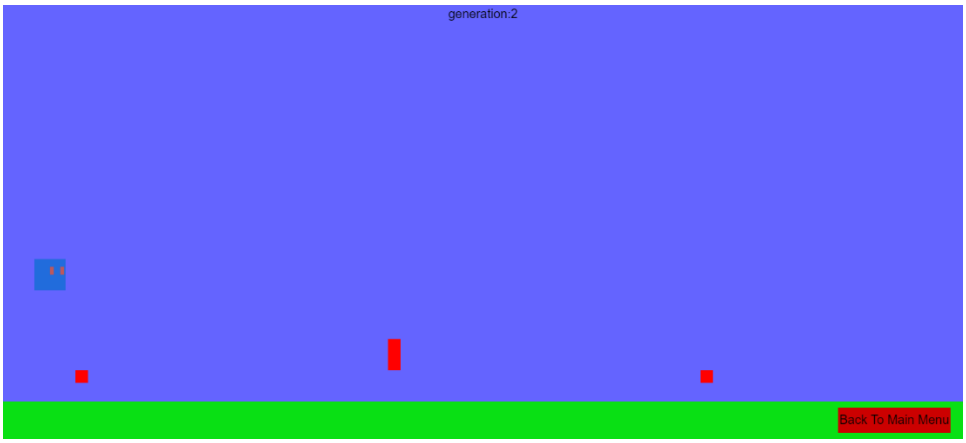
Upon reloading the page, an error appeared which had something to do with trying to find the first 2 closest obstacles. To fix this problem, I simply pushed 2 FlyingObstacles at the start of the program which fixed the error.



Now the A.I works within the game and now I can work on the restart function

```
let SavedRunners = []
function Restart(){
  //next generation
  generation++
  //reset everything
  obstacles = []
  FrameCount = 0
  //find the best runner
  let BestRunner = SavedRunners[0]
  let CurrentBestScore = SavedRunners[0].score
  //get the best bird
  for(let i = 0; i < SavedRunners.length; i++){
      if(SavedRunners[i].score > BestRunner.score){
          BestRunner = SavedRunners[i]
      }
  }
  //save the brain
  localStorage.setItem("bestRunnerBrain",JSON.stringify(BestRunner.brain))
  for(let i = 0; i < population; i++){
    Runners.push(new Runner)
  }
  for(let i = 0; i < Runners.length; i++){
    Runners[i].brain = JSON.parse(localStorage.getItem("bestRunnerBrain"))
    NeuralNetwork.mutate(Runners[i].brain,0.25)
  }
  SavedRunners = []
  //prevents the program from breaking
  obstacles.push(new FlyingObstacle)
  obstacles.push(new FlyingObstacle)
}
```

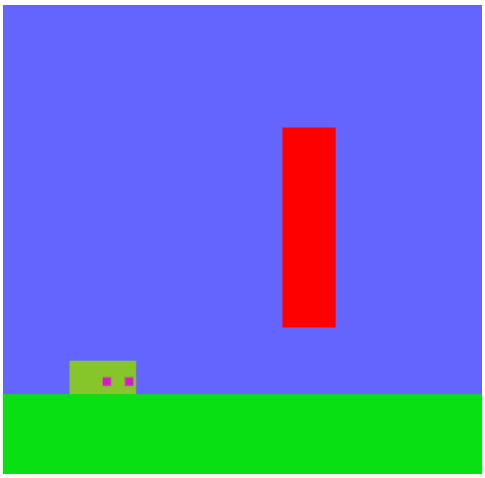
Similarly, to the flappy bird restart function, Cube runner works the exact same as the 2 games are similar. It first deletes all remaining obstacles from the game. It then uses a linear search to find the cube with the highest score, after that, it saves its neural network so it can copy it onto the new population of runners. The neural networks are then mutated for variety and the 2 obstacles are then pushed into the obstacle array.



Here is the result of the restart function. Now the A.I should improve with each generation. To see what generation the program is currently on, I added some text which displays the generation number.

```
//Long Flying Obstacle
class LongFlyingObstacle extends ObstacleBlock{
  constructor(){
    super()
    this.y = GroundHeight - 200
    this.height = 150
    this.Id = 2
  }
}
```

I created a new obstacle which would be a longer version of the flying obstacle. The idea with this obstacle is to hope that they crouch under it



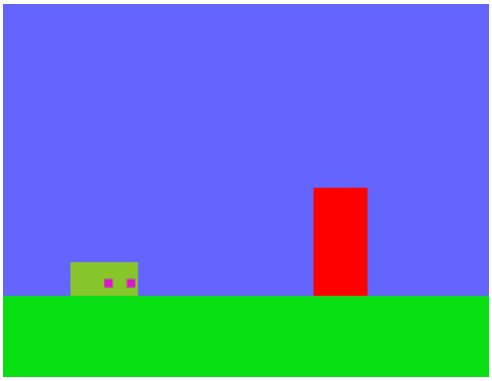
Here is an example of this new obstacle I have created. As I mentioned previously, it is advised that the A.I crouches under the obstacle instead of trying to jump over as it will be exceedingly difficult to do so.

```
think(obstacles){
  let closest1 = obstacles[0]
  let closestD1 = Infinity
  for(let i = 0; i < obstacles.length; i++){
    let d = obstacles[i].x - this.x
    if (d < closestD1 && d > -closest1.width){
        closest1 = obstacles[i]
        closestD1 = d
    }
  }
  //set the inputs
  this.inputs[0] = this.y
  this.inputs[1] = this.Yspeed
  if(this.IsCrouching){
    this.inputs[2] = 1
  }else{
    this.inputs[2] = 0
  }
  if(this.IsInAir){
    this.inputs[3] = 1
  }else{
    this.inputs[3] = 0
  }
  this.inputs[4] = this.y - closest1.y
  this.inputs[5] = this.y - (closest1.y+closest1.height)
  this.inputs[6] = this.x - closest1.x
  this.inputs[7] = this.x - (closest1.x+closest1.width)
  this.inputs[8] = closest1.Id
}
```

While testing the A.Is to make sure I had gotten the inputs right, I realised that sometimes, the A.Is would still struggle to play the game. At first, I thought it may have been too difficult for them to play so I decided to reduce the difficulty. This however did not work so I decided to change the inputs so it could only see the first closest obstacle instead of the first 2 and this fixed the problem. The image above shows the new inputs that the A.I has.

```
//simple obstacle
class ObstacleBlock{
  constructor(){
    this.x = box.width
    this.y = GroundHeight - 20
    this.width = 40
    this.height = 20
    this.color = "red"
    this.Id = 0
  }
  draw(){···
  update(){···
}
//Flying Obstacle. You can go over or under this one.
class FlyingObstacle extends ObstacleBlock{
  constructor(){
    super()
    this.y = GroundHeight - 50
    //this.height = 50
    this.Id = 1
  }
}
//Long Flying Obstacle, jumping over this one is not advised
class LongFlyingObstacle extends ObstacleBlock{
  constructor(){
    super()
    this.y = GroundHeight - 200
    this.height = 150
    this.Id = 2
  }
}
//Taller Obstcale, harder to jump over
class TallerObstcale extends ObstacleBlock{
  constructor(){
    super()
    this.height = 80
    this.y = GroundHeight - 80
    this.Id = 3
  }
}
```

I then created a new obstacle called “TallerObstacle” to challenge the A.Is once again. The image above shows the code for it, like the others, it uses inheritance and polymorphism where its parent is the ObstacleBlock.



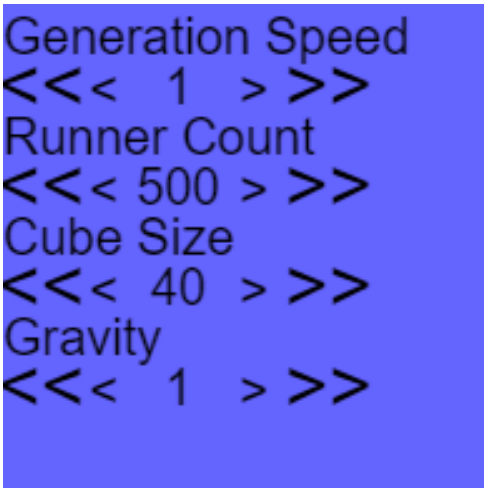
Here is an example of the new TallerObstacle

```
//stats for game that can be edited
let GenSpeed = new GameStat(0,20,cycles,"Generation Speed",0,100,1,buttons,"black",true)
let RunnerCount = new GameStat(0,60,population,"Runner Count",1,1000,1,buttons,"black",true)
let CubeSizes = new GameStat(0,100,CubeSize,"Cube Size",20,120,1,buttons,"black",true,10)
let Grav = new GameStat(0,140,gravity,"Gravity",1,5,0.1,buttons,"black",true,1)
```

Now that I have added all the main game functionality, I decided to start adding the editable stats to the game. I reused the Gamestat object I had made for flappy bird and added 4 new editable stats that the user can change to customize their experience. These stats are:

- GenSpeed – speed up or slow down the generation
- RunnerCount – choose the number of runners per generation
- CubeSizes – select the size of the runners
- Grav – choose the gravity present in the game

These so far are the only ones I have decided to add, however, I may add more later.



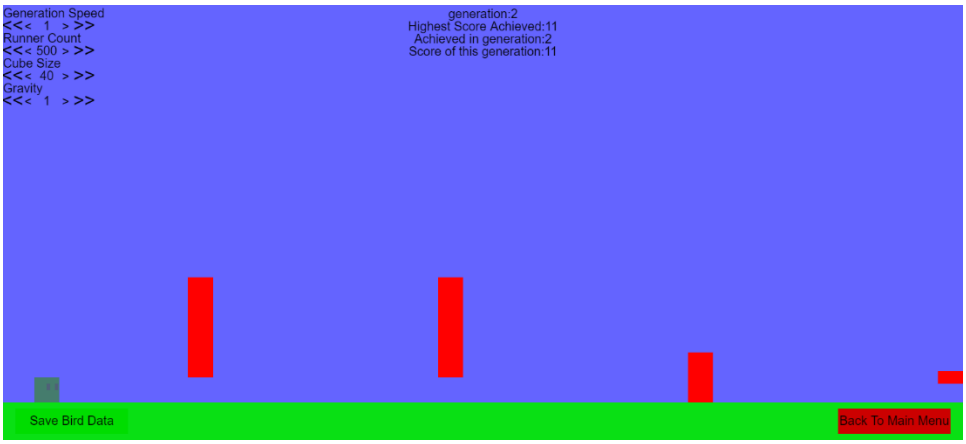
Here is the outcome of adding these stats into the program. Now they will appear in the top left of the screen where the user can change them.

```
pen.fillText("generation:"+generation,box.width/2,15)
pen.fillText("Highest Score Achieved:"+TotalScore,box.width/2,35)
pen.fillText("Achieved in generation:"+HighestGen,box.width/2,55)
pen.fillText("Score of this generation:"+score,box.width/2,75)
if(GameRecentlySaved){
  pen.fillText("game saved",box.width/2,95)
}
```

I added the recently saved feature into this program as a way of telling the user that the game has been saved.

```
//save Runner data
buttons.push(new Button(
  20,
  (box.height) - 50,
  180,
  40,
  "#00DD00",
  "Save Runner Data",
  20,
  "black",
  function () {
    this.color = "#00FF00";
    this.textcolor = "#FFFFFF";
  },
  () => {
    let data = {
      brain:JSON.parse(localStorage.getItem("bestRunnerBrain")),
      HighestScore:score,
      cubesize:CubeSize,
      grav:gravity
    }
    localStorage.setItem("SavedDataCubeRunner",JSON.stringify(data))
    console.log("saved")
    GameRecentlySaved = true
  }
))
```

Here is the save runner data button. It is the same as the flappy bird save button with a few changes



And here is the result. One minor problem I encountered when adding this button is realising that the colour of the button is the exact same colour as the ground. As a result of this, I decided to change the ground so you can easily distinguish the button from the ground.

```
//grass
class Grass{
  constructor(x){
    this.x = x
    this.y = GroundHeight
    this.width = grasswidth
    this.height = Math.random() * (40 - (20)) + (20)
    this.color = "rgba(9,224,20,1)"
  }
  draw(){
    pen.fillStyle = this.color
    pen.fillRect(this.x,this.y,this.width,this.height)
  }
}
```

To change the ground, I decided to add grass onto the floor. Here I created a grass class which only takes in 1 parameter which will determine its x value. The height of the grass will be a random height and the width of the grass will be determined on the grasswidth variable.

```
//grass columns
let grasswidth = 10
let cols = Math.round(box.width/grasswidth)
```

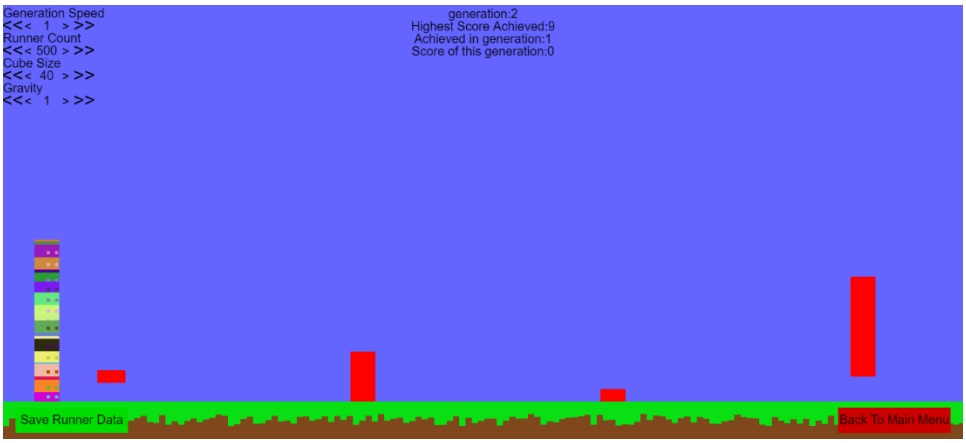
Here is where the grasswidth and cols variable are declared. The cols variable is determined by the width of the screen divided by the grasswidth. The result is then rounded to the nearest whole number.

```
let grass = []
for(let i = 0; i < cols; i++){
  grass.push(new Grass(i*grasswidth))
}
```

This for loop simply adds the grass so it fills the width of the program.

```
//Floor and text
pen.fillStyle = "#80471C"
pen.fillRect(0,GroundHeight,box.width,60)
grass.forEach(grass=>{
  grass.draw()
})
```

I updated the floor code by changing the colour of the ground to a brownish colour and by drawing the grass.



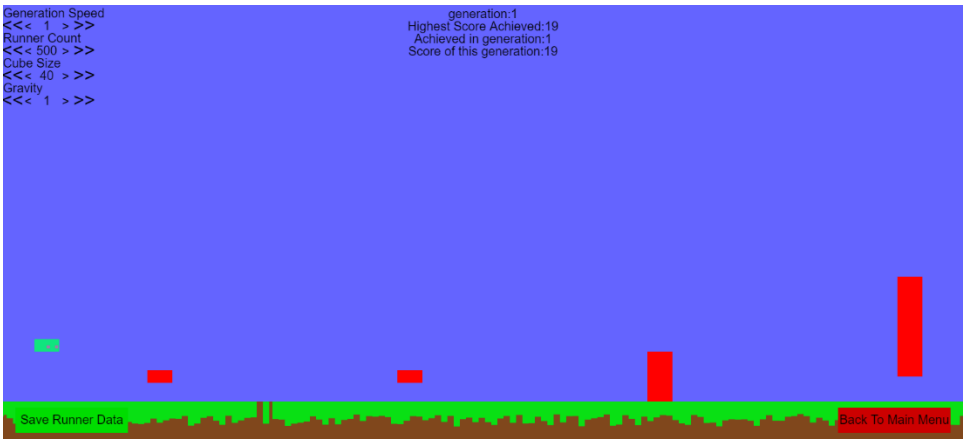
Here is the result of adding grass into the program. One problem I have with it however is that the grass is just stationary and is not moving which destroys the illusion that the runners are running through the game.

```
//grass
class Grass{
  constructor(x){
    this.x = x
    this.y = GroundHeight
    this.width = grasswidth
    this.height = Math.random() * (40 - (20)) + (20)
    this.color = "rgba(9,224,20,1)"
  }
  draw(){
    pen.fillStyle = this.color
    pen.fillRect(this.x,this.y,this.width,this.height)
  }
  update(){
    this.x -=5
  }
}
```

To fix this visual problem, I added an update method which simply moves the grass by a constant value.

```
    //update the grass
    grass.forEach((Grasss,i)=>{
      if(Grasss.x <= -Grasss.width){
        grass.splice(i,1)
        grass.push(new Grass(box.width))
      }
      Grasss.update()
    })
```

Here is the code that updates each grass object within the grass array. It also checks if a grass object has moved off screen and if so, it removes it and adds another grass object.



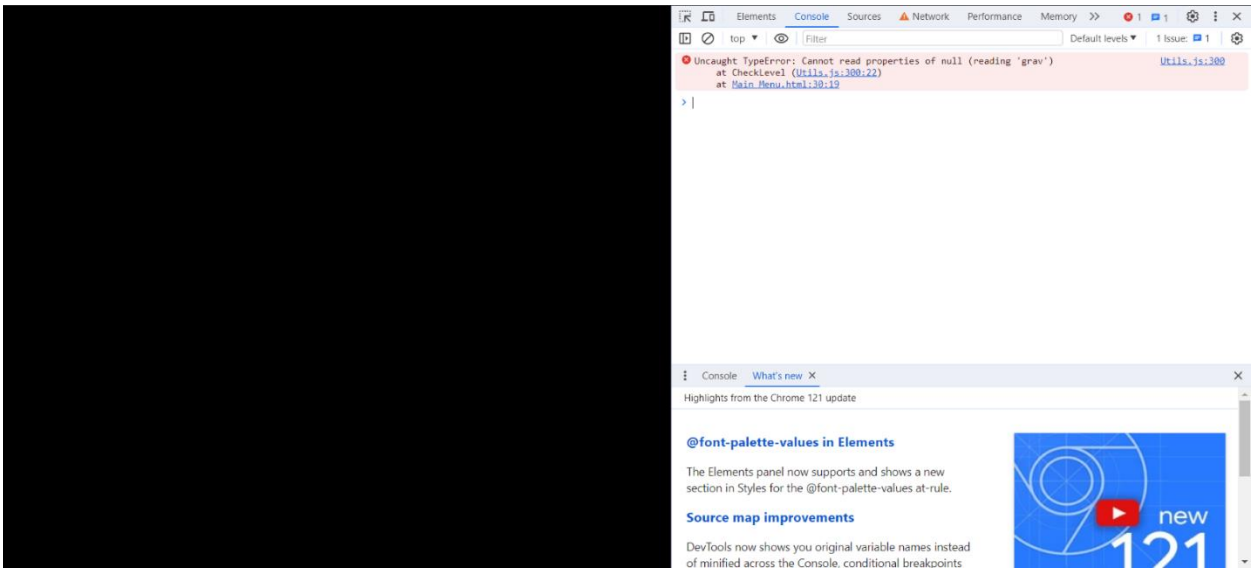
Here is the result of this implementation. One new problem arose when implementing this and that are the holes which do not contain any grass objects.

```
let grass = []
for(let i = 0; i < cols+5; i++){
  grass.push(new Grass(i*grasswidth))
}
```

This was an easy fix however as all I had to do was slightly change the for loop of initialising the grass objects which is shown in the image above.

Updating the main menu #2

Once I finished creating the Cube Runner A.I game, I was going to add it onto the main menu. However, upon opening the main menu, this error appeared.



This error appeared due to me being on a different device which did not have any saved data so as a result, the program failed. This was an easy bug to fix.

```
let AILevels = {
    FlappyBird: localStorage.getItem("SavedDataFlappyBird") == null ? null : CheckLevel("FlappyBird",JSON.parse(localStorage.getItem("SavedDataFlappyBird"))),
    CubeRunner: localStorage.getItem("SavedDataCubeRunner") == null ? null : CheckLevel("CubeRunner",JSON.parse(localStorage.getItem("SavedDataCubeRunner")))
}
```

The change I had to make was to the AI levels variable. Now it uses a ternary operator to check if a value is present if so then it assigns the value to the variable. If not, then it will assign a null value to the variable. I also added the cube runner variable which does the same for the cube runner game.

```
function CheckLevel(GameName,Data){
  let level = 0
  switch(GameName){
    case "FlappyBird":
      level = ((Data.grav*Data.PipeSpeed)/250)+(((1/Data.grav)*Data.diff)/0.2)+((1/Data.Spacing)/(500))
      level = Data.HighestScore/level
      level /=100
    break
    case "CubeRunner":
      level = Math.max((((Data.cubesize/Data.grav)+(Data.cubesize/Data.grav))*Data.HighestScore),100)
    break
  }
  return Math.round(level)
}
```

I also went into the utils file and updated the Checklevel function which now supports the cube runner A.I game. Like I mentioned with the Flappy bird version, it may change later throughout development.

I also added the cube runner to the training section of the main menu with the text “Teach AI cubes to avoid obstacles and run the farthest distance”

```
//Cube Runner Game
buttons.push(new Button(
  (box.width / 3) - 400,//X
  (box.height / 2) - 50,//Y
  400,//Width
  100,//Height
  ColorScheme.RegularButtonColors,//Color
  "Cube Runner",//text
  65,//text size
  ColorScheme.RegularButtonText,//text color
  function () {
    this.color = ColorScheme.RegularButtonColorsLight;
    this.textcolor = ColorScheme.RegularButtonTextLight;
    TextDescription = "Teach AI cubes to avoid obstacles and run the farthest distance"
  },//function when mouse is over button
  () => {
    window.open("Cube Runner AI.html")
  }//function when mouse clicks button
))
```

Here is the code for the button.

I have also removed the “CoolBackground” function from the main menu.html file as I have decided to rewrite the code to a new file.

Creating Background.js

This file will be where I store the animated backgrounds for the main menu

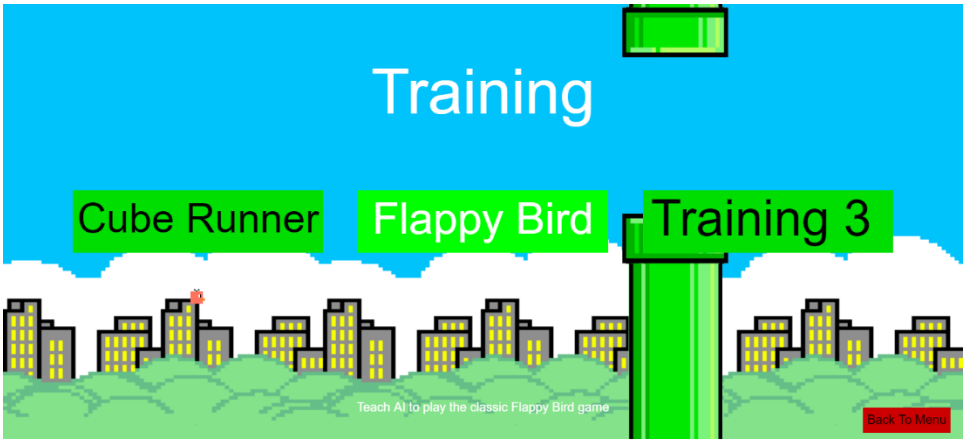
```
var box = document.getElementById("box")
box.width = innerWidth
box.height = innerHeight
let pen = box.getContext("2d")
//flappy bird game
let gravity = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).grav
let PipeSpeed = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).PipeSpeed
let PipeFreq = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).diff
let spacing = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).Spacing
let BG = new Image(box.width,box.height)
BG.src = "Flappy Bird BG.png"
//pipes
let pipes = []
pipes.push(new Pipe)
//bird
let bird = new Bird()
bird.brain = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).brain
//frame count
let FrameCount = 0
function FlappyBirdGameplay(){
    console.log(FrameCount % Math.floor((500 / (PipeSpeed*PipeFreq))))
    //updates
    FrameCount++
    //update the bird
    bird.update()
    //update the pipes
    for(let i = 0; i < pipes.length; i++){
        pipes[i].update()
        //remove the pipe if it's off the screen
        if(pipes[i].x < -pipes[i].width){
            pipes.splice(i,1)
        }
    }
    //push a new pipe into the list
    if(FrameCount % Math.floor((500 / (PipeSpeed*PipeFreq))) == 0){
        pipes.push(new Pipe)
    }
    //drawing
    //draw the background
    pen.fillStyle = "rgba(100,100,255,1)"
    pen.fillRect(0,0,box.width,box.height)
    pen.drawImage(BG, 0, 0, box.width, box.height+5)
    //draw everything else
    //draw the bird
    bird.draw()
    //draw the pipes
    pipes.forEach(pipe=>{
        pipe.draw()
    })
    requestAnimationFrame(FlappyBirdGameplay)
}
```

I added the “CoolBackground” function into this file but renamed it to “FlappyBirdGameplay” after adding this. I went back to the main menu.html file and called this function to see if it worked.

This is the following result, Now the game plays in the background. All I need to do now is make sure the game plays only when the Flappy bird A.I button is hovered over.

```
function loadBackground(){
  //draw the background
  switch (ButtonHoveredOver) {
    case "FlappyBird":
    FlappyBirdGameplay()
    break;
    default:
    pen.fillStyle = ColorScheme.BackgroundColor
    pen.fillRect(0, 0, box.width, box.height)
    break;
  }
}
function Loop() {
  loadBackground()
```

I updated the main menu by adding a new function called loadBackground which uses a switch statement to find out what button the mouse has hovered over then deciding what background to load based on that information.



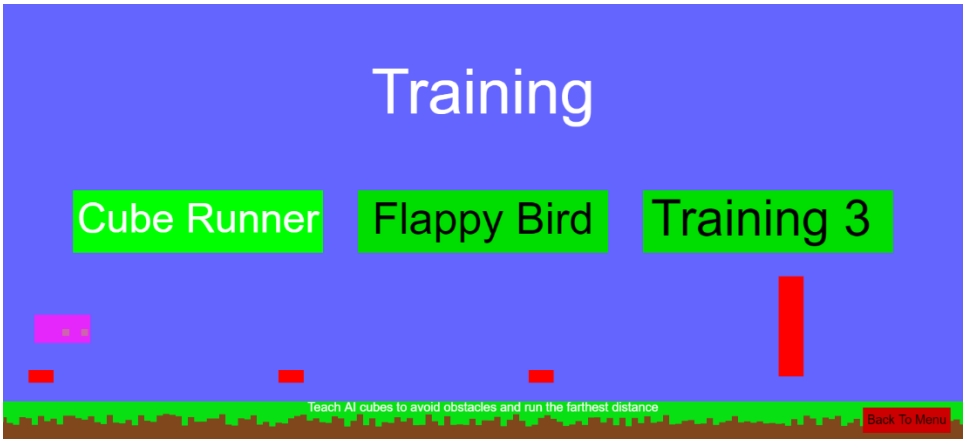
Here is the result of that implementation. Now hovering over the Flappy Bird button will play flappy bird in the background and when the mouse is off the button, it will go back to displaying the normal background.

```
//cube runner game
if(localStorage.getItem("SavedDataCubeRunner") != null){
    var gravity = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).grav
    var CubeSize = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).cubesize
    //ground
    var GroundHeight = (box.height) - 60
    //grass columns
    var grasswidth = 10
    var cols = Math.round(box.width/grasswidth)
    //obstacles
    var AllObstacles = [ObstacleBlock, LongFlyingObstacle, FlyingObstacle, TallerObstcale]
    var obstacles = []
    obstacles.push(new FlyingObstacle)
    //grass
    var grass = []
    for(let i = 0; i < cols+5; i++){
      grass.push(new Grass(i*grasswidth))
    }
    //runner
    var runner = new Runner()
    runner.brain = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).brain
}
function CubeRunnerGameplay(){
    //updating
     FrameCount++
     //runner
     runner.update()
    //Obstacles
    obstacles.forEach(obstacle=>{
        obstacle.update()
    })
    if(FrameCount % 80 == 0){
        let NewItem = Math.floor(Math.random()* AllObstacles.length)
        obstacles.push(new AllObstacles[NewItem])
    }
    //update the grass
    grass.forEach((Grasss,i)=>{
        if(Grasss.x <= -Grasss.width){
          grass.splice(i,1)
          grass.push(new Grass(box.width))
        }
        Grasss.update()
    })
    //drawing
    //draw the background
    pen.fillStyle = "rgba(100,100,255,1)"
    pen.fillRect(0,0,box.width,box.height)
    runner.draw()
    //Obstacles
    obstacles.forEach(obstacle=>{
        obstacle.draw()
    })
    //Floor
    pen.fillStyle = "#80471C"
    pen.fillRect(0,GroundHeight,box.width,60)
    grass.forEach(grass=>{
        grass.draw()
    })
}
```

Here is the code which will display the gameplay for cube runner. To do this, I had to move the runner class, the grass class and all the obstacle classes to the utils.js file.

```
function loadBackground(){
  //draw the background
  switch (ButtonHoveredOver) {
    case "FlappyBird":
      if(localStorage.getItem("SavedDataFlappyBird") != null){
        FlappyBirdGameplay()
      }else{
        pen.fillStyle = ColorScheme.BackgroundColor
        pen.fillRect(0, 0, box.width, box.height)
      }
    break;
    case "CubeRunner":
    if(localStorage.getItem("SavedDataCubeRunner") != null){
      CubeRunnerGameplay()
    }else{
      pen.fillStyle = ColorScheme.BackgroundColor
      pen.fillRect(0, 0, box.width, box.height)
    }
    break
    default:
    pen.fillStyle = ColorScheme.BackgroundColor
    pen.fillRect(0, 0, box.width, box.height)
    break;
  }
}
```

I have also updated the loadBackground function so it can display the cube runner gameplay too. I have also added a check to make sure that there is data present before drawing the background. If there is not, then it will just draw the blank background.



Here is the result of adding that feature into the program, now whenever the user hovers over the cube runner game plays in the background like the flappy bird background.

Updating the main menu #3

After fixing the background, I decided to work on implementing the settings and stats for the program as these are features of my program which will improve the experience for the user.

```
//like game stats but for settings
class SettingStat{
  constructor(x,y,statname,options,optionFunc,color){
    this.x = x || 0
    this.y = y || 0
    this.statname = statname || "setting"
    this.options = options || ["on","off"]
    this.optionFunc = optionFunc || [function(){console.log("true")}, function(){console.log("false")}]
    this.color = color || "black"
  }
  draw(){
    pen.textBaseline = "alphabetic"
    pen.font= "30px Arial"
    pen.textAlign = "center"
    pen.fillStyle = this.color
    pen.fillText(this.statname,this.x,this.y)
  }
}
```

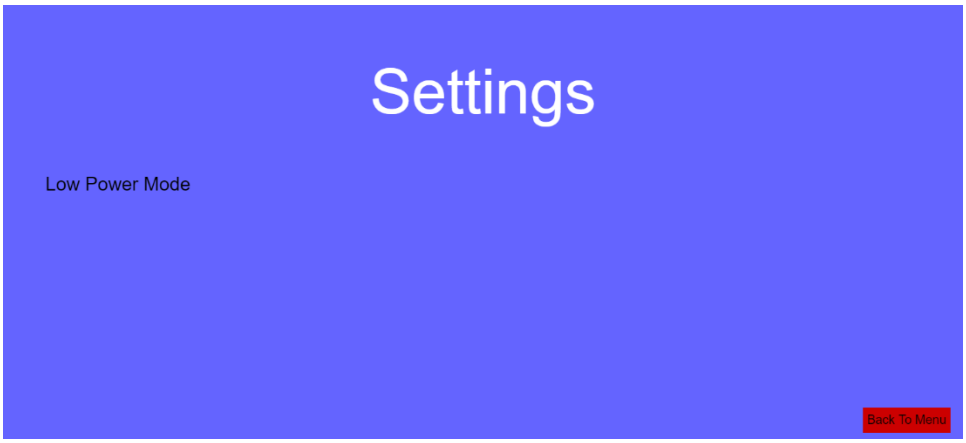
I first made a new class in the Utils.js file called SettingStat. It will take in 6 parameters, and it will function in a comparable way to the GameStat class I implemented a while ago. I also added a draw method which so far will just print the name of the stat on the screen.

```
case "Settings":
MenuTitle = "Settings"
settingsstats.push(new SettingStat(
  (box.width/4)-200,
  (box.height / 2) - 50,
  "Low Power Mode",
  ["On","Off"],
  [function(){console.log("on")},function(){console.log("off")}]
))
//Back to Options button
buttons.push(new Button(
    (box.width)-160,//X
    (box.height) - 50,//Y
    140,//Width
    40,//Height
    ColorScheme.QuitButtonColors,//Color
    "Back To Menu",//text
    20,//text size
    ColorScheme.RegularButtonText,//text color
    function () {
      this.color = ColorScheme.QuitButtonColorsLight;
      this.textcolor = ColorScheme.RegularButtonTextLight;
    },//function when mouse is over button
    () => {
      LoadMenu("Options")
    }//function when mouse clicks button
  ))
break
```

Back in the Main Menu.html file, I added a new array called settingsstats which will be the same as the buttons array but just for the SettingStat objects. I then pushed a new SettingStat which will eventually become the low power mode setting. The parameters that I had to give for this class are:

- X and Y position which can be a float value or an integer
- The name of the stat which will be a string

- An array of options where the options within the array could be any value
- An array of functions which will correspond to the array options from the options parameter
- The colour of the text which should be a colour value



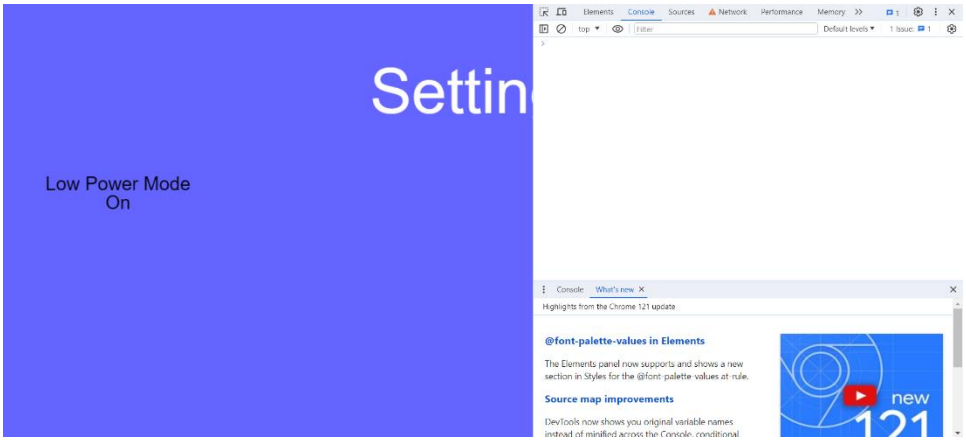
Here is the result of adding the low power mode object in the program. It works as intended without any bugs or errors so I can move on to adding functionality to it.

```
//like game stats but for settings
class SettingStat{
  constructor(x,y,statname,options,optionFunc,color){
    this.x = x || 0
    this.y = y || 0
    this.statname = statname || "setting"
    this.options = options || ["on","off"]
    this.optionFunc = optionFunc || [function(){console.log("true")}, function(){console.log("false")}]
    this.color = color || "black"
    this.pointer = 0
  }
  draw(){
    pen.textBaseline = "alphabetic"
    pen.font= "30px Arial"
    pen.textAlign = "center"
    pen.fillStyle = this.color
    pen.fillText(this.statname,this.x,this.y)
    pen.fillText(this.options[this.pointer],this.x,this.y+30)
  }
  //move to the next item in the list
  MoveNext(){
    //go back to the first item if we're at the end
    if(this.pointer >= this.options.length-1){
      this.pointer = 0
    }else{
      this.pointer++
    }
  }
  //move back in the list
  MoveBack(){
    //go back to the last item if we're at the start
    if(this.pointer <= 0){
      this.pointer = this.options.length-1
    }else{
      this.pointer--
    }
  }
}
```

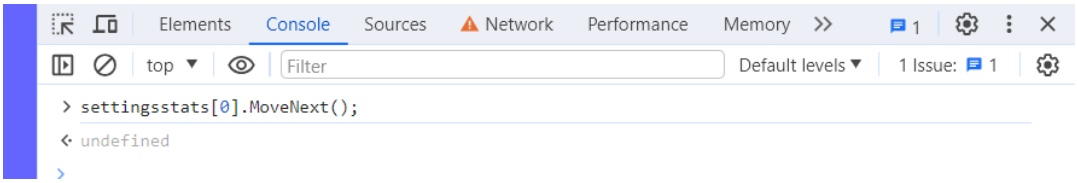
I updated the draw method by making it, so it displays one of the options below the name of the setting. I also added 2 more methods known as MoveBack and MoveNext which will move the pointer through the options array. It also loops back to the front/back of the array if it is at the end/start of it.

```
//low power mode
settingsstats.push(new SettingStat(
  (box.width/4)-200,
  (box.height / 2) - 50,
  "Low Power Mode",
  ["On","In the middle","Off"],
  [function(){console.log("on")},function(){console.log("off")}]
))
```

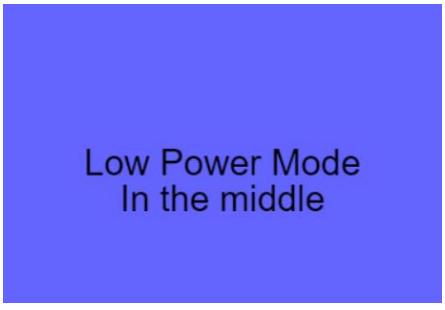
To test if this feature were working as intended, I added a new item into the options array called “In the middle” to see if this would work with more than 2 items.



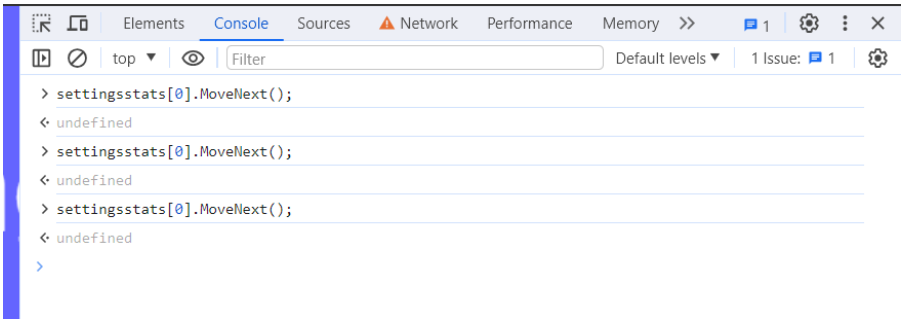
When reloading the page, I saw that the option I was currently on was displayed at the bottom which means that the updated draw method had worked.



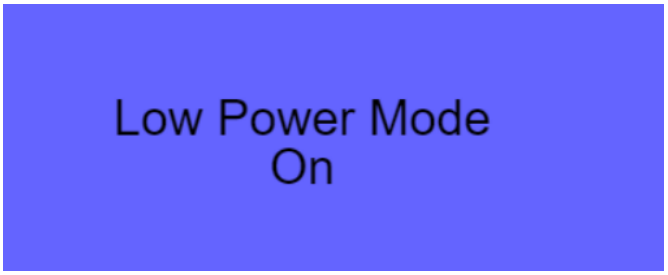
Using the console, I called the MoveNext method on the settingsstat which should change the “On” text to “In the middle”.



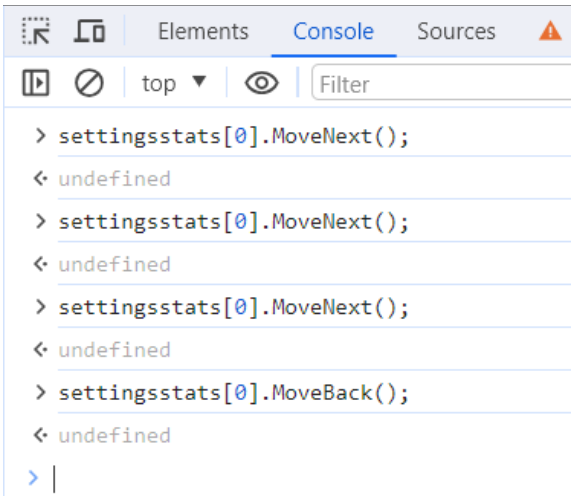
This was the output which means it worked as expected.



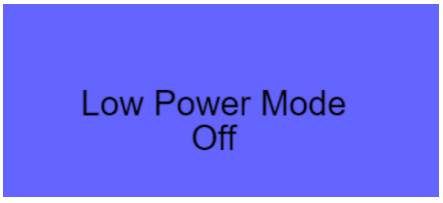
I then called it 2 more times which should now loop over and now display “On”.



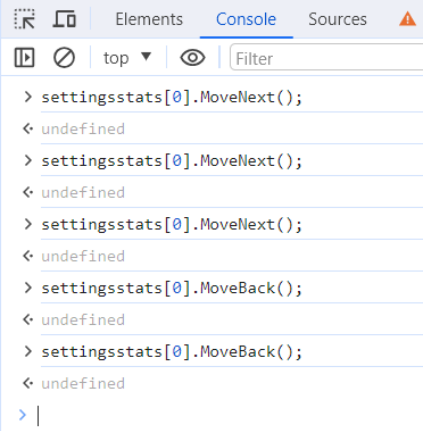
As expected, it displays on and that means going from the last item to the first item works.



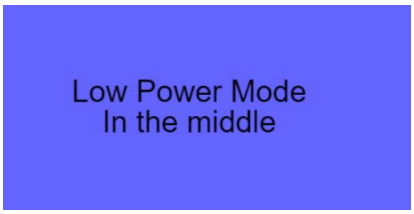
I then used the MoveBack method which should now loop back to the last item which is “Off”.



It now displays the “Off” option which means it worked as it is supposed to.



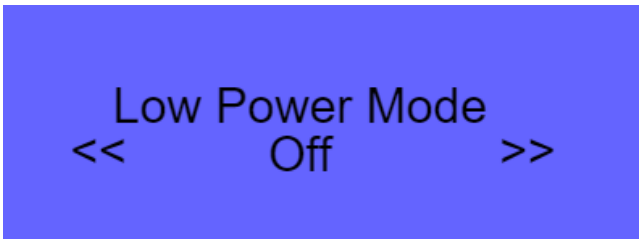
Finally, I called the MoveBack method again which should now display “In the middle”.



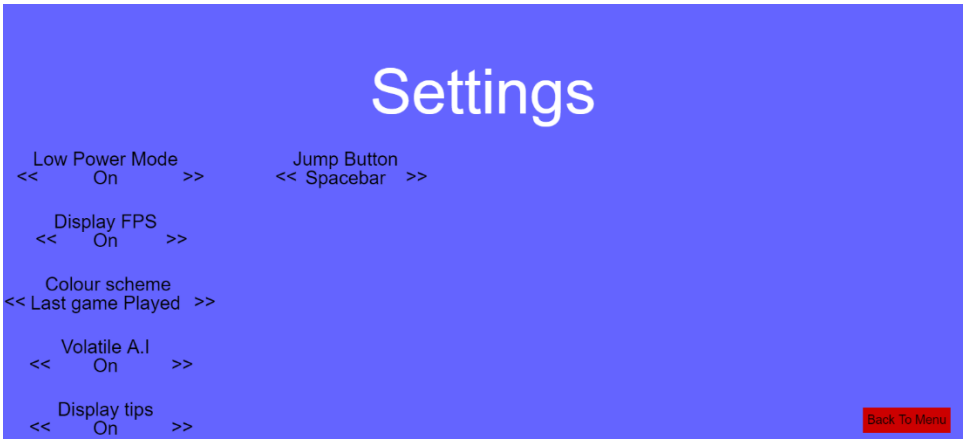
And as expected, it displays “In the middle” which means that the 2 methods I have added worked.

```
buttons.push(new Button(
  this.x + (this.statname.length*9),//X
  this.y+5,//Y
  30,//Width
  30,//Height
  "rgba(0,0,0,0)",//Color
  ">>",//text
  30,//text size
  this.color,//text color
  function () {
    this.textcolor = "#FFFFFF";
  },//function when mouse is over button
  () => {
    this.MoveNext()
  }//function when mouse clicks button
))
buttons.push(new Button(
  this.x - (this.statname.length*10),//X
  this.y+5,//Y
  30,//Width
  30,//Height
  "rgba(0,0,0,0)",//Color
  "<<",//text
  30,//text size
  this.color,//text color
  function () {
    this.textcolor = "#FFFFFF";
  },//function when mouse is over button
  () => {
    this.MoveBack()
  }//function when mouse clicks button
))
```

Now I added 2 buttons which when pressed, will call the MoveBack and MoveNext methods.



Here is the result of adding the buttons, Now pressing the << and the >> button will call the MoveBack and MoveNext methods respectively. Now that this works, I can add more settings in the program.



Here I have added the following settings:

- Low power mode- when turned on, it will disable certain power intensive features such as the animated background.
- Display FPS – displays the frame rate of the program
- Colour scheme – changes the colour scheme of the menu
- Volatile A.I - when turned on, A.Is and generations will be deleted upon refresh unless they have been saved manually
- Display tips – when hovering over buttons, tips will display informing the user about whatever they've pressed
- Jump button – decides on what key should be used to jump in flappy bird and cube runner

```
//choosing the colour scheme
let SelectedColorScheme = "Default"
let ColorScheme = GetColorScheme().then(data => {
  if (data == "Failure") {
    ColorScheme = {
      RegularButtonColors: "#0000CC",
      RegularButtonColorsLight: "#0000FF",
      QuitButtonColors: "#CC0000",
      QuitButtonColorsLight: "#FF0000",
      RegularButtonText: "black",
      RegularButtonTextLight: "#555555",
      SettingsText: "#DDDDDD",
      TitleText: "#FFFFFF",
      BackgroundColor: "black"
    }
  } else if(SelectedColorScheme == "LastGamePlayed"){
    let ColorSchemeToBeUsed;
    switch (LastGamePlayed) {
      case "FlappyBird":
        ColorSchemeToBeUsed = data.FlappyBird
        break;

      default:
        ColorSchemeToBeUsed = data.Default
        break;
    }
    ColorScheme = ColorSchemeToBeUsed
  }else if(SelectedColorScheme == "FlappyBird"){
    let ColorSchemeToBeUsed;
    ColorSchemeToBeUsed = data.FlappyBird
    ColorScheme = ColorSchemeToBeUsed
  }else if(SelectedColorScheme == "Default"){
    let ColorSchemeToBeUsed;
    ColorSchemeToBeUsed = data.Default
    ColorScheme = ColorSchemeToBeUsed
  }
})
```

I then created a variable called “SelectedColorScheme” which be used to select the colour scheme. I then made a series of if statements which will select a colour scheme based on the value of the Selected Colour scheme.

This is an example of the SelectedColorScheme being set to “Default”, as shown here, it is displaying the default colour scheme as expected.

```
//saving data
let SettingsData ={
  MenuTheme: "Last game played",
  MenuPointer:0,
}
if(!localStorage.getItem("SettingData")){
  localStorage.setItem("SettingData",JSON.stringify(SettingsData))
}
```

I then created a new variable called settings data which will store the settings that the user chose along with the pointer of the option.

```
  //Colour scheme
  settingsstats.push(new SettingStat(
    (box.width / 4) - 220,
    (box.height / 2) + 70,
    "  Colour scheme ",
    ["Last game played", "Default", "Flappy Bird", "Cube Runner", "Inverted"],
    [
      function () {
        let d = JSON.parse(localStorage.getItem("SettingData"))
        d.MenuTheme = "Last game played"
        d.MenuPointer = 0
        localStorage.setItem("SettingData",JSON.stringify(d))
      },
      function () {
        let d = JSON.parse(localStorage.getItem("SettingData"))
        d.MenuTheme = "Default"
        d.MenuPointer = 1
        localStorage.setItem("SettingData",JSON.stringify(d))
      },
      function () {
        let d = JSON.parse(localStorage.getItem("SettingData"))
        d.MenuTheme = "Flappy Bird"
        d.MenuPointer = 2
        localStorage.setItem("SettingData",JSON.stringify(d))
      },
      function () {
        let d = JSON.parse(localStorage.getItem("SettingData"))
        d.MenuTheme = "Cube Runner"
        d.MenuPointer = 3
        localStorage.setItem("SettingData",JSON.stringify(d))
      },
      function () {
        let d = JSON.parse(localStorage.getItem("SettingData"))
        d.MenuTheme = "Inverted"
        d.MenuPointer = 4
        localStorage.setItem("SettingData",JSON.stringify(d))
      }
    ],
    ColorScheme.SettingsText,
    JSON.parse(localStorage.getItem("SettingData")).MenuPointer
  ))
```

I then updated the colour scheme setting stat and now it should be able to function as intended.

Here is an example of the inverted menu being selected. As shown in this screenshot, the value of the pointer has also been updated to show the inverted option every time I go onto the settings.

```
//saving data
let SettingsData ={
  LowPowerMode:false,
  LowPointer:0,
  ShowFps:false,
  ShowPointer:0,
  MenuTheme: "Last game played",
  MenuPointer:0,
  VolatileAI: true,
  VolatilePointer:0,
  ShowTips:false,
  TipsPointer:0,
  JumpPref:"Spacebar",
  JumpPointer:0
}
```

I then updated the settings data to include more of the different settings available. I then created their own functions which saves and updates them. After this, I went through the process of implementing saving and updating features for each of the settings, like the colour scheme setting.

```
case "FlappyBird":
  if (localStorage.getItem("SavedDataFlappyBird") != null && !JSON.parse(localStorage.getItem("SettingData")).LowPowerMode) {
    FlappyBirdGameplay()
  } else {
    pen.fillStyle = ColorScheme.BackgroundColor
    pen.fillRect(0, 0, box.width, box.height)
  }
  break;
case "CubeRunner":
  if (localStorage.getItem("SavedDataCubeRunner") != null && !JSON.parse(localStorage.getItem("SettingData")).LowPowerMode) {
    CubeRunnerGameplay()
  } else {
    pen.fillStyle = ColorScheme.BackgroundColor
    pen.fillRect(0, 0, box.width, box.height)
  }
```

Part of the low power mode which include the animated backgrounds not loading when it is activated so here, I have updated some of the background code and now it checks if low power mode is off before activating the background.

```
//fps
let FC = 0
let lastTime
function FpsCounter() {
  let currentTime = performance.now()
  let deltaTime = currentTime - (lastTime || currentTime)
  lastTime = currentTime
  let fps = Math.round(1000 / deltaTime)
  FC++
  if(JSON.parse(localStorage.getItem("SettingData")).MenuTheme == "Inverted"){
    pen.fillStyle = "black"
  }else{
    pen.fillStyle = "white"
  }
  pen.font= "40px Arial"
  pen.fillText(fps,40,box.height-20)
}
```

In the Utils.js file, I created a new function called FpsCounter which will calculate the current fps of the program and output it to the screen. The colour of it will usually be white unless the menu theme is inverted in which it will be black.

```
//fps
if(JSON.parse(localStorage.getItem("SettingData")).ShowFps){
  FpsCounter()
}
```

I then went into all the files and added this line of code which will allow the fps to be visible if the option is shown.

Due to time constraints, I have removed the show tips option. However, this should not be a big problem as all the features of this program are easy and simple to understand.

```
    //Cube Runner
    buttons.push(new Button(
      50,//X
      (box.height / 2),//Y
      250,//Width
      100,//Height
      ColorScheme.RegularButtonColors,//Color
      "Cube Runner",//text
      42,//text size
      ColorScheme.RegularButtonText,//text color
      function () {
        this.color = ColorScheme.RegularButtonColorsLight;
        this.textcolor = ColorScheme.RegularButtonTextLight;
        GameDescription.GameName = "Cube Runner"
        GameDescription.GameDes = "Run dodging obstacles using your own trained A.I"
        GameDescription.AILevel = "A.I Level:" + AILevels.CubeRunner
        GameDescription.MinsPlayed = "Minutes Played: " + 0
        GameDescription.GameImg.src = "LogoRunner.png"
      },//function when mouse is over button
```

I then decided to add cube runner to the play games menu. The code here is like the flappy bird version of it I created a while ago with a few changes due to the game being different.

Here is the result of that implementation. Now hovering over the cube runner button will display the following information and clicking it will launch the cube runner game. This image also displays the fps counter in the top right corner as I have turned the show fps setting on.

Creating Cube Runner Game.html

This part will contain the creation of the cube runner game which I have already made the A.I for.

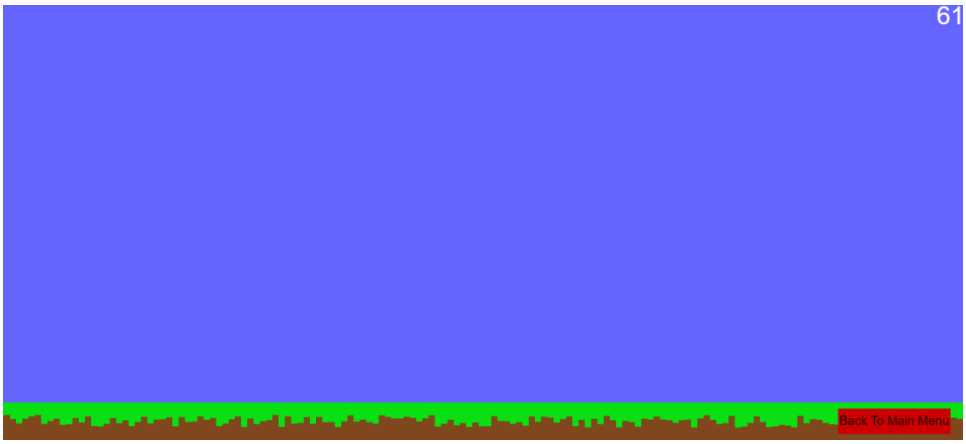
```
var box = document.getElementById("box")
box.width = innerWidth
box.height = innerHeight
var pen = box.getContext("2d")
//button functionality
let pos = {
    x: 0,
    y: 0
}
buttons = []
//Return to main menu button
buttons.push(new Button(
    (box.width)-200,
    (box.height) - 50,
    180,
    40,
    "#CC0000",
    "Back To Main Menu",
    20,
    "black",
    function () {
        this.color = "#FF0000";
        this.textcolor = "#FFFFFF";
    },
    () => {
        if(confirm("Are you sure you want to quit and go back to menu?")){
            window.open("Main Menu.html")
        }
    }
))
function GameLoop(){
  //Drawing
  //draw the background
  pen.fillStyle = "rgba(100,100,255,1)"
  pen.fillRect(0,0,box.width,box.height)
  //fps
  if(JSON.parse(localStorage.getItem("SettingData")).ShowFps){
    FpsCounter()
  }
  ButtonInteraction(pos,buttons,false)
  requestAnimationFrame(GameLoop)
}
GameLoop()
```

I first copied the basic boilerplate code I will need to create this game which contains the game loop, background and button functionality.

Here is the result. Now we have a blue screen with the go back to main menu button along with the fps counter in the top corner.

```
function GameLoop(){
  //updating
  FrameCount++
  //update the grass
  grass.forEach((Grasss,i)=>{
    if(Grasss.x <= -Grasss.width){
      grass.splice(i,1)
      grass.push(new Grass(box.width))
    }
    Grasss.update()
  })
  //Drawing
  //draw the background
  pen.fillStyle = "rgba(100,100,255,1)"
  pen.fillRect(0,0,box.width,box.height)
  //Floor
  pen.fillStyle = "#80471C"
  pen.fillRect(0,GroundHeight,box.width,60)
  grass.forEach(grass=>{
    grass.draw()
  })
  //fps
  if(JSON.parse(localStorage.getItem("SettingData")).ShowFps){
    FpsCounter()
  }
  ButtonInteraction(pos,buttons,false)
  requestAnimationFrame(GameLoop)
}
```

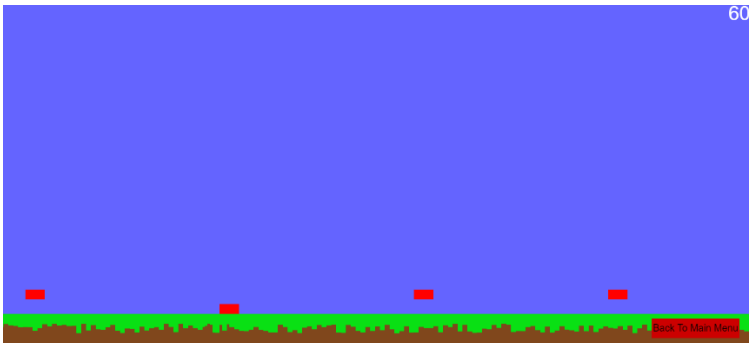
I then decided to start adding the floor and grass into the game which is shown in the screenshot above.



Here is the result. Now the grass is working just like it did in the A.I version of this game.

```
function GameLoop(){
  //updating
  FrameCount++
  //update the grass
  grass.forEach((Grasss,i)=>{
    if(Grasss.x <= -Grasss.width){
      grass.splice(i,1)
      grass.push(new Grass(box.width))
    }
    Grasss.update()
  })
  //Obstacles
  obstacles.forEach(obstacle=>{
    obstacle.update()
  })
  if(FrameCount % 80 == 0){
    let NewItem = Math.floor(Math.random()* AllObstacles.length)
    obstacles.push(new AllObstacles[NewItem])
  }
  //Drawing
  //draw the background
  pen.fillStyle = "rgba(100,100,255,1)"
  pen.fillRect(0,0,box.width,box.height)
  //Floor
  pen.fillStyle = "#80471C"
  pen.fillRect(0,GroundHeight,box.width,60)
  grass.forEach(grass=>{
    grass.draw()
  })
  //Obstacles
  obstacles.forEach(obstacle=>{
    obstacle.draw()
  })
  //fps
  if(JSON.parse(localStorage.getItem("SettingData")).ShowFps){
    FpsCounter()
  }
  ButtonInteraction(pos,buttons,false)
  requestAnimationFrame(GameLoop)
}
```

After adding the ground and grass, I started to add the obstacles which the player and A.I will have to jump over/ crouch under.



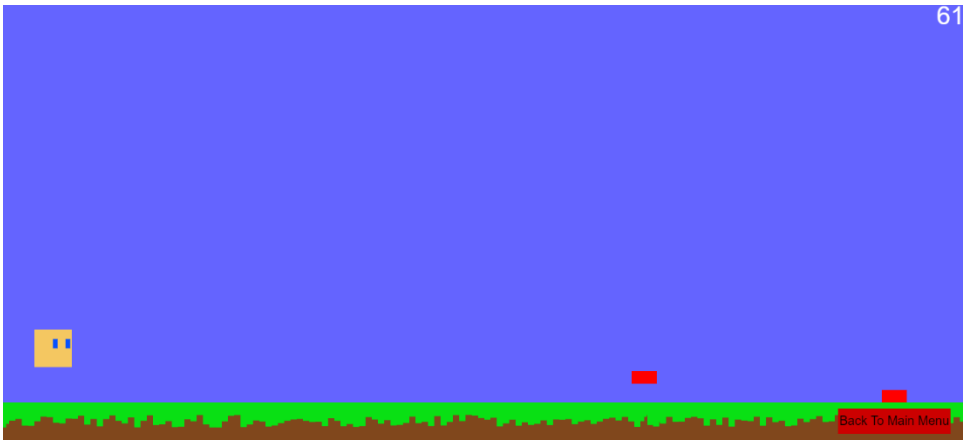
Here is the result. Now the obstacles are in the game working as intended.

```
//Runners
let Runners = []
//the AI
Runners.push(new Runner)
Runners[0].brain = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).brain
```

I then decided to start adding the runners into the game. I first started on adding the A.I runner. I first created the Runners array and pushed a new runner into it and gave it the brain of the last saved runner.

```
//runners
Runners.forEach(runner=>{
  runner.update()
  //check if any of the runners have touched the obstacles. If so, rip
  obstacles.forEach(obstacle=>{
    if(touching(runner,obstacle)){
      runner.isDead = true
    }
  })
})
//kill any runners if needed
Runners.forEach((runner,i)=>{
  if(runner.isDead){
    Runners.splice(i,1)
  }
})
```

I then went and added the updating code for the runners which includes the collision detection and the runner update method.



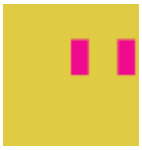
Here is the result. Now we have the A.I in the game and now I need to add the player.

```
if(!this.IsHumanControlled){
  this.think(obstacles)
  let outputs = NeuralNetwork.FeedForward(this.brain,this.inputs)
  if(outputs[0] > 0.5){
    this.Jump()
  }
  if(outputs[1] > 0.5){
    this.Crouch()
  }
  if(outputs[2] > 0.5){
    this.UnCrouch()
  }
}
```

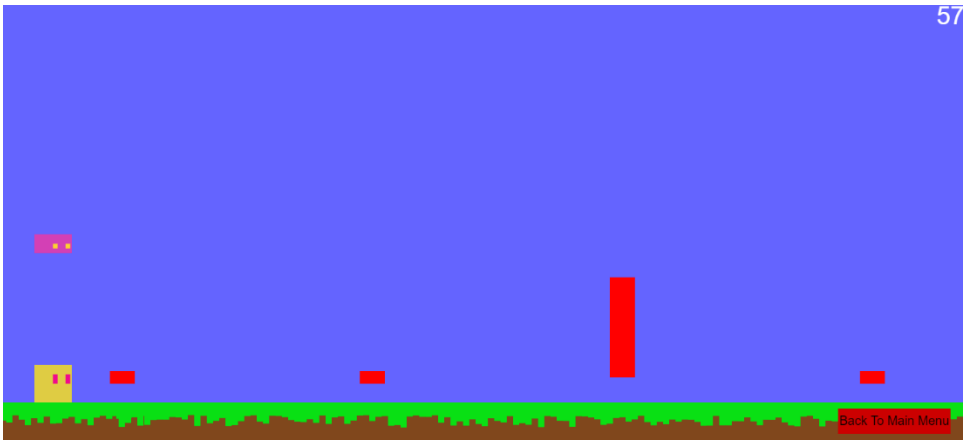
I first went to the runner class and updated the code. Now the Runner class takes in a parameter to see if a human is controlling it or not. If there is a human controlling it, it will not call its think method and it will not be controlled by the neural network.

```
//The Player
Runners.push(new Runner(true))
Runners[1].color = "rgba(223,204,67,1)"
Runners[1].eyesColor = "rgba(240,10,141,1)"
```

I then added the second runner into the program and changed its colours so it will always stay consistent, and it will be easy to distinguish between the player and the A.I.



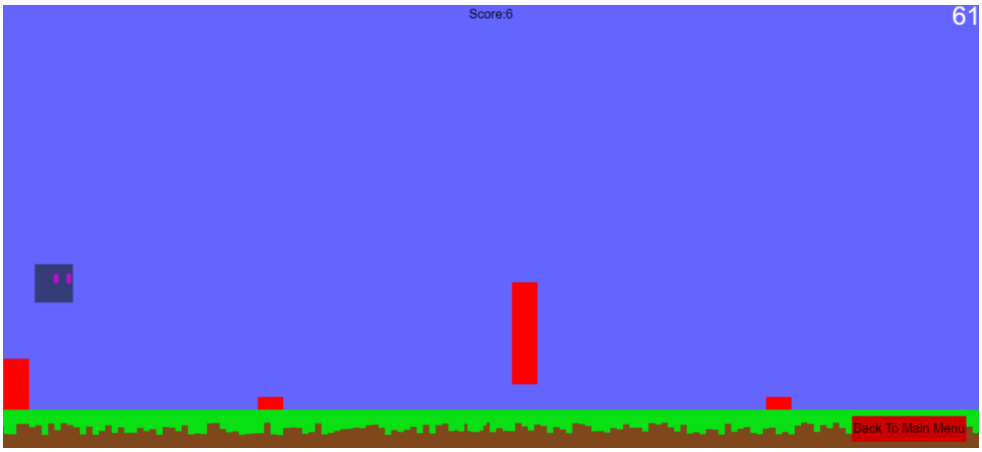
Here is what the player will look like in the game.



Now the player and the A.I are both in the game. Now I need to add the controls so the player can move like the A.I.

```
//score
pen.fillStyle = "black"
pen.font= "20px Arial"
pen.fillText("Current Score:"+score,box.width/2,15)
```

Before adding player movement, I decided to add the score which will just show how much score the player and the A.I has.



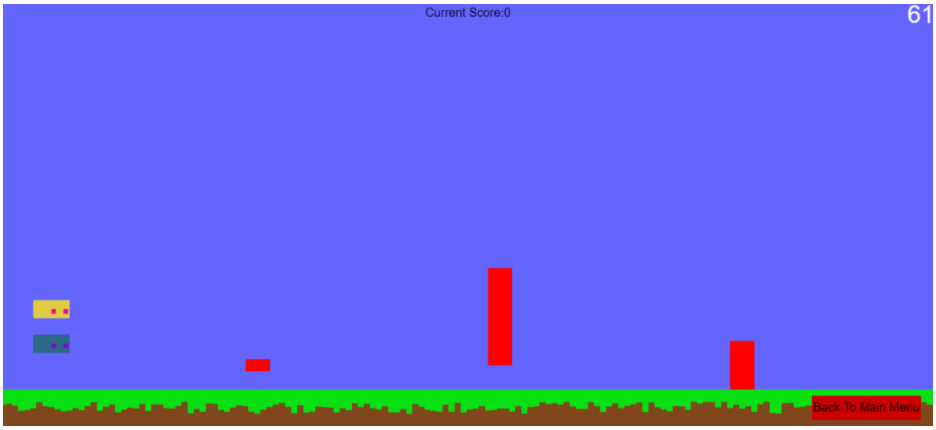
Here is the result, Now the score is displayed in the top centre of the screen.

```
//player movement
switch (JSON.parse(localStorage.getItem("SettingData")).JumpPref) {
  case "Click":
    document.addEventListener("click",()=>{
      Runners[1].Jump()
    })
    break;
    case "W key":
    document.addEventListener("keydown",(e)=>{
      if(e.keyCode == 87){
        Runners[1].Jump()
      }
    })
    break;
    case "Spacebar":
    document.addEventListener("keydown",(e)=>{
      if(e.keyCode == 32){
        Runners[1].Jump()
      }
    })
    break;
    case "Up key":
    document.addEventListener("keydown",(e)=>{
      if(e.keyCode == 38){
        Runners[1].Jump()
      }
    })
    break;
}
```

I then decided to add player movement. I created a switch statement which will choose the appropriate settings based on what jump setting the user has selected.

```
document.addEventListener("keydown",(e)=>{
  if(e.keyCode == 49){
    if(Runners[1].IsCrouching){
      Runners[1].UnCrouch()
    }else{
      Runners[1].Crouch()
    }
  }
})
```

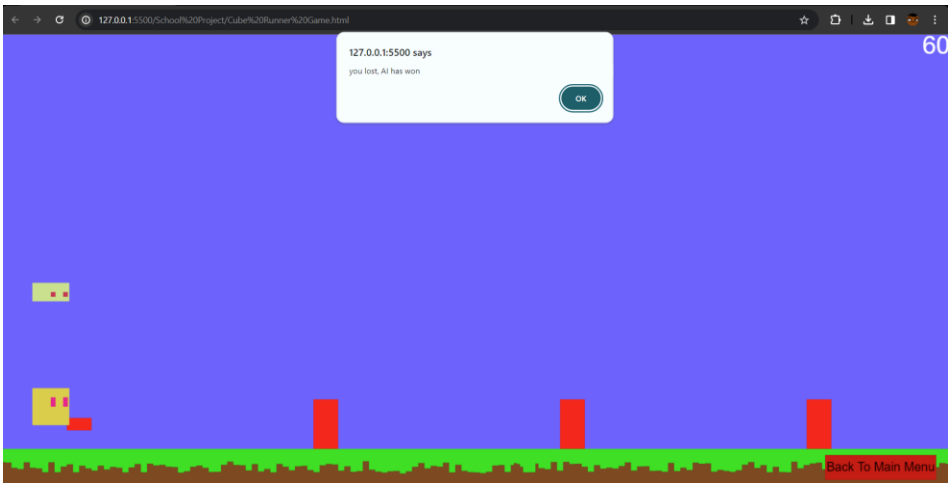
After that, I then added the crouch and uncrouch ability. This will work by pressing the 1 key on the keyboard and depending on if the runner is already crouching or not, it will crouch/uncrouch.



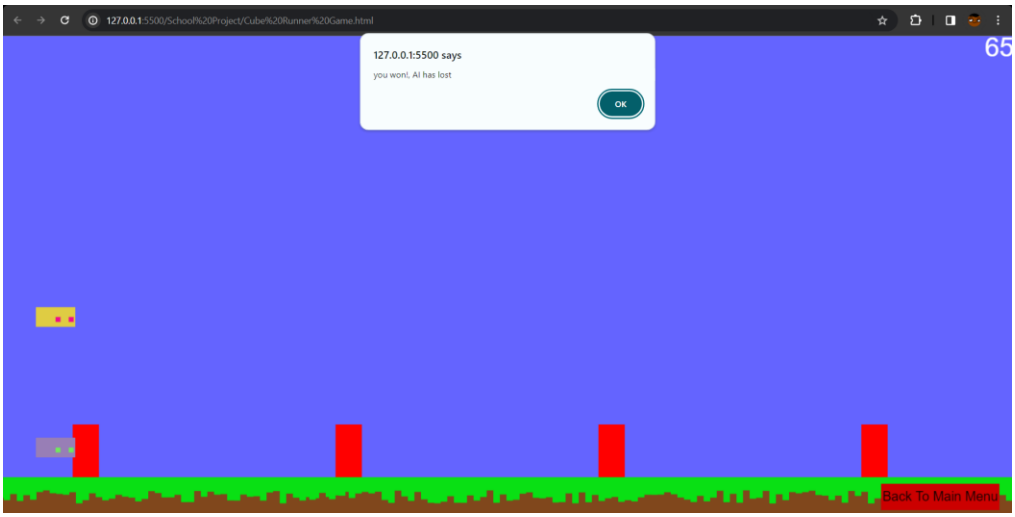
Here is the result of player controls. Now the player and the A.I can move together and now all I need to do is add the win/lose conditions.

```
//kill any runners if needed
Runners.forEach((runner,i)=>{
  if(runner.isDead){
    if(runner.IsHumanControlled){
      window.alert("you lost, AI has won")
    }else{
      window.alert("you won!, AI has lost")
    }
    Runners.splice(i,1)
    window.open("Main Menu.html")
    window.close()
  }
})
```

Once a runner dies, the program will check to see if it was human controlled or not. If it was, that must mean that the player has died and the A.I has won, if not, it must mean the A.I has died, and the player has won. After one of the runners have died, an alert will be displayed to the user and then the menu will be opened.



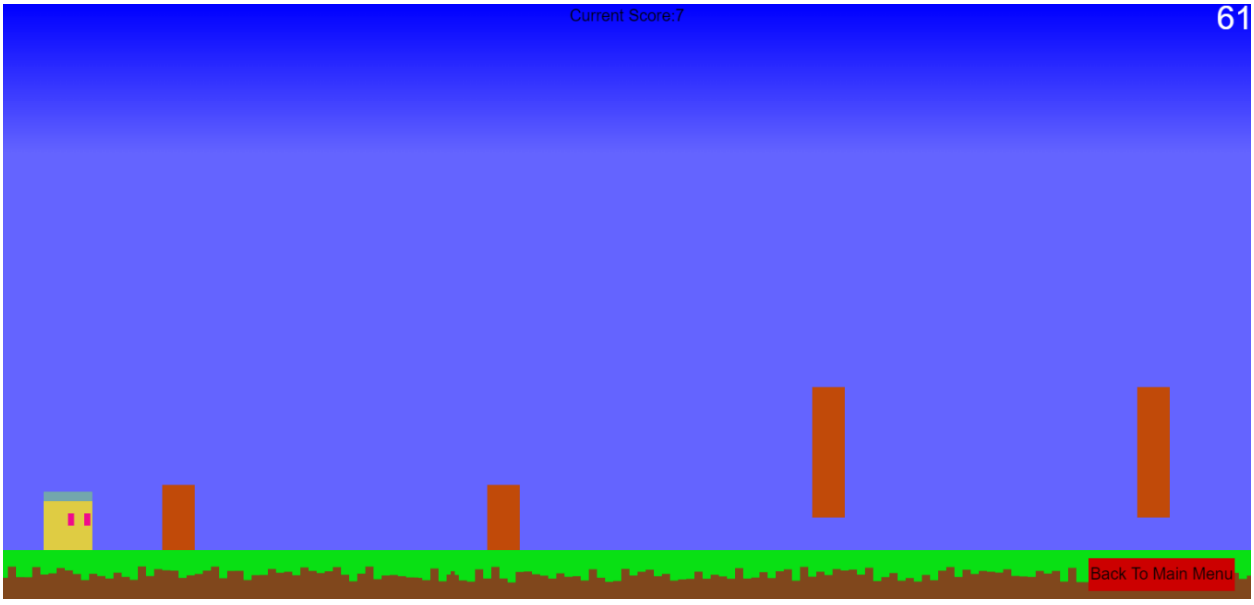
Here is the screen if the player loses.



Here is the screen if the player wins.

```
//gradient background
let gradient = pen.createLinearGradient(0,0,0,box.height)
gradient.addColorStop(0, 'blue')
gradient.addColorStop(0.25, "rgba(100,100,255,1)")
```

To make the background less blank in the cube runner games, I decided to add a gradient from a darker shade of blue to a lighter shade. I also changed the colour of the blocks so they weren't a flashy red.



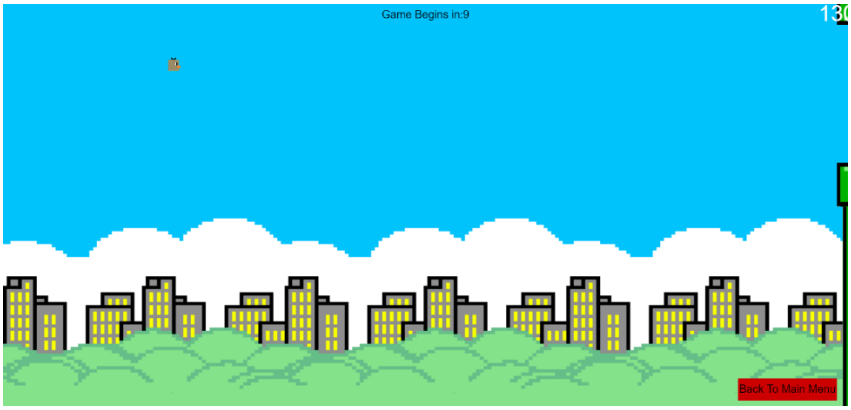
Here is the result of these changes. Now in my opinion, the game looks a lot better to look at than it did before.

Final Changes

This part will be the final few changes/implementations I need to add before the program is finished entirely.

```
let CountDownStart = setInterval(CountDown,1000)
window.addEventListener("load", () => {Loop();CountDown();})
```

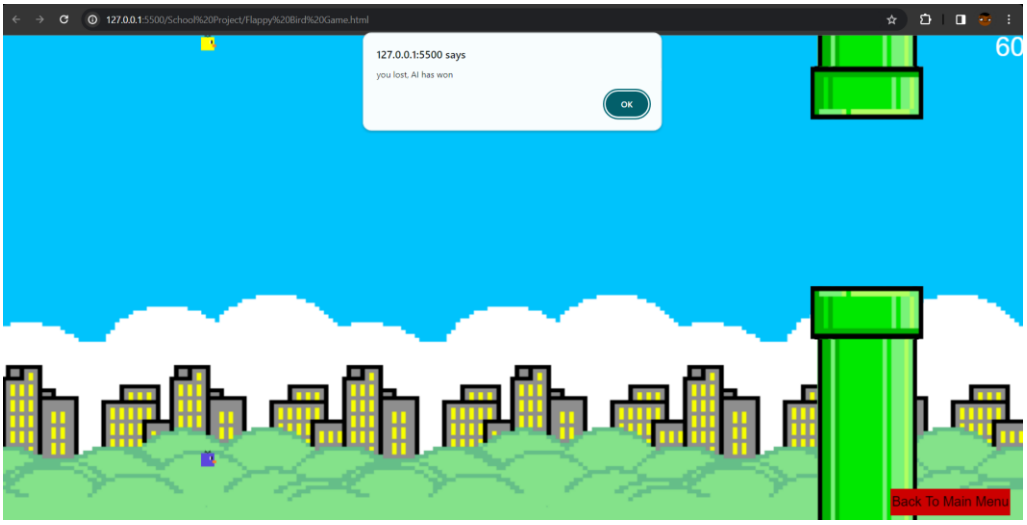
First, I decided to fix the error in which the textures would not load during the countdown of the flappy bird game. This was an easy fix as all I had to do was start the countdown and loop after everything had loaded using an event listener.



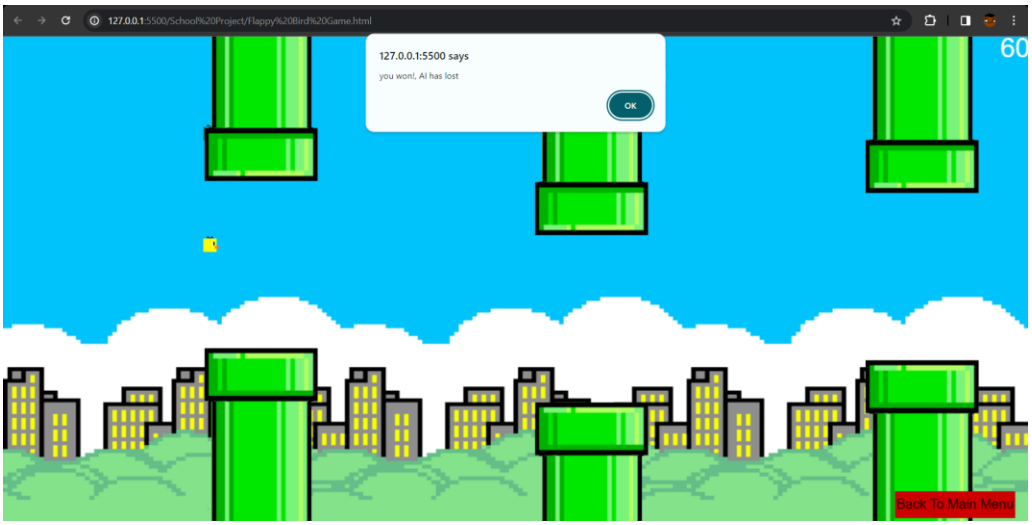
As shown in this screenshot, all the textures now load before the countdown begins.

```
//kill the bird and declare a winner
birds.forEach((bird,i)=>{
    if(bird.isDead){
        if(bird.HumanControlled){
            window.alert("you lost, AI has won")
        }else{
            window.alert("you won!, AI has lost")
        }
        birds.splice(i,1)
        window.open("Main Menu.html")
        window.close()
    }
})
```

I then decided to update the win/lose conditions for the flappy bird game so they would be identical to the one in the cube runner game.



The alert box if the player loses.



The alert box if the player wins.

```
//save non-volatile A.I
if(JSON.parse(localStorage.getItem("SettingData")).VolatileAI){
    document.addEventListener("visibilitychange",() => {
        let data = {
            brain:JSON.parse(localStorage.getItem("bestFlappyBrain")),
            HighestScore:score,
            PipeSpeed:PipeSpeed,
            Spacing:spacing,
            diff:PipeFreq,
            grav:gravity,
            gen:generation
        }
        localStorage.setItem("SavedDataFlappyBird",JSON.stringify(data))
    })
}
```

I then decided to work on the non-volatile A.I. This would work by automatically saving the A.I as soon as the visibility of the program changed. This could be due to the user exiting the program or switching tabs. I also changed it so it would include the generation of the A.I as well.

```
//if there's non-volatile AI
if(JSON.parse(localStorage.getItem("SettingData")).VolatileAI == true){
    generation = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).gen
    gravity = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).grav
    spacing = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).Spacing
    PipeFreq = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).diff
    PipeSpeed = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).PipeSpeed
}
```

Here is the code which checks to see if the volatile A.I is false (in the code it says true, however, I realised the mistake and changed it later throughout development). If it is, then it restores the conditions and generation of the A.I.

```
  for(let i  = 0; i < population; i++){
      birds.push(new Bird)
      if(JSON.parse(localStorage.getItem("SettingData")).VolatileAI == true){
          birds[i].brain = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).brain
          NeuralNetwork.mutate(birds[i].brain,0.25)
      }
  }
```

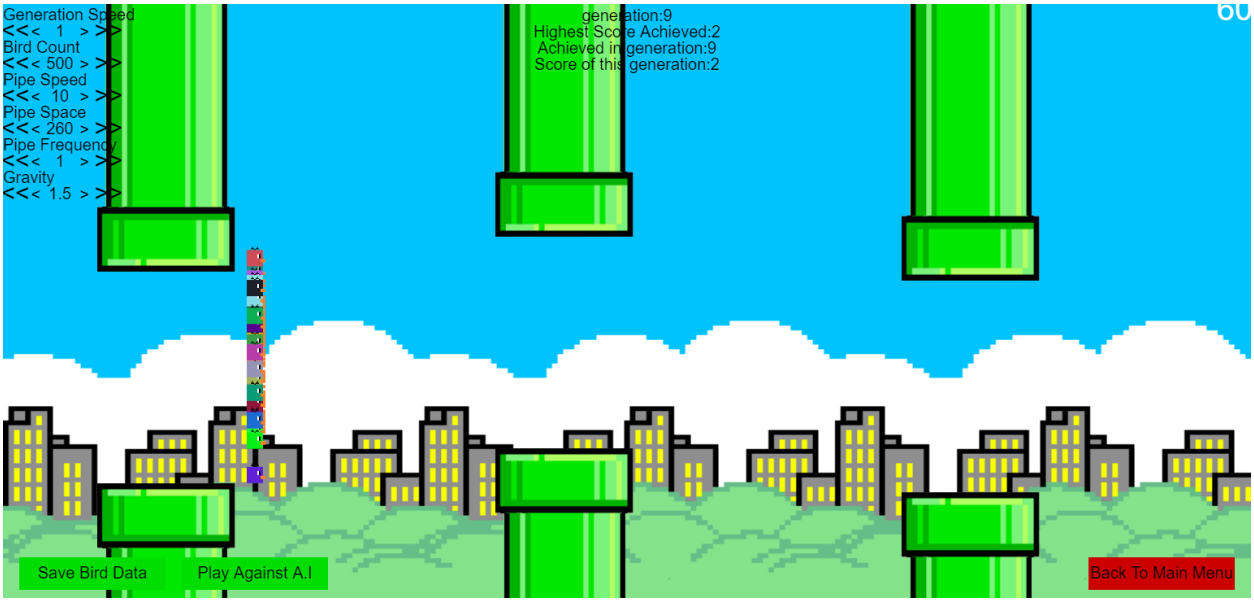
I also restored the A.I of the birds if the volatile A.I setting was turned off too.

```
//if there's non-volatile AI
if(JSON.parse(localStorage.getItem("SettingData")).VolatileAI == true){
  CubeSize = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).cubesize
  generation = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).gen
  Gravity = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).grav
}
```

Here is the same feature implemented for the cube runner AI game.

```
//directly play against AI
buttons.push(new Button(
    220,
    (box.height) - 50,
    180,
    40,
    "#00DD00",
    "Play Against A.I",
    20,
    "black",
    function () {
        this.color = "#00FF00";
        this.textcolor = "#FFFFFF";
    },
    () => {
        let data = {
            brain:JSON.parse(localStorage.getItem("bestFlappyBrain")),
            HighestScore:score,
            PipeSpeed:PipeSpeed,
            Spacing:spacing,
            diff:PipeFreq,
            grav:gravity,
            gen:generation
        }
        localStorage.setItem("SavedDataFlappyBird",JSON.stringify(data))
        console.log("saved")
        GameRecentlySaved = true
        window.open("Flappy Bird Game.html")
    }
))
```

Part of the mock up GUI involved a button in which you could go against the A.I immediately without having to travel back to the main menu. So I created a button which would save the A.I and load the flappy bird game.



Here is the result of that addition, now we have the ability to go from the training game straight to the actual flappy bird game.

Here is the same addition in the cube runner game.

```
function () {
  this.color = ColorScheme.RegularButtonColorsLight;
  this.textcolor = ColorScheme.RegularButtonTextLight;
  if(localStorage.getItem("SavedDataFlappyBird")){
    GameDescription.GameName = "Flappy Bird"
    GameDescription.GameDes = "Play the classic flappy bird using your own trained A.I"
    GameDescription.AILevel = "A.I Level:" + AILevels.FlappyBird
    GameDescription.MinsPlayed = "Minutes Played: " + 0
    GameDescription.GameImg.src = "LogoFlappy.png"
  }else{
    GameDescription.GameImg.src = "LogoFlappy.png"
    GameDescription.GameName = "Flappy Bird"
    GameDescription.GameDes = "you must train a flappy bird A.I before you can play one."
  }
},//function when mouse is over button
() => {
  if(localStorage.getItem("SavedDataFlappyBird")){
    window.open("Flappy Bird Game.html")
    window.close()
  }
}//function when mouse clicks button
```

To prevent users from attempting to play the games without training the A.I, I added in a condition to check if there was any saved data from the selected game, if not, it would disable the button from being pressed and when the user hovered over it, It would display a message which would display information on how they needed to train an A.I before they could play it. The screenshot above shows the code for the flappy bird game.

Here is how it would look if the user attempted to play the flappy bird game without training the A.I first.

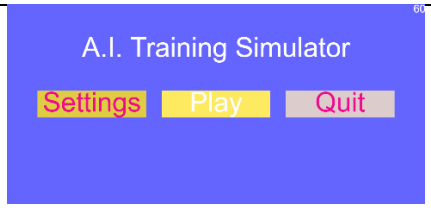
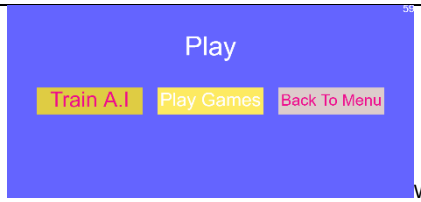
```
function () {
  this.color = ColorScheme.RegularButtonColorsLight;
  this.textcolor = ColorScheme.RegularButtonTextLight;
  if(localStorage.getItem("SavedDataCubeRunner")){
    GameDescription.GameName = "Cube Runner"
    GameDescription.GameDes = "Run dodging obstacles using your own trained A.I"
    GameDescription.AILevel = "A.I Level:" + AILevels.CubeRunner
    GameDescription.MinsPlayed = "Minutes Played: " + 0
    GameDescription.GameImg.src = "LogoRunner.png"
  }else{
    GameDescription.GameImg.src = "LogoRunner.png"
    GameDescription.GameName = "Cube Runner"
    GameDescription.GameDes = "Train a runner A.I before you play against one."
  }
},//function when mouse is over button
() => {
  if(localStorage.getItem("SavedDataCubeRunner")){
    window.open("Cube Runner Game.html")
    window.close()
  }
```

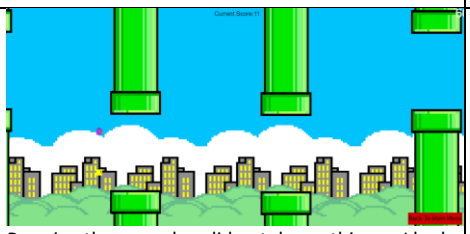
Here is the same code but for the cube runner game.

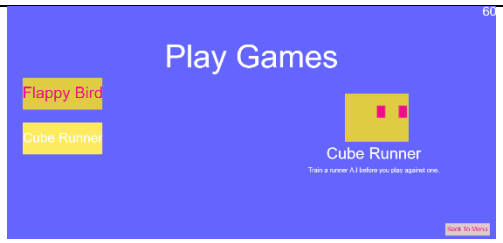
And here is how the program would look if the user tried to play the cube runner game without training the A.I first.

Testing

Throughout the iterative development, there were many tests I did to make sure functions and parts of the program worked. But now that I have finished the program, I can see how the program will work overall when I apply these tests.

Item Tested	Applied Test	Data	Expected outcome	Actual Outcome	Comments
Hover over buttons	Move the mouse until it hovers over the button. And do the same when the mouse is not over the button.	N/A	The button should perform the mouseover method which could be anything that it is set to be e.g., the colour changes when the mouse is over the button, and it should not do anything while the mouse is not over the button.	 Here, I have hovered over the play button and it preformed the mouseover event, Which in this case, the button lights up, so this test was a success.	There is nothing to comment on as the test was a success.
Pressing the buttons	Left and right click over and outside the button.	N/A	The button should perform the action method which again can be whatever the button's task	 When pressing the Play button on the main menu, it	There is nothing to comment on as the test was a success.

			is e.g., loading up a new menu. When the mouse has been clicked outside of the button, the button should not do anything.	activated the click event which here it loads the "Play menu" which means it works as it was supposed to, so this test was a success.	
Flappy bird flap ability	Pressing the space bar whilst the flappy bird game is playing normally and again when the game is paused or when the game is not playing.	N/A	While on the flappy bird game, the player bird should bounce upwards in a flapping motion. Anywhere else and nothing should happen.	 Pressing the space bar did not do anything as I had changed the setting to the mouse click, However, upon changing the settings back to the space bar, the flap ability worked as intended.	The test was a success, but it was only a success as I was able to switch the jump controls back to the spacebar.
Deleting data	Save some data to the program then go to the settings to delete it	The A.I. from one of the games e.g. flappy bird	The data should be erased and should not be able to be recovered.	This test is not possible as I have not added a feature which allows me to delete the data.	This test was a failure as I was not able to perform the test.
Neural network works	Use the neural network to play one game and sees if it manages to learn to play it.	A neural network	The neural network should be able to eventually play the game in a successful way and within reasonable time.	The Neural networks have clearly been able to learn and play both flappy bird and cube runner effectively in a reasonable amount of time, so this test was a success.	There is nothing to comment on as the test was a success.
Menu navigation.	Press all the buttons in the main menu to see if the program will break. E.g. press the play button, press the train A.I. button then press the go back to menu button.	N/A	The program should not break at all, and it should be able to move through the menus in reasonable time	Pressing through all the buttons allows me to navigate to all the different parts of the program without taking too long to load or without causing errors so this test was a success.	There is nothing to comment on as the test was a success.
Settings	Change some of the settings and see if the program will apply those settings	N/A	The program should apply these settings without any errors.	I have changed my settings to this: • Volatile A.I is turned on. • Menu theme is set to Cube runner. • The fps counter is on. • The jump button is set to a click. The settings I have applied have worked and not caused any errors whilst using the program, so this test was a success.	There is nothing to comment on as the test was a success.

Play without training	Attempt to play one of the games without first training an A.I	N/A	The program should not allow me to play any of the games.	 I try to play cube runner without first training a cube runner A.I but the program will not allow me as it displays the text "Train a runner A.I before you go against one" so this test is a success.	There is nothing to comment on as the test was a success.

Evaluation

Point of Evaluation

Evaluation and usability testing are essential components of any project, particularly in areas like software development and product design. These processes serve several key purposes. Firstly, they help assess the effectiveness of the product in meeting its intended goals, ensuring it functions correctly and addresses user needs. Additionally, evaluation and usability testing pinpoint any issues or flaws in the design or functionality, enabling developers to make necessary improvements. By observing user behaviour and preferences, these processes facilitate user-centric design, guiding decisions to enhance the user experience. Through iterative cycles of testing and refinement, developers can continuously improve the product, mitigating risks and preventing dissatisfaction among users. Usability testing also validates design choices and ensures compliance with industry standards and regulations. Overall, evaluation and usability testing play a crucial role in creating user-friendly, effective products that meet the needs of their target audience.

Initial Objectives

My initial objectives were listed in my analysis stage, and I had to make sure that I had hit all the essentials for my project to be successful. Here are all the objectives I managed to complete and how I completed them:

- Create a menu for my game- This objective was successful as I managed to complete a menu for my game which contains, settings, a game menu to select the game you want to play and a training menu for A.I that you want to train.
- Create a neural network which will function as the brain for the A.I- this objective was also met as I was able to create a neural network from the ground up, which can perform a forward propagation algorithm. This allows the A.I to work and without creating this, the entire program would have not worked.
- Create a game for the A.I. to play – this objective was met with the implementation of flappy bird and cube runner, 2 games that the user can train A.I and play them.
- Add an ability to compete against the A.I. in the game – this objective was met as I added the play games menu which allows the user to play against the A.I it has trained and try to beat it.

- Add a saving system for scores and A.I. neural networks – this objective was partially met. I was able to add a saving system for the neural networks, I was not able to create one for permanently saving player scores or A.I scores.
- Implement a player customization feature – sadly this objective was not met as I was not able to implement an ability for users to customize their own players.

Limitations

Whilst developing my project, I came across many limitations which stopped me from achieving certain goals in my project. Whilst somewhere expected, a few were not and were not planned for. As a result, this hindered progression of my project dramatically. Here are a few of the limitations I faced:

- Time Constraint:
  During the project, implementing certain elements like buttons ended up taking more time than I initially anticipated. Unfortunately, this time crunch forced me to make tough decisions, such as having to remove features like character customization. It's disappointing because character customization is usually a big hit with users, but I just couldn't fit it in given the time constraints.
- Database Knowledge:
  I didn't have much experience with databases, so I had to resort to using local storage to store all the project's information. However, local storage has its limitations, particularly in terms of how much data it can handle efficiently. This meant I couldn't store as much data as I would have liked, which might affect the project's scalability and performance.
- Not Using Libraries:
  I made a conscious decision not to use libraries for certain functionalities, like creating neural networks and buttons. While I thought it would give me more control, it ended up being quite time-consuming to develop these features from scratch. This decision might have delayed the project's completion and could have introduced potential errors or performance issues compared to using established libraries. In summary, these limitations significantly impacted the project's development process and outcome. They highlight the importance of careful planning and considering available resources to mitigate such challenges effectively.

Possible Improvements

There are a few possible improvements I could have made to this program to improve the enjoyment of it. Before I decided to come up with possible improvements, I first decided to make a form to get feedback from certain users that have used the program to see what their thoughts were on it. The form only consists of 5 questions which were:

- How did you enjoy the program?
- How did you find the games added? were they enjoyable?
- How did you find the layout of the program?
- How did you find the training part of this program?
- Any improvements?

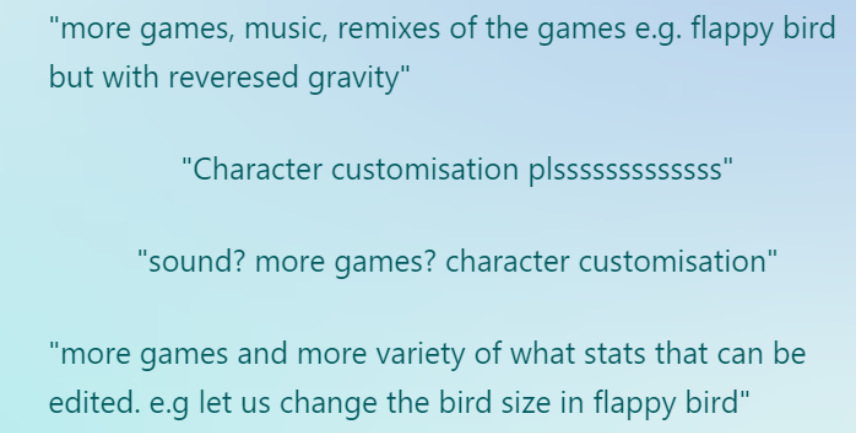
Here were some of the results from the questionnaire.

By these results from this question, it seems that lots of people enjoyed the program and what it had to offer which means that despite not being able to finish everything, people still enjoyed it.

From the results of this question, it seems that a lot of people found the games in the program to be fun to play as well as easy. However, some people thought that they were a bit repetitive and boring to play.

Based on the results from this question it seems that lots of people found the layout of this program to be simplistic and “Pretty” I assume by the words easy and eyes that many people found it easy on the eyes which is a positive as that means it was easy to navigate around the menus.

Based on these results, it looks like again, lots of people found it fun, easy and enjoyable but a few people did find it boring and repetitive.



For the final question, I decided to take a sample of answers and see if I could see a theme. Based on the sample I took; it seems like people wanted a wider variety of games and more features as well as a few people wanting character customization and some sort of audio.

Based on all the information I have on the questionnaire and from my own judgement, I have come up with few improvements I could make to the program:

1. Wider variety of games – based on the questionnaire it seems that a lot of people wanted more games to play and train A.I on, due to time constraints, this was not possible but if I had the chance to revisit this project in the future, I would add more games with more variety such as shooting games.
2. Character customization – This was one of the objectives in my success criteria I could not meet and one feature that many users seem to have wanted. If I had more time on this project, I would have added this feature.
3. More game settings – for games like cube runner, there were only 2 stats available to be changed which were the cube size and gravity. This at times can make the program a boring and repetitive, if I were to go back to this program in the future, I would make sure to add more editable settings such as jump power or obstacle speed.

4. Pre-built A.Is – this feature would have given users an idea of how the A.Is should play the game and It would also give users other competition to go against instead of A.I they created, and they trained. This feature might even make some of the games a lot harder to play as these pre-built A.Is may have stronger neural networks compared to those which were trained.

Final Code

This contains the screenshots of the final code of my project.

NeuralNetwork.js

```
//functions for this NN
function lerp(a,b,t){
    return a+(b-a)*t
}
//SIGMOID function
function Sigmoid(num){
    FinalNum = 1/(1+Math.pow(Math.E,-(num)))
    return FinalNum
}
//square the numbers
function SquareNum(Num){
    return Num*Num
}
```

```
//Matrix class
class Matrix{
    constructor(rows,cols){
        this.rows = rows || 2
        this.cols = cols || 2
        if(!Number.isInteger(this.rows) || !Number.isInteger(this.cols)){
            throw new Error("Please enter an integer when defining a matrix")
        }else{
            this.matrix = []
            for(let i = 0; i < this.rows; i++){
                this.matrix[i] = []
                for(let j = 0; j < this.cols; j++){
                this.matrix[i][j] = 0
                }
            }
        }
    }
    //randomizes the values of the matrix
    randomize(){
        for(let i = 0; i < this.rows; i++){
            for(let j = 0; j < this.cols; j++){
            this.matrix[i][j] =  Math.random()*2-1
            }
        }
    }
    static add(m0,m1){
        //adding matrices
        if(m0.rows == m1.rows && m0.cols == m1.cols){
            for(let i = 0; i < m0.rows; i++){
                for(let j = 0; j < m0.cols; j++){
                    m0.matrix[i][j] += m1.matrix[i][j]
                }
            }
        }else{
            throw new Error("Matrix dimensions must be the same in order to add them")
        }
    }
```

```
//Matrix multiplication
static multiply(m0,m1){
    //rows n cols must be equal
    if(m0.cols == m1.rows){
        var rows = m0.rows
        var cols = m1.cols
        var newMatrix = new Matrix(m0.rows,m1.cols)
        for(let i = 0; i < rows; i++){
            for(let j = 0; j < cols; j++){
               var sum = 0
               for(let k = 0; k < m0.cols; k++){
                   sum += m0.matrix[i][k] * m1.matrix[k][j]
               }
               newMatrix.matrix[i][j] = sum
            }
        }
        return newMatrix
    }else{
        throw new Error("the 1st Matrix rows must be equal to the 2nd Matrix columns")
    }
}
//this is so I can see the matrix so much easier
print(){
    console.table(this.matrix)
}
//add functions onto the matrix
static AddFunc(m0,Func){
for(let i = 0; i < m0.rows; i++){
    for(let j = 0; j < m0.cols; j++){
        m0.matrix[i][j] = Func(m0.matrix[i][j])
        }
    }
}
```

```
    //this is so I can see the matrix so much easier
    print(){
        console.table(this.matrix)
    }
    //add functions onto the matrix
    static AddFunc(m0,Func){
    for(let i = 0; i < m0.rows; i++){
        for(let j = 0; j < m0.cols; j++){
            m0.matrix[i][j] = Func(m0.matrix[i][j])
            }
        }
    }
}
function TurnArrayToMatrix(array){
    let matrix = new Matrix(array.length,1)
    for(let i = 0; i < matrix.rows; i++){
        for(let j = 0; j < matrix.cols; j++){
        matrix.matrix[i][j] = array[i]
        }
    }
    return matrix
}
function TurnMatrixToArray(Matrix){
    let array = []
    for(let i = 0; i < Matrix.rows; i++){
        for(let j = 0; j < Matrix.cols; j++){
        array.push(Matrix.matrix[i][j])
        }
    }
    return array
}
```

```
//the Neural Network (the brain)
class NeuralNetwork{
    constructor(I,H,O){
        //get the neuron counts
        this.inputsCount = I
        this.HiddenCount = H
        this.OutputsCount = O
        //initalise the matrices
        this.weights_IH = new Matrix(this.HiddenCount,this.inputsCount)
        this.bias_H = new Matrix(this.HiddenCount,1)
        this.weights_HO = new Matrix(this.OutputsCount,this.HiddenCount)
        this.bias_O = new Matrix(this.OutputsCount,1)
        this.weights_IH.randomize()
        this.bias_H.randomize()
        this.weights_HO.randomize()
        this.bias_O.randomize()
    }
     static FeedForward(brain,inputs){
        //turn the array of inputs into a matrix
        let Inputs = TurnArrayToMatrix(inputs)
        let Hidden = Matrix.multiply(brain.weights_IH,Inputs)
        Matrix.add(Hidden,brain.bias_H)
        Matrix.AddFunc(Hidden,Sigmoid)
        let outputs = Matrix.multiply(brain.weights_HO,Hidden)
        Matrix.add(outputs,brain.bias_O)
        Matrix.AddFunc(outputs,Sigmoid)
        return TurnMatrixToArray(outputs)

    }
    static mutate(network,amount){
        for(let i = 0; i < network.weights_IH.rows; i++){
            for(let j = 0; j < network.weights_IH.cols; j++){
                network.weights_IH.matrix[i][j] = lerp(network.weights_IH.matrix[i][j],Math.random()*2-1,amount)
            }
        }
        for(let i = 0; i < network.bias_H.rows; i++){
            for(let j = 0; j < network.bias_H.cols; j++){
                network.bias_H.matrix[i][j] = lerp(network.bias_H.matrix[i][j],Math.random()*2-1,amount)
            }
        }
```

```
        for(let i = 0; i < network.weights_HO.rows; i++){
            for(let j = 0; j < network.weights_HO.cols; j++){
                network.weights_HO.matrix[i][j] = lerp(network.weights_HO.matrix[i][j],Math.random()*2-1,amount
            }
        }
        for(let i = 0; i < network.bias_O.rows; i++){
            for(let j = 0; j < network.bias_O.cols; j++){
                network.bias_O.matrix[i][j] = lerp(network.bias_O.matrix[i][j],Math.random()*2-1,amount)
            }
        }
    }
}
```

Utils.js

```
//////////////////////////essential Util algorithms/////////////////////////////
//checks to see if a value is within a certain range
function InRange(value, min, max) {
  return value >= Math.min(min, max) && value <= Math.max(min, max);
}
//checks to see if a point is inside an object
function Interact(x, y, object) {
  return InRange(x, object.x, (object.x + object.width)) && InRange(y, object.y, (object.y + object.height))
}
//for simple collision detection between non rotated rectangles or squares
function touching(item1,item2){
  if(item1.x+item1.width >= item2.x && item1.x <= item2.x+item2.width && item2.y <= item1.y+item1.height&& item1.y <= item2.y+item2.height){
    return true
  }
}
//truncate a value to 1 d.p
function Truncate(num){
  NewNum = num.toString()
  return NewNum.substring(0, 3)
}
function GenerateRandomColor(){
  return `rgba(${Math.floor(Math.random() * (255 - 0 + 1) + 0)},${Math.floor(Math.random() * (255 - 0 + 1) + 0)},${Math.floor(Math.random() * (255 - 0 + 1) + 0)},1)`
}
//fps
let FC = 0
let lastTime
function FpsCounter() {
  let currentTime = performance.now()
  let deltaTime = currentTime - (lastTime || currentTime)
  lastTime = currentTime
  let fps = Math.round(1000 / deltaTime)
  FC++
  if(JSON.parse(localStorage.getItem("SettingData")).MenuTheme == "Inverted"){
    pen.fillStyle = "black"
  }else{
    pen.fillStyle = "white"
  }
  pen.font= "40px Arial"
  pen.fillText(fps,box.width-20,(box.height-box.height)+20)
}
```

```
////////////////////Menu Management Utils/////////////////////////////////////////
//A button class
class Button {
  constructor(x, y, width, height, color, text, textsize, textcolor, MouseOver, click) {
    //actual values
    this.x = x || Math.random() * 500;
    this.y = y || Math.random() * 500;
    this.width = width || 100;
    this.height = height || 40;
    this.color = color || "red";
    this.text = text || "Button";
    this.textsize = textsize || 20;
    this.textsize = String(this.textsize) + "px"
    this.textcolor = textcolor || "black";
    //Original values
    this.Ox = this.x
    this.Oy = this.y
    this.Owidth = this.width
    this.Oheight = this.height
    this.Ocolor = this.color
    this.Otext = this.text
    this.Otextsize = this.textsize
    this.Otextcolor = this.textcolor
    //button functions
    this.MouseIsOver = false
    this.clicked = false
    this.MouseOver = MouseOver || function() { this.color = "orange"; this.text = "Click?"; };
    this.click = click || function() { console.log("clicked") };
  }
  OriginState() {
    this.x = this.Ox
    this.y = this.Oy
    this.width = this.Owidth
    this.height = this.Oheight
    this.color = this.Ocolor
    this.text = this.Otext
    this.textsize = this.Otextsize
    this.textcolor = this.Otextcolor
  }
```

```
  draw() {
    pen.fillStyle = this.color;
    pen.fillRect(this.x, this.y, this.width, this.height);
    pen.font = this.textsize + " Arial";
    pen.fillStyle = this.textcolor;
    pen.textAlign = "center";
    pen.textBaseline = "middle";
    this.text = this.text
    pen.fillText(this.text, this.x + this.width / 2, this.y + this.height / 2);
    if (this.MouseIsOver) {
  this.MouseOver()
    } else {
      this.OriginState()
    }
    if (this.clicked) {
      this.click()
      this.clicked = false
    }
  }
}
```

```
//Function for button Interaction
function ButtonInteraction(pos,Arrays,IsMainMenu){
  document.addEventListener("mousemove", function (e) {
    pos.x = e.clientX
    pos.y = e.clientY
  })
  document.addEventListener("click", function (e) {
    Arrays.forEach(array => {
      if (Interact(pos.x, pos.y, array)) {
        array.clicked = true
      }
    })
  })
  Arrays.forEach(array => {
    array.draw()
    if (Interact(pos.x, pos.y, array)) {
      array.MouseIsOver = true
      if(IsMainMenu){
        InteractedButtons += 1
      }
    }else{
      array.MouseIsOver = false
    }
  })
}
```

```
//Edit game stats
class GameStat{
  constructor(x,y,stat,statname,min,max,amt,button,color,hasExtremes,ExtremeAmt){
    this.x = x || 0
    this.y = y || 0
    this.stat = stat
    this.statname = statname || "Gen speed"
    this.min = min || 0
    this.max = max || 100
    this.amt = amt || 1
    this.color = color || "black"
    this.hasExtremes = hasExtremes
    this.ExtremeAmt = ExtremeAmt || 10
    if(!stat){
      console.error("a stat value is required!")
    }
    if(!this.hasExtremes){
      button.push(new Button(
        this.x+0.1,
        this.y,
        30,
        30,
        "rgba(0,0,0,0)",
        "<",
        20,
        "black",
        function () {

            this.textcolor = "#FFFFFF";
        },
        () => {
          if(this.stat <= this.min){

          }else{
            this.stat -= this.amt
          }
        }
        ))
```

```
        button.push(new Button(
        this.x+50,
        this.y,
        30,
        30,
        "rgba(0,0,0,0)",
        ">",
        20,
        "black",
        function () {

            this.textcolor = "#FFFFFF";
        },
        () => {
          if(this.stat >= this.max){

          }else{
            this.stat += this.amt
          }
        }
        ))
    }else{
      button.push(new Button(
        this.x+0.1,
        this.y,
        30,
        30,
        "rgba(0,0,0,0)",
        "<<",
        30,
        this.color,
        function () {

            this.textcolor = "#FFFFFF";
        },
```

```
    () => {
      if(this.stat <= this.min+this.ExtremeAmt){
        //do nothing
      }else{
        this.stat -= this.ExtremeAmt
      }
    }
  ))
button.push(new Button(
  this.x+25,
  this.y,
  30,
  30,
  "rgba(0,0,0,0)",
  "<",
  20,
  this.color,
  function () {

    this.textcolor = "#FFFFFF";
  },
  () => {
    if(this.stat <= this.min){
      //do nothing
    }else{
      this.stat -= this.amt
    }
  }
  ))
```

```
button.push(new Button(
this.x+85,
this.y,
30,
30,
"rgba(0,0,0,0)",
">",
20,
this.color,
function () {

    this.textcolor = "#FFFFFF";
},
() => {
  if(this.stat >= this.max){

    }else{
      this.stat += this.amt
  }
  }
))
```


button.push(new Button(
this.x+115,
this.y,
30,
30,
"rgba(0,0,0,0)",
">>",
30,
this.color,
function () {
this.textcolor = "#FFFFFF";
},
() => {
if(this.stat >= this.max-this.ExtremeAmt){
}else{
this.stat += this.ExtremeAmt
}
}
))
}
}

```
  draw(){
    this.stat = this.stat
    pen.fillStyle = this.color
    pen.font= "20px Arial"
    pen.textAlign = "start"
    pen.textBaseline = "alphabetic"
    pen.fillText(this.statname,this.x,this.y)
    pen.textBaseline = "alphabetic"
    pen.font= "20px Arial"
    pen.textAlign = "center"
    if(this.hasExtremes){
      if(Math.abs(this.amt) < 1){
        pen.fillText(Truncate(this.stat),this.x+70,this.y+20)
      }else{
        pen.fillText(this.stat,this.x+70,this.y+20)
      }
    }else{
      if(Math.abs(this.amt) < 1){
      pen.fillText(Truncate(this.stat),this.x+40,this.y+20)
      }else{
        pen.fillText(this.stat,this.x+40,this.y+20)
      }
    }
    return this.stat
  }
}
```

```
//like game stats but for settings
class SettingStat{
  constructor(x,y,statname,options,optionFunc,color,pointer){
    this.x = x || 0
    this.y = y || 0
    this.statname = statname || "setting"
    this.options = options || ["on","off"]
    this.optionFunc = optionFunc || [function(){console.log("true")}, function(){console.log("false")}]
    this.color = color || "black"
    this.pointer = pointer || 0
    buttons.push(new Button(
      this.x + (this.statname.length*9),//X
      this.y+5,//Y
      30,//Width
      30,//Height
      "rgba(0,0,0,0)",//Color
      ">>",//text
      30,//text size
      this.color,//text color
      function () {
        this.textcolor = "#FFFFFF";
      },//function when mouse is over button
      () => {
        this.MoveNext()
      }//function when mouse clicks button
    ))
```

```
    buttons.push(new Button(
      this.x - (this.statname.length*10),//X
      this.y+5,//Y
      30,//Width
      30,//Height
      "rgba(0,0,0,0)",//Color
      "<<",//text
      30,//text size
      this.color,//text color
      function () {
        this.textcolor = "#FFFFFF";
      },//function when mouse is over button
      () => {
        this.MoveBack()
      }//function when mouse clicks button
    ))

  }
  draw(){
    pen.textBaseline = "alphabetic"
    pen.font= "30px Arial"
    pen.textAlign = "center"
    pen.fillStyle = this.color
    pen.fillText(this.statname,this.x,this.y)
    pen.fillText(this.options[this.pointer],this.x,this.y+30)
  }
```

```
  draw(){
    pen.textBaseline = "alphabetic"
    pen.font= "30px Arial"
    pen.textAlign = "center"
    pen.fillStyle = this.color
    pen.fillText(this.statname,this.x,this.y)
    pen.fillText(this.options[this.pointer],this.x,this.y+30)
  }
  //move to the next item in the list
  MoveNext(){
    //go back to the first item if we're at the end
    if(this.pointer >= this.options.length-1){
      this.pointer = 0
    }else{
      this.pointer++
    }
    this.optionFunc[this.pointer]()
  }
  //move back in the list
  MoveBack(){
    //go back to the last item if we're at the start
    if(this.pointer <= 0){
      this.pointer = this.options.length-1
    }else{
      this.pointer--
    }
    this.optionFunc[this.pointer]()
  }
}
```

```
//get the color schemes
async function GetColorScheme(){
  let HasFailed = false
  let FilePath = 'ColorScheme.json';
  // Fetch the JSON data
  let response = await fetch(FilePath).catch(error=>{console.error("Failed to fetch Data. Sorry :("); HasFailed = true})
  if(HasFailed){
    return "Failure"
  }else{
    let data = await response.json()
    return data.ColorScheme
  }
}
```

```
////////////////////GAME UTILS/////////////////////////////////////////////
////////////////////AI level checker//////////////////////////
function CheckLevel(GameName,Data){
  let level = 0
  switch(GameName){
    case "FlappyBird":
      level = ((Data.grav*Data.PipeSpeed)/250)+(((1/Data.grav)*Data.diff)/0.2)+((1/Data.Spacing)/(500))
      level = Math.min(level,100)
      //level /=100
    break
    case "CubeRunner":
      level = Math.min((((Data.cubesize/Data.grav)+(Data.cubesize/Data.grav))*Data.HighestScore),100)
    break
  }
  return Math.round(level)
}
```

```
//////////////////FLAPPY BIRD/////////////////////////////////////////////
//Pipe class
class Pipe{
  constructor(){
      this.x = box.width
      this.HasCollectedScore = false
      this.top_height = Math.floor(Math.random() * ((box.height/2) - (box.height/35)+1) + (box.height/35))
      this.top_y = 300
      this.spacing = spacing
      this.bottom_y = this.top_height+this.spacing
      this.width = 150
      this.bottom_height = box.height-(this.top_height+this.spacing)
      this.color = "rgb(100,255,100)"
      this.TopPipeImg = new Image()
      this.TopPipeImg.src = "Flappy Bird Pipe.png"
      this.BottomPipeImg = new Image()
      this.BottomPipeImg.src = "Flappy Bird Pipe.png"
  }
```

```
  draw(){
      pen.fillStyle = this.color
      pen.fillRect(this.x,0,this.width,this.top_height)
      pen.fillRect(this.x,this.bottom_y,this.width,this.bottom_height)
      //top part
      //bottom of pipe
      pen.drawImage(this.BottomPipeImg,82,5,249,399,this.x,-10,this.width,this.top_height+10)
      //top of pipe
      pen.drawImage(this.TopPipeImg, 74, 405, 269, 108,this.x-10,this.top_height-80,this.width+20, 80)
      //bottom part
      pen.save()
      pen.translate(this.x-10, this.bottom_y)
      pen.scale(1,-1)
      //bottom of pipe
      pen.drawImage(this.BottomPipeImg,82,5,249,399,10,-this.bottom_height-10,this.width,this.bottom_height+10)
      //top of pipe
      pen.drawImage(this.TopPipeImg, 74, 405, 269, 108,0,-80, this.width+20, 80)
      pen.restore()
  }
  update(){
      this.x -= PipeSpeed
  }
}
```

```
//bird class
class Bird{
  constructor(IsHuman){
      this.x = 300
      this.y = 100
      this.width = 20
      this.height = 20
      this.color = GenerateRandomColor()
      this.y_speed = 0
      this.UpForce = 15
      this.score = 0
      this.inputs = []
      this.brain = new NeuralNetwork(9,10,1)
      this.isDead = false
      this.Img = new Image
      this.Img.src = "Flappy Bird Icon.png"
      if(IsHuman){
        this.color = "yellow"
        this.HumanControlled = true
      }
  }
  draw(){
      pen.fillStyle = this.color
      pen.fillRect(this.x,this.y,this.width,this.height)
      pen.drawImage(this.Img,0,0,25,25,this.x,this.y-5,25,25)
  }
```

```
  think(pipes){
      let closest = pipes[0]
      let closestD = Infinity
      for(let i = 0; i < pipes.length; i++){
        let d = pipes[i].x - this.x
        if(d < closestD && d > -closest.width){
          closest = pipes[i]
          closestD = d
        }
      }
      this.inputs[0] = this.y
      this.inputs[1] = this.y_speed
      this.inputs[2] = (this.y-closest.top_height)
      this.inputs[3] = (this.y-closest.bottom_y)
      this.inputs[4] = (this.x-closest.x)
      this.inputs[5] = (this.x-closest.x+closest.width)
      this.inputs[6] = this.UpForce
      this.inputs[7] = closest.spacing
      this.inputs[8] = closest.spacing

  }
  update(){
      this.score++
      this.y_speed+= gravity
      this.y_speed *=0.9
      this.y += this.y_speed
      if(!this.HumanControlled){
        this.think(pipes)
        let outputs = NeuralNetwork.FeedForward(this.brain,this.inputs)
        if(outputs[0] > 0.5){
          this.flap()
        }
      }
    }
```

```
  flap(){
      this.y_speed -= this.UpForce
  }
}
```

```
//////////////////CUBE RUNNER/////////////////////////////////////////////
class Runner{
  constructor(IsHuman){
    this.x = 50
    this.y = GroundHeight - 50
    this.size = CubeSize
    this.width = this.size
    this.height = this.size
    this.Oheight = this.height
    this.crouchHeight = this.height/2
    this.color = GenerateRandomColor()
    this.Yspeed = 0
    this.JumpForce = 30
    this.IsInAir = false
    this.eyesColor = GenerateRandomColor()
    this.IsCrouching = false
    this.isDead = false
    this.inputs = []
    this.brain = new NeuralNetwork(9,25,3)
    this.score = 0
    this.IsHumanControlled = IsHuman
  }
```

```
  draw(){
  pen.fillStyle = this.color
  pen.fillRect(this.x,this.y,this.width,this.height)
  //eyes
  pen.save()
  pen.translate(this.x,this.y)
  //if it's crouching
  if(this.IsCrouching){
    pen.fillStyle =  this.eyesColor
    pen.fillRect(this.width/2,this.width/4,this.width/8,this.width/8)
    pen.fillRect((this.width/2)+(this.width/3),this.width/4,this.width/8,this.width/8)
  }else{
    //if it's not
    pen.fillStyle =  this.eyesColor
    pen.fillRect(this.width/2,this.width/4,this.width/8,this.width/4)
    pen.fillRect((this.width/2)+(this.width/3),this.width/4,this.width/8,this.width/4)
  }
  pen.restore()
  }
```

```
think(obstacles){
  let closest1 = obstacles[0]
  let closestD1 = Infinity
  for(let i = 0; i < obstacles.length; i++){
    let d = obstacles[i].x - this.x
    if (d < closestD1 && d > -closest1.width){
        closest1 = obstacles[i]
        closestD1 = d
    }
  }
  //set the inputs
  this.inputs[0] = this.y
  this.inputs[1] = this.Yspeed
  if(this.IsCrouching){
    this.inputs[2] = 1
  }else{
    this.inputs[2] = 0
  }
  if(this.IsInAir){
    this.inputs[3] = 1
  }else{
    this.inputs[3] = 0
  }
  this.inputs[4] = this.y - closest1.y
  this.inputs[5] = this.y - (closest1.y+closest1.height)
  this.inputs[6] = this.x - closest1.x
  this.inputs[7] = this.x - (closest1.x+closest1.width)
  this.inputs[8] = closest1.Id
}
```

```
update(){
  this.score++
  this.y += this.Yspeed
  this.Yspeed *=0.9
  if(this.y <= GroundHeight - this.height){
    if(this.IsCrouching){
      this.Yspeed +=Gravity/4
    }else{
      this.Yspeed +=Gravity
    }
  }else{
    this.IsInAir = false
    this.Yspeed = 0
    this.y = GroundHeight - this.height
  }
  if(!this.IsHumanControlled){
    this.think(obstacles)
    let outputs = NeuralNetwork.FeedForward(this.brain,this.inputs)
    if(outputs[0] > 0.5){
      this.Jump()
    }
    if(outputs[1] > 0.5){
      this.Crouch()
    }
    if(outputs[2] > 0.5){
      this.UnCrouch()
    }
  }
}
```

```
  Jump(){
    if(!this.IsInAir){
      if(this.IsCrouching){
        this.Yspeed -=this.JumpForce
      }else{
        this.Yspeed -=this.JumpForce/2
      }
      this.IsInAir = true
    }
  }
  Crouch(){
    this.height = this.crouchHeight
    this.IsCrouching = true
  }
  UnCrouch(){
    this.height = this.Oheight
    this.IsCrouching = false
  }
}
```

```
//simple obstacle
class ObstacleBlock{
  constructor(){
    this.x = box.width
    this.y = GroundHeight - 20
    this.width = 40
    this.height = 20
    this.color = "#c14a09"
    this.Id = 0
  }
  draw(){
    pen.fillStyle = this.color
    pen.fillRect(this.x,this.y,this.width,this.height)
  }
  update(){
    this.x -=5
    this.width = this.width
    this.height = this.height
  }
}
```

```
//Flying Obstacle. You can go over or under this one.
class FlyingObstacle extends ObstacleBlock{
  constructor(){
    super()
    this.y = GroundHeight - 50
    //this.height = 50
    this.Id = 1
  }
}
//Long Flying Obstacle, jumping over this one is not advised
class LongFlyingObstacle extends ObstacleBlock{
  constructor(){
    super()
    this.y = GroundHeight - 200
    this.height = 160
    this.Id = 2
  }
}
//Taller Obstcale, harder to jump over
class TallerObstcale extends ObstacleBlock{
  constructor(){
    super()
    this.height = 80
    this.y = GroundHeight - 80
    this.Id = 3
  }
}
```

```
//grass
class Grass{
  constructor(x){
    this.x = x
    this.y = GroundHeight
    this.width = grasswidth
    this.height = Math.random() * (40 - (20)) + (20)
    this.color = "rgba(9,224,20,1)"
  }
  draw(){
    pen.fillStyle = this.color
    pen.fillRect(this.x,this.y,this.width,this.height)
  }
  update(){
    this.x -=5
  }
}
```

Main Menu.html

```
<title>A.I Training Simulator</title>
<script src="NeuralNetwork.js"></script>
<script src="Utils.js"></script>
<canvas id="box" width="500" height="500" style="border:0px solid black"></canvas>
<script src="Background.js"></script>
</canvas>
<style>
  body {
    margin: 0;
    overflow: hidden;
  }
</style>
<script>
  var box = document.getElementById("box")
  box.width = innerWidth
  box.height = innerHeight
  pen = box.getContext("2d")
  pen.fillStyle = "rgba(0,0,0,1)"
  pen.fillRect(0, 0, box.width, box.height)
  let IsOnGameMenu = false
  let MenuTitle;
  let TextDescription = " "
  let GameDescription = {
    GameImg: new Image(),
    GameName: " ",
    GameDes: " ",
    MinsPlayed: " ",
    AILevel: " ",
  }
  let AILevels = {
    FlappyBird: localStorage.getItem("SavedDataFlappyBird") == null ? null : CheckLevel("FlappyBird", JSON.parse(localStorage.getItem("SavedDataFlappyBird"))),
    CubeRunner: localStorage.getItem("SavedDataCubeRunner") == null ? null : CheckLevel("CubeRunner", JSON.parse(localStorage.getItem("SavedDataCubeRunner")))
  }
  let ButtonHoveredOver = ""
  let buttons = []
  let settingsstats = []
  //last game played
  let LastGamePlayed = localStorage.getItem("LastGamePlayed")
```

```
//saving data
let SettingsData ={
  LowPowerMode:false,
  LowPointer:0,
  ShowFps:false,
  ShowPointer:0,
  MenuTheme: "Last game played",
  MenuPointer:0,
  VolatileAI: true,
  VolatilePointer:0,
  ShowTips:false,
  TipsPointer:0,
  JumpPref:"Spacebar",
  JumpPointer:0
}
if(!localStorage.getItem("SettingData")){
  localStorage.setItem("SettingData",JSON.stringify(SettingsData))
}
```

```
//choosing the colour scheme
let SelectedColorScheme = JSON.parse(localStorage.getItem("SettingData")).MenuTheme
let ColorScheme = GetColorScheme().then(data => {
  if (data == "Failure") {
    ColorScheme = {
      RegularButtonColors: "#0000CC",
      RegularButtonColorsLight: "#0000FF",
      QuitButtonColors: "#CC0000",
      QuitButtonColorsLight: "#FF0000",
      RegularButtonText: "black",
      RegularButtonTextLight: "#555555",
      SettingsText: "#DDDDDD",
      TitleText: "#FFFFFF",
      BackgroundColor: "black"
    }
  } else if(SelectedColorScheme == "Last game played"){
    let ColorSchemeToBeUsed;
    switch (LastGamePlayed) {
      case "FlappyBird":
        ColorSchemeToBeUsed = data.FlappyBird
        break;
        case "Cube Runner":
        ColorSchemeToBeUsed = data.CubeRunner
        break;

      default:
        ColorSchemeToBeUsed = data.Default
        break;
    }
```

```
    ColorScheme = ColorSchemeToBeUsed
  }else if(SelectedColorScheme == "Flappy Bird"){
    let ColorSchemeToBeUsed;
    ColorSchemeToBeUsed = data.FlappyBird
    ColorScheme = ColorSchemeToBeUsed
  }else if(SelectedColorScheme == "Default"){
    let ColorSchemeToBeUsed;
    ColorSchemeToBeUsed = data.Default
    ColorScheme = ColorSchemeToBeUsed
  }else if(SelectedColorScheme == "Cube Runner"){
    let ColorSchemeToBeUsed;
    ColorSchemeToBeUsed = data.CubeRunner
    ColorScheme = ColorSchemeToBeUsed
  }else if(SelectedColorScheme == "Inverted"){
    let ColorSchemeToBeUsed;
    ColorSchemeToBeUsed = data.Inverted
    ColorScheme = ColorSchemeToBeUsed
  }
})
```

```
//change menu
function LoadMenu(Menu) {
  IsOnGameMenu = false
  buttons = []
  settingsstats = []
  switch (Menu) {
    case "Options":
      MenuTitle = "Options"
      //Settings Button
      buttons.push(new Button(
        (box.width / 2) - 200,//X
        (box.height / 2) - 50,//Y
        400,//Width
        100,//Height
        ColorScheme.RegularButtonColors,//Color
        "Settings",//text
        100,//text size
        ColorScheme.RegularButtonText,//text color
        function () {
          this.color = ColorScheme.RegularButtonColorsLight;
          this.textcolor = ColorScheme.RegularButtonTextLight;
        },//function when mouse is over button
        () => {
          LoadMenu("Settings")
        }//function when mouse clicks button
      ))
      //Stats Button
      buttons.push(new Button(
        (box.width / 3) - 400,//X
        (box.height / 2) - 50,//Y
        400,//Width
        100,//Height
        ColorScheme.RegularButtonColors,//Color
        "Stats",//text
        100,//text size
        ColorScheme.RegularButtonText,//text color
        function () {
          this.color = ColorScheme.RegularButtonColorsLight;
          this.textcolor = ColorScheme.RegularButtonTextLight;
        },//function when mouse is over button
        () => {
          console.log("Stats")
        }//function when mouse clicks button
      ))
```

```
    ))
    //Back To Menu Button
    buttons.push(new Button(
      (2 * box.width / 3),//X
      (box.height / 2) - 50,//Y
      400,//Width
      100,//Height
      ColorScheme.QuitButtonColors,//Color
      "Back To Menu",//text
      60,//text size
      ColorScheme.RegularButtonText,//text color
      function () {
        this.color = ColorScheme.QuitButtonColorsLight;
        this.textcolor = ColorScheme.RegularButtonTextLight;
      },//function when mouse is over button
      () => {
        LoadMenu()
      }//function when mouse clicks button
    ))
    break;
```

```
case "Settings":
  MenuTitle = "Settings"
  //low power mode
  settingsstats.push(new SettingStat(
    (box.width / 4) - 220,
    (box.height / 2) - 90,
    "Low Power Mode",
    ["On", "Off"],
    [
    function () {
        let d = JSON.parse(localStorage.getItem("SettingData"))
        d.LowPowerMode = true
        d.LowPointer = 0
        localStorage.setItem("SettingData",JSON.stringify(d))
      },
      function () {
        let d = JSON.parse(localStorage.getItem("SettingData"))
        d.LowPowerMode = false
        d.LowPointer = 1
        localStorage.setItem("SettingData",JSON.stringify(d))
      }
    ],
    ColorScheme.SettingsText,
    JSON.parse(localStorage.getItem("SettingData")).LowPointer
  ))
```

```
//show FPS
settingsstats.push(new SettingStat(
  (box.width / 2) + 180,
  (box.height / 2) - 90,
  "Display FPS",
  ["On", "Off"],
  [
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.ShowFps = true
      d.ShowPointer = 0
      localStorage.setItem("SettingData",JSON.stringify(d))
    },
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.ShowFps = false
      d.ShowPointer = 1
      localStorage.setItem("SettingData",JSON.stringify(d))
    }
  ],
  ColorScheme.SettingsText,
  JSON.parse(localStorage.getItem("SettingData")).ShowPointer
))
```

```
//Colour scheme
settingsstats.push(new SettingStat(
  (box.width / 2) + 520,
  (box.height / 2) - 90,
  "  Colour scheme ",
  ["Last game played", "Default", "Flappy Bird", "Cube Runner", "Inverted"],
  [
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.MenuTheme = "Last game played"
      d.MenuPointer = 0
      localStorage.setItem("SettingData",JSON.stringify(d))
    },
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.MenuTheme = "Default"
      d.MenuPointer = 1
      localStorage.setItem("SettingData",JSON.stringify(d))
    },
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.MenuTheme = "Flappy Bird"
      d.MenuPointer = 2
      localStorage.setItem("SettingData",JSON.stringify(d))
    },
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.MenuTheme = "Cube Runner"
      d.MenuPointer = 3
      localStorage.setItem("SettingData",JSON.stringify(d))
    },
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.MenuTheme = "Inverted"
      d.MenuPointer = 4
      localStorage.setItem("SettingData",JSON.stringify(d))
    }
  ],
  ColorScheme.SettingsText,
  JSON.parse(localStorage.getItem("SettingData")).MenuPointer
))
```

```
//Volatile A.I
settingsstats.push(new SettingStat(
  (box.width / 4) - 220,
  (box.height / 2) + 150,
  "Volatile A.I",
  ["On", "Off"],
  [
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.VolatileAI = true
      d.VolatilePointer = 0
      localStorage.setItem("SettingData",JSON.stringify(d))
    },
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.VolatileAI = false
      d.VolatilePointer = 1
      localStorage.setItem("SettingData",JSON.stringify(d))
    }
  ],
  ColorScheme.SettingsText,
  JSON.parse(localStorage.getItem("SettingData")).VolatilePointer
))
```

```
/*        //Show tips on hover
        settingsstats.push(new SettingStat(
          (box.width / 4) - 220,
          (box.height / 2) + 230,
          "Display tips",
          ["On", "Off"],
          [
            function () {
              let d = JSON.parse(localStorage.getItem("SettingData"))
              d.ShowTips = true
              d.TipsPointer = 0
              localStorage.setItem("SettingData",JSON.stringify(d))
            },
            function () {
              let d = JSON.parse(localStorage.getItem("SettingData"))
              d.ShowTips = false
              d.TipsPointer = 1
              localStorage.setItem("SettingData",JSON.stringify(d))
            }
          ],
          ColorScheme.SettingsText,
          JSON.parse(localStorage.getItem("SettingData")).TipsPointer
        ))
```

```
 *///Jump button preference
settingsstats.push(new SettingStat(
  (box.width / 2) - 220,
  (box.height / 2) - 90,
  "Jump Button",
  ["Spacebar", "W key", "Up key", "Click"],
  [
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.JumpPref = "Spacebar"
      d.JumpPointer = 0
      localStorage.setItem("SettingData",JSON.stringify(d))
    },
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.JumpPref = "W key"
      d.JumpPointer = 1
      localStorage.setItem("SettingData",JSON.stringify(d))
    },
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.JumpPref = "Up key"
      d.JumpPointer = 2
      localStorage.setItem("SettingData",JSON.stringify(d))
    },
    function () {
      let d = JSON.parse(localStorage.getItem("SettingData"))
      d.JumpPref = "Click"
      d.JumpPointer = 3
      localStorage.setItem("SettingData",JSON.stringify(d))
    }
  ],
  ColorScheme.SettingsText,
  JSON.parse(localStorage.getItem("SettingData")).JumpPointer
))
```

```
//Back to Options button
buttons.push(new Button(
  (box.width) - 160,//X
  (box.height) - 50,//Y
  140,//Width
  40,//Height
  ColorScheme.QuitButtonColors,//Color
  "Back To Menu",//text
  20,//text size
  ColorScheme.RegularButtonText,//text color
  function () {
    this.color = ColorScheme.QuitButtonColorsLight;
    this.textcolor = ColorScheme.RegularButtonTextLight;
  },//function when mouse is over button
```

```
() => {
  SelectedColorScheme = JSON.parse(localStorage.getItem("SettingData")).MenuTheme
  ColorScheme = GetColorScheme().then(data => {
    if (data == "Failure") {
      ColorScheme = {
        RegularButtonColors: "#0000CC",
        RegularButtonColorsLight: "#0000FF",
        QuitButtonColors: "#CC0000",
        QuitButtonColorsLight: "#FF0000",
        RegularButtonText: "black",
        RegularButtonTextLight: "#555555",
        SettingsText: "#DDDDDD",
        TitleText: "#FFFFFF",
        BackgroundColor: "black"
      }
    } else if(SelectedColorScheme == "Last game played"){
      let ColorSchemeToBeUsed;
      switch (LastGamePlayed) {
        case "FlappyBird":
          ColorSchemeToBeUsed = data.FlappyBird
          break;
          case "Cube Runner":
          ColorSchemeToBeUsed = data.CubeRunner
          break;

        default:
          ColorSchemeToBeUsed = data.Default
          break;
      }
      ColorScheme = ColorSchemeToBeUsed
```

```
            }else if(SelectedColorScheme == "Flappy Bird"){
              let ColorSchemeToBeUsed;
              ColorSchemeToBeUsed = data.FlappyBird
              ColorScheme = ColorSchemeToBeUsed
            }else if(SelectedColorScheme == "Default"){
              let ColorSchemeToBeUsed;
              ColorSchemeToBeUsed = data.Default
              ColorScheme = ColorSchemeToBeUsed
            }else if(SelectedColorScheme == "Cube Runner"){
              let ColorSchemeToBeUsed;
              ColorSchemeToBeUsed = data.CubeRunner
              ColorScheme = ColorSchemeToBeUsed
            }else if(SelectedColorScheme == "Inverted"){
              let ColorSchemeToBeUsed;
              ColorSchemeToBeUsed = data.Inverted
              ColorScheme = ColorSchemeToBeUsed
            }
          })
          LoadMenu()
        }//function when mouse clicks button
      ))
      break
```

```
case "Play":
  MenuTitle = "Play"
  //Play games Button
  buttons.push(new Button(
    (box.width / 2) - 200,//X
    (box.height / 2) - 50,//Y
    400,//Width
    100,//Height
    ColorScheme.RegularButtonColors,//Color
    "Play Games",//text
    70,//text size
    ColorScheme.RegularButtonText,//text color
    function () {
      this.color = ColorScheme.RegularButtonColorsLight;
      this.textcolor = ColorScheme.RegularButtonTextLight;
    },//function when mouse is over button
    () => {
      LoadMenu("Play Games")
    }//function when mouse clicks button
  ))
  //Train Button
  buttons.push(new Button(
    (box.width / 3) - 400,//X
    (box.height / 2) - 50,//Y
    400,//Width
    100,//Height
    ColorScheme.RegularButtonColors,//Color
    "Train A.I",//text
    80,//text size
    ColorScheme.RegularButtonText,//text color
    function () {
      this.color = ColorScheme.RegularButtonColorsLight;
      this.textcolor = ColorScheme.RegularButtonTextLight;
    },//function when mouse is over button
    () => {
      LoadMenu("Train")
    }//function when mouse clicks button
  ))
```

```
//Back To Menu Button
buttons.push(new Button(
  (2 * box.width / 3),//X
  (box.height / 2) - 50,//Y
  400,//Width
  100,//Height
  ColorScheme.QuitButtonColors,//Color
  "Back To Menu",//text
  60,//text size
  ColorScheme.RegularButtonText,//text color
  function () {
    this.color = ColorScheme.QuitButtonColorsLight;
    this.textcolor = ColorScheme.RegularButtonTextLight;
  },//function when mouse is over button
  () => {
    LoadMenu()
  }//function when mouse clicks button
))
break;
```

```
    case "Train":
      MenuTitle = "Training"
      //Flappy Bird Game
      buttons.push(new Button(
/*          (box.width / 2) - 200,//X
        (box.height / 2) - 50,//Y */
        (2 * box.width / 3),
        (box.height / 2) - 50,
        400,//Width
        100,//Height
        ColorScheme.RegularButtonColors,//Color
        "Flappy Bird",//text
        70,//text size
        ColorScheme.RegularButtonText,//text color
        function () {
          this.color = ColorScheme.RegularButtonColorsLight;
          this.textcolor = ColorScheme.RegularButtonTextLight;
          TextDescription = "Teach AI to play the classic Flappy Bird game"
          ButtonHoveredOver = "FlappyBird"
        },//function when mouse is over button
        () => {
          window.open("Flappy Bird AI.html")
          window.close()
        }//function when mouse clicks button
      ))
```

```
      //Cube Runner Game
      buttons.push(new Button(
        (box.width / 3) - 400,//X
        (box.height / 2) - 50,//Y
        400,//Width
        100,//Height
        ColorScheme.RegularButtonColors,//Color
        "Cube Runner",//text
        65,//text size
        ColorScheme.RegularButtonText,//text color
        function () {
          this.color = ColorScheme.RegularButtonColorsLight;
          this.textcolor = ColorScheme.RegularButtonTextLight;
          TextDescription = "Teach AI cubes to avoid obstacles and run the farthest distance"
          ButtonHoveredOver = "CubeRunner"
        },//function when mouse is over button
        () => {
          window.open("Cube Runner AI.html")
          window.close()
        }//function when mouse clicks button
      ))
```

```
/* // game 3
buttons.push(new Button(
  (2 * box.width / 3),//X
  (box.height / 2) - 50,//Y
  400,//Width
  100,//Height
  ColorScheme.RegularButtonColors,//Color
  "Training 3 ",//text
  80,//text size
  ColorScheme.RegularButtonText,//text color
  function () {
    this.color = ColorScheme.RegularButtonColorsLight;
    this.textcolor = ColorScheme.RegularButtonTextLight;
  },//function when mouse is over button
  () => {
    console.log("Train 3")
  }//function when mouse clicks button
)) */
//Back to Play button
buttons.push(new Button(
  (box.width) - 160,//X
  (box.height) - 50,//Y
  140,//Width
  40,//Height
  ColorScheme.QuitButtonColors,//Color
  "Back To Menu",//text
  20,//text size
  ColorScheme.RegularButtonText,//text color
  function () {
    this.color = ColorScheme.QuitButtonColorsLight;
    this.textcolor = ColorScheme.RegularButtonTextLight;
  },//function when mouse is over button
  () => {
    LoadMenu("Play")
  }//function when mouse clicks button
))
```

```
    break;
  case "Play Games":
    MenuTitle = "Play Games"
    //Flappy Bird Game
    buttons.push(new Button(
      50,//X
      (box.height / 2) - 140,//Y
      250,//Width
      100,//Height
      ColorScheme.RegularButtonColors,//Color
      "Flappy Bird",//text
      50,//text size
      ColorScheme.RegularButtonText,//text color
      function () {
        this.color = ColorScheme.RegularButtonColorsLight;
        this.textcolor = ColorScheme.RegularButtonTextLight;
        if(localStorage.getItem("SavedDataFlappyBird")){
          GameDescription.GameName = "Flappy Bird"
          GameDescription.GameDes = "Play the classic flappy bird using your own trained A.I"
          GameDescription.AILevel = "A.I Level:" + AILevels.FlappyBird
          GameDescription.MinsPlayed = "Minutes Played: " + 0
          GameDescription.GameImg.src = "LogoFlappy.png"
        }else{
          GameDescription.GameImg.src = "LogoFlappy.png"
          GameDescription.GameName = "Flappy Bird"
          GameDescription.GameDes = "you must train a flappy bird A.I before you can play one."
        }
      },//function when mouse is over button
      () => {
        if(localStorage.getItem("SavedDataFlappyBird")){
          window.open("Flappy Bird Game.html")
          window.close()
        }
      }//function when mouse clicks button
    ))
```

```
//Cube Runner
buttons.push(new Button(
  50,//X
  (box.height / 2),//Y
  250,//Width
  100,//Height
  ColorScheme.RegularButtonColors,//Color
  "Cube Runner",//text
  42,//text size
  ColorScheme.RegularButtonText,//text color
  function () {
    this.color = ColorScheme.RegularButtonColorsLight;
    this.textcolor = ColorScheme.RegularButtonTextLight;
    if(localStorage.getItem("SavedDataCubeRunner")){
      GameDescription.GameName = "Cube Runner"
      GameDescription.GameDes = "Run dodging obstacles using your own trained A.I"
      GameDescription.AILevel = "A.I Level:" + AILevels.CubeRunner
      GameDescription.MinsPlayed = "Minutes Played: " + 0
      GameDescription.GameImg.src = "LogoRunner.png"
    }else{
      GameDescription.GameImg.src = "LogoRunner.png"
      GameDescription.GameName = "Cube Runner"
      GameDescription.GameDes = "Train a runner A.I before you play against one."
    }
  },//function when mouse is over button
  () => {
    if(localStorage.getItem("SavedDataCubeRunner")){
      window.open("Cube Runner Game.html")
      window.close()
    }
  }//function when mouse clicks button
))
```

```
        //Game 3
/*          buttons.push(new Button(
          50,//X
          (box.height / 2) + 140,//Y
          250,//Width
          100,//Height
          ColorScheme.RegularButtonColors,//Color
          "Game 3",//text
          70,//text size
          ColorScheme.RegularButtonText,//text color
          function () {
            this.color = ColorScheme.RegularButtonColorsLight;
            this.textcolor = ColorScheme.RegularButtonTextLight;
          },//function when mouse is over button
          () => {
            console.log("Game 3")
          }//function when mouse clicks button
        )) */
        //Back to Play button
        buttons.push(new Button(
          (box.width) - 160,//X
          (box.height) - 50,//Y
          140,//Width
          40,//Height
          ColorScheme.QuitButtonColors,//Color
          "Back To Menu",//text
          20,//text size
          ColorScheme.RegularButtonText,//text color
          function () {
            this.color = ColorScheme.QuitButtonColorsLight;
            this.textcolor = ColorScheme.RegularButtonTextLight;
          },//function when mouse is over button
          () => {
            LoadMenu("Play")
          }//function when mouse clicks button
        ))
        break
```

```
case "Flappy Bird":
  MenuTitle = "Flappy Bird"
  IsOnGameMenu = true
  //PreBuilt A.I 1
  buttons.push(new Button(
    (box.width / 3) - 400,//X
    (box.height / 2) - 50,//Y
    400,//Width
    100,//Height
    ColorScheme.RegularButtonColors,//Color
    "PreBuilt 1",//text
    80,//text size
    ColorScheme.RegularButtonText,//text color
    function () {
      this.color = ColorScheme.RegularButtonColorsLight;
      this.textcolor = ColorScheme.RegularButtonTextLight;
    },//function when mouse is over button
    () => {
      console.log("A.I 1")
    }//function when mouse clicks button
  ))
  //PreBuilt A.I 2
  buttons.push(new Button(
    (box.width / 2) - 200,//X
    (box.height / 2) - 50,//Y
    400,//Width
    100,//Height
    ColorScheme.RegularButtonColors,//Color
    "PreBuilt 1",//text
    80,//text size
    ColorScheme.RegularButtonText,//text color
    function () {
      this.color = ColorScheme.RegularButtonColorsLight;
      this.textcolor = ColorScheme.RegularButtonTextLight;
    },//function when mouse is over button
    () => {
      console.log("A.I 2")
    }//function when mouse clicks button
  ))
```

```
//Custom A.I
buttons.push(new Button(
  (2 * box.width / 3),//X
  (box.height / 2) - 50,//Y
  400,//Width
  100,//Height
  ColorScheme.RegularButtonColors,//Color
  "Custom A.I",//text
  80,//text size
  ColorScheme.RegularButtonText,//text color
  function () {
    this.color = ColorScheme.RegularButtonColorsLight;
    this.textcolor = ColorScheme.RegularButtonTextLight;
    TextDescription = "Play against your own A.I and see who is better!"
  },//function when mouse is over button
  () => {
    window.open("Flappy Bird Game.html")
    window.close()
  }//function when mouse clicks button
))
//Back to Play button
buttons.push(new Button(
  (box.width) - 160,//X
  (box.height) - 50,//Y
  140,//Width
  40,//Height
  ColorScheme.QuitButtonColors,//Color
  "Back To Menu",//text
  20,//text size
  ColorScheme.RegularButtonText,//text color
  function () {
    this.color = ColorScheme.QuitButtonColorsLight;
    this.textcolor = ColorScheme.RegularButtonTextLight;
  },//function when mouse is over button
  () => {
    LoadMenu("Play Games")
  }//function when mouse clicks button
))
break
```

```
default:
  MenuTitle = "A.I. Training Simulator"
  //Play Button
  buttons.push(new Button(
    (box.width / 2) - 200,//X
    (box.height / 2) - 50,//Y
    400,//Width
    100,//Height
    ColorScheme.RegularButtonColors,//Color
    "Play",//text
    100,//text size
    ColorScheme.RegularButtonText,//text color
    function () {
      this.color = ColorScheme.RegularButtonColorsLight;
      this.textcolor = ColorScheme.RegularButtonTextLight;
    },//function when mouse is over button
    () => {
      LoadMenu("Play")
    }//function when mouse clicks button
  ))
  //Options Button
  buttons.push(new Button(
    (box.width / 3) - 400,//X
    (box.height / 2) - 50,//Y
    400,//Width
    100,//Height
    ColorScheme.RegularButtonColors,//Color
    "Settings",//text
    100,//text size
    ColorScheme.RegularButtonText,//text color
    function () {
      this.color = ColorScheme.RegularButtonColorsLight;
      this.textcolor = ColorScheme.RegularButtonTextLight;
    },//function when mouse is over button
    () => {
      LoadMenu("Settings")
    }//function when mouse clicks button
  ))
```

```
        //quit Button
        buttons.push(new Button(
          (2 * box.width / 3),//X
          (box.height / 2) - 50,//Y
          400,//Width
          100,//Height
          ColorScheme.QuitButtonColors,//Color
          "Quit",//text
          100,//text size
          ColorScheme.RegularButtonText,//text color
          function () {
            this.color = ColorScheme.QuitButtonColorsLight;
            this.textcolor = ColorScheme.RegularButtonTextLight;
          },//function when mouse is over button
          () => {
            if (confirm("Are you sure you want to quit?")) {
              window.close()
            }
          }//function when mouse clicks button
        ))
        break;
    }
}
//mouse movement for the button logic
let InteractedButtons = 0
let Pos = {
  x: 0,
  y: 0
}
```

```
//choosing a background
function loadBackground() {
  //draw the background
  switch (ButtonHoveredOver) {
    case "FlappyBird":
      if (localStorage.getItem("SavedDataFlappyBird") != null && !JSON.parse(localStorage.getItem("SettingData")).LowPowerMode) {
        FlappyBirdGameplay()
      } else {
        pen.fillStyle = ColorScheme.BackgroundColor
        pen.fillRect(0, 0, box.width, box.height)
      }
      break;
    case "CubeRunner":
      if (localStorage.getItem("SavedDataCubeRunner") != null && !JSON.parse(localStorage.getItem("SettingData")).LowPowerMode) {
        CubeRunnerGameplay()
      } else {
        pen.fillStyle = ColorScheme.BackgroundColor
        pen.fillRect(0, 0, box.width, box.height)
      }
      break
    default:
      pen.fillStyle = ColorScheme.BackgroundColor
      pen.fillRect(0, 0, box.width, box.height)
      break;
  }
}
```

```
function Loop() {
  loadBackground()
  //Main title
  pen.font = "100px Arial";
  pen.fillStyle = ColorScheme.TitleText;
  pen.textAlign = "center";
  pen.textBaseline = "middle";
  pen.fillText(MenuTitle, box.width / 2, (box.height / 2) - 200);
  //Description text
  pen.font = "20px Arial";
  pen.fillStyle = "white";
  pen.textAlign = "center";
  pen.textBaseline = "middle";
  pen.fillText(TextDescription, box.width / 2, (box.height - 50));
  //fps
  if(JSON.parse(localStorage.getItem("SettingData")).ShowFps){
    FpsCounter()
  }
  //Description for games
  pen.font = "50px Arial";
  pen.fillStyle = "white";
  pen.textAlign = "center";
  pen.textBaseline = "middle";
  pen.fillText(GameDescription.GameName, (box.width / 2) + (box.width / 4), (box.height / 2) + 100);
  pen.font = "20px Arial";
  pen.fillText(GameDescription.GameDes, (box.width / 2) + (box.width / 4), (box.height / 2) + 150);
  pen.fillText(GameDescription.AILevel, (box.width / 2) + (box.width / 4), (box.height / 2) + 180);
  pen.fillText(GameDescription.MinsPlayed, (box.width / 2) + (box.width / 4), (box.height / 2) + 210);
  pen.drawImage(GameDescription.GameImg, ((box.width / 2) + (box.width / 4)) - 90, (box.height / 2) - 90, 200, 150)
  //button logic
  InteractedButtons = 0
  ButtonInteraction(Pos, buttons, true)
  //setting stat interaction
  settingsstats.forEach(settingstat => {
    settingstat.draw()
  })
```

```
    if (InteractedButtons < 1) {
      TextDescription = " "
      GameDescription.GameName = " "
      GameDescription.GameDes = " "
      GameDescription.AILevel = " "
      GameDescription.MinsPlayed = " "
      GameDescription.GameImg.src = "blank.png"
      ButtonHoveredOver = ""
    }
    requestAnimationFrame(Loop)
  }
  window.addEventListener("load", () => {Loop();LoadMenu();})
</script>
```

Background.js

```
var box = document.getElementById("box")
box.width = innerWidth
box.height = innerHeight
let pen = box.getContext("2d")
//frame count
var FrameCount = 0
//background
var BG = new Image(box.width,box.height)
//flappy bird game
if(localStorage.getItem("SavedDataFlappyBird") != null){
    var gravity = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).grav
    var PipeSpeed = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).PipeSpeed
    var PipeFreq = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).diff
    var spacing = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).Spacing
    BG.src = "Flappy Bird BG.png"
    //pipes
    var pipes = []
    pipes.push(new Pipe)
    //bird
    var bird = new Bird()
    bird.brain = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).brain
}
```

```
function FlappyBirdGameplay(){
    //updates
    FrameCount++
    //update the bird
    bird.update()
     //update the pipes
     for(let i = 0; i < pipes.length; i++){
        pipes[i].update()
        //remove the pipe if it's off the screen
        if(pipes[i].x < -pipes[i].width){
            pipes.splice(i,1)
        }
    }
    //push a new pipe into the list
    if(FrameCount % Math.floor((500 / (PipeSpeed*PipeFreq))) == 0){
        pipes.push(new Pipe)
    }
    //drawing
    //draw the background
    pen.fillStyle = "rgba(100,100,255,1)"
    pen.fillRect(0,0,box.width,box.height)
    pen.drawImage(BG, 0, 0, box.width, box.height+5)
    //draw everything else
    //draw the bird
    bird.draw()
    //draw the pipes
    pipes.forEach(pipe=>{
        pipe.draw()
    })
}
```

```
//cube runner game
if(localStorage.getItem("SavedDataCubeRunner") != null){
    var Gravity = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).grav
    var CubeSize = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).cubesize
    //ground
    var GroundHeight = (box.height) - 60
    //grass columns
    var grasswidth = 10
    var cols = Math.round(box.width/grasswidth)
    //obstacles
    var AllObstacles = [ObstacleBlock, LongFlyingObstacle, FlyingObstacle, TallerObstcale]
    var obstacles = []
    obstacles.push(new FlyingObstacle)
    //grass
    var grass = []
    for(let i = 0; i < cols+5; i++){
      grass.push(new Grass(i*grasswidth))
    }
    //background
    var gradient = pen.createLinearGradient(0,0,0,box.height)
    gradient.addColorStop(0, 'blue')
    gradient.addColorStop(0.25, "rgba(100,100,255,1)")
    //runner
    var runner = new Runner()
    runner.brain = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).brain
}
```

```
function CubeRunnerGameplay(){
    //updating
     FrameCount++
     //runner
     runner.update()
    //Obstacles
    obstacles.forEach(obstacle=>{
        obstacle.update()
    })
    if(FrameCount % 80 == 0){
        let NewItem = Math.floor(Math.random()* AllObstacles.length)
        obstacles.push(new AllObstacles[NewItem])
    }
    //update the grass
    grass.forEach((Grasss,i)=>{
        if(Grasss.x <= -Grasss.width){
          grass.splice(i,1)
          grass.push(new Grass(box.width))
        }
        Grasss.update()
    })
    //drawing
    //draw the background
    pen.fillStyle = gradient
    pen.fillRect(0,0,box.width,box.height)
    runner.draw()
    //Obstacles
    obstacles.forEach(obstacle=>{
        obstacle.draw()
    })
    //Floor
    pen.fillStyle = "#80471C"
    pen.fillRect(0,GroundHeight,box.width,60)
    grass.forEach(grass=>{
        grass.draw()
    })
}
```

ColorScheme.json

```
"ColorScheme":{
    "Default":{
        "RegularButtonColors":"#0000CC",
        "RegularButtonColorsLight":"#0000FF",
        "QuitButtonColors":"#CC0000",
        "QuitButtonColorsLight":"#FF0000",
        "RegularButtonText":"black",
        "RegularButtonTextLight":"#555555",
        "SettingsText":"#CCCCCC",
        "TitleText":"#FFFFFF",
        "BackgroundColor":"black"
    },
    "FlappyBird":{
        "RegularButtonColors":"#00DD00",
        "RegularButtonColorsLight":"#00FF00",
        "QuitButtonColors":"#CC0000",
        "QuitButtonColorsLight":"#FF0000",
        "RegularButtonText":"black",
        "RegularButtonTextLight":"white",
        "SettingsText":"black",
        "TitleText":"white",
        "BackgroundColor":"rgba(100,100,255,1)"
    },
    "CubeRunner":{
        "RegularButtonColors":"rgba(223,204,67,1)",
        "RegularButtonColorsLight":"rgba(253,234,97,1)",
        "QuitButtonColors":"#DDCCCC",
        "QuitButtonColorsLight":"#FFDDDD",
        "RegularButtonText":"rgba(240,10,141,1)",
        "RegularButtonTextLight":"white",
        "SettingsText":"black",
        "TitleText":"white",
        "BackgroundColor":"rgba(100,100,255,1)"
    },
    "Inverted":{
        "RegularButtonColors": "#FFFF33",
        "RegularButtonColorsLight": "#FFFF00",
        "QuitButtonColors": "#33FFFF",
        "QuitButtonColorsLight": "#00FFFF",
        "RegularButtonText": "white",
        "RegularButtonTextLight": "#AAAAAA",
        "SettingsText": "#333333",
        "TitleText": "#000000",
        "BackgroundColor": "white"
    }
```

Flappy Bird AI.html

```
<title>Flappy Bird AI</title>
<script src="NeuralNetwork.js"></script>
<script src="Utils.js"></script>
<canvas id="box" width="500" height="500" style="border:0px solid ☐black"></canvas>
</canvas>
<style>
  body {
    margin: 0;
    overflow: hidden;
  }
</style>
<script>
    var box = document.getElementById("box")
    box.width = innerWidth
    box.height = innerHeight
    var pen = box.getContext("2d")
    let population = 500
    let gravity = 0.5
    let generation = 1
    let cycles = 1
    let PipeSpeed = 5
    let PipeFreq = 0.3
    let spacing = 250
    let score = 0
    //if there's non-volatile AI
    if(!JSON.parse(localStorage.getItem("SettingData")).VolatileAI){
        generation = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).gen
        gravity = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).grav
        spacing = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).Spacing
        PipeFreq = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).diff
        PipeSpeed = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).PipeSpeed
    }
    let BG = new Image(box.width,box.height)
    localStorage.setItem("LastGamePlayed","FlappyBird")
    BG.src = "Flappy Bird BG.png"
    let pos = {
        x: 0,
        y: 0
    }
```

```
let pipes = []
pipes.push(new Pipe)
let birds = []
let SavedBirds = []
for(let i  = 0; i < population; i++){
    birds.push(new Bird)
    if(!JSON.parse(localStorage.getItem("SettingData")).VolatileAI){
        birds[i].brain = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).brain
        NeuralNetwork.mutate(birds[i].brain,0.25)
    }
}
let buttons = []
//Return to main menu button
buttons.push(new Button(
    (box.width)-200,
    (box.height) - 50,
    180,
    40,
    "#CC0000",
    "Back To Main Menu",
    20,
    "black",
    function () {
        this.color = "#FF0000";
        this.textcolor = "#FFFFFF";
    },
    () => {
        if(confirm("Are you sure you want to quit and go back to menu?")){
            window.open("Main Menu.html")
            window.close()
        }
      }
    ))
```

```
//save bird data
buttons.push(new Button(
    20,
    (box.height) - 50,
    180,
    40,
    "#00DD00",
    "Save Bird Data",
    20,
    "black",
    function () {
        this.color = "#00FF00";
        this.textcolor = "#FFFFFF";
    },
    () => {
        let data = {
            brain:JSON.parse(localStorage.getItem("bestFlappyBrain")),
            HighestScore:score,
            PipeSpeed:PipeSpeed,
            Spacing:spacing,
            diff:PipeFreq,
            grav:gravity,
            gen:generation
        }
        localStorage.setItem("SavedDataFlappyBird",JSON.stringify(data))
        console.log("saved")
        GameRecentlySaved = true
      }
    ))
```

```
//directly play against AI
buttons.push(new Button(
    220,
    (box.height) - 50,
    180,
    40,
    "#00DD00",
    "Play Against A.I",
    20,
    "black",
    function () {
        this.color = "#00FF00";
        this.textcolor = "#FFFFFF";
    },
    () => {
        let data = {
            brain:JSON.parse(localStorage.getItem("bestFlappyBrain")),
            HighestScore:score,
            PipeSpeed:PipeSpeed,
            Spacing:spacing,
            diff:PipeFreq,
            grav:gravity,
            gen:generation
        }
        localStorage.setItem("SavedDataFlappyBird",JSON.stringify(data))
        console.log("saved")
        GameRecentlySaved = true
        window.open("Flappy Bird Game.html")
      }
    ))
```

```
//save non-volatile A.I
if(!JSON.parse(localStorage.getItem("SettingData")).VolatileAI){
    document.addEventListener("visibilitychange",() => {
        let data = {
            brain:JSON.parse(localStorage.getItem("bestFlappyBrain")),
            HighestScore:score,
            PipeSpeed:PipeSpeed,
            Spacing:spacing,
            diff:PipeFreq,
            grav:gravity,
            gen:generation
        }
        localStorage.setItem("SavedDataFlappyBird",JSON.stringify(data))
      })
}
//stats for game that can be edited
let GenSpeed = new GameStat(0,20,cycles,"Generation Speed",0,100,1,buttons,"black",true)
let BirdCount = new GameStat(0,60,population,"Bird Count",1,1000,1,buttons,"black",true)
let PipeSpeedStat = new GameStat(0,100,PipeSpeed,"Pipe Speed",1,50,1,buttons,"black",true,5)
let PipeSpace = new GameStat(0,140,spacing,"Pipe Space",200,500,1,buttons,"black",true)
let Diff = new GameStat(0,180,PipeFreq,"Pipe Frequency",0.3,1,0.1,buttons,"black",true,0.5)
let Grav = new GameStat(0,220,gravity,"Gravity",0.2,5,0.1,buttons,"black",true,0.5)
let FrameCount = 0
let BestScore = 0
let BestBirdAllTime;
```

```
function restart(){
    //next generation
    generation++
    //reset everything
    score = 0
    pipes = []
    FrameCount = 0
    //determine the best bird
    let BestBird = SavedBirds[0]
    let CurrentBestScore = SavedBirds[0].score
    //get the best bird
    for(let i = 0; i < SavedBirds.length; i++){
        if(SavedBirds[i].score > BestBird.score){
            BestBird = SavedBirds[i]
        }
    }
    //save the brain
        localStorage.setItem("bestFlappyBrain",JSON.stringify(BestBird.brain))
        for(let i = 0; i < population; i++){
            birds.push(new Bird)
        }
        for(let i = 0; i < birds.length; i++){
            birds[i].brain = JSON.parse(localStorage.getItem("bestFlappyBrain"))
            NeuralNetwork.mutate(birds[i].brain,0.25)
        }
        SavedBirds = []
}
let TotalScore = 0
let HighestGen = 0
let GameRecentlySaved = false
let DisappearTimer = 500
```

```
function GameLoop(){
    //updates
    for (let i = 0; i < cycles; i++) {
        FrameCount++
        //update the birds
        for(let i = 0; i < birds.length; i++){
            birds[i].update()
            //check if the birds are out of bounds
            if(birds[i].y < 0 || birds[i].y > box.height){
                birds[i].isDead = true
            }
            //collision detection with birds and pipe
            pipes.forEach(pipe=>{
                //touches the top pipe
                if(birds[i].x+birds[i].width >= pipe.x && birds[i].x <= pipe.x+pipe.width && 0 <= birds[i].y+birds[i].height && birds[i].y <= pipe.top_height){
                    birds[i].isDead = true
                }
                //touching the bottom pipe
                 else if(birds[i].x+birds[i].width >= pipe.x && birds[i].x <= pipe.x+pipe.width && pipe.bottom_y <= birds[i].y+birds[i].height&& birds[i].y <= pipe.
                bottom_y+pipe.bottom_height){
                    birds[i].isDead = true
                }
                //if the birds have collected the score from the pipe or not
                if(birds[i].x > pipe.x+pipe.width && !pipe.HasCollectedScore){
                 score++
                 pipe.HasCollectedScore = true
                }
            })
        }
        //checks if a bird is dead
        birds.forEach((bird,i)=>{
            if(bird.isDead){
                 SavedBirds.push(bird)
                 birds.splice(i,1)
            }
        })
```

```
        //if the score is larger than total score
        if(score > TotalScore){
         HighestGen = generation
         TotalScore = score
        }
        //check if all the birds are dead
        if(birds.length < 1){
            restart()
        }
        //update the pipes
        for(let i = 0; i < pipes.length; i++){
            pipes[i].update()
            //remove the pipe if it's off the screen
            if(pipes[i].x < -pipes[i].width){
                pipes.splice(i,1)
            }
        }
        //push a new pipe into the list
        if(FrameCount % Math.floor((500 / (PipeSpeed*PipeFreq))) == 0){
            pipes.push(new Pipe)
        }
    }
    //drawing
    //draw the background
    pen.fillStyle = "rgba(100,100,255,1)"
    pen.fillRect(0,0,box.width,box.height)
    pen.drawImage(BG, 0, 0, box.width, box.height+5)
    //draw the birds
    birds.forEach(bird=>{
        bird.draw()
    })
    //draw the pipes
    pipes.forEach(pipe=>{
        pipe.draw()
    })
    //fps
    if(JSON.parse(localStorage.getItem("SettingData")).ShowFps){
        FpsCounter()
    }
```

```
        //draw and update the buttons
        ButtonInteraction(pos,buttons,false)
        //check if game was recently saved
        if(GameRecentlySaved){
            DisappearTimer--
        }
        if(DisappearTimer <= 0 ){
            GameRecentlySaved = false
            DisappearTimer = 500
        }
        pen.fillStyle = "black"
        pen.font= "20px Arial"
        pen.fillText("generation:"+generation,box.width/2,15)
        pen.fillText("Highest Score Achieved:"+TotalScore,box.width/2,35)
        pen.fillText("Achieved in generation:"+HighestGen,box.width/2,55)
        pen.fillText("Score of this generation:"+score,box.width/2,75)
        if(GameRecentlySaved){
            pen.fillText("game saved",box.width/2,95)
        }
        pen.textAlign = "start"
        pen.textBaseline = "alphabetic"
        cycles = GenSpeed.draw()
        population = BirdCount.draw()
        PipeSpeed = PipeSpeedStat.draw()
        spacing = PipeSpace.draw()
        PipeFreq = Diff.draw()
        gravity = Grav.draw()
        requestAnimationFrame(GameLoop)
    }
    GameLoop()
/script>
```

Flappy Bird Game.html

```
<title>Flappy Bird Game</title>
<script src="NeuralNetwork.js"></script>
<script src="Utils.js"></script>
<canvas id="box" width="500" height="500" style="border:0px solid ☐black"></canvas>
</canvas>
<style>
  body {
    margin: 0;
    overflow: hidden;
  }
</style>
<script>
    var box = document.getElementById("box")
    box.width = innerWidth
    box.height = innerHeight
    var pen = box.getContext("2d")
    let buttons = []
    let CanStartNow = false
    let WinnerText = " "
    let gravity = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).grav
    let PipeSpeed = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).PipeSpeed
    let PipeFreq = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).diff
    let spacing = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).Spacing
    let score = 0
    let BG = new Image(box.width,box.height)
    BG.src = "Flappy Bird BG.png"
    let pos = {
        x: 0,
        y: 0
    }
```

```
//Return to main menu button
buttons.push(new Button(
    (box.width)-200,
    (box.height) - 50,
    180,
    40,
    "#CC0000",
    "Back To Main Menu",
    20,
    "black",
    function () {
        this.color = "#FF0000";
        this.textcolor = "#FFFFFF";
    },
    () => {
        if(confirm("Are you sure you want to quit and go back to menu?")){
            window.open("Main Menu.html")
            window.close()
        }
    }
))
//pipes
let pipes = []
pipes.push(new Pipe)
//birds
let birds = []
birds.push(new Bird(true))
birds.push(new Bird)
//give the second bird the A.I
birds[1].brain = JSON.parse(localStorage.getItem("SavedDataFlappyBird")).brain
```

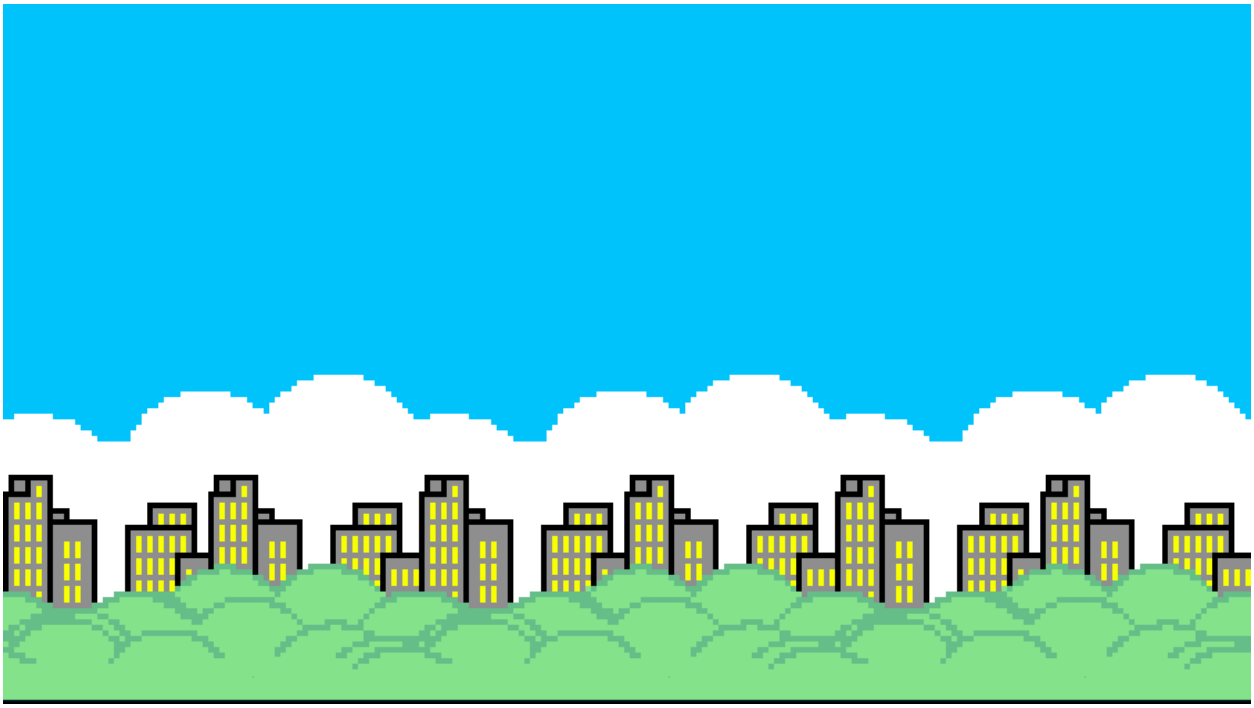
```
//control the first bird
 switch (JSON.parse(localStorage.getItem("SettingData")).JumpPref) {
  case "Click":
    document.addEventListener("click",()=>{
      birds[0].flap()
    })
    break;
    case "W key":
    document.addEventListener("keydown",(e)=>{
      if(e.keyCode == 87){
        birds[0].flap()
      }
    })
    break;
    case "Spacebar":
    document.addEventListener("keydown",(e)=>{
      if(e.keyCode == 32){
        birds[0].flap()
      }
    })
    break;
    case "Up key":
    document.addEventListener("keydown",(e)=>{
      if(e.keyCode == 38){
        birds[0].flap()
      }
    })
    break;
}
//Game items
let FrameCount = 0
```

```
//the gameloop
function Loop(){
    //updates
    FrameCount++
    //update the birds
    for(let i = 0; i < birds.length; i++){
        birds[i].update()
        //check if the birds are out of bounds
        if(birds[i].y < 0 || birds[i].y > box.height){
            birds[i].isDead = true
        }
        //collision detection with birds and pipe
        pipes.forEach(pipe=>{
        //touches the top pipe
        if(birds[i].x+birds[i].width >= pipe.x && birds[i].x <= pipe.x+pipe.width && 0 <= birds[i].y+birds[i].height && birds[i].y <= pipe.top_height){
            birds[i].isDead = true
        }
        //touching the bottom pipe
        else if(birds[i].x+birds[i].width >= pipe.x && birds[i].x <= pipe.x+pipe.width && pipe.bottom_y <= birds[i].y+birds[i].height&& birds[i].y <= pipe.bottom_y
        +pipe.bottom_height){
            birds[i].isDead = true
        }
        //if the birds have collected the score from the pipe or not
        if(birds[i].x > pipe.x+pipe.width && !pipe.HasCollectedScore){
            score++
            pipe.HasCollectedScore = true
            }
        })
    }
    //kill the bird and declare a winner
    birds.forEach((bird,i)=>{
        if(bird.isDead){
            if(bird.HumanControlled){
                window.alert("you lost, AI has won")
            }else{
                window.alert("you won!, AI has lost")
            }
            birds.splice(i,1)
            window.open("Main Menu.html")
            window.close()
        }
    })
```

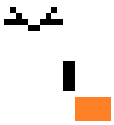
```
    //update the pipes
    for(let i = 0; i < pipes.length; i++){
        pipes[i].update()
        //remove the pipe if it's off the screen
        if(pipes[i].x < -pipes[i].width){
            pipes.splice(i,1)
        }
    }
    //push a new pipe into the list
    if(FrameCount % Math.floor((500 / (PipeSpeed*PipeFreq))) == 0){
        pipes.push(new Pipe)
    }
    //draw the background
    pen.fillStyle = "rgba(100,100,255,1)"
    pen.fillRect(0,0,box.width,box.height)
    pen.drawImage(BG, 0, 0, box.width, box.height+5)
    //draw everything else
    //draw the birds
    birds.forEach(bird=>{
        bird.draw()
    })
    //draw the pipes
    pipes.forEach(pipe=>{
        pipe.draw()
    })
    //fps
    if(JSON.parse(localStorage.getItem("SettingData")).ShowFps){
        FpsCounter()
    }
    //text
    pen.fillStyle = "black"
    pen.font= "20px Arial"
    pen.fillText("Current Score:"+score,box.width/2,20)
    pen.fillText(WinnerText,box.width/2,50)
    ButtonInteraction(pos,buttons,false)
    if(CanStartNow){
        requestAnimationFrame(Loop)
    }
}
```

```
    let Count = 11
    let CountDownText = "Game Begins in:"+Count
    for(let i = 0; i < 1; i++){
            Loop()
        }
    function CountDown(){
        Count--
        CountDownText = "Game Begins in:"+Count
        pen.fillStyle = "rgba(0,195,252,1)"
        pen.fillRect((box.width/2)-100,0,200,100)
        pen.fillStyle = "black"
        pen.font= "20px Arial"
        pen.fillText(CountDownText,box.width/2,20)
        if(Count <= 0){
            pen.fillStyle = "rgba(195,252,255,1)"
            CountDownText = " "
            pen.fillRect((box.width/2)-100,0,200,100)
            CanStartNow = true
            Loop()
            clearInterval(CountDownStart)
        }
    }
    let CountDownStart = setInterval(CountDown,1000)
    window.addEventListener("load", () => {Loop();CountDown();})
</script>
```

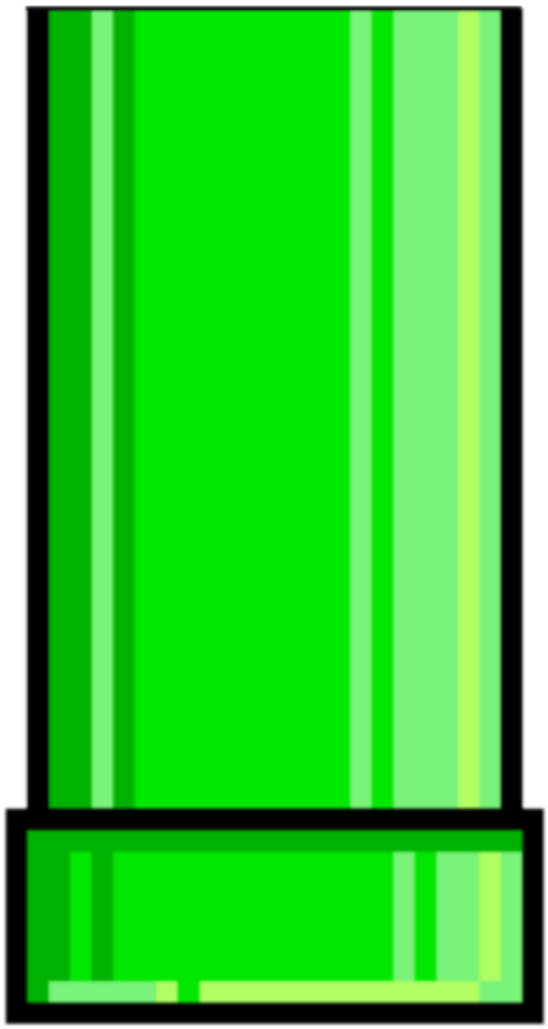
Flappy Bird BG.png



Flappy Bird Icon.png



Flappy Bird Pipe.png



LogoFlappy.png

Cube Runner AI.html

```
<title>Cube Runner AI</title>
<script src="NeuralNetwork.js"></script>
<script src="Utils.js"></script>
<canvas id="box" width="500" height="500" style="border:0px solid black"></canvas>
</canvas>
<style>
  body {
    margin: 0;
    overflow: hidden;
  }
</style>
<script>
    var box = document.getElementById("box")
    box.width = innerWidth
    box.height = innerHeight
    var pen = box.getContext("2d")
    localStorage.setItem("LastGamePlayed","Cube Runner")
    //gravity
    let Gravity = 1
    //Population
    let population = 500
    //generation
    let generation = 1
    //cycles
    let cycles = 1
    //scores
    let score = 0
    let TotalScore = 0
    let HighestGen = 1
    //saving system
    let DisappearTimer = 500
    let GameRecentlySaved = false
    //background
    var gradient = pen.createLinearGradient(0,0,0,box.height)
    gradient.addColorStop(0, 'blue')
    gradient.addColorStop(0.25, "rgba(100,100,255,1)")
    //ground
    const GroundHeight = (box.height) - 60
    //cube size
    let CubeSize = 40
```

```
//if there's non-volatile AI
if(!JSON.parse(localStorage.getItem("SettingData")).VolatileAI){
  CubeSize = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).cubesize
  generation = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).gen
  Gravity = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).grav
}
//grass columns
let grasswidth = 10
let cols = Math.round(box.width/grasswidth)
//button functionality
let pos = {
    x: 0,
    y: 0
}
buttons = []
//Return to main menu button
buttons.push(new Button(
    (box.width)-200,
    (box.height) - 50,
    180,
    40,
    "#CC0000",
    "Back To Main Menu",
    20,
    "black",
    function () {
        this.color = "#FF0000";
        this.textcolor = "#FFFFFF";
    },
    () => {
        if(confirm("Are you sure you want to quit and go back to menu?")){
            window.open("Main Menu.html")
            window.close()
        }
    }
))
```

```
//save Runner data
buttons.push(new Button(
  20,
  (box.height) - 50,
  180,
  40,
  "#00DD00",
  "Save Runner Data",
  20,
  "black",
  function () {
    this.color = "#00FF00";
    this.textcolor = "#FFFFFF";
  },
  () => {
    let data = {
      brain:JSON.parse(localStorage.getItem("bestRunnerBrain")),
      HighestScore:score,
      cubesize:CubeSize,
      grav:Gravity,
      gen:generation
    }
    localStorage.setItem("SavedDataCubeRunner",JSON.stringify(data))
    console.log("saved")
    GameRecentlySaved = true
  }
))
```

```
//directly play against AI
buttons.push(new Button(
  220,
  (box.height) - 50,
  180,
  40,
  "#00DD00",
  "Play Against A.I",
  20,
  "black",
  function () {
      this.color = "#00FF00";
      this.textcolor = "#FFFFFF";
  },
  () => {
    let data = {
      brain:JSON.parse(localStorage.getItem("bestRunnerBrain")),
      HighestScore:score,
      cubesize:CubeSize,
      grav:Gravity,
      gen:generation
    }
    localStorage.setItem("SavedDataCubeRunner",JSON.stringify(data))
    console.log("saved")
    GameRecentlySaved = true
    window.open("Cube Runner Game.html")
  }
))
```

```
//save non-volatile A.I
if(!JSON.parse(localStorage.getItem("SettingData")).VolatileAI){
  document.addEventListener("visibilitychange",() => {
    let data = {
      brain:JSON.parse(localStorage.getItem("bestRunnerBrain")),
      HighestScore:score,
      cubesize:CubeSize,
      grav:Gravity,
      gen:generation
    }
    localStorage.setItem("SavedDataCubeRunner",JSON.stringify(data))
  })
}
let Runners = []
for(let i = 0; i < population; i++){
  Runners.push(new Runner)
  if(!JSON.parse(localStorage.getItem("SettingData")).VolatileAI){
    Runners[i].brain = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).brain
        NeuralNetwork.mutate(Runners[i].brain,0.25)
    }
}
let AllObstacles = [ObstacleBlock, LongFlyingObstacle, FlyingObstacle, TallerObstcale]
let obstacles = []
obstacles.push(new FlyingObstacle)
let grass = []
for(let i = 0; i < cols+5; i++){
  grass.push(new Grass(i*grasswidth))
}
//stats for game that can be edited
let GenSpeed = new GameStat(0,20,cycles,"Generation Speed",0,100,1,buttons,"black",true)
let RunnerCount = new GameStat(0,60,population,"Runner Count",1,1000,1,buttons,"black",true)
let CubeSizes = new GameStat(0,100,CubeSize,"Cube Size",20,120,1,buttons,"black",true,10)
let Grav = new GameStat(0,140,Gravity,"Gravity",1,5,0.1,buttons,"black",true,1)
let FrameCount = 0
let SavedRunners = []
```

```
function Restart(){
  //next generation
  generation++
  //reset everything
  obstacles = []
  FrameCount = 0
  score = 0
  //find the best runner
  let BestRunner = SavedRunners[0]
  let CurrentBestScore = SavedRunners[0].score
  //get the best bird
  for(let i = 0; i < SavedRunners.length; i++){
      if(SavedRunners[i].score > BestRunner.score){
          BestRunner = SavedRunners[i]
      }
  }
  //save the brain
  localStorage.setItem("bestRunnerBrain",JSON.stringify(BestRunner.brain))
  for(let i = 0; i < population; i++){
    Runners.push(new Runner)
  }
  for(let i = 0; i < Runners.length; i++){
    Runners[i].brain = JSON.parse(localStorage.getItem("bestRunnerBrain"))
    NeuralNetwork.mutate(Runners[i].brain,0.25)
  }
  SavedRunners = []
  //prevents the program from breaking
  obstacles.push(new FlyingObstacle)
}
```

```
function GameLoop(){
  for(let i = 0; i < cycles; i++) {
    //updating
    FrameCount++
    //runners
    Runners.forEach(runner=>{
      runner.update()
      //check if any of the runners have touched the obstacles. If so, rip
      obstacles.forEach(obstacle=>{
        if(touching(runner,obstacle)){
          runner.isDead = true
        }
      })
    })
    //kill any runners if needed
    Runners.forEach((runner,i)=>{
      if(runner.isDead){
        SavedRunners.push(runner)
        Runners.splice(i,1)
      }
    })
    //restart the game if all the runners are dead
    if(Runners.length == 0){
      Restart()
    }
    //Obstacles
    obstacles.forEach(obstacle=>{
      obstacle.update()
    })
    obstacles.forEach((obstacle,i)=>{
      if(obstacle.x < -obstacle.width){
        obstacles.splice(i,1)
        score++
      }
    })
    //if the score is larger than total score
    if(score > TotalScore){
      HighestGen = generation
      TotalScore = score
    }
```

```
    if(FrameCount % 80 == 0){
      let NewItem = Math.floor(Math.random()* AllObstacles.length)
      obstacles.push(new AllObstacles[NewItem])
    }
    //update the grass
    grass.forEach((Grasss,i)=>{
      if(Grasss.x <= -Grasss.width){
        grass.splice(i,1)
        grass.push(new Grass(box.width))
      }
      Grasss.update()
    })
  }
  //Drawing
  //draw the background
  pen.fillStyle = gradient
  pen.fillRect(0,0,box.width,box.height)
  //The runners
  Runners.forEach(runner=>{
    runner.draw()
  })
  //Obstacles
  obstacles.forEach(obstacle=>{
    obstacle.draw()
  })
  //check if game was recently saved
  if(GameRecentlySaved){
      DisappearTimer--
  }
  if(DisappearTimer <= 0 ){
    GameRecentlySaved = false
    DisappearTimer = 500
  }
  //Floor and text
  pen.fillStyle = "#80471C"
  pen.fillRect(0,GroundHeight,box.width,60)
  grass.forEach(grass=>{
    grass.draw()
  })
```

```
    //fps
    if(JSON.parse(localStorage.getItem("SettingData")).ShowFps){
      FpsCounter()
    }
    pen.fillStyle = "black"
    pen.font= "20px Arial"
    pen.fillText("generation:"+generation,box.width/2,15)
    pen.fillText("Highest Score Achieved:"+TotalScore,box.width/2,35)
    pen.fillText("Achieved in generation:"+HighestGen,box.width/2,55)
    pen.fillText("Score of this generation:"+score,box.width/2,75)
    if(GameRecentlySaved){
      pen.fillText("game saved",box.width/2,95)
    }
    cycles = GenSpeed.draw()
    population = RunnerCount.draw()
    CubeSize = CubeSizes.draw()
    Gravity = Grav.draw()
    ButtonInteraction(pos,buttons,false)
    requestAnimationFrame(GameLoop)
  }
  GameLoop()
</script>
```

Cube Runner Game.html

```
<title>Cube Runner Game</title>
<script src="NeuralNetwork.js"></script>
<script src="Utils.js"></script>
<canvas id="box" width="500" height="500" style="border:0px solid ■black"></canvas>
</canvas>
<style>
  body {
    margin: 0;
    overflow: hidden;
  }
</style>
<script>
    var box = document.getElementById("box")
    box.width = innerWidth
    box.height = innerHeight
    var pen = box.getContext("2d")
    localStorage.setItem("LastGamePlayed","Cube Runner")
    //gravity
    let Gravity = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).grav
    //cube size
    let CubeSize = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).cubesize
    //gradient background
    let gradient = pen.createLinearGradient(0,0,0,box.height)
    gradient.addColorStop(0, 'blue')
    gradient.addColorStop(0.25, "rgba(100,100,255,1)")
    //ground
    const GroundHeight = (box.height) - 60
    //grass columns
    let grasswidth = 10
    let cols = Math.round(box.width/grasswidth)
    //grass
    let grass = []
    for(let i = 0; i < cols+5; i++){
      grass.push(new Grass(i*grasswidth))
    }
    //scores
    let score = 0
    //button functionality
    let pos = {
        x: 0,
        y: 0
    }
```

```
buttons = []
//Return to main menu button
buttons.push(new Button(
    (box.width)-200,
    (box.height) - 50,
    180,
    40,
    "#CC0000",
    "Back To Main Menu",
    20,
    "black",
    function () {
        this.color = "#FF0000";
        this.textcolor = "#FFFFFF";
    },
    () => {
        if(confirm("Are you sure you want to quit and go back to menu?")){
            window.open("Main Menu.html")
            window.close()
        }
    }
))
//Runners
let Runners = []
//the AI
Runners.push(new Runner)
Runners[0].brain = JSON.parse(localStorage.getItem("SavedDataCubeRunner")).brain
//The Player
Runners.push(new Runner(true))
Runners[1].color = "rgba(223,204,67,1)"
Runners[1].eyesColor = "rgba(240,10,141,1)"
```

```
//player movement
switch (JSON.parse(localStorage.getItem("SettingData")).JumpPref) {
  case "Click":
    document.addEventListener("click",()=>{
      Runners[1].Jump()
    })
    break;
    case "W key":
    document.addEventListener("keydown",(e)=>{
      if(e.keyCode == 87){
        Runners[1].Jump()
      }
    })
    break;
    case "Spacebar":
    document.addEventListener("keydown",(e)=>{
      if(e.keyCode == 32){
        Runners[1].Jump()
      }
    })
    break;
    case "Up key":
    document.addEventListener("keydown",(e)=>{
      if(e.keyCode == 38){
        Runners[1].Jump()
      }
    })
    break;
}
document.addEventListener("keydown",(e)=>{
  if(e.keyCode == 49){
    if(Runners[1].IsCrouching){
      Runners[1].UnCrouch()
    }else{
      Runners[1].Crouch()
    }
  }
})
```

```
//obstacles
let AllObstacles = [ObstacleBlock, LongFlyingObstacle, FlyingObstacle, TallerObstcale]
let obstacles = []
obstacles.push(new FlyingObstacle)
//Game Loop
let FrameCount = 0
function GameLoop(){
  //updating
  FrameCount++
  //update the grass
  grass.forEach((Grasss,i)=>{
    if(Grasss.x <= -Grasss.width){
      grass.splice(i,1)
      grass.push(new Grass(box.width))
    }
    Grasss.update()
  })
  //runners
  Runners.forEach(runner=>{
    runner.update()
    //check if any of the runners have touched the obstacles. If so, rip
    obstacles.forEach(obstacle=>{
      if(touching(runner,obstacle)){
        runner.isDead = true
      }
    })
  })
  //kill any runners if needed
   Runners.forEach((runner,i)=>{
    if(runner.isDead){
      if(runner.IsHumanControlled){
        window.alert("you lost, AI has won")
      }else{
        window.alert("you won!, AI has lost")
      }
      Runners.splice(i,1)
      window.open("Main Menu.html")
      window.close()
    }
  })
```

```
//Obstacles
obstacles.forEach(obstacle=>{
  obstacle.update()
})
obstacles.forEach((obstacle,i)=>{
  if(obstacle.x < -obstacle.width){
    obstacles.splice(i,1)
    score++
  }
})
if(FrameCount % 80 == 0){
  let NewItem = Math.floor(Math.random()* AllObstacles.length
  obstacles.push(new AllObstacles[NewItem])
}
//Drawing
//draw the background
pen.fillStyle = gradient
pen.fillRect(0,0,box.width,box.height)
//Floor
pen.fillStyle = "#80471C"
pen.fillRect(0,GroundHeight,box.width,60)
grass.forEach(grass=>{
  grass.draw()
})
//Obstacles
obstacles.forEach(obstacle=>{
  obstacle.draw()
})
//The runners
Runners.forEach(runner=>{
  runner.draw()
})
//score
pen.fillStyle = "black"
pen.font= "20px Arial"
pen.fillText("Current Score:"+score,box.width/2,15)
//fps
if(JSON.parse(localStorage.getItem("SettingData")).ShowFps){
  FpsCounter()
}
```

```
    ButtonInteraction(pos,buttons,false)
    requestAnimationFrame(GameLoop)
  }
  GameLoop()
/script>
```

LogoRunner.png