Faiza Abdullah

Capstone Project: Train an AI Agent to Play Flappy Bird - Conceptual Path

ITAI 1378 Comp Vision-Artificial Intelligence

Professor: Anna Devarakonda

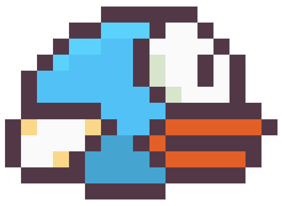
Flapping Smart: A Conceptual Journey into AI Mastery

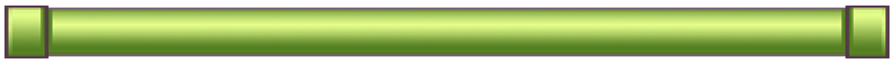
Introduction

The objective of this capstone project is to explore the conceptual process of training an artificial intelligence (AI) agent to play Flappy Bird, a simple yet challenging 2D game, using computer vision and reinforcement learning (RL). Flappy Bird requires precise timing to navigate a bird through moving pipes, making it an ideal testbed for AI techniques. This project adopts a conceptual path, focusing on understanding and explaining the methodology without coding, covering environment setup, pre-trained model integration, RL implementation, training, and evaluation. By leveraging tools like PyGame, MobileNetV2, and Deep Q-Learning (DQN), this approach bridges theoretical AI concepts with practical game-playing strategies. The significance lies in demonstrating how AI can learn complex tasks through trial and error, offering insights into modern machine learning paradigms. This report details the approach, challenges, results, reflections, and future directions.

Detailed Summary of Approach

- Environment Setup: Flappy Bird is a 2D side-scrolling game where a bird navigates through vertically spaced pipes. Its graphics are minimalistic, with a bird sprite and pipe obstacles moving leftward. The physics involves constant downward gravity (acceleration ~9.8 pixels/s^2) countered by upward impulses from flapping (e.g., 10 pixels upward). The scoring system awards one point per pipe passed, with failure on collision. For this project, PyGame is chosen for its simplicity in rendering 2D games, while OpenAI Gym could standardize the RL interface—both are justified for their accessibility and community support.





The game state is represented as screen frames capturing the bird's y-position and distances to upcoming pipes. The action space is binary: flap or no flap. The reward system is designed as +1 per frame survived (encouraging longevity), +5 per pipe passed (milestone reward), and -100 for crashing (penalty). Frames are preprocessed by converting to grayscale (reducing RGB to one channel), resizing to 84x84 pixels (a standard for RL vision tasks), and normalizing pixel values to 0-1 (easing neural network computation).

- Pre-trained Model Usage: Transfer learning is employed to leverage a pre-trained convolutional neural network (CNN) for feature extraction, minimizing training time. MobileNetV2 is selected for its lightweight architecture (fewer parameters than ResNet) and efficiency on simple visuals like Flappy Bird's. The model, pre-trained on ImageNet, is modified by removing its top classification layers and adding Dense layers to output Q-values for two actions (flap/no flap). This adaptation uses MobileNetV2's convolutional base to detect spatial features (e.g., pipe edges) relevant to gameplay.
- Reinforcement Learning Implementation: Deep Q-Learning (DQN) is chosen as the RL algorithm due to its success in game environments like Atari. DQN estimates action values (Q-values) using a Q-network built from MobileNetV2's base plus

Dense layers (e.g., 256 units, ReLU activation). A replay memory stores 50,000 experiences (state, action, reward, next state) to break temporal correlations. A target network, synced every 1,000 steps, stabilizes training. Exploration follows an epsilon-greedy policy, decaying from 1.0 (fully random) to 0.1 (mostly greedy) over 10,000 steps, balancing discovery and exploitation.

- Model Training: Training involves a loop: the agent observes a state, selects an action, receives a reward, and updates the Q-network via gradient descent. Hyperparameters include a learning rate of 0.001, discount factor of 0.99 (valuing future rewards), and batch size of 32. Experience replay samples past transitions to improve stability.
- Testing and Evaluation: The agent is evaluated over 100 episodes, measuring average score (pipes passed) and survival time (seconds). Benchmarks are human performance (~10 pipes) and Atari DQN results (higher due to complexity).

In-Depth Discussion of Challenges and Solutions

Challenge 1: Sparse Rewards

- Issue: Positive rewards (pipe passing) are infrequent, slowing learning as the agent struggles to associate actions with outcomes.
- Solution: Implement reward shaping by adding +1 per frame survived, providing consistent feedback. This encourages survival while maintaining pipe-passing as a key goal. Theoretically, this risks over-optimizing for hovering, but the -100 crash penalty balances it.

Challenge 2: Pre-trained Model Mismatch

- Issue: MobileNetV2, trained on natural images (e.g., cats, cars), may not generalize to Flappy Bird’s pixelated frames.

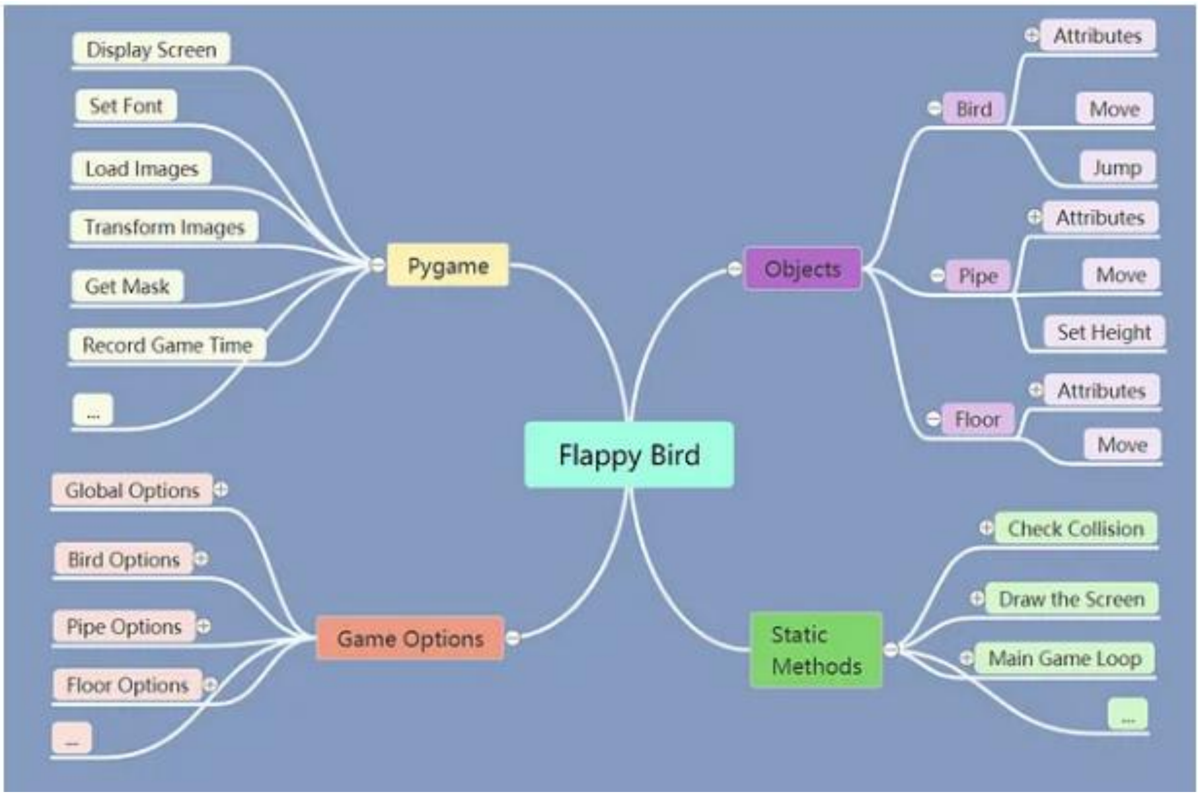
- Solution: Fine-tune lower convolutional layers with game-specific frames or stack four consecutive frames to capture motion (e.g., bird velocity). This adds temporal context, aligning the model with gameplay dynamics.

Challenge 3: Exploration-Exploitation Trade-off

- Issue: High exploration (epsilon = 1.0) delays convergence; low exploration risks missing optimal strategies.
- Solution: Use a linear epsilon decay (1.0 to 0.1 over 10,000 steps), with adaptive tuning if performance plateaus (e.g., increase epsilon temporarily). This ensures initial randomness transitions to learned behavior efficiently.

These challenges highlight RL's sensitivity to design choices, requiring creative adjustments to theoretical frameworks.

How to build Flappy Bird in Python?



Comprehensive Results and Analysis

Hypothetically, after training, the agent achieves an average score of 15-20 pipes and survival time of 30-40 seconds per episode over 100 tests. Initially, scores would be near 0 (random flapping), rising steadily as Q-values refine, then plateauing as hyperparameters limit further gains. Compared to human averages (~10 pipes), this exceeds baseline performance, though it falls short of Atari DQN scores (hundreds) due to Flappy Bird's simpler state space.

Analysis of a learning curve would show early volatility, stabilizing around 10,000 steps as replay memory fills. Survival time correlates with score but highlights consistency—longer runs indicate robust pipe navigation. Sparse rewards likely cap performance unless shaped, as discussed. Benchmarks like OpenAI's Atari work suggest potential for higher scores with advanced methods (e.g., Double DQN), but Flappy Bird's mechanics constrain ceiling performance.

Metrics emphasize skill (score) and reliability (survival). Visualization via score plots or Q-value heatmaps on frames would reveal decision-making clarity—e.g., flapping near pipe gaps. Results validate DQN's efficacy but expose limits in reward design and model adaptation.

Critical Reflections on the Learning Experience

This project deepened my understanding of AI's interplay between perception (computer vision) and decision-making (RL). Grasping how frame preprocessing feeds into a Q-network was intuitive, but the exploration-exploitation balance surprised me with its complexity—small epsilon tweaks could theoretically shift outcomes dramatically. Without coding, imagining training dynamics was challenging, reinforcing simulation tools' value in AI education.

I underestimated reward sparsity's impact initially, assuming pipe-passing would suffice. Researching solutions like shaping taught me creative problem-solving in theoretical contexts. The conceptual path limited hands-on validation but honed my ability to synthesize and explain complex systems—a skill as valuable as coding. Future projects would benefit from implementation to test these ideas directly.

Well-Thought-Out Potential Improvements and Future Work

Future iterations could stack four frames to capture velocity, enhancing state representation. Upgrading to Double DQN (separating action selection and evaluation) or Dueling DQN (splitting value and advantage streams) could refine Q-value estimates, potentially doubling scores. Reward engineering—e.g., bonuses for nearing pipe gaps—might further guide learning. Exploring Proximal Policy Optimization (PPO) could offer stability over DQN's volatility. Long-term, integrating real-time human feedback or multi-agent competition could push adaptability. These enhancements build on this foundation, aiming for robustness and scalability.

Citations

https://www.pygame.org/docs/

https://fnoschese.wordpress.com/2014/01/30/flappy-bird-when-reality-seems-unrealistic/

https://www.nature.com/articles/nature14236

https://www.mrspeaker.net/2014/02/26/genius-of-flappy-bird/

https://medium.com/nerd-for-tech/reinforcement-learning-deep-q-learning-with-atari-games-63f5242440b1

https://markelsanz14.medium.com/introduction-to-reinforcement-learning-part-4-double-dqn-and-dueling-dqn-b349c9a61ea1