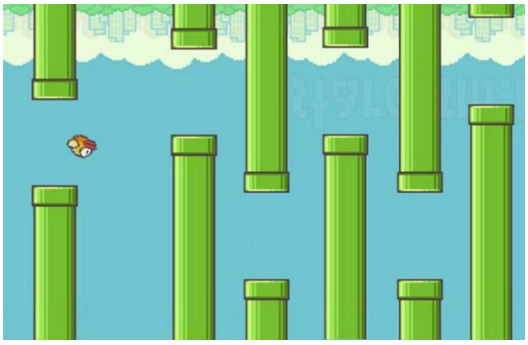
Jessica Navarro
ITAI 1378
CP



Flappy Bird

What is Flappy Bird?

Flappy Bird is a mobile game developed in 2013, which involved a bird controlled by the player. The objective of the game is to navigate the bird through the gaps of columns of pipes for as long as possible, without hitting them or hitting the ground. The game includes:

- A bird: The bird's position and velocity are key aspects that define the state of the game. It starts from a specific height and falls due to gravity unless it is flapped.
- Pipes: The pipes come in pairs with an equally sized gap in between them, placed at random heights. The pipes are also placed at different distances from each other. These pipes move from right to left at a constant speed. If the bird passes through the gap between pipes, it scores a point.
- Gravity and flap mechanism: Gravity constantly pulls the bird down unless the player flaps, which temporarily lifts the bird. This action needs to be timed well to avoid collisions.
- Score: If the bird passes through the gap between pipes, it scores a point. If the player reached 10 or more points, they were awarded a bronze medal which was displayed during the game over screen. If they reached 30 points, they were awarded a gold medal. If they reached 40 points, they were awarded a platinum medal.
- Game over: The game ends when the bird collides with the pipes or the ground.

PyGame is a beginner friendly Python library, used for creating multimedia applications, such as graphics and sounds, and allows user input through a mouse or keyboard. The reason for choosing PyGame is its ease of use for creating 2D games and its flexibility to fine-tune the mechanics of the game environment.

Setting up Flappy Bird for AI interaction

State Representation is the state that represents the essential information the agent needs to decide its next action in the game. Between giving the screen raw pixel values or a preprocessed state, I chose a preprocessed state. I like to have a bit of control of things; therefore I'd like to provide the height of the gaps between the pipes,

the distance between the pipes from each other, the velocity of the bird, and the position of the bird. As well as the height and width of the bird, and how the measurements of the bird compare to the gap of the gaps. This is significant information because it gives a sense of knowing if the bird can easily pass through the gaps or will it be a tight squeeze. The original game sets the gap between the pipes at an equal distance. I will have 2 different heights for the gaps. Take a look at the photo I used above the title, the last few columns are placed closely next to each other, making it a challenge to successfully allow Faby, the bird, to pass through. When there is enough distance between the gaps, as in the photo with the first and second column, I will set the gap to be “1 inch.” I will have the model understand that if the columns are “0.5 inches” or less apart, the gap will be set to be “1 inch.” When the columns are closely adjacent, I will set the gap to be “1.5 inches.” I will have the model understand that if the columns are more than “0.5 inches” apart, the gap will be set to be “1.5 inches,” allowing for a more successful outcome.

The action space is discrete, as the bird only has two actions it can take: (1) flap its wings or (0) not flap its wings and fall. The reinforcement learning agent will decide which of these actions to take based on the preprocessed state I provide. For instance, if the gap is large and the bird is positioned well, the agent might decide not to flap (0). But if the gap is small and the bird needs to gain altitude quickly to avoid hitting the pipes, the agent might choose to flap (1). Over time, it will improve its decisions through trial and error (reinforcement learning) to maximize its score (surviving longer or passing through more pipes).

The agent’s goal is to maximize its reward. I increased the point system and allowed for additional rewards because I believe making the gap taller gives the agent a better chance of passing through the gaps, staying alive longer and being rewarded with more points.

- +3 points for every 15 seconds they manage to stay alive
- +2 points for passing through a pipe gap
- +0.5 points for passing through 10 pipes
- -2 for colliding with a pipe
- Game over if the bird hits the ground

Preprocessing Game Frames for AI Input: Making the game more manageable for the neural network model. It reduces complexity, speeds up training and improves performance and efficiency.

- Resizing: Resize each game frame to a fixed size (such as 84x84 or 128x128 pixels). This helps reduce the dimensionality of the input, making it less computationally expensive while retaining essential features.
- Grayscale Conversion: Convert the game frames to grayscale. This reduces the complexity of the input by eliminating color information, focusing only on intensity (brightness). This simplification works well for Flappy Bird, as the core task is navigating through obstacles and reacting to the bird’s position, not color.
- Stacking Frames: Instead of using a single frame, stack a few frames together (e.g., 4 consecutive frames). For example, instead of feeding a single frame, you

can provide a sequence of the last 4 frames. This way, the AI can learn to predict movement (like when the bird is falling or rising).

- Region of Interest (ROI) Cropping: In games like Flappy Bird, you don't need the whole screen; you just need the area where the action is happening (e.g., the bird and the pipes). Cropping the image to focus on just this region will reduce unnecessary information, making the task easier for the AI.
- Edge Detection or Feature Extraction: Instead of feeding the raw pixels, you can use techniques like edge detection (e.g., using algorithms like Canny edge detection) to highlight the outlines of important objects like the pipes and the bird. This makes it easier for the AI to identify obstacles and navigate through the gaps.

Pre-trained Model Usage

Transfer learning involves reusing a pre-trained model, often trained on a large dataset for one task, and adapting it to solve a different but related problem. The main advantage is that the model already has learned useful features (e.g., edges, textures, shapes) from the original task, which can be applied to the new task, reducing the need for extensive training. Rather than building a model from the ground up, transfer learning enables the agent to utilize a pre-trained model to extract valuable features from the visual input of the game. For Flappy Bird, a pre-trained model like MobileNetV2 or ResNet could help the agent identify objects such as pipes and the bird itself, even without extensive training on the specific game.

MobileNetV2 is a lightweight model with fewer restrictions, making it suitable for real-time applications where computational efficiency is critical. The model is designed for mobile and embedded devices, and uses depthwise separable convolutions, which reduces the computation cost. Since Flappy Bird is a relatively simple game with lower resolution graphics, MobileNetV2 can efficiently extract features like edges, pipes, and bird position. To use MobileNetV2 for feature extraction, you would remove the top classification layers of the network (those used for ImageNet classification) and extract features from earlier layers, such as convolutional layers. The output from these layers can then be used as input for the reinforcement learning algorithms.

Adjusting pre-trained models for tasks like Flappy Bird comes with challenges. One key challenge is the domain gap, as pre-trained models are typically optimized for datasets like ImageNet, which consist of natural images rather than the specific visuals of Flappy Bird. To address this, fine-tuning the pre-trained model on a smaller set of Flappy Bird frames can help tailor it to the game’s requirements. Another challenge includes feature selection. Since Flappy Bird is a active game with moving objects, some features from the pre-trained model may not directly map to game-specific features. You may need to experiment with different layers of the model for feature extraction to find the most useful ones.

Reinforcement Learning Implementation

Jessica Navarro
ITAI 1378
CP

In reinforcement learning, an agent learns by interacting with an environment and receiving feedback in the form of rewards. The objective is to maximize the rewards by taking actions that lead to positive outcomes.

- States: A state is a representation of the environment at a particular time step. In Flappy Bird, this could include the bird's position, velocity, the positions of pipes, the gap between pipes, and the distance between pipes.
- Actions: The agent can perform one of two actions: flap (1) or do nothing (0).
- Reward System: The agent's goal is to maximize its reward.
    - +3 points for every 15 seconds they manage to stay alive
    - +2 points for passing through a pipe gap
    - +0.5 points for passing through 10 pipes
    - -2 for colliding with a pipe
    - Game over if the bird hits the ground
- Policy: A policy defines the agent's strategy for selecting actions based on states. The policy is learned through trial and error, guided by the rewards.

The Deep Q-Network (DQN) algorithm is a reinforcement learning algorithm that uses deep learning to approximate the Q-values (expected future rewards) for each action in each state.

- Q-Network: The Q-network is a neural network used in reinforcement learning that estimates the Q-value function, which helps an agent decide the best action to take in a given state.
- Replay Memory: Replay memory stores past experiences, which include state, action, reward, and next state. This is used for training the Q-network. Randomly sampling experiences from memory helps break correlations between consecutive states and ensures stable learning.
- Target Network: The target network is a copy of the Q-network, which is periodically updated. This prevents the Q-values from fluctuating wildly and stabilizes the learning process.

In the early stages of training, the agent explores the environment (taking random actions) to learn about the world. Over time, it shifts to exploitation, where it uses the learned Q-values to choose the best action. The exploration rate (epsilon) decays over time, balancing exploration and exploitation.

When an agent interacts with an environment, it generates experiences. Instead of immediately learning from each experience in the order it occurs, the agent stores these experiences in a replay buffer, a kind of memory storage. By storing experiences and sampling from them randomly, the agent can avoid overfitting the most recent experiences. This also helps the agent learn better by exposing it to diverse experiences.

Jessica Navarro
ITAI 1378
CP

Model Training

Training Process

1. Initialize Environment: Set up the game using PyGame and the OpenAI Gym interface.
2. Initialize Q-Network: Set up the neural network that approximates the Q-value function.
3. Interact with Environment: The agent interacts with the environment, selecting actions based on the epsilon-greedy policy.
4. Store Experience: The agent stores its experiences in replay memory.
5. Train the Q-Network: Sample batches from the replay memory and update the Q-Network using the Bellman equation.
6. Update Target Network: Periodically update the target network to improve stability.

Hyperparameters

- Learning Rate: Controls the size of updates to the Q-network.
- Discount Factor (γ): Determines how much future rewards are valued.
- Epsilon Decay: Controls how quickly the agent shifts from exploration to exploitation.

Training Challenges

- The agent might forget what it has learned as it trains. Using experience replay helps alleviate this problem by ensuring that past experiences are not forgotten.
- Rewards are sparse in Flappy Bird, meaning the agent might go through many time steps without receiving a significant reward. To address this, you could use techniques like reward shaping, where intermediate rewards are given for certain behaviors.

Performance Evaluation

- **Average Score**: Track how well the agent performs over multiple games.
- **Survival Time**: Track how long the agent survives before colliding with a pipe or the ground.
- Analyze gap height: Determine if the variation of gap sizes play a factor in the rewards earned and if they contribute to Faby's increased survival duration

Testing and Evaluation

Testing Strategy: Run the trained agent in various scenarios to evaluate its generalization to different levels of difficulty (e.g., longer survival time, faster pipes, etc.).

- A scenario will only have pipes set to 0.5 inches or less, apart from each other. This will set the gap size to be 1 inch.

- A scenario will only have pipes set to more than 0.5 inches, apart from each other. This will set the gap size to be 1.5 inches.
- A scenario will have half of the pipes set to 0.5 inches or less and the other set to more than 0.5 inches. They will be set randomly, but will ensure there are equal pipes set at both distances.

Evaluation Metrics

- Average Score: Evaluate how consistently the agent achieves a high score over several games.
- Survival Time: Track how long the agent survives before making a mistake.

Visualization: Visualize the agent's decisions during gameplay to understand its decision-making process. This could involve visualizing Q-values for each action at each time step or showing the agent's chosen action at each frame.

Improvements and Future Work

- Model Optimization: Experiment with different architectures (e.g., convolutional layers) or other deep RL algorithms like Double DQN or DDPG.
- Curriculum Learning: Gradually introduce more challenging pipe configurations to help the agent learn progressively harder tasks. Such as making the gaps smaller and/or larger when the pipes are adjacent to each other, making the pipes larger and/or when there is more distance between them. Also creating more distance between the pipes to create anticipation as to when a pipe will appear.
- Multi-agent Training: Experiment with training multiple agents to see how cooperation or competition influences performance.