Capstone Project: Train an AI Agent to Play Flappy Bird

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­­­­­­­

**Capstone Project: Train an AI Agent to Play Flappy Bird** is a comprehensive endeavor designed to serve as a culminating academic and intellectual experience, providing hands-on engagement with AI, game development, computer vision, and reinforcement learning. Though complex, it is a highly rewarding project. The objective is to understand and/or implement the process of training an AI agent to play the Flappy Bird game.

### **1. Environment Setup**

**Flappy Bird Game Environment:**

* **Graphics:** Flappy Bird is a straightforward 2D side-scrolling game featuring pixel art graphics. The player controls a bird, guiding it through gaps between columns of green pipes without colliding with them. The game’s simplicity and retro aesthetic contribute to its charm and make it an excellent candidate for AI training.
* **Physics:** The bird is subject to gravity, causing it to fall unless the player intervenes. By tapping the screen (or pressing a button), the player makes the bird flap its wings, causing it to ascend briefly. The pipes move from right to left at a constant speed, creating a continuous challenge for the player to navigate through the gaps.
* **Scoring System:** The game's score increases by one point for each set of pipes the bird successfully passes through. The game ends when the bird hits a pipe or the ground, prompting the player to try again to beat their previous score.

**Libraries and Tools:**

* **PyGame:** PyGame is a widely-used library for creating 2D games in Python. It is well-suited for recreating the Flappy Bird environment due to its capabilities in handling graphics, user inputs, events, and physics. PyGame provides a robust framework for building the game environment, making it easier to focus on developing and testing the AI agent.
* **OpenAI Gym:** OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It provides a standardized API for communicating with various environments, including custom ones like Flappy Bird. Using OpenAI Gym helps standardize the interaction between the AI agent and the game, facilitating smoother development and testing of reinforcement learning models.

**Setting Up the Game for AI Interaction:**

* **State Representation:** The state of the game can be represented by capturing the game screen (a frame) at each time step. This frame includes crucial information such as the bird's position, velocity, and the positions of the nearest pipes. By processing these frames, the AI agent can understand the current state of the game and make decisions accordingly.
* **Action Space:** The action space for the AI agent consists of two possible actions: flap (to make the bird ascend) or do nothing (to let the bird fall). This binary action space simplifies the decision-making process for the AI agent.
* **Reward System:** To train the AI agent, a reward system needs to be defined. A reward of +1 is given for each set of pipes successfully passed, encouraging the agent to navigate through as many pipes as possible. Conversely, a penalty (e.g., -100) is given for crashing into a pipe or the ground, discouraging such actions. This reward structure helps the agent learn to maximize its score by staying alive and passing through pipes.

**Preprocessing Game Frames:**

* **Resizing:** To reduce computational load and improve processing speed, the game frames can be resized to a smaller resolution. This step helps in managing the data efficiently without losing essential information.
* **Grayscale Conversion:** Converting frames to grayscale simplifies the input data by reducing the complexity. Since color information is not critical for the task of playing Flappy Bird, grayscale frames are sufficient for the AI agent to understand the game state.
* **Normalization:** Normalizing pixel values to a range [0, 1] improves the stability and performance of the training process. This step ensures that the input data is standardized, which helps in faster convergence and better learning for the AI agent.

By carefully setting up the environment and preprocessing the game frames, we create a solid foundation for training the AI agent to play Flappy Bird using computer vision and reinforcement learning techniques.

**2. Pre-trained Model Usage**

**Pre-trained Model Usage**

**Transfer Learning:**

* **Concept:** Transfer learning is a machine learning technique where a model developed for one task is reused as the starting point for a model on a second task. In this context, a pre-trained model, initially trained on a large dataset such as ImageNet, is fine-tuned for the specific task of playing Flappy Bird. The primary advantage of transfer learning is that it leverages the feature extraction capabilities learned by the pre-trained model, which are often generalized and robust, to reduce training time and improve performance on the new task.
* **Benefits:**
  + **Faster Convergence:** Since the pre-trained model has already learned low-level features such as edges, textures, and shapes, and even some high-level features, the training process on the new task can start from these already useful features rather than from scratch. This leads to quicker convergence.
  + **Requires Less Data:** Pre-trained models require significantly less data to achieve high performance on the new task because they already encapsulate useful feature representations. This is particularly beneficial when the new task has limited data available, like a small dataset of Flappy Bird frames.
  + **Powerful Feature Representations:** The feature representations learned from large and diverse datasets are typically robust and powerful, capturing essential characteristics of images that can be beneficial for the new task.

**Pre-trained Model Selection:**

* **Model Choice:**
  + **MobileNetV2:** MobileNetV2 is a popular choice for transfer learning due to its efficient architecture and strong performance on image classification tasks. It is designed to be lightweight, making it suitable for environments with limited computational resources, while still providing high accuracy.
  + **Other Potential Models:** Depending on specific needs, other models such as ResNet, VGG, or EfficientNet could also be considered. The choice depends on the trade-off between computational efficiency and accuracy requirements.
* **Feature Extraction:**
  + **Convolutional Layers:** The convolutional layers of the pre-trained model are typically retained to extract features from the input frames. These layers act as feature detectors, identifying various patterns and structures within the images.
  + **Using Extracted Features:** The extracted features from these convolutional layers can then be fed into the reinforcement learning algorithm. This process involves flattening the output of the convolutional layers and passing it through fully connected layers or directly into the reinforcement learning model, depending on the architecture.

**Challenges and Solutions:**

* **Challenge:**
  + The pre-trained model might not capture game-specific features effectively because it was originally trained on a general-purpose dataset. This discrepancy can lead to suboptimal performance when applied directly to the Flappy Bird task.
* **Solution:**
  + **Fine-Tuning:** Fine-tuning involves training the pre-trained model further on a small dataset of Flappy Bird frames to adapt its feature extraction capabilities to the specific characteristics of the game. This process allows the model to adjust its parameters to better recognize and interpret game-specific features, such as the bird's movement, pipe positions, and the background environment.
  + **Steps for Fine-Tuning:**
    1. **Dataset Preparation:** Collect a small dataset of Flappy Bird frames, ensuring it covers various game scenarios (e.g., different pipe positions, bird positions, and velocities).
    2. **Freezing Layers:** Initially, freeze the lower layers of the pre-trained model to retain the generic feature extraction capabilities and only train the higher layers.
    3. **Training:** Fine-tune the higher layers (or the entire model if necessary) on the Flappy Bird dataset. Use a smaller learning rate to make gradual adjustments to the weights, preventing the model from forgetting the previously learned features.
    4. **Validation:** Continuously validate the model’s performance on a separate validation set to ensure it generalizes well to new, unseen game scenarios.

By incorporating transfer learning and fine-tuning a pre-trained model, we can leverage the strengths of existing models to efficiently and effectively train an AI agent for the specific task of playing Flappy Bird. This approach not only saves time and computational resources but also enhances the model's performance through the use of robust, pre-learned feature representations.

**3. Reinforcement Learning Implementation**

**Key Concepts:**

* **States:**
  + In the context of Flappy Bird, states are the representations of the game environment at any given time. This can be the raw game frames or processed versions of these frames. The state includes vital information such as the bird’s position, velocity, and the relative positions of the pipes.
  + **Example:** A state could be a grayscale, resized frame from the game, capturing the bird’s location and the gap between pipes.
* **Actions:**
  + Actions are the possible moves the AI agent can take. In Flappy Bird, the action space is discrete and consists of two actions: flap (which causes the bird to ascend) or do nothing (which lets the bird descend).
  + **Example:** If the bird is about to hit a pipe, the action would be to flap; otherwise, the action might be to do nothing and let gravity pull the bird down.
* **Rewards:**
  + Rewards are feedback signals from the environment that guide the learning process. Positive rewards encourage the agent to repeat certain actions, while negative rewards discourage them.
  + **Example:** A reward of +1 is given for each set of pipes successfully passed, and a penalty (e.g., -100) is given for crashing into a pipe or the ground.
* **Policies:**
  + A policy is the strategy used by the agent to decide which action to take based on the current state. It maps states to actions.
  + **Example:** A policy might dictate that if the bird is close to the top of the screen, it should do nothing, whereas if it’s near the bottom, it should flap to avoid crashing.

**Algorithm Choice:**

* **Algorithm:** Deep Q-Network (DQN)
  + DQN is a reinforcement learning algorithm that combines Q-learning with deep neural networks to handle high-dimensional input spaces like images. It approximates the Q-value function, which estimates the expected return (reward) of taking a particular action in a given state and following the optimal policy thereafter.
* **Components:**
  + **Q-network Architecture:**
    - A neural network that approximates the Q-value function. It takes the current state (game frame) as input and outputs Q-values for each possible action. The network learns to predict the Q-value for each action, guiding the agent to choose actions that maximize cumulative rewards.
    - **Example Architecture:** A convolutional neural network (CNN) with several convolutional layers followed by fully connected layers. The output layer has two nodes (one for each action in Flappy Bird).
  + **Replay Memory:**
    - A buffer to store experiences (state, action, reward, next state). During training, random samples from this buffer are used to update the Q-network. This process helps to stabilize training by breaking the correlation between consecutive experiences.
    - **Example:** After each action, the tuple (state, action, reward, next state) is stored in the replay memory. During training, a mini-batch of these tuples is sampled to train the Q-network.
  + **Target Network:**
    - A separate network that is a copy of the Q-network but updated less frequently. It provides stable Q-value targets during training, which reduces the risk of divergence in Q-learning.
    - **Example:** The target network is updated every fixed number of steps with the weights of the Q-network, ensuring the targets used in the Q-learning update remain stable.

**Exploration-Exploitation Trade-off:**

* **Handling:**
  + The trade-off between exploring new actions and exploiting known ones is managed using an epsilon-greedy policy. The agent explores (takes random actions) with probability ε and exploits (chooses the best-known action) with probability 1-ε. Initially, ε is high to encourage exploration and gradually decreases over time to favor exploitation.
  + **Example:** Start with ε = 1 (pure exploration). Over time, decay ε to a lower value (e.g., 0.1) to shift towards exploitation.

**Experience Replay:**

* **Implementation:**
  + Store experiences in a replay buffer. When training the Q-network, sample mini-batches of experiences from this buffer instead of using consecutive experiences. This method improves training stability and efficiency by diversifying the training data and breaking the correlation between consecutive experiences.
  + **Example:** After each step, add the experience tuple (state, action, reward, next state) to the replay buffer. During training, randomly sample a mini-batch of experiences to update the Q-network.
* **Importance:**
  + Experience replay helps to break the temporal correlation between consecutive experiences, which can lead to more stable and efficient training. By using a random sample of past experiences, the learning algorithm can generalize better and converge more reliably.
  + **Example:** Sampling a diverse set of experiences (different states, actions, and rewards) from the replay buffer helps the Q-network learn more robust and generalizable features, leading to better performance.

**Detailed Step-by-Step Explanation**

1. **State Representation:**
   * Capture the game frame at each time step.
   * Process the frame (resize, convert to grayscale, normalize).
   * Input the processed frame to the Q-network to determine the current state.
2. **Action Selection:**
   * Use the epsilon-greedy policy to select an action.
   * With probability ε, select a random action (exploration).
   * With probability 1-ε, select the action with the highest Q-value from the Q-network (exploitation).
3. **Perform Action:**
   * Execute the selected action in the game environment.
   * Observe the reward and the next state (next game frame).
4. **Store Experience:**
   * Store the experience tuple (state, action, reward, next state) in the replay buffer.
5. **Training:**
   * Sample a mini-batch of experiences from the replay buffer.
   * For each experience in the mini-batch:
     + Calculate the target Q-value using the target network.
     + Compute the loss between the predicted Q-value and the target Q-value.
   * Update the Q-network by minimizing the loss.
6. **Update Target Network:**
   * Periodically copy the weights from the Q-network to the target network to provide stable Q-value targets.
7. **Repeat:**
   * Continue the process of action selection, performing actions, storing experiences, and training until the agent achieves satisfactory performance.

By following these steps and utilizing the components and techniques discussed, you can effectively implement a reinforcement learning agent to play Flappy Bird using Deep Q-Networks (DQN).

**4. Model Training**

**Training Process**

**Step-by-Step:**

1. **Initialize the Q-network and target network:**
   * **Q-network:** Initialize the neural network with random weights. This network will be trained to approximate the Q-value function.
   * **Target network:** Create a copy of the Q-network. The target network’s weights will be updated periodically to provide stable Q-value targets during training.
2. **Set up the replay memory:**
   * **Replay Memory:** Initialize a buffer to store experience tuples (state, action, reward, next state). This buffer will hold a fixed number of experiences to sample from during training.
3. **For each episode:**
   * **Start the game and initialize the state:**
     + Reset the game environment to the initial state.
     + Capture the initial game frame, preprocess it, and set it as the current state.
4. **For each time step:**
   * **Select an action using the epsilon-greedy policy:**
     + With probability ε, select a random action (exploration).
     + With probability 1-ε, select the action with the highest predicted Q-value from the Q-network (exploitation).
   * **Execute the action and observe the reward and next state:**
     + Perform the selected action in the game.
     + Observe the immediate reward and capture the next game frame.
     + Preprocess the next game frame to represent the next state.
   * **Store the experience in replay memory:**
     + Store the tuple (state, action, reward, next state) in the replay memory.
   * **Sample a mini-batch from replay memory:**
     + Randomly sample a mini-batch of experience tuples from the replay memory.
   * **Perform a gradient descent step on the loss derived from the Bellman equation:**
     + For each experience in the mini-batch, calculate the target Q-value using the reward and the maximum predicted Q-value from the target network for the next state.
     + Compute the loss between the predicted Q-value (from the Q-network) and the target Q-value.
     + Perform a gradient descent step to minimize this loss, updating the Q-network’s weights.
   * **Update the target network periodically:**
     + Every fixed number of steps, copy the weights from the Q-network to the target network to provide stable Q-value targets.

**Hyperparameters:**

* **Potential Hyperparameters:**
  + **Learning rate:** Controls the step size of the gradient descent updates.
  + **Discount factor (γ):** Determines the importance of future rewards. A higher γ values future rewards more.
  + **Epsilon decay rate:** Determines how quickly the exploration rate (ε) decreases over time.
  + **Mini-batch size:** Number of experiences sampled from replay memory for each training update.
  + **Replay memory size:** Maximum number of experiences stored in the replay buffer.
* **Tuning:**
  + Use grid search or random search to explore different combinations of hyperparameter values and identify the optimal settings.

**Handling Training Issues:**

* **Catastrophic Forgetting:**
  + **Target Network:** Use a separate target network to provide stable Q-value targets.
  + **Replay Memory:** Store and sample experiences from replay memory to break the correlation between consecutive experiences and ensure more stable learning.
* **Reward Sparsity:**
  + **Reward Shaping:** Modify the reward structure to provide more frequent feedback, such as small positive rewards for staying alive or slight penalties for being too close to the pipes.
  + **Auxiliary Tasks:** Introduce additional learning tasks that provide more frequent rewards, such as avoiding certain zones in the game.

**Performance Evaluation:**

* **During Training:**
  + **Track Metrics:** Monitor the average reward per episode, loss values, and Q-values to gauge the agent’s learning progress and stability.

**Testing and Evaluation**

**Testing Strategy:**

* **Comprehensive Testing:**
  + Evaluate the agent’s performance over multiple episodes to account for variability in game scenarios and ensure robustness.
* **Metrics:**
  + **Average Score:** Average number of pipes passed per episode.
  + **Survival Time:** Average time the agent survives in the game.
  + **Number of Pipes Passed:** Total number of pipes the agent successfully navigates.

**Result Interpretation:**

* **Benchmarks:**
  + Compare the agent’s performance against a random policy (randomly selecting actions) and a heuristic-based policy (simple predefined rules).
* **Visualization:**
  + Plot learning curves showing the average reward per episode over time.
  + Visualize game frames with the agent’s actions to understand its decision-making process.
  + Analyze decision-making patterns to identify strengths and weaknesses in the agent’s policy.

**Potential Improvements:**

* **Future Work:**
  + **Advanced Algorithms:** Explore more advanced reinforcement learning algorithms like Double DQN (which reduces overestimation bias) or Dueling DQN (which separately estimates the value of states and the advantage of actions).
  + **Sophisticated State Representations:** Incorporate more complex state representations, such as using recurrent neural networks (RNNs) to capture temporal dependencies in the game frames.
  + **Reward Structures:** Experiment with different reward structures to provide more informative feedback to the agent, potentially improving learning efficiency and performance.

By carefully implementing and evaluating these steps, you can effectively train an AI agent to play Flappy Bird using reinforcement learning, while continually assessing and improving its performance through rigorous testing and analysis.

**Bibliography Report**

1. **Academic Papers and Books:**
   * Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533.
   * Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.
2. **Online Articles and Tutorials:**
   * Brownlee, J. (2016). An introduction to Q-learning: reinforcement learning. *Machine Learning Mastery*. Retrieved from https://machinelearningmastery.com/q-learning/.
   * Karpathy, A. (2016). Deep reinforcement learning: Pong from pixels. *karpathy.github.io*. Retrieved from http://karpathy.github.io/2016/05/31/rl/.
3. **Documentation and Libraries:**
   * OpenAI Gym. (2020). Environments documentation. Retrieved from <https://gym.openai.com/docs/>.
   * PyGame. (2021). Pygame documentation. Retrieved from https://www.pygame.org/docs/.