Hayden Wood

Andre Ellis

## Path 1: Conceptual Path

For students choosing this path, provide a detailed explanation for each of the following steps:

### Environment Setup

* Flappy Bird’s simple yet engaging environment features a colorful sky, green pipes and a cartoonish bird. Scoring is straightforward, one point per pipe successfully passed.
* For development, PyGame is ideal for its 2D capabilities and ease of use, while OpenAI Gym would facilitate AI integration.
* In setting up the game for AI interaction, the representation of the game state can be simplified by focusing on the bird's position and velocity. The proximity of the pipes, and the gaps between them. Allowing AI to make the decisions such as “flap” or “do nothing” based on its training due to discrete actions. Rewards system could be designed for successfully navigating through the pipes and negative reward for collisions.
* To preprocess game frames for AI input, one effective method would be to resize the frames to a consistent input dimension suitable for neural networks. Additionally, converting frames to grayscale could help reduce computational complexity while preserving essential structural information.

### Pre-trained Model Usage

**Transfer Learning**

Transfer learning is taking a pre-trained model (like one trained on ImageNet) and using it for a new task. It's helpful because these models already know how to recognize shapes, edges, etc, so we wouldn't have to train the AI from scratch. This saves a ton of time and computing power.

**Model Choice**

We'd go with MobileNetV2 or ResNet because they're lightweight and good at feature extraction. MobileNetV2, in particular, works well for simpler problems where we don't want a super-heavy model.

**Modifications**

I'd cut off the last layers of the pre-trained model (the ones that classify objects) and add a few new layers tailored to Flappy Bird. These new layers would feed into a Q-network that helps the AI make decisions.

**Challenges**

One problem might be that the pre-trained model is designed for real-world images rather than game frames. To handle this, I'd fine-tune the model on Flappy Bird data (like a bunch of screenshots). I could also preprocess the game frames in ways that make them easier for the model to understand, like edge detection.

### Reinforcement Learning Implementation

* Reinforcement Learning trains agents to make optimal decisions via interaction with an environment. The key concepts are states, actions, rewards, policies.
* As for the algorithm for Flappy bird, DQN components that uses a neural network to approximate the Q-values. And to implement this by initializing the Q-network with random weights and training it.
* Components that would be needed would be Q-network (neural network) a neural network with input layers representing the state. Replay memory is a buffer that stores a collection of past experiences. Target network is a separate neural network that stabilizes training by providing consistent Qivalue targets and is also updated less frequently than the main Q-network.
* We would use a strategy of “epsilon-greedy” where the agent explores new actions with a probability of epsilon and exploits known information.
* Experience replay would store past experiences for training and improve stability and learning efficiency.

### Model Training

**Training Process**

1. Start the game and initialize the AI.
2. Let the AI interact with the environment (e.g., flap or not).
3. Record experiences (state, action, reward, next state).
4. Store these in replay memory.
5. Train the Q-network using random samples from memory.
6. Periodically update the target network.

**Training Loop**

For every episode:

* Reset the game.
* Loop until the bird crashes:
  + Use the current Q-network to decide an action (flap or not).
  + Observe the new state and reward.
  + Save this experience.
  + Sample a batch from memory to train the network.

**Hyperparameters**

Some important ones:

* Learning rate (e.g., 0.001) controls how fast the network learns.
* Discount factor (e.g., 0.99) decides how much future rewards matter.
* Batch size (e.g., 32 or 64) affects how many experiences the network learns from at once.

**Common Issues**

* Reward Sparsity: If the AI rarely gets rewards, I’d add small rewards for survival.
* Catastrophic Forgetting: Use experience replay and periodically update the target network.
* Overfitting: Monitor training and testing performance to catch this early.

**Evaluating During Training**

I’d plot the average score over episodes and look for a steady upward trend. If scores stop improving, I’d check for bugs or tweak the hyperparameters.

### Testing and Evaluation

* Describe a comprehensive testing strategy for the trained agent.
* Evaluation metrics used are average score, the average number of points accomplished over multiple game runs. Higher scores indicate better performance. Survival time would be the average duration the agent survives in each game. Longer survival times show improved ability to avoid obstacles. Success rate would be based on the percentage of games completed without crashing. This metric offers an insight into the consistency and reliability of the agent's decision making skills.
* This would be done by establishing a baseline by comparing the agents performance against a naive agent or human player. Then this would be assessed whether the agents metrics surpass the baseline significantly indicating successful learning and adaptation. Lastly analyzing the performance variance to identify inconsistencies in the agents behavior. High variance suggests the agents strategy lacks robustness.
* This would be done by creating graphs plotting average score and survival time over training episodes to showcase the learning progress.
* As for the potential improvements if the agent performs poorly adjusting the hyperparameters (learning rate, discount factor, exploration rate). Investigating the agents behavior in specific scenarios like does it struggle with certain obstacles. Explore incorporating more sophisticated state representation to provide agent with richer contextual information.