**Capstone Project: Train an AI Agent to Play Flappy Bird**

**Team EyeConic**

**Brooke Broderick - W217065417**

**Matthew Choo - W206744145**

**Cameroun White-W216530571**

**Melvis Maduagwu-W216890463**

**Erick Banegas - W216200956**

**NancyChieu-W000098778**

**Yoana Cook - W215913890**

**6252-ITAI-1378-Comp Vision-Artificial Intel-RT-15698 - Spring 2025**

**Professor Anna Devarakonda**

**April 1, 2025**

**Intro:**

* **In this capstone project, team eyeconic is tasked to understand or implement the process training of an AI agent to play a game called Flappy Bird by using computer vision and reinforcement learning. There are two paths for this project where we can either do the conceptual path and a coding path.**

**Path 1: Conceptual Path (No Coding Required)**

**1. Environment Setup**

* **Flappy Bird Game Environment**
* **Flappy Bird is a 2D game where a player controls a bird attempting to navigate through gaps between pipes. The key components of the environment include:**
* **Graphics: Pixel-based 2D environment.**
* **Physics: Gravity constantly pulls the bird downward, while the player can apply an upward force by flapping.**
* **Scoring System: +1 point for passing a pipe; the game ends if the bird collides with a pipe or the ground.**

**2. Libraries and Tools**

* **PyGame: Useful for replicating the Flappy Bird environment from scratch.**
* **OpenAI Gym: Provides ready-to-use reinforcement learning (RL) environments, including Flappy Bird-like setups.**

**3. Game Setup for AI Interaction**

* **State Representation: The agent perceives the game through features like the bird’s position, velocity, and distance to pipes.**
* **Action Space: Two discrete actions: flap (jump) or do nothing (fall).**
* **Reward System: +1 for passing a pipe, -1 for collision, and small positive rewards for survival time.**

**4. Preprocessing Game Frames for AI Input**

* **Resizing: Reduce frame size to decrease computational complexity.**
* **Grayscale Conversion: Remove redundant color information to focus on structure.**
* **Frame Stacking: Stack multiple frames to capture motion over time.**

**2. Pre-trained Model Integration**

1. **Concept of Transfer Learning**

* **Transfer learning involves using a pre-trained model on a related task to improve efficiency and performance while reducing training time.**

1. **Selecting a Pre-trained Model**

* **MobileNetV2 or ResNet: Useful for extracting visual features from game frames.**

**Justification: These models are trained on large datasets (e.g., ImageNet) and excel at feature extraction.**

1. **Modifying the Pre-trained Model**

* **Remove fully connected layers.**
* **Use convolutional layers to extract spatial features relevant to the game environment.**

1. **Challenges & Solutions**

* **Challenge: The model is trained on real-world images while Flappy Bird is pixel-based.**
* **Solution: Fine-tune the model on game-specific data.**
* **Challenge: Adapting CNN features to an RL framework.**
* **Solution: Use extracted features as inputs to an RL network.**

**3. Reinforcement Learning Implementation**

1. **Basics of RL**

* **State: The bird’s position, velocity, and distance to pipes.**
* **Actions: Flap (jump) or do nothing (fall).**
* **Reward: +1 for passing a pipe, -1 for collision, and small survival rewards.**
* **Policy: Strategy that determines actions based on the current state.**

1. **Algorithm Choice**

* **Deep Q-Networks (DQN): Uses a neural network to estimate Q-values for each action, and learns to take actions that maximize future rewards.**

1. **Components Needed**

* **Q-network architecture: Convolutional neural network (CNN) to process frames and output Q-values.**
* **Replay memory: Stores past experiences for training.**
* **Target network: Stabilizes learning by reducing fluctuations in Q-value updates.**

1. **Handling Exploration-Exploitation**

* **Use an ε-greedy strategy:**
* **Start with high exploration (random actions).**
* **Gradually shift toward exploitation (choosing the best-known action).**

1. **Experience Replay**

* **Stores past experiences to improve sample efficiency and break correlation between consecutive game states.**

**4. Model Training**

1. **Training Process**

**Initialize Q-network, replay memory, and hyperparameters.**

**Play the game, collecting (state, action, reward, next state) data.**

**Store experiences in replay memory.**

**Sample batches from memory and update the Q-network using the Bellman equation.**

**Repeat until performance stabilizes.**

1. **Training Loop Setup**

**Define episodes (one episode = one full game session).**

**At each step:**

**Observe state → Choose action → Get reward → Store experience → Train model.**

1. **Hyperparameter Tuning**

**Learning rate: Determines how much the model updates (e.g., 0.001).**

**Batch size: Number of samples per training iteration (e.g., 32, 64).**

**Discount factor (γ): Balances immediate vs. long-term rewards.**

1. **Handling Training Issues**

**Catastrophic Forgetting: Use target networks to stabilize learning.**

**Reward Sparsity: Implement small survival rewards to encourage learning.**

1. **Model Evaluation During Training**

**Track average score and survival time.**

**Observe Q-value trends to check policy learning improvements.**

**5. Testing and Evaluation**

1. **Testing Strategy**

**Run multiple episodes with the trained agent.**

**Evaluate performance across different game conditions.**

1. **Performance Metrics**

**Average score over multiple games.**

**Survival time (number of frames survived).**

**Action efficiency (fewer unnecessary flaps).**

1. **Interpreting Results & Benchmarking**

**Compare against a random agent (flapping randomly).**

**Assess generalization across different obstacle placements.**

1. **Visualizing Performance**

**Plot scores over time to track improvement.**

**Use heatmaps to analyze where the agent struggles most.**

1. **Potential Improvements & Future Work**

**Implement policy gradient methods (e.g., PPO, A3C) for better performance.**

**Use curriculum learning to gradually increase difficulty.**

**Train in different variations (e.g., faster pipes, changing gravity).**

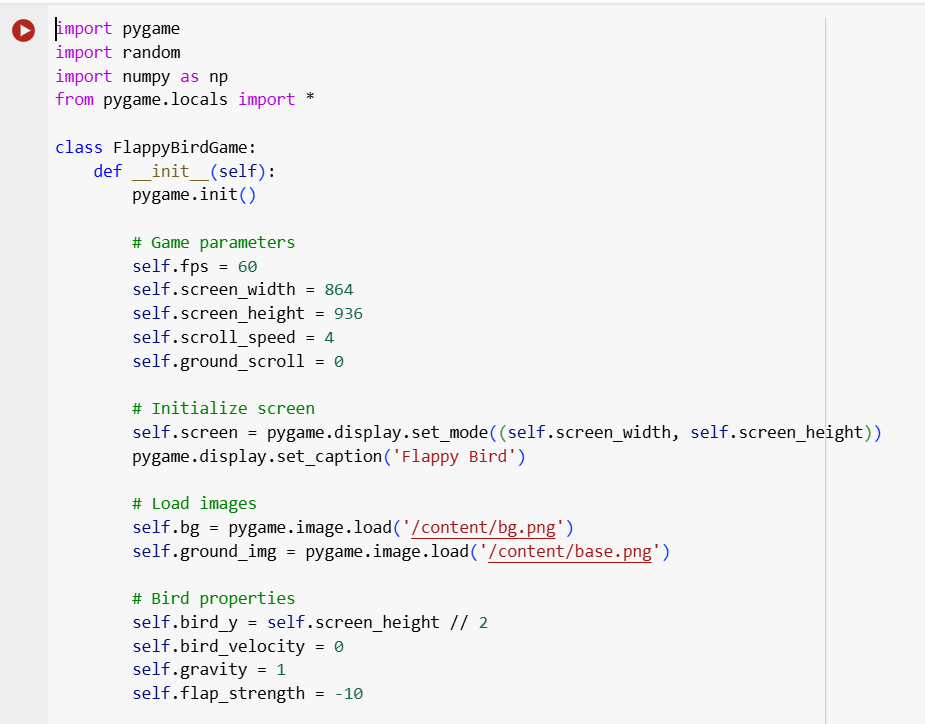
**Path 2: Coding Path**

**Intro: In this path, team eyeconic will create a code where the AI agent will play a game called Flappy Bird.**

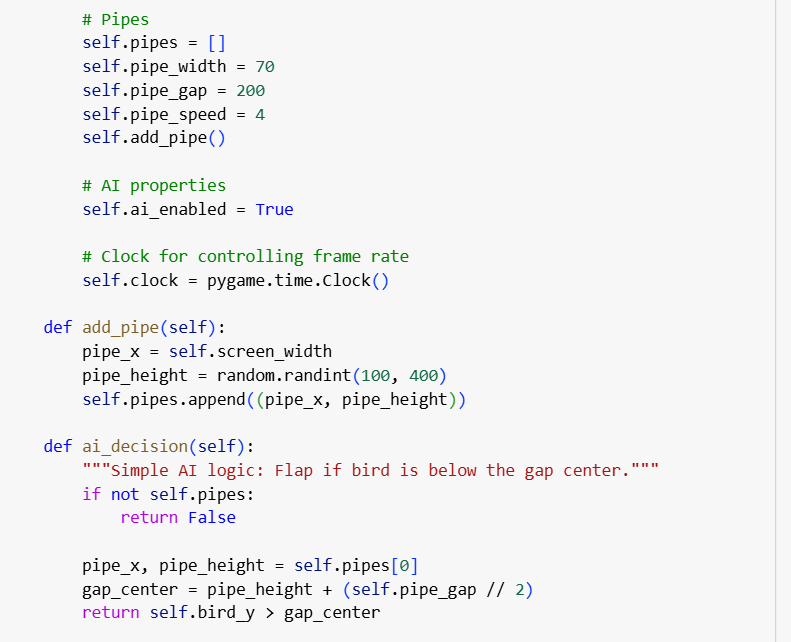
**1. Environment Setup = Matthew Choo and Melvis**

* **We used both google colab and Pycharm to make an AI agent play the Flappy Bird Game. In figures 1 - 5, it shows that after running the code, it will produce a time loop when running it. Then on figures 6 - 14 the same code is used on pycharm and after running it, the game is produced showing the AI playing the Flappy Bird Game.** **Then in figures 15 - 19 we did a second version of the google colab showing the same thing but trying to have the AI lose. Lastly in figures 20 - 24 Vscode was used on the V2of Flappy bird and it was showing the same results.**

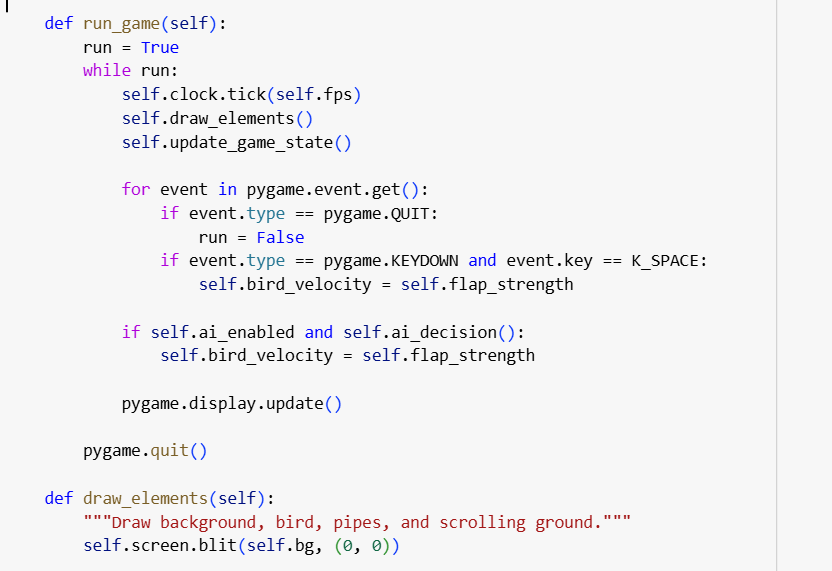
**Google Collab**



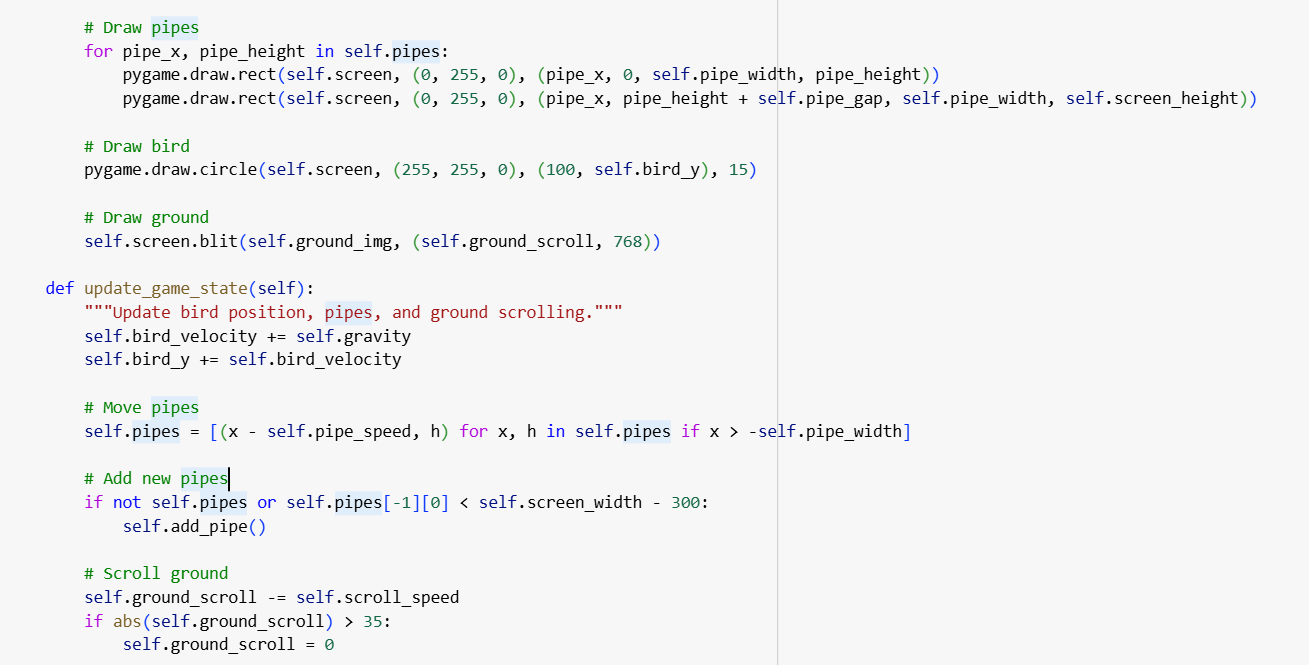
**Fig 1: Google Collab Results Part 1**



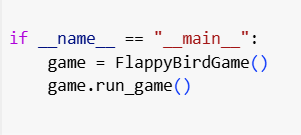
**Fig 2: Google Collab Results Part 2**



**Fig 3: Google Collab Results Part 3**

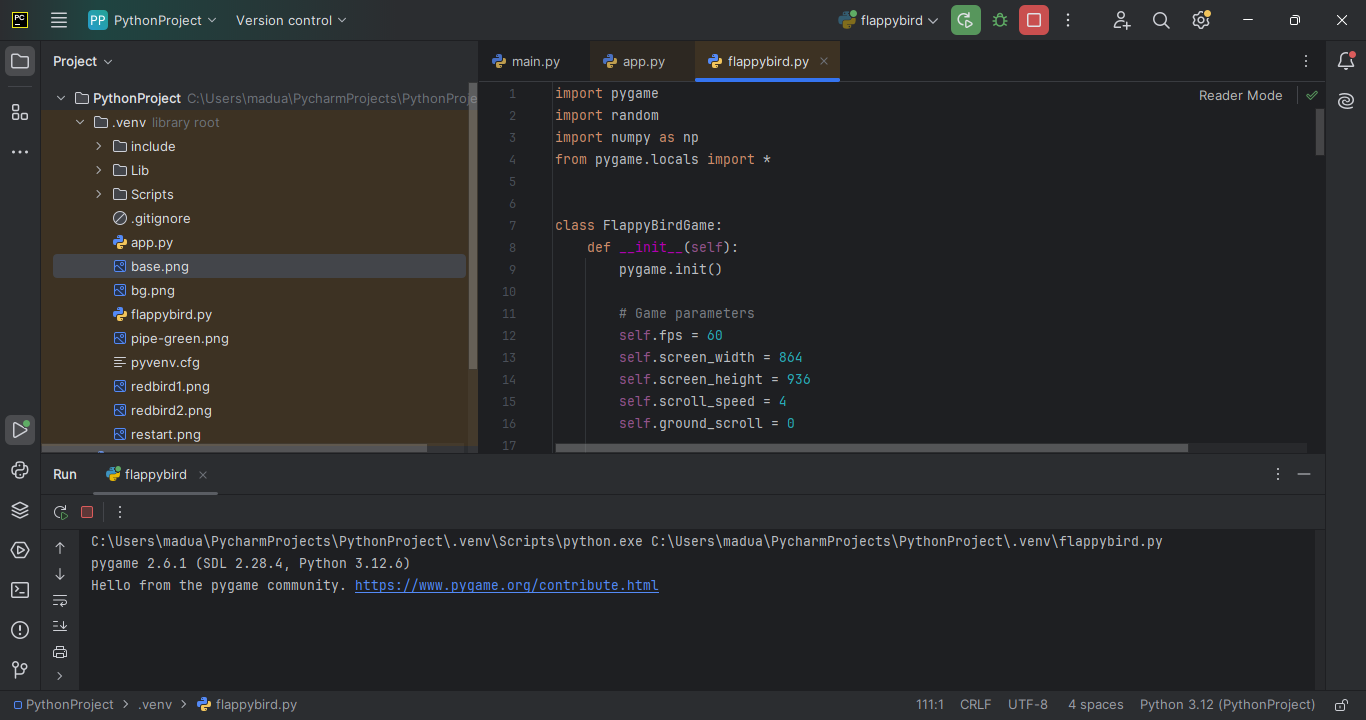


**Fig 4: Google Collab Results Part 4**

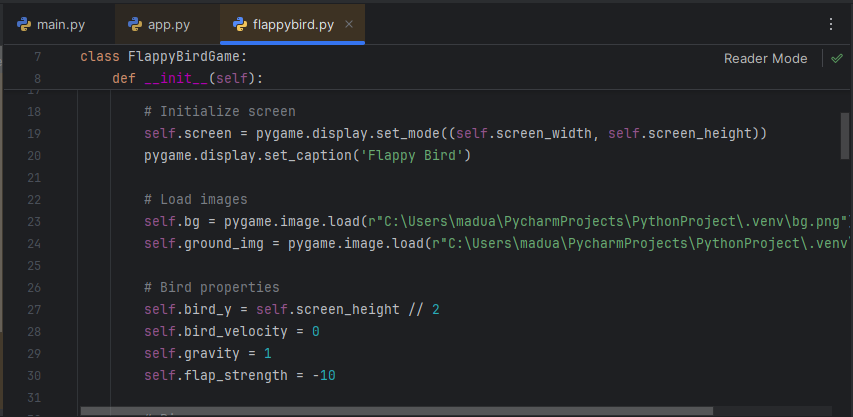


**Fig 5: Google Collab Results Part 5**

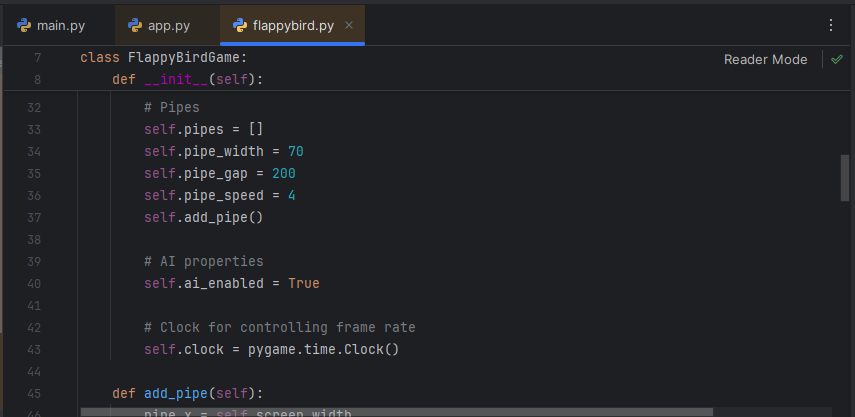
**PyCharm**



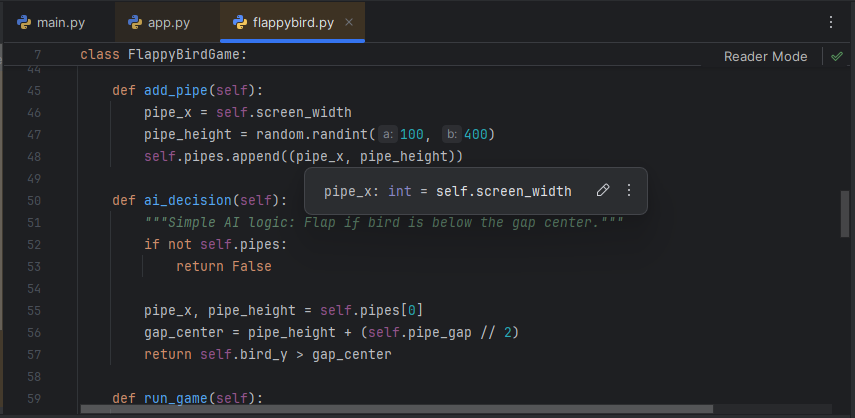
**Fig 6: PyCharm Results Part 1**



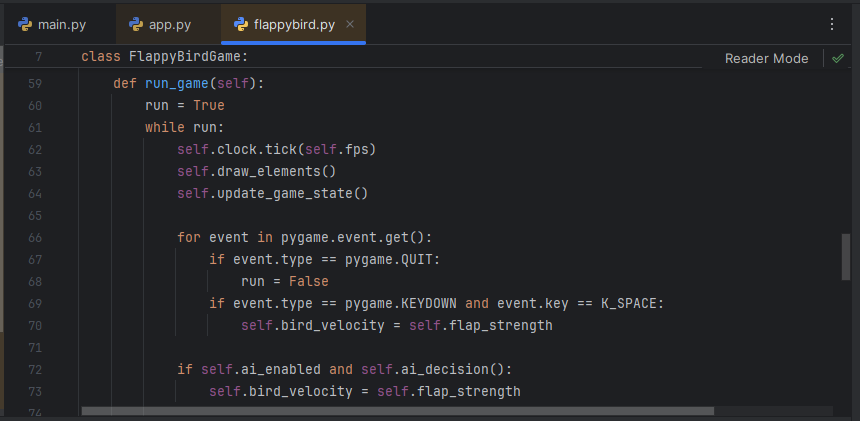
**Fig 7: PyCharm Results Part 2**



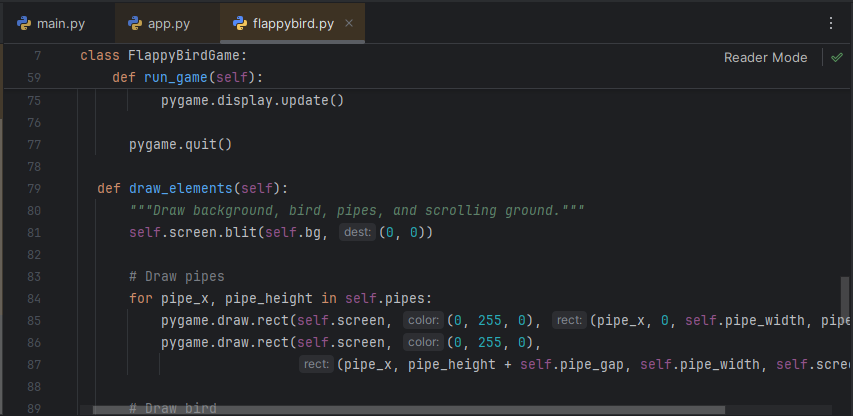
**Fig 8: PyCharm Results Part 3**



**Fig 9: PyCharm Results Part 4**



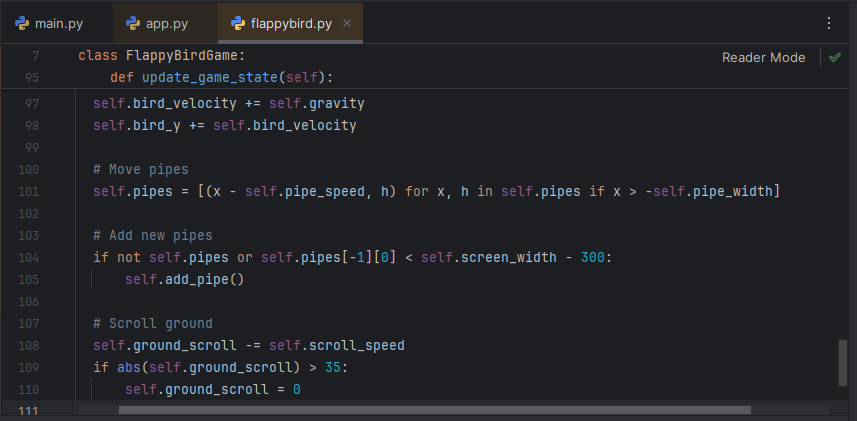
**Fig 10: PyCharm Results Part 4**



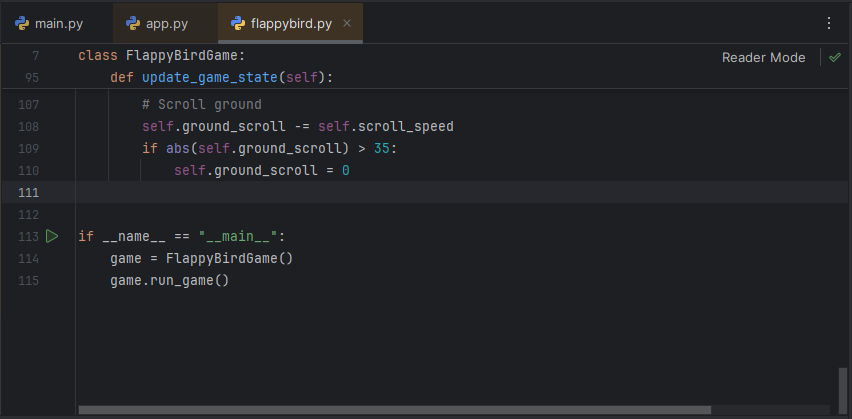
**Fig 11: PyCharm Results Part 5**



**Fig 12: PyCharm Results Part 6**



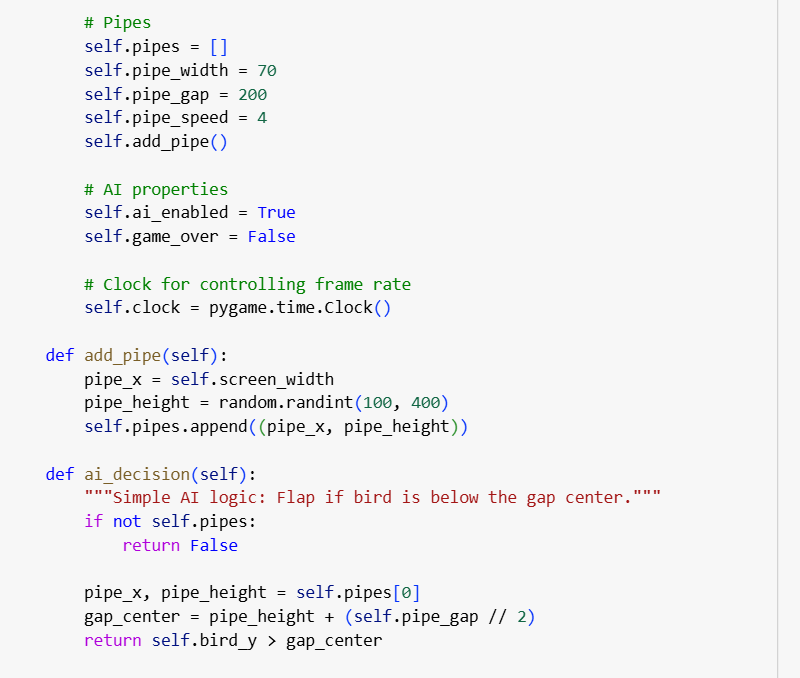
**Fig 13: PyCharm Results Part 7**



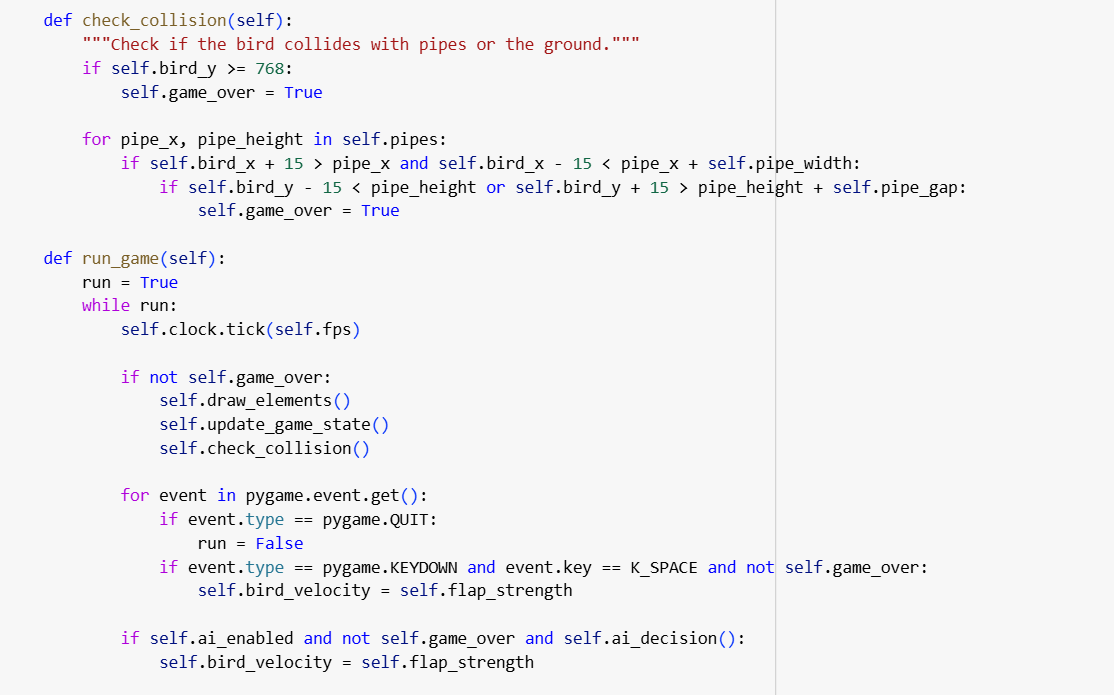
**Fig 14: PyCharm Results Part 8**

**Google Collab V2**

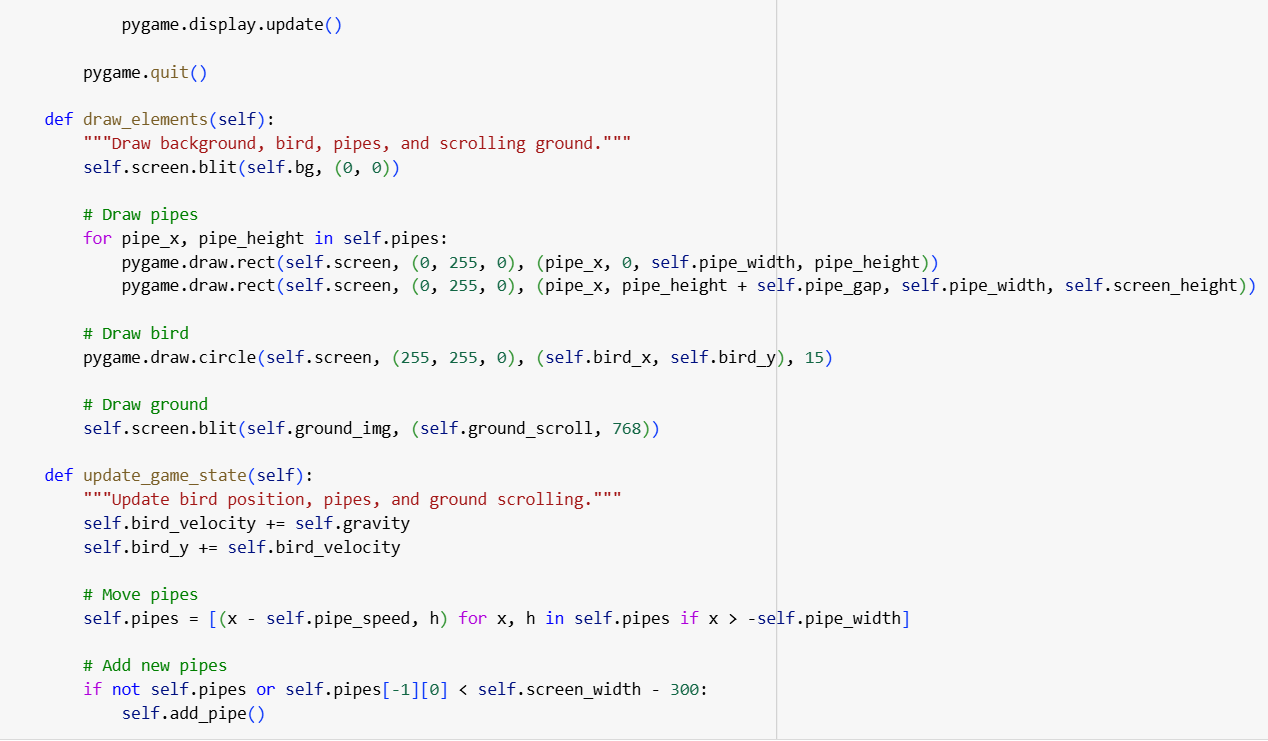
**Fig 15: Google Collab Results of V2 Part 1**

****

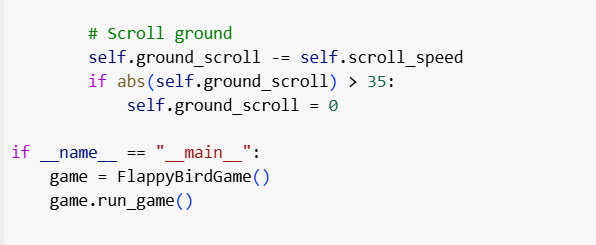
**Fig 16: Google Collab Results of V2 Part 2**

****

**Fig 17: Google Collab Results of V2 Part 3**

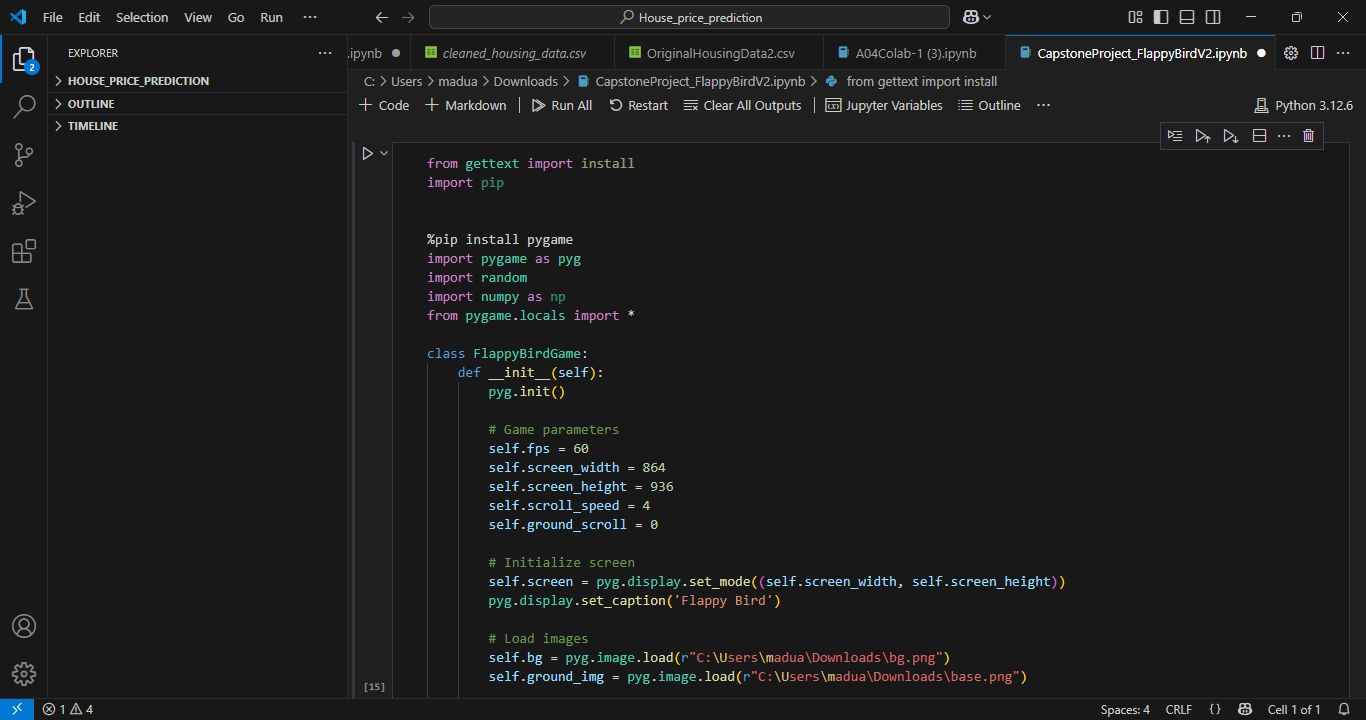
****

**Fig 18: Google Collab Results of V2 Part 4**

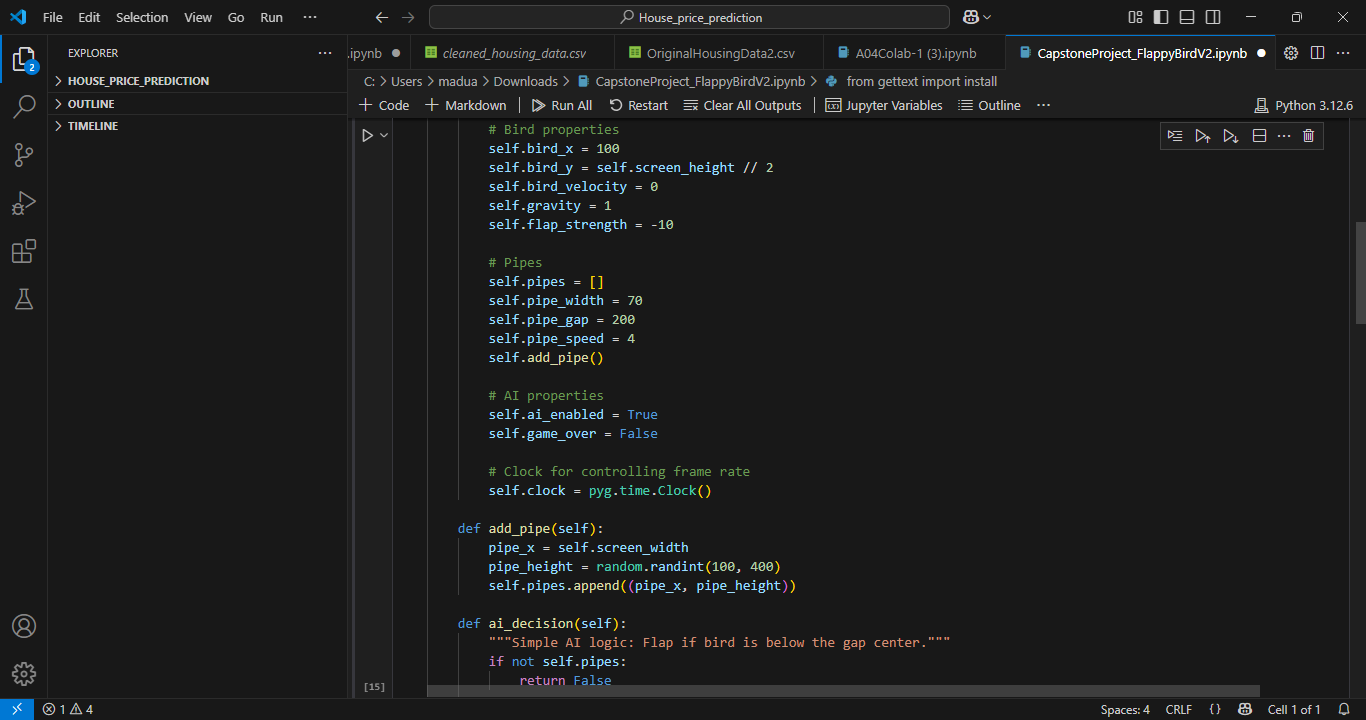
****

**Fig 19: Google Collab Results of V2 Part 5**

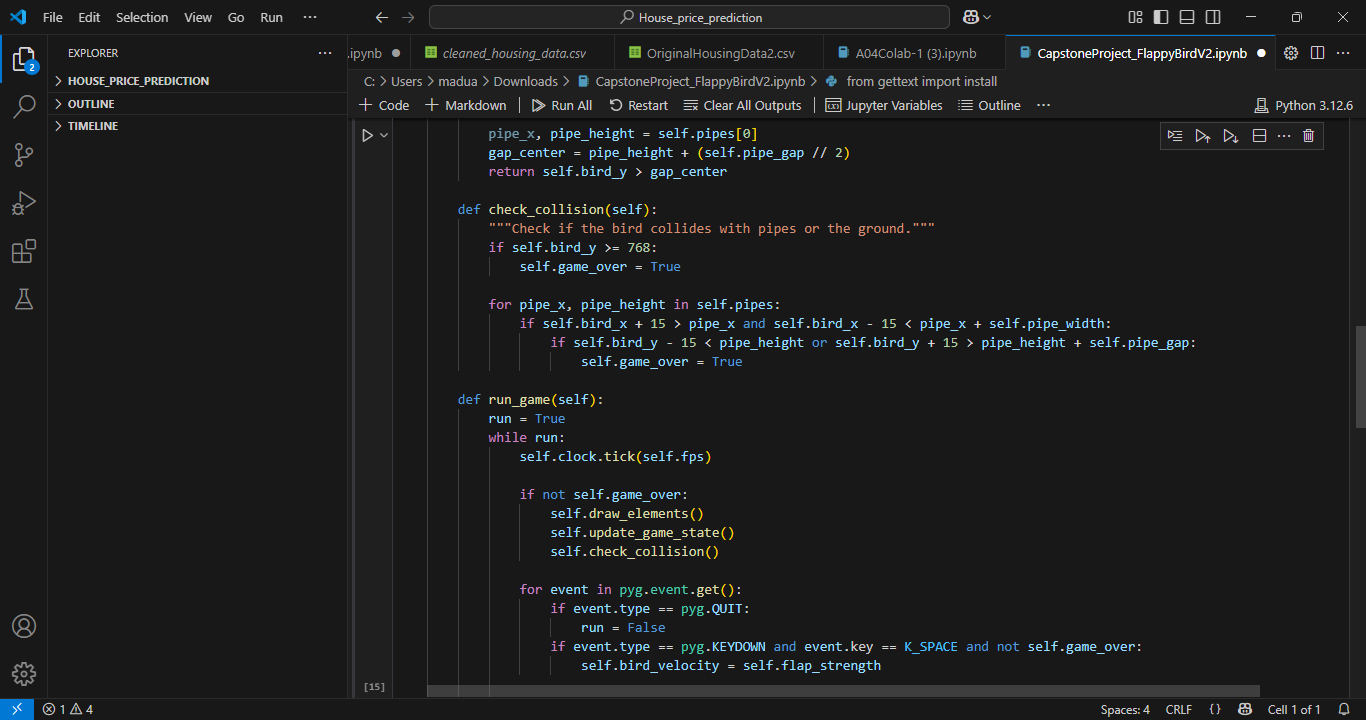
**Vscode**

****

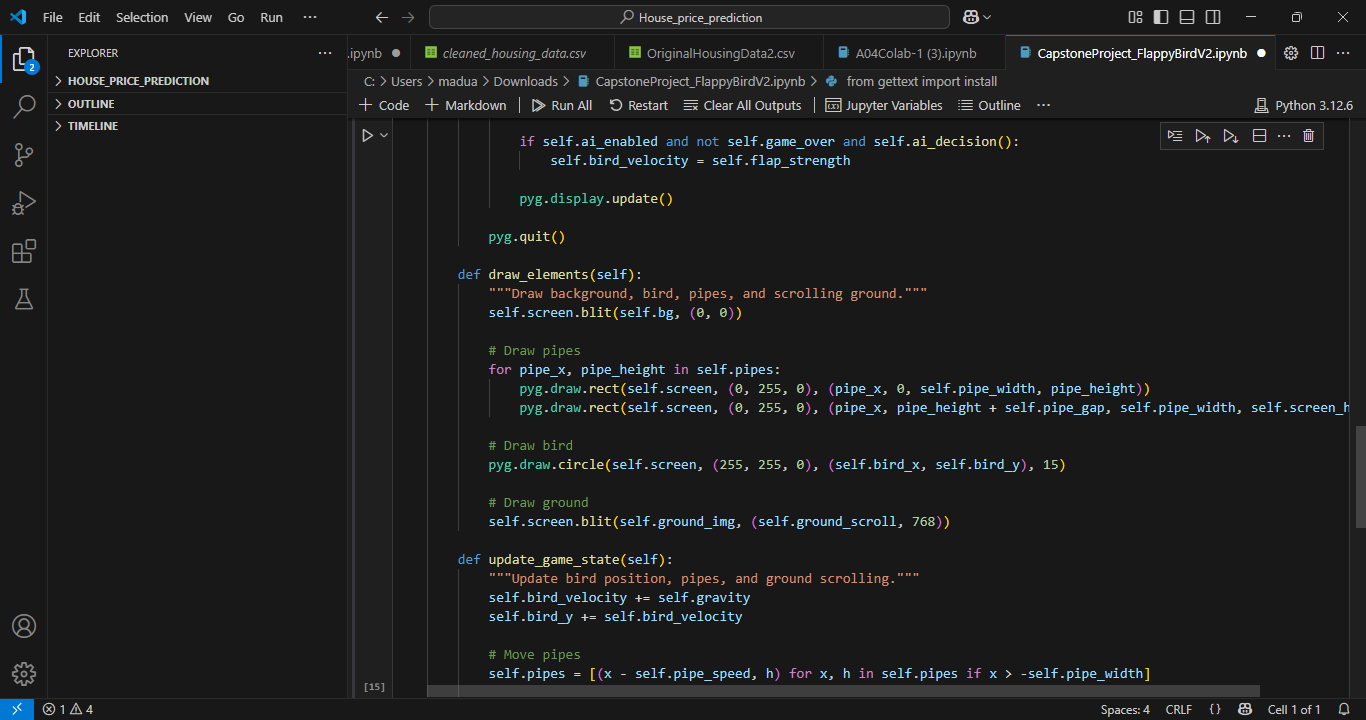
**Fig 20: Vscode Results Part 1**

****

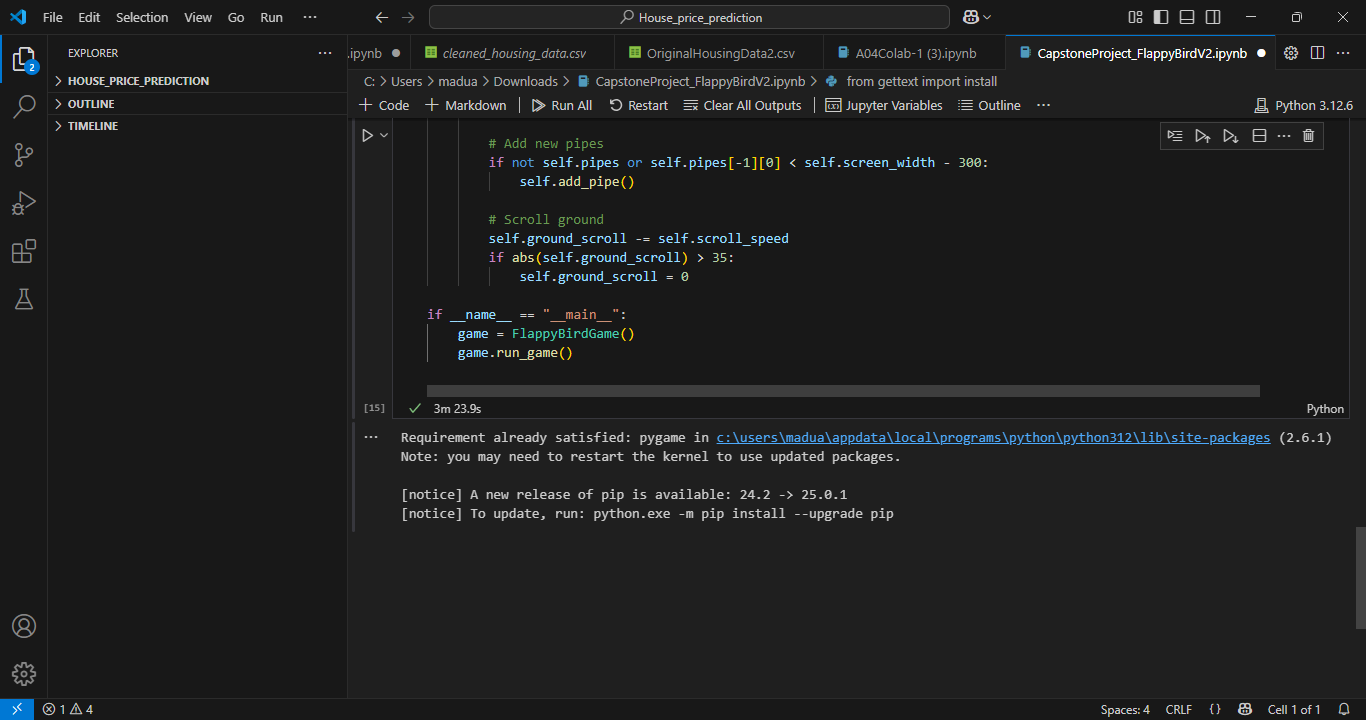
**Fig 21: Vscode Results Part 2**

****

**Fig 22: Vscode Results Part 3**

****

**Fig 23: Vscode Results Part 4**

****

**Fig 24: Vscode Results Part 5**

**2. Pre-trained Model Integration**

* **We integrated a pre-trained model into a Flappy Bird environment to enhance gameplay automation and decision-making. Utilizing reinforcement learning techniques, particularly deep Q-networks (DQN), the model is trained on numerous gameplay iterations to optimize bird movements efficiently. The implementation is carried out in PyCharm, leveraging Python libraries such as TensorFlow/PyTorch for deep learning, OpenAI Gym for simulation, and Pygame for game rendering. By integrating a pre-trained model, the Flappy Bird agent can make real-time decisions, avoiding obstacles and improving its score without requiring extensive retraining. This integration demonstrates the effectiveness of transfer learning, where a model trained on similar tasks can be fine-tuned for specific applications. The approach not only reduces computational cost but also showcases the potential of AI in game automation and reinforcement learning applications.**

**3. Reinforcement Learning Implementation**

**Basics of Reinforcement Learning (RL)**

**\*State: The environment observation:**

**Bird’s position (x,y coordination), Bird’s velocity,**

**Bird’s horizontal and vertical distances to the next pipe**

**\*Actions: Flap (jump) or do nothing (fall with gravity).**

**\*Reward: +1 for passing a pipe, -1 for collision, and +0.1 for small survival rewards.**

**\*Strategy: select an action based on the current state to maximize future cumulative reward.**

**\*Algorithm Choice**

* **Deep Q-Networks (DQN): Uses a neural network to estimate Q-values- Q(*s, a*) for each action given a state *s*, and learns to take actions that maximize future rewards.**

**\*Components Needed**

* **Q-network architecture: Convolutional neural network (CNN) to process input frames (visual or structured state) and output Q-values for each possible action.**
* **Replay memory: Stores past experiences for training, and improves data efficiency.**
* **Target network: Stabilizes learning by reducing fluctuations in Q-value updates.**

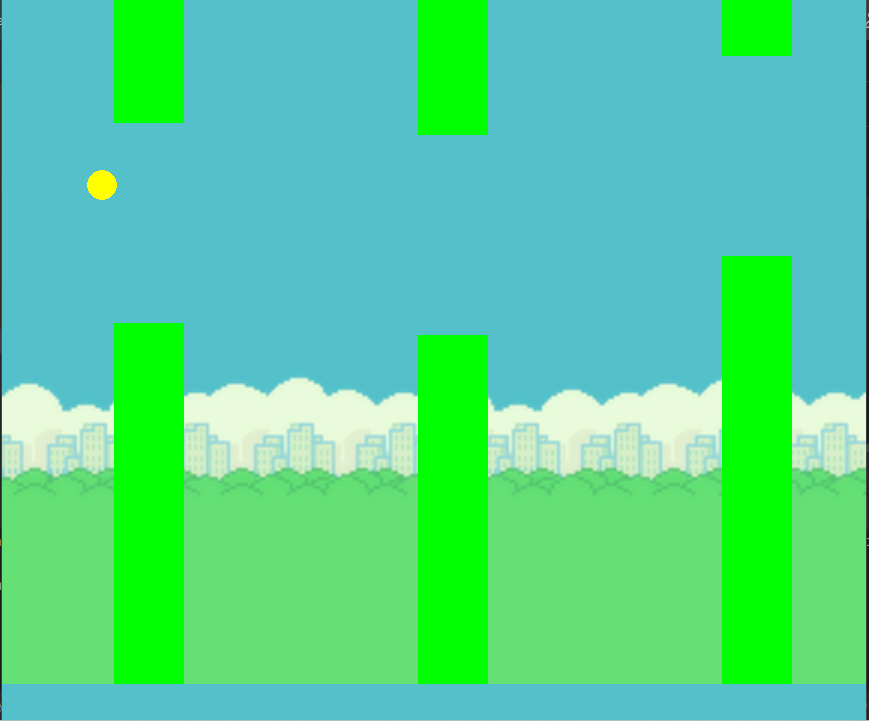
**\*Handling Exploration-Exploitation**

* **Use probability ε = 1 to start with high exploration (random actions).**
* **Gradually shift to ε = 0.01 over training toward exploitation (best prediction).**

**\*Experience Replay**

* **Stores past experiences for reuse to improve data efficiency and break correlation between consecutive game states.**

**4. Model Training - Results**



**Fig 25: Result of AI Agent Playing Flappy Birds**

* **We developed this project as a team using Python and the Pygame library with the goal of creating a simple yet functional version of the Flappy Bird game. All the game logic is organized within a class called FlappyBird Game, where we define the main configurations such as screen size, scroll speed, bird gravity, and dynamic pipe generation. From the very beginning of the game, we generate an initial pipe to ensure there’s an obstacle right away, and as the game progresses, the pipes move to the left to simulate forward movement. We draw all elements on the screen in each frame: the background, the pipes (both top and bottom), the bird represented by a circle, and the ground, which scrolls continuously to create the illusion of motion. We programmed the bird to fall due to gravity and also implemented functionality to make it jump when the space key is pressed. Additionally, we included a basic artificial intelligence function that determines whether the bird should jump based on its position relative to the gap in the pipes. We also handled the ground scrolling logic and created a system for continuously generating new pipes while removing the old ones once they exit the screen. While we haven’t yet implemented collision detection, scoring, or start/end screens, we’ve built a solid and clean structure that will allow us to easily add these features later on. This project has been a great opportunity for us to learn more about how video games are built from scratch, how to work with real-time graphics and game loops, and how to handle keyboard events. It also gave us an introductory experience in incorporating basic AI logic into an interactive game environment. Overall, it was a very useful and enriching process for strengthening our skills in programming and game development.**

**5. Testing and Evaluation - Reflection = Yoana Cook**

### **Reflection on Training an AI Agent for Flappy Bird**

Throughout this capstone project, our team embarked on the ambitious journey of training an AI agent to play Flappy Bird using computer vision and reinforcement learning. We explored different methodologies, experimented with various implementations, and utilized multiple resources, including YouTube tutorials and online forums, such as the Medium and GitHub. This experience provided us with a deeper understanding of reinforcement learning principles, model training processes, and the challenges of working with different machine learning environments.

From the outset, our team investigated multiple versions of AI training for Flappy Bird. We looked at both conceptual explanations and practical implementations, analyzing different reinforcement learning algorithms such as *Deep Q-Networks (DQN*) and *Proximal Policy Optimization (PPO)*. Initially, we attempted to implement a DQN-based solution using existing frameworks and tutorials, refining our approach based on insights from YouTube videos and online documentation.

One of our main goals was to leverage a pre-trained model for feature extraction, which would help accelerate training by reducing the complexity of the input state space. We explored *MobileNetV2* and *ResNet* as potential candidates for feature extraction but faced difficulties in integrating them effectively with our reinforcement learning pipeline. While these models offered computational efficiency, adapting them to the Flappy Bird environment required significant preprocessing and tuning.

To implement our project, we utilized several key tools and libraries: *Stable-Baselines3:* A reinforcement learning framework that provided pre-implemented algorithms, such as DQN and PPO, making it easier to structure our agent’s learning process. *Gymnasium (OpenAI Gym Fork):* Used to create and interact with the Flappy Bird environment, defining the action and observation spaces required for training. *TensorFlow & Keras:* Enabled the creation and training of deep learning models, particularly the Q-network for our reinforcement learning agent. *OpenCV:* Played a crucial role in preprocessing game frames by converting them to grayscale, resizing them, and normalizing pixel values to improve learning efficiency. *PyGame:* Used as the underlying game framework for rendering the Flappy Bird environment and handling real-time game interactions. *NumPy:* Assisted in handling large numerical computations efficiently, including experience replay memory storage and processing. Matplotlib: Helped visualize training performance, such as plotting reward curves and evaluating the agent’s progress over time.

Each of these libraries played a vital role in different aspects of the project, from setting up the training environment to handling data processing, model training, and evaluation.

One of the biggest roadblocks we encountered was the execution environment. Initially, we attempted to run our various implementations in Google Colab, expecting it to provide a smooth workflow with GPU acceleration. However, we quickly ran into compatibility issues with Colab’s setup. Some key challenges included: *Library and Package Conflicts:* Google Colab’s pre-installed packages at times conflicted with the specific versions required for our project, particularly the Keras. *Rendering Issues*: Since Colab runs in a cloud environment, rendering the Flappy Bird game environment in real-time proved difficult, making it hard to visualize the AI agent’s learning progress. *Session Timeouts:* Colab's automatic disconnection caused interruptions in long training sessions, affecting our ability to train the model effectively.

After several unsuccessful attempts, we decided to transition to a local development environment using PyCharm. This shift allowed us greater flexibility in managing dependencies, visualizing the environment, and maintaining stable training sessions. However, migrating from Colab to PyCharm also introduced new challenges, such as setting up virtual environments, ensuring compatibility across different TensorFlow and Keras versions, and managing memory constraints.

Despite the technical difficulties, this project provided us with invaluable insights into reinforcement learning. We gained firsthand experience in: *Setting Up AI Training Environments*: Understanding the importance of choosing the right platform and ensuring package compatibility. *Experimentation and Adaptability:* Learning that AI development is not a linear process, multiple iterations and trials are necessary to refine the approach, lots of trial and error. *Overcoming Computational Limitations:* Recognizing the need for efficient training techniques, such as experience replay and target network updates, to improve performance while managing limited resources. *Theoretical and Practical Integration:* Bridging the gap between conceptual AI knowledge and real-world implementation challenges.

Moving forward, we see several potential improvements to our approach. Optimizing hyperparameters, exploring alternative architectures, and leveraging more robust reinforcement learning algorithms could enhance the model’s efficiency. Additionally, using a dedicated GPU environment, such as a local high-performance machine or an external cloud-based service, would likely mitigate the computational limitations we faced.

Computer vision played a fundamental role in our project, as it allowed the AI agent to interpret the game environment effectively. By converting visual input into a structured representation, our model could make informed decisions about when to flap and when to stay idle. The preprocessing steps, including grayscale conversion and frame resizing, ensured that the neural network received a simplified yet informative version of the game screen, improving learning efficiency and generalization.

In conclusion, after many late nights discussing the capstone project, we believe we also gained valuable skills in teamwork. While we encountered multiple roadblocks, our persistence in trying different methods, watching instructional videos, and troubleshooting technical issues allowed us to gain a deeper understanding of reinforcement learning and AI training processes. This project was not just about developing an AI agent for Flappy Bird, it was an exercise in problem-solving, adaptation, and continuous learning.

**Video Link:**

**Flappy bird using pycharm**

* [**https://www.youtube.com/watch?v=grbsRgVm33E**](https://www.youtube.com/watch?v=grbsRgVm33E)

**Flappy bird version 2**

* [**https://www.youtube.com/watch?v=-Qq1FPioDvo**](https://www.youtube.com/watch?v=-Qq1FPioDvo)

**Github Links**

[**https://github.com/MattChoo0/CapstoneProject-FlappyBird/tree/main**](https://github.com/MattChoo0/CapstoneProject-FlappyBird/tree/main)

**Bibliography**

“Pygame - Flappy Bird.” YouTube, YouTube,

www.youtube.com/playlist?list=PLjcN1EyupaQkz5Olxzwvo1OzDNaNLGWoJ.

Accessed 30 Mar. 2025.

“Python Flappy Bird AI Tutorial (with NEAT) - Creating the Bird.” YouTube, YouTube,

[www.youtube.com/watch?v=MMxFDaIOHsE&list=PLzMcBGfZo4-lwGZWXz5Qgta\_Y](http://www.youtube.com/watch?v=MMxFDaIOHsE&list=PLzMcBGfZo4-lwGZWXz5Qgta_YNX3)

[NX3](http://www.youtube.com/watch?v=MMxFDaIOHsE&list=PLzMcBGfZo4-lwGZWXz5Qgta_YNX3)\_vLS2. Accessed 30 Mar. 2025.

russs123. “Russs123/Flappy\_bird.” GitHub, github.com/russs123/flappy\_bird. Accessed 30 Mar.

2025.

Techwithtim. “Techwithtim/Neat-Flappy-Bird: An Ai That Plays Flappy Bird! Using the NEAT

Python Module.” GitHub, github.com/techwithtim/NEAT-Flappy-Bird. Accessed 30 Mar.

2025.

Zhu, Danny. “How I Built an AI to Play Flappy Bird.” Medium, Analytics Vidhya, 26 May 2020,

medium.com/analytics-vidhya/how-i-built-an-ai-to-play-flappy-bird-81b672b66521.