Jesus Chavez

Capstone Project: Train an AI Agent to Play Flappy Bird

Conceptual Path:

Flappy Bird is a basic 2D side-scrolling game in which a bird must navigate pipes without colliding with them. The game's physics and environment include the following elements:The game's graphics are pixel art, with the bird, pipes, and background. The bird is operated by tapping or pressing a button to flap, and gravity drags it downward. Gravity influences the bird's movements, forcing it to fall until it flaps its wings. When the player taps, the bird accelerates upward. The pipes travel from right to left across the screen, and the player must navigate between them. The height difference between pipes fluctuates randomly. The player wins one point for each pair of pipes that the bird successfully passes through without colliding with. The game ends when the bird hits a pipe or the ground.

PyGame is an excellent choice for this project since it can efficiently manage the graphical rendering and simulation of the Flappy Bird environment. It is lightweight and extensively used for creating small games, with utilities for managing graphics, collisions, and events. While OpenAI Gym offers a standardized interface for AI training, which facilitates the implementation of reinforcement learning algorithms, PyGame is helpful for game simulation. You can effectively set up the environment and make sure the agent interacts with it by utilizing both tools.

The vertical position of the bird, its velocity, and the location of the closest pipes would probably make up the agent's state. The AI can be programmed with this data. Binary could be used for the action space. Action 0: Take no action; gravity causes the bird to fall. Action 1: The bird flaps, moving upward. System of Rewards: A positive reward (+1) is given for each pipe that is successfully passed. A negative reward (-1) is given for hitting the ground or a pipe. For all other states, such as continuing to fly between pipes, there is no reward (0).

To lower processing demands and make it easier for AI models to handle, the game frames should be resized to a smaller size. By eliminating the color channels, converting the frames to grayscale streamlines the input and lowers complexity. Lightness and contrast allow for the separation of the background, pipes, and bird. Enhancing training stability, normalizing pixel values (scaling between 0 and 1) guarantees consistent input to the neural network.

The process of fine-tuning or adapting a model that was trained on one task to another related task is known as transfer learning. When training data is scarce or you wish to speed up the training process, this enables you to use pre-trained features from a large dataset. Using a pre-trained model such as MobileNetV2 or ResNet, transfer learning can be helpful in extracting high-level features (such as shapes and patterns) from game frames. Because Flappy Bird's visual input is comparable to image classification tasks, this can drastically cut down on the amount of time required to learn effective feature representations.

A portable deep learning model made for mobile devices is called MobileNetV2. It is effective and offers a reasonable balance between accuracy and computational expense. Because the Flappy Bird game frames are small, MobileNetV2 can extract pertinent features without incurring high computational costs. By keeping the convolutional layers and eliminating the fully connected layers (also known as the output layers), the pre-trained MobileNetV2 model can be utilized for feature extraction. After processing the input frames, these layers will produce a feature map that the AI agent can use to guide its decision-making.

Because the model was trained on much larger and more complex datasets (such as ImageNet), it might not be directly applicable to the 2D game environment of Flappy Bird. The agent could better adapt the learned features to the task of identifying obstacles and the position of the bird by fine-tuning the final few layers of the pre-trained model for the particular task of Flappy Bird.

Fundamental Ideas in Reinforcement Learning (RL) States: Indicate the environment in which the agent is currently interacting (e.g., the position of the pipes, the velocity of the bird, etc.). Actions: The agent's available options, such as flapping or doing nothing. Incentives: The agent is rewarded for its actions (e.g., -1 for colliding with pipes, +1 for passing them). Policy: A method the agent uses to determine what to do in light of the situation at hand.

Q-learning: The agent can learn the benefits of performing particular actions in particular states thanks to this model-free algorithm. The expected reward for an agent beginning in a particular state and performing a specific action is represented by the Q-value. The Q-learning algorithm iteratively updates Q-values using a Q-table or neural network.

The Q-network, which approximates the Q-value function, can be a deep neural network. The game frame that has been processed (features taken from the previously trained model) is the input layer. Hidden Layers: Layers that record the game's temporal and spatial elements. Output Layer: The anticipated Q-values for every action (do nothing or flap).

Exploration: The agent acts haphazardly to investigate the surroundings and find potentially more effective tactics. Exploitation: By selecting actions that have produced high rewards in the past, the agent maximizes rewards by applying its learned policy.

Why It Matters: Training stability is increased and the correlation between successive samples is broken by storing and reusing prior experiences. Implementation: Store states, actions, and rewards in a replay buffer. Sample this buffer at random to train the Q-network.

Procedure for Training Set up the Q-network: Set up the neural network using arbitrary weights. Collect Experience: Using the current Q-values as a guide, the agent interacts with the surroundings. Revise the Q-values: After every step, update the Q-values using the Bellman equation. Repeat: In order to enhance its policy, the agent keeps interacting with the environment for numerous episodes.

Learning Rate: Regulates the model's rate of update. The usual range is between 0.001 and 0.0001. The Gamma discount factor establishes the relative value of future rewards in relation to current ones. The ratio of exploration to exploitation is regulated by the exploration rate (Epsilon).

Problems with training are Catastrophic Forgetting: When the model keeps learning new things, it forgets previously learned behaviors. Experience replay can help lessen this. Reward Sparsity: The agent might only get feedback after a collision or after passing pipes, for example. Offering intermediate rewards based on progress (e.g., distance traveled) is one way to solve the problem.

Method of testing performance metrics include number of pipes passed, survival time, and average score per episode. Assess Learning Progress: Keep track of the agent's performance as it gets better over time. Plotting the agent's performance and using visualizations (such as action heatmaps) can help you understand how the agent makes decisions.

Interpreting the findings of the Benchmark: Evaluate the agent's performance against a predetermined, straightforward policy (e.g., always flap or never flap) or random actions. Enhancements and Upcoming Projects: To increase generalization, further refine the model. Try out different reinforcement learning algorithms, such as Proximal Policy Optimization (PPO) or Deep Q-Networks (DQN).