Deep Q-Learning for FlappyBird

Lavric Adrian-Gabriel 3A3
Ioja Stefan-Valentin 3B2
Facultatea de Informatica, UAIC

January 12, 2025

Introduction

This report details the architecture of the neural network, the implementation of the DQN algorithm, the experimental setup, and the results obtained from various training sessions. By analyzing the performance logs and graphs, we assess the effectiveness of the DQN approach in mastering the FlappyBird environment.

Neural Network Architecture

The Deep Q-Network (DQN) used in this implementation is designed to approximate the action-value function $Q(s, a)$. The architecture incorporates a dueling network structure to separately estimate the value of being in a state and the advantage of each action. This helps improve stability and learning efficiency.

Structure:

- **Input Layer:** The input consists of features representing the game state, such as the bird's position, velocity, and distances to the next pipe.
- **Fully Connected Layer:** A hidden layer with 256 neurons and ReLU activation.
- **Dueling Network (if enabled):**
  - **Value Stream:** Computes the value $V(s)$ of the current state using a fully connected layer with 256 nodes and ReLU activation, followed by a linear layer to output a single value.

  - **Advantage Stream:** Computes the advantage $A(s, a)$ for each action. A fully connected layer with 256 nodes and ReLU activation is followed by a linear layer to output action advantages.
  - The final Q-value output is computed as:

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a')$$

- **Output Layer (if dueling is disabled):** Directly maps the hidden layer to Q-values for each action.

Key Techniques:

- **Weight Initialization:** Kaiming uniform initialization is applied to ensure proper weight scaling.
- **Activation Function:** ReLU is used for non-linearity, promoting faster training.

Best Parameter Configuration:

- **Learning Rate (α)**: 0.00025
- **Discount Factor (γ)**: 0.99
- **Replay Memory Size**: 100,000 transitions
- **Mini-batch Size**: 32
- **Initial Epsilon (ϵ)**: 1.0
- **Epsilon Decay Rate**: 0.9995
- **Minimum Epsilon**: 0.01
- **Target Network Sync Frequency**: 500
- **Hidden Layer Size**: 256 nodes

This combination balances exploration and exploitation while ensuring stable learning, allowing the agent to effectively learn optimal policies for FlappyBird.

Q-Learning Algorithm Implementation

The Deep Q-Learning agent implemented for the FlappyBird environment integrates several advanced features to ensure stable and efficient training. Below, we detail the core components and their roles in the algorithm.

1. Initialization

- **Q-Networks:**
  - The agent initializes two neural networks:
    - The **policy network** (Q_{policy}) predicts the Q-values for the current state-action pairs.
    - The **target network** (Q_{target}) provides stable Q-value targets during optimization.
  - The weights of the target network are periodically synchronized with those of the policy network to decouple Q-value targets from the policy updates, improving stability.
- **Replay Memory:** A **Replay Buffer** stores transitions (s, a, r, s'), sampled randomly to break the correlation between consecutive experiences. This ensures that training data is independent and identically distributed, which is crucial for effective gradient-based learning.
- **Hyperparameters:** Key parameters, such as learning rate, discount factor (γ), exploration rate (ϵ), and network synchronization frequency, are defined and optimized to balance exploration and exploitation.

2. Training Loop

- **Episode Management:** The agent iterates through episodes until termination (e.g., maximum reward achieved or game over).
- **Action Selection:** An ϵ-greedy policy is used:
  - With probability ϵ, the agent chooses a random action (exploration).
  - Otherwise, it selects the action with the highest predicted Q-value from the policy network (exploitation).
- **Transition and Reward Processing:** The environment returns the next state, reward, and termination flag after the agent performs an action. The transition is stored in the replay buffer for later sampling.

- **Optimization:** After enough transitions are stored, a mini-batch is sampled to optimize the policy network:

    1. Compute the **target Q-value** using the Bellman equation:

    $$Q_{\text{target}} = r + \gamma \max_{a'} Q_{\text{target}}(s', a')$$

    2. For **Double DQN**, the target is computed as:

    $$Q_{\text{target}} = r + \gamma Q_{\text{target}}(s', \arg\max_{a'} Q_{\text{policy}}(s', a'))$$

    3. Calculate the loss between predicted and target Q-values:

    $$\text{Loss} = \frac{1}{N} \sum_{i=1}^{N} \left(Q_{\text{policy}}(s_i, a_i) - Q_{\text{target}}(s_i, a_i)\right)^2$$

    4. Update the policy network using backpropagation and gradient descent.

- **Target Network Update:** The weights of the target network are periodically updated to match the policy network, providing stable Q-value targets.

3. Exploration-Exploitation Tradeoff

The exploration rate (ϵ) is decayed over time:

$$\epsilon = \max(\epsilon \times \text{decay rate}, \epsilon_{\min})$$

This ensures the agent transitions from exploration (random actions) to exploitation (optimal actions) as training progresses.

4. Experience Replay

- The replay buffer stores transitions up to a fixed capacity.
- Sampling uniformly from the buffer improves stability and reduces variance during training.
- A mini-batch size of 32 is used for stochastic gradient descent updates.

5. Results Logging and Model Saving

- Cumulative rewards for each episode are tracked and logged for analysis.
- If a new best reward is achieved, the policy network's weights are saved.

6. Key Features

- **Double DQN:** Reduces overestimation bias of Q-values by decoupling action selection and evaluation.
- **Replay Buffer:** Promotes stability and reduces variance during training.
- **Target Network Synchronization:** Stabilizes learning by providing consistent Q-value targets.

Optimization Process

The `optimize` function executes the following steps:

1. Extract transitions from the mini-batch.
2. Compute predicted Q-values for current state-action pairs.
3. Compute target Q-values using the target network.
4. Calculate the loss between predicted and target Q-values.
5. Update the policy network using backpropagation.

Experimental Results

Below are results from multiple training runs with varying hyperparameters.

```
01-12 01:37:31: Start:
01-12 01:37:32: Best reward -6.9 (-100.0%) at episode 0, model saved.
01-12 01:37:32: Best reward -3.9 (-43.5%) at episode 2, model saved.
01-12 01:37:33: Best reward -0.9 (-76.9%) at episode 58, model saved.
01-12 01:37:39: Best reward 0.3 (-133.3%) at episode 958, model saved.
01-12 01:37:47: Best reward 3.3 (+1000.0%) at episode 1751, model saved.
01-12 01:37:53: Best reward 3.9 (+18.2%) at episode 2285, model saved.
01-12 01:38:48: Best reward 4.4 (+12.8%) at episode 6793, model saved.
01-12 01:39:57: Best reward 4.7 (+6.8%) at episode 11607, model saved.
01-12 01:40:10: Best reward 4.9 (+4.3%) at episode 12611, model saved.
01-12 01:40:11: Best reward 6.6 (+34.7%) at episode 12710, model saved.
01-12 01:40:41: Best reward 8.4 (+27.3%) at episode 15022, model saved.
01-12 01:45:48: Best reward 9.0 (+7.1%) at episode 35113, model saved.
01-12 01:46:27: Best reward 9.2 (+2.2%) at episode 37441, model saved.
01-12 01:46:31: Best reward 9.4 (+2.2%) at episode 37699, model saved.
01-12 01:47:15: Best reward 13.4 (+42.6%) at episode 40406, model saved.
01-12 01:49:39: Best reward 13.6 (+1.5%) at episode 48702, model saved.
01-12 01:49:56: Best reward 16.1 (+18.4%) at episode 49659, model saved.
01-12 01:51:06: Best reward 17.9 (+11.2%) at episode 53579, model saved.
01-12 01:51:33: Best reward 22.4 (+25.1%) at episode 55056, model saved.
01-12 01:52:18: Best reward 26.9 (+20.1%) at episode 57290, model saved.
01-12 01:55:19: Best reward 41.8 (+55.4%) at episode 66260, model saved.
01-12 02:06:45: Best reward 50.4 (+20.6%) at episode 97672, model saved.
01-12 02:13:58: Best reward 55.8 (+10.7%) at episode 114354, model saved.
01-12 02:15:57: Best reward 57.1 (+2.3%) at episode 118819, model saved.
01-12 02:16:46: Best reward 59.9 (+4.9%) at episode 120588, model saved.
01-12 02:18:12: Best reward 60.8 (+1.5%) at episode 123717, model saved.
01-12 02:18:30: Best reward 62.1 (+2.1%) at episode 124327, model saved.
01-12 02:18:50: Best reward 67.2 (+8.2%) at episode 124980, model saved.
01-12 02:24:50: Best reward 74.5 (+10.9%) at episode 137100, model saved.
01-12 02:26:10: Best reward 78.4 (+5.2%) at episode 139730, model saved.
01-12 02:39:59: Best reward 80.7 (+2.9%) at episode 165937, model saved.
01-12 03:01:51: Best reward 92.4 (+14.5%) at episode 204883, model saved.
01-12 03:09:58: Best reward 111.2 (+20.3%) at episode 218159, model saved.
```

Figure 1: Training log for Run 1.

```
self.learning_rate_a     = 0.00025
self.discount_factor_g   = 0.99
self.network_sync_rate   = 100
self.replay_memory_size  = 200000
self.mini_batch_size     = 64
self.epsilon_init        = 1
self.epsilon_decay       = 0.9995
self.epsilon_min         = 0.05
self.stop_on_reward      = 10000
self.fc1_nodes           = 512
self.enable_double_dqn   = True
self.loss_fn = nn.MSELoss()
```


Figure 2: Hyperparameter configuration for Run 1.

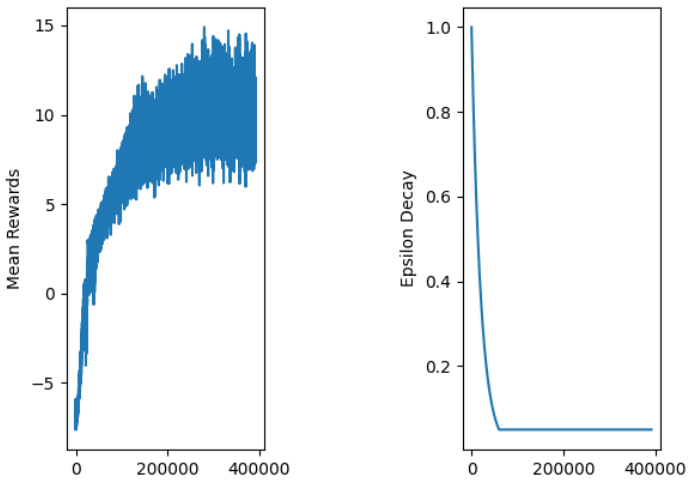

Figure 3: Mean reward and epsilon decay during Run 1.

```
01-12 15:56:06: Start:
01-12 15:56:07: Best reward -6.9 (-100.0%) at episode 0, model saved.
01-12 15:56:07: Best reward -6.3 (-8.7%) at episode 1, model saved.
01-12 15:56:07: Best reward -5.7 (-9.5%) at episode 6, model saved.
01-12 15:56:08: Best reward -4.5 (-21.1%) at episode 79, model saved.
01-12 15:56:09: Best reward -2.1 (-53.3%) at episode 154, model saved.
01-12 15:56:11: Best reward -1.5 (-28.6%) at episode 445, model saved.
01-12 15:56:11: Best reward -0.9 (-40.0%) at episode 452, model saved.
01-12 15:56:11: Best reward 0.9 (-200.0%) at episode 497, model saved.
01-12 15:56:12: Best reward 2.1 (+133.3%) at episode 560, model saved.
01-12 15:56:13: Best reward 3.9 (+85.7%) at episode 668, model saved.
01-12 15:56:21: Best reward 4.1 (+5.1%) at episode 1336, model saved.
01-12 15:56:23: Best reward 4.5 (+9.8%) at episode 1522, model saved.
01-12 15:56:26: Best reward 8.4 (+86.7%) at episode 1717, model saved.
01-12 15:57:24: Best reward 8.5 (+1.2%) at episode 5696, model saved.
01-12 15:57:25: Best reward 8.7 (+2.4%) at episode 5753, model saved.
01-12 15:57:30: Best reward 10.7 (+23.0%) at episode 6067, model saved.
01-12 15:57:31: Best reward 11.2 (+4.7%) at episode 6089, model saved.
01-12 15:57:33: Best reward 12.9 (+15.2%) at episode 6231, model saved.
01-12 15:57:35: Best reward 17.9 (+38.8%) at episode 6348, model saved.
01-12 15:58:41: Best reward 26.9 (+42.3%) at episode 10265, model saved.
01-12 15:59:04: Best reward 32.3 (+20.1%) at episode 11555, model saved.
01-12 15:59:07: Best reward 43.3 (+34.1%) at episode 11706, model saved.
01-12 15:59:59: Best reward 46.5 (+7.4%) at episode 14238, model saved.
01-12 16:00:13: Best reward 50.4 (+8.4%) at episode 14864, model saved.
01-12 16:00:45: Best reward 64.4 (+27.8%) at episode 16310, model saved.
01-12 16:00:51: Best reward 106.4 (+65.2%) at episode 16533, model saved.
01-12 16:03:09: Best reward 110.9 (+4.2%) at episode 21282, model saved.
01-12 16:03:31: Best reward 135.3 (+22.0%) at episode 22080, model saved.
01-12 16:03:52: Best reward 135.6 (+0.2%) at episode 22793, model saved.
01-12 16:09:35: Best reward 151.3 (+11.6%) at episode 32426, model saved.
01-12 16:10:44: Best reward 167.5 (+10.7%) at episode 34129, model saved.
01-12 16:11:07: Best reward 186.3 (+11.2%) at episode 34617, model saved.
01-12 16:14:19: Best reward 230.6 (+23.8%) at episode 38916, model saved.
01-12 16:17:07: Best reward 236.9 (+2.7%) at episode 42152, model saved.
01-12 16:18:28: Best reward 264.9 (+11.8%) at episode 43423, model saved.
01-12 16:24:09: Best reward 281.3 (+6.2%) at episode 49128, model saved.
01-12 16:30:10: Best reward 391.0 (+39.0%) at episode 55042, model saved.
01-12 16:37:42: Best reward 419.7 (+7.3%) at episode 62220, model saved.
01-12 17:53:28: Best reward 498.4 (+18.8%) at episode 112616, model saved.
```

Figure 4: Training log for Run 2.

```
self.learning_rate_a    = 0.00025
self.discount_factor_g  = 0.99
self.network_sync_rate  = 500
self.replay_memory_size = 100000
self.mini_batch_size    = 32
self.epsilon_init       = 1
self.epsilon_decay      = 0.9995
self.epsilon_min        = 0.01
self.stop_on_reward     = 5000
self.fc1_nodes          = 256
self.enable_double_dqn  = True
self.loss_fn = nn.MSELoss()
```

Figure 5: Hyperparameter configuration for Run 2.

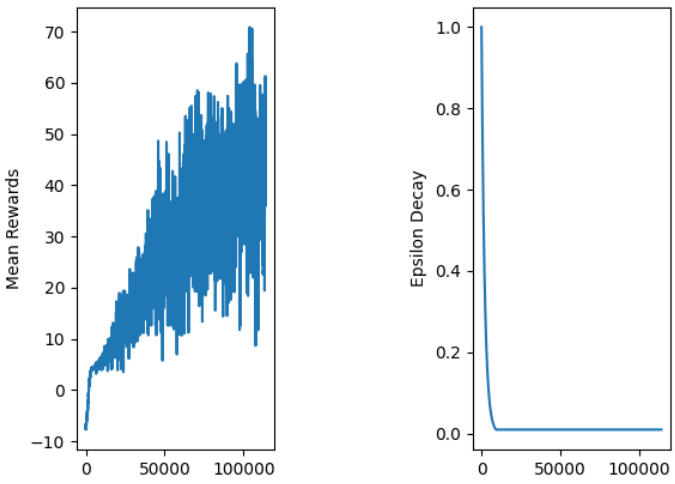

Figure 6: Mean reward and epsilon decay during Run 2.

```
01-12 14:03:36: Start:
01-12 14:03:37: Best reward -7.5 (-100.0%) at episode 0, model saved.
01-12 14:03:37: Best reward -6.9 (-8.0%) at episode 2, model saved.
01-12 14:03:37: Best reward -6.3 (-8.7%) at episode 3, model saved.
01-12 14:03:37: Best reward -3.9 (-38.1%) at episode 16, model saved.
01-12 14:03:37: Best reward -3.3 (-15.4%) at episode 74, model saved.
01-12 14:03:40: Best reward 0.3 (-109.1%) at episode 458, model saved.
01-12 14:03:40: Best reward 0.9 (+200.0%) at episode 495, model saved.
01-12 14:03:41: Best reward 3.3 (+266.7%) at episode 585, model saved.
01-12 14:03:42: Best reward 3.9 (+18.2%) at episode 671, model saved.
01-12 14:03:45: Best reward 4.3 (+10.3%) at episode 1108, model saved.
01-12 14:03:47: Best reward 6.3 (+46.5%) at episode 1216, model saved.
01-12 14:03:52: Best reward 6.8 (+7.9%) at episode 1805, model saved.
01-12 14:03:57: Best reward 8.4 (+23.5%) at episode 2214, model saved.
01-12 14:04:05: Best reward 8.8 (+4.8%) at episode 2973, model saved.
01-12 14:04:42: Best reward 9.1 (+3.4%) at episode 5460, model saved.
01-12 14:04:48: Best reward 12.9 (+41.8%) at episode 5916, model saved.
01-12 14:04:59: Best reward 13.5 (+4.7%) at episode 6603, model saved.
01-12 14:05:37: Best reward 15.2 (+12.6%) at episode 9035, model saved.
01-12 14:05:41: Best reward 15.3 (+0.7%) at episode 9336, model saved.
01-12 14:06:08: Best reward 17.9 (+17.0%) at episode 11082, model saved.
01-12 14:06:10: Best reward 22.4 (+25.1%) at episode 11195, model saved.
01-12 14:06:43: Best reward 26.9 (+20.1%) at episode 13086, model saved.
01-12 14:09:34: Best reward 29.5 (+9.7%) at episode 22132, model saved.
01-12 14:09:59: Best reward 36.4 (+23.4%) at episode 23316, model saved.
01-12 14:11:52: Best reward 48.5 (+33.2%) at episode 28860, model saved.
01-12 14:13:20: Best reward 48.6 (+0.2%) at episode 32890, model saved.
01-12 14:13:46: Best reward 50.4 (+3.7%) at episode 33948, model saved.
01-12 14:14:01: Best reward 53.0 (+5.2%) at episode 34470, model saved.
01-12 14:15:30: Best reward 59.9 (+13.0%) at episode 38128, model saved.
01-12 14:16:58: Best reward 64.4 (+7.5%) at episode 41980, model saved.
01-12 14:20:14: Best reward 73.9 (+14.8%) at episode 50130, model saved.
01-12 14:41:20: Best reward 78.4 (+6.1%) at episode 97697, model saved.
01-12 14:42:40: Best reward 82.9 (+5.7%) at episode 100657, model saved.
01-12 14:44:32: Best reward 96.9 (+16.9%) at episode 104461, model saved.
01-12 14:57:07: Best reward 101.9 (+5.2%) at episode 129532, model saved.
01-12 15:49:52: Best reward 111.6 (+9.5%) at episode 221499, model saved.
```

Figure 7: Training log for Run 3.

```
self.learning_rate_a     = 0.0001
self.discount_factor_g   = 0.98
self.network_sync_rate   = 1000
self.replay_memory_size  = 50000
self.mini_batch_size     = 32
self.epsilon_init        = 1
self.epsilon_decay       = 0.9995
self.epsilon_min         = 0.05
self.stop_on_reward      = 5000
self.fc1_nodes           = 256
self.enable_double_dqn   = True
self.loss_fn = nn.MSELoss()
```


Figure 8: Hyperparameter configuration for Run 3.

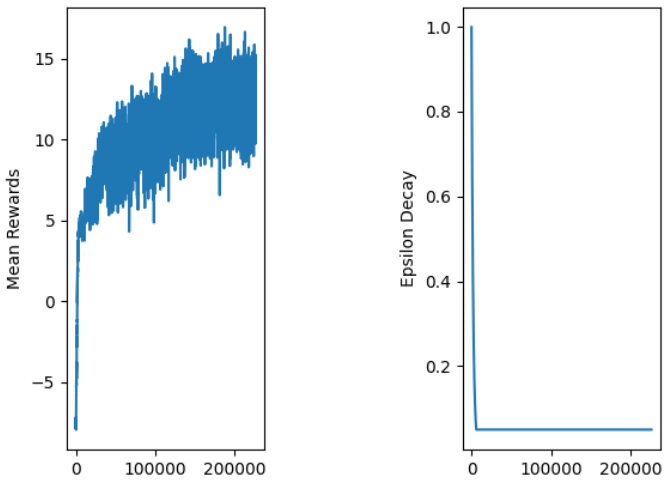

Figure 9: Mean reward and epsilon decay during Run 3.

Conclusion

This project demonstrated the successful application of a Deep Q-Network (DQN) to train an agent for the FlappyBird game. By integrating techniques such as experience replay, target network updates, and double Q-learning, the agent was able to learn a robust policy in a challenging and dynamic environment.

The experimental results highlighted the importance of hyperparameter tuning and stabilizing methods for achieving consistent performance. Future improvements could explore advanced architectures and alternative reinforcement learning algorithms to further enhance efficiency and scalability.