Jacob Ehling

Prof McManus

ITAI 1378

Capstone Flappy Bird AI Report

**Environment and Setup**

**Game and Environment**

Flappy Bird is a simple 2D pixel art game, featuring a harmless flying bird. This bird must navigate pipes of various heights, to see how far the bird can fly in this game. The game itself uses pixel art and scrolls horizontally, with the pipe obstacles aligning vertically. The background scrolls continuously to mimic motion, as the bird progresses through the level. The key here is the bird is affected by gravity, so you must tap the screen to maintain flight, or to “flap”. The game’s physics here are pretty straightforward, gravity pulls down, while taps flap the bird upwards and continue the game. Every time the bird passes one of the pipe obstacles, the user gains one point. This keeps the scoring system pretty simple, one point per obstacle passed.

**Libraries and Tools for Environment**

PyGame will be used to create the game environment, and render graphics. PyGame is a great choice here due to the simplicity of the game and its graphics. PyGame has many tools we can use as well, for example collision, event handling, and more. We will also be using the OpenAI Gym. The OpenAI Gym has equipment for action and observation spaces, this makes it easy for the reinforcement learning integration. The Gym also supports existing reinforcement libraries, in order to maximise compatibility.

**AI Interaction and Setup**

The state holds the bird’s x coordinate, y coordinate, pipe position, and current velocity. This info is needed for the AI to make accurate decisions, with this info, we can convey the game’s current status. The actions the AI can take are binary, flap or nothing. The fact this choice is so limited means the AI can lean faster, as the controls are simpler to understand. The reward system works almost in conjunction with the scoring system. Rewards are +1 on pipe pass, and -1 on collision. This simple reward system, paired with the game’s overall simplicity, will help reduce errors and make our AI more efficient.

**Preprocessing Game Frames**

The game frames will be reduced for consistency, to 84x84. This smaller size will mean the neural network processes inputs faster and without losing too many details. Frames are also converted to greyscale for this exact reason, removing color information while keeping crucial data in image. After these steps, pixel values are normalized from 0 to 1. This is done to prevent any instabilities while training.

**Pre-trained Model Use**

**Transfer Learning**

Transfer Learning, as seen in class, reduces training time by reusing learned features from large datasets. This allows our model to focus on more task-specific learning, since general features are already taught.

**MobileNetV2**

MobileNetV2 is the model selected for our use case, due to its lightweight design and many feature extraction abilities. Since MobileNetV2 can run quickly on its small design, we can use it for real time use. Since our game is simple as well, MobileNetV2 would not struggle with our game images. This means our model is not only fast, accurate, and useful, but it comes packed with features.

**Modifying the Model**

We will remove the final layer and replace it with our own custom layer. This new final layer is fully connected, helping to predict actions in game. This change allows the model to output probabilities based on our game environment and action space.

**Challenges and Solutions**

Immediately, one challenge that arises is Flappy Bird’s unique visual data. Flappy Bird has pixel-based graphics, meaning the style needs to be understood or interpreted by our model. The datasets for our model are not in pixel art style, meaning inputs must match expected format. This was accounted for in preprocessing, where game frames are made uniform and greyscale for the model. When we resize and normalize our inputs, the model will have less difficulty with the pixel style, especially given it’s simplified data and reduced scale.

**Reinforcement Learning**

**Overview**

Reinforcement learning utilizes states, actions, and rewards to train an AI agent on a specific “learned” skill. This is saved under a policy, and ours will be applied to our AI agent to excel at Flappy Bird

**Algorithm Selection**

For the actual implementation, we will use the Deep Q-Network algorithm. According to research, the DQN combines Q-learning with deep neural networks. The algorithm has a remarkable ability to learn from visual inputs, which makes it extremely useful for us. The DQN is also best because it can handle both the continuous state space, and the action space.

**Necessary Components**

Implementing our DQN requires a few key factors. The Q-network architecture is made up of many convolutional layers that process visual input, and then fully connected layers for the action prediction. According to AI on the topic, we will implement a replay memory buffer of 100,000 experiences to store state transitions, to ensure the AI understands separate game states. This network, which gets updated every 1000 steps, helps train our model best.

**Exploration-Exploitation Balance**

In order to balance exploration and exploitation, we must implement an “epsilon-greedy strategy with a decaying epsilon value,” according to PapersWithCode. This means the model will alternate between exploiting learned behaviors, and exploring new ones. We need this for learning, and we will setup our code this way. The epsilon value will begin at 1.0, meaning all actions are completely random. After a while, this value slowly decreases, so we can see the model’s learned strategies. The delay happens slowly over our 100,000 experiences, so that the agent can discover and remix its strategies to learn best.

**Model Training**

**Proposed Training Process**

The training process would begin with initialization of networks and our replay buffer. Each run through would then start a fresh game, and let the agent make its decisions according to our action policy. Every 4 steps, the experiences are collected in the buffer and update our game. This balance and frequency should help the model maintain speed and quality well.

**Hyperparameters**

Based on research papers and AI consultation, I settled on a learning rate of .00025. This value should provide stable updates. We also find the discount factor, which was recommended at .99. The value of .99 should ensure the agent both focuses on immediate survival, and the long term goal of game completion.

**Common Training Issues**

To address issues such as catastrophic forgetting, we could implement a dual memory system. This would ensure old and new strategies are retained. We could also give more weight to pipe passes, making sure the agent came to grips with its task faster.

**Evaluation**

We would track and evaluate a few key aspects while running our code. We would track the average scores over 100 iterations, the success rate of passing the obstacles, average survival time, and action distribution. This info would help us really “get in the mind of” our model, to see where and when things go awry. This process should ensure a coherent agent, one that can flap until its heart’s content.

**Working with Results**

Based on our findings from above, we would extrapolate out the data and compare it to other sources. For example, we would compare our agent to the average human’s performance of flappy bird. Additionally, we would compare our agent against simpler agent types, and more “random” policy variations, until the best Flappy Bird agent is achieved. This data can be visualized in heat maps, trajectory trackers, or decision tree maps. All of these will help ensure we are trained best, to train our model best, and ensure we all learn skills together,

Works Cited

Patel, Yash. "PyGame: A Primer on Game Programming in Python." <https://realpython.com/pygame-a-primer/>

Wilson, Ashia C., et al. "The Marginal Value of Adaptive Gradient Methods in Machine Learning." Advances in Neural Information Processing Systems, 2017.

Howard, Andrew G., et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." arXiv preprint arXiv:1704.04861, 2017

Silver, David, et al. "Deep Reinforcement Learning: An Overview." arXiv preprint arXiv:1701.07274, 2017.

Zhang, Shangtong, and Richard S. Sutton. "A Deeper Look at Experience Replay." arXiv preprint arXiv:1712.01275, 2017.

Masters, Dominic, and Carlo Luschi. "Revisiting Small Batch Training for Deep Neural Networks." arXiv preprint arXiv:1804.07612, 2018.

Sutton, Richard S., and Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press, 2018.