Flappy Bird - Controlul unui agent prin Q-Learning

Carare Theodora-Sabina și Crăiescu Iustin-Darius

I. INTRODUCERE

Scopul acestui proiect este implementarea și antrenarea unui agent inteligent capabil să joace jocul Flappy Bird folosind algoritmul **Q-learning cu rețea neuronală**.
Agentul învață exclusiv din **input vizual (pixeli)**, fără acces la starea internă a jocului, folosind:

- o rețea neuronală convoluțională (CNN),
- replay buffer,
- politică epsilon-greedy,
- antrenare off-policy.

Obiectul este maximizarea scorului (numărul de pipe-uri consecutive trecute).

II. ENVIRONMENT ȘI SETUP-UL PROBLEMEI

A. *Mediul utilizat*

Am folosit mediul:

```
FlappyBird-v0 (flappy-bird-gymnasium)
```

Instalare:

```
pip install gymnasium flappy-bird-gymnasium
    opencv-python
```

Inițializare mediu:

```
env = gym.make("FlappyBird-v0", render_mode="
    rgb_array")
```

B. *Spațiul de acțiuni*

Agentul are 2 acțiuni posibile:

- 0 - nu sare
- 1 - sare

C. *Recompensă (reward)*

Mediul oferă:

- +1 pentru fiecare pipe trecut
- episodul se termină la coliziune

Am aplicat penalizare suplimentară:

```
adj_reward = reward if not done else -15
```

pentru a penaliza moartea și a forța agentul să evite coliziunile.

III. REPREZENTAREA ȘI PREPROCESAREA INPUTULUI

Agentul primește ca observație imaginea jocului randată de motorul grafic.

A. *Preprocesare:*

- conversie la grayscale
- redimensionare la 64x64
- normalizare în rețea

```
def preprocess_frame(frame):
gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
resized = cv2.resize(gray, (64, 64))
return resized.astype(np.uint8)
```

B. *Dimensiune input*

```
1 x 64 x 64 (grayscale)
```

IV. ARHITECTURA REȚELEI NEURONALE

Pentru a aproxima funcția Q din algoritmul Deep Q-Learning am folosit o rețea neuronală convoluțională potrivită pentru procesarea imaginilor.
Inputul rețelei este un frame din joc, preprocesat în format grayscale și redimensionat la 64x64 pixeli.

- Rețeaua este formată din **3 straturi convoluționale** care extrag automat informații relevante din imagine (poziția păsării, poziția pipe-urilor și distanța până la următorul obstacol).
- După partea convoluțională, **un strat fully-connected cu 512 neuroni** combină toate informațiile extrase și învață o reprezentare globală a stării jocului. Stratul de ieșire produce două valori Q, câte una pentru fiecare acțiune posibilă.
- Funcția de activare folosită este **ReLU**, iar la ieșire se aplică softmax deoarece valorile estimate sunt Q-values, nu probabilități. Acțiunea este aleasă prin selecția valorii maxime. Această arhitectură este potrivită pentru Flappy Bird deoarece permite agentului să învețe direct din pixeli ce înseamnă o poziție bună și când trebuie să sară, fără a fi nevoie de informații explicite despre pozitia pipe-urilor.

```
class FlappyDQN(nn.Module):
def __init__(self, action_size=2):
    super(FlappyDQN, self).__init__()
    self.conv1 = nn.Conv2d(1, 32, kernel_size
        =8, stride=4)
    self.conv2 = nn.Conv2d(32, 64, kernel_size
        =4, stride=2)
    self.conv3 = nn.Conv2d(64, 64, kernel_size
        =3, stride=1)
    self.fc1 = nn.Linear(64 * 4 * 4, 512)
    self.fc2 = nn.Linear(512, action_size)
```

V. IMPLEMENTAREA ALGORITMULUI Q-LEARNING

Agentul este antrenat folosind algoritmul Q-learning cu aproximare prin rețea neuronală. Rețeaua aproximează funcția de valoare **Q(s,a)**, care estimează recompensa cumulată așteptată pentru executarea unei acțiuni într-o anumită stare.

Procesul de învățare se desfășoară ciclic, parcurgând următoarele etape:

1) **Observarea stării**
Starea mediului este reprezentată de un frame al jocului, preprocesat sub forma unei imagini grayscale de dimensiune 64x64.

```
state = preprocess_frame(env.render())
```

Acesta constituie inputul rețelei neuronale.

2) **Selectarea acțiunii (politica epsilon-greedy)**
Agentul selectează acțiunea folosind o politică epsilon-greedy:

- cu probabilitatea ϵ alege o acțiune aleatorie (explorare),
- cu probabilitatea $1-\epsilon$ alege o acțiune cu valoarea Q maximă (exploatare).

```
def select_action(self, state):
    if random.random() < self.epsilon:
        return random.randrange(self.
            action_size)

    with torch.no_grad():
        state_t = torch.as_tensor(state).
            unsqueeze(0).unsqueeze(0)
        return torch.argmax(self.
            policy_net(state_t)).item()
```

3) **Executarea acțiunii și obținerea recompensei**
Acțiunea este aplicată în mediu, iar agentul primește recompensa și următoarea stare.

```
_, reward, terminated, truncated, info = env.
    step(action)
done = terminated or truncated
next_state = preprocess_frame(env.render())
```

Pentru penalizarea coliziunii se aplică:

```
    adj_reward = reward if not done else -15
```

4) **Salvarea experienței (Replay Buffer)**
Tranziția este salvată în buffer-ul de experiență sub forma:

$$(s, a, r, s', done)$$

```
    self.memory.append((state, action,
        adj_reward, next_state, done))
```

5) **Eșantionarea unui batch de antrenare**
Se extrage un batch aleator din replay buffer.

```
batch = random.sample(self.memory, self.
    batch_size)
states, actions, rewards, next_states, dones =
     zip(*batch)
```

6) **Calculul valorii Q curente**
Se calculează valorile Q estimate din rețea pentru acțiunile executate.

7) **Calculul valorii Q țintă**
Se aplică regula Q-learning:

$$Q(s,a) = r + \gamma \max_{a'} Q(s', a')$$

```
with torch.no_grad():
    max_next_q = self.policy_net(next_states_t
        ).max(1)[0]
    target_q = rewards_t + self.gamma *
        max_next_q * (1 - dones_t)
```

8) **Actualizarea rețelei neuronale**
Se minimizează eroarea dintre valoarea Q curentă și valoarea Q țintă folosind MSE (Mean Squared Error) :

$$\text{MSE} = \frac{1}{N} \sum_{i=n}^{N} \left(Q_{\text{pred}}^{(i)} - Q_{\text{target}}^{(i)} \right)^2$$

unde:

- Q_{pred} = valoarea Q estimată de rețea
- Q_{target} = valaore Q țintă (din regula Q-learning)
- N = dimensiunea batch ului

```
loss = nn.MSELoss()(current_q, target_q)
self.optimizer.zero_grad()
loss.backward()
self.optimizer.step()
```

9) **Actualizarea rețelei neuronale**
Se minimizează eroarea dintre valoarea Q curentă și valoarea Q țintă folosind MSE.

```
loss = nn.MSELoss()(current_q, target_q)
self.optimizer.zero_grad()
loss.backward()
self.optimizer.step()
```

10) **Acutalizarea politicii de explorare**
După fiecare episod, epsilon este redus gradual pentru a favoriza exploatarea politicii învățate.

```
    self.epsilon *= self.epsilon_decay
```

VI. SETUP-UL LA ANTRENARE ȘI HIPERPARAMETRI

Parametru	Valoare
Learning rate	1×10^{-4}
Gamma	0.99
Batch size	64
Replay buffer size	20 000
Epsilon inițial	1.0
Epsilon final	0.01
Decay	0.99
Episoade	2000 / 5000

VII. EXPERIMENTE

A. Configurație Comună

Toate experimentele au folosit aceeași arhitectură CNN, aceeași preprocesare (grayscale) și același mecanism DQN (replay buffer + epsilon-greedy + update Q-learning).

B. Experimentul 1 - Antrenare 2000 de episoade

Scop: determinarea dacă agentul poate învăța o politică funcțională plecând doar de la pixeli și estimarea timpului necesar pentru apariția primelor comportamente utile.
Setup: 2000 de episoade de antrenare, cu epsilon decăzut până la 0.01.
Observații din log:

- În primele 200–300 episoade, scorul (pipe-uri) rămâne predominant 0, ceea ce este așteptat în condiții de explorare ridicată și reward rar.
- Primele episoade cu pipe-uri apar după câteva sute de episoade, indicând începutul învățării unui comportament non-aleator.
- Spre finalul antrenării, apar episoade cu scor semnificativ (de ordinul 10+ pipe uri), iar în ep. 1970 s-au obținut 13 pipe uri.

```
Episod: 1945 | Pipe-uri:  6 | Scor RL:   9.9 |
    Epsilon: 0.010
Episod: 1950 | Pipe-uri:  2 | Scor RL:  -0.9 |
    Epsilon: 0.010
Episod: 1955 | Pipe-uri:  3 | Scor RL:   0.2 |
    Epsilon: 0.010
Episod: 1960 | Pipe-uri:  0 | Scor RL:  -2.0 |
    Epsilon: 0.010
Episod: 1965 | Pipe-uri:  0 | Scor RL:   2.7 |
    Epsilon: 0.010
Episod: 1970 | Pipe-uri: 13 | Scor RL:  -4.6 |
    Epsilon: 0.010
--- Performanta buna! Model salvat cu 13 pipe-uri
    ---
Episod: 1975 | Pipe-uri:  4 | Scor RL:   1.4 |
    Epsilon: 0.010
Episod: 1980 | Pipe-uri:  0 | Scor RL:   1.2 |
    Epsilon: 0.010
Episod: 1985 | Pipe-uri:  6 | Scor RL:   5.5 |
    Epsilon: 0.010
Episod: 1990 | Pipe-uri:  6 | Scor RL:   2.2 |
    Epsilon: 0.010
Episod: 1995 | Pipe-uri:  3 | Scor RL:   4.9 |
    Epsilon: 0.010
```

C. Experimentul 2 - Antrenare 5000 de episoade

Scop: determinarea dacă agentul își îmbunătățește politica față de varianta cu 2000 episoade și estimarea performanței finale în regim de exploatare, pe baza numarului de pipe uri trecute
Setup: 5000 episoade de antrenare, cu epsilon decăzut până la 0.01
Observații din log:

- În primele sute de episoade scorul rămâne aproape mereu 0m ceea ce este compatibil cu faza de explorare și cu dificultatea obținerii reward ului util
- După ce epsilon scade, apar episoade cu scor pozitiv mai dez și se observă stabilizarea treptată a deciziilor (salturi mai controlate)
- Spre finalul antrenării apar episoade cu scor foarte mare, indicând o politică deja funcțională

```
--- Performanta buna ! Model salvat cu 13 pipe-uri
    ---
Episod: 4940 | Pipe-uri: 13 | Scor RL:  19.3 |
    Epsilon: 0.010
--- Performanta buna ! Model salvat cu 13 pipe-uri
    ---
--- Performanta buna ! Model salvat cu 19 pipe-uri
    ---
--- Performanta buna ! Model salvat cu 16 pipe-uri
    ---
Episod: 4945 | Pipe-uri:  8 | Scor RL:   9.1 |
    Epsilon: 0.010
Episod: 4950 | Pipe-uri:  7 | Scor RL:   8.2 |
    Epsilon: 0.010
Episod: 4955 | Pipe-uri:  4 | Scor RL:   7.9 |
    Epsilon: 0.010
Episod: 4960 | Pipe-uri:  2 | Scor RL:  -4.5 |
    Epsilon: 0.010
Episod: 4965 | Pipe-uri:  9 | Scor RL:  19.1 |
    Epsilon: 0.010
--- Performanta buna ! Model salvat cu 12 pipe-uri
    ---
Episod: 4970 | Pipe-uri:  0 | Scor RL:   0.9 |
    Epsilon: 0.010
Episod: 4975 | Pipe-uri:  1 | Scor RL:   4.6 |
    Epsilon: 0.010
Episod: 4980 | Pipe-uri:  1 | Scor RL:  -1.2 |
    Epsilon: 0.010
Episod: 4985 | Pipe-uri:  4 | Scor RL:  11.2 |
    Epsilon: 0.010
--- Performanta buna ! Model salvat cu 12 pipe-uri
    ---
Episod: 4990 | Pipe-uri:  4 | Scor RL:  -1.6 |
    Epsilon: 0.010
--- Performanta buna ! Model salvat cu 29 pipe-uri
    ---
Episod: 4995 | Pipe-uri:  0 | Scor RL:   3.1 |
    Epsilon: 0.010
```

D. Evaluare finală - "multiple runs" (20 de episoade, ϵ)

Scop: măsurarea performanței reale a politicii învățate, fără explorare, pe mai multe episoade independente.

- **Setup evaluare 1:**
  - 2000 de episoade la antrenare
  - 20 episoade consecutive
  - explorare dezactivată: $\epsilon = 0$
  - nu se mai actualizează rețeaua

  Rezultate obținute:

```
=== EVALUARE (multiple runs, epsilon = 0) ===
Eval Episod 01     Pipe-uri: 5
Eval Episod 02     Pipe-uri: 20
Eval Episod 03     Pipe-uri: 3
Eval Episod 04     Pipe-uri: 2
Eval Episod 05     Pipe-uri: 2
Eval Episod 06     Pipe-uri: 3
Eval Episod 07     Pipe-uri: 5
Eval Episod 08     Pipe-uri: 21
Eval Episod 09     Pipe-uri: 6
Eval Episod 10     Pipe-uri: 19
Eval Episod 11     Pipe-uri: 3
Eval Episod 12     Pipe-uri: 0
Eval Episod 13     Pipe-uri: 24
Eval Episod 14     Pipe-uri: 1
Eval Episod 15     Pipe-uri: 16
Eval Episod 16     Pipe-uri: 9
Eval Episod 17     Pipe-uri: 4
Eval Episod 18     Pipe-uri: 14
Eval Episod 19     Pipe-uri: 2
Eval Episod 20     Pipe-uri: 0
```

```
=== REZULTATE FINALE ===
Mean pipes   : 7.95
Std deviation: 7.71
Best run     : 24
Worst run    : 0
```

- **Setup evaluare 2:**
  - 5000 de episoade la antrenare
  - 20 episoade consecutive
  - explorare dezactivată: $\epsilon = 0$
  - nu se mai actualizează rețeaua

  Rezultate obținute:

```
=== EVALUARE (multiple runs, epsilon = 0) ===
Eval Episod 01     Pipe-uri: 8
Eval Episod 02     Pipe-uri: 8
Eval Episod 03     Pipe-uri: 20
Eval Episod 04     Pipe-uri: 2
Eval Episod 05     Pipe-uri: 13
Eval Episod 06     Pipe-uri: 1
Eval Episod 07     Pipe-uri: 26
Eval Episod 08     Pipe-uri: 14
Eval Episod 09     Pipe-uri: 25
Eval Episod 10     Pipe-uri: 8
Eval Episod 11     Pipe-uri: 3
Eval Episod 12     Pipe-uri: 4
Eval Episod 13     Pipe-uri: 33
Eval Episod 14     Pipe-uri: 11
Eval Episod 15     Pipe-uri: 15
Eval Episod 16     Pipe-uri: 20
Eval Episod 17     Pipe-uri: 19
Eval Episod 18     Pipe-uri: 1
Eval Episod 19     Pipe-uri: 2
Eval Episod 20     Pipe-uri: 1

=== REZULTATE FINALE ===
Mean pipes   : 11.70
Std deviation: 9.36
Best run     : 33
Worst run    : 1
```

VIII. CONCLUZII

În acest proiect am implementat cu succes un agent inteligent capabil să joace Flappy Bird folosind Deep Reinforcement Learning.
Contribuții principale:

- implementare completă DQN cu CNN
- inplut exclusiv din pixeli
- replay buffer + epsilon greedy
- evaluare corectă prin multiple runs
- performanță competitivă

Agentul demonstrează capacitatea rețelei neuronale de a învăța comportamente complexe pornind doar de la observații vizuale brute.