Proiect Programarea Aplicațiilor de Simulare – Echipa Fivers

Documentație proiect

1. Descrierea jocului Flappy Bird:

1.1.Prezentare generala

Flappy Bird este un joc de tip side-scroller, creat pentru telefonul mobil.

Prima variantă a jocului a apărut în anul 2013 și a fost dezvoltată de Dong Nguyen prin compania sa de game development .GEARS Studios.

1.2 Scopul jocului

Scopul jocului Flappy Bird este să controlezi o pasăre, încercând să parcurgi o distanță cât mai mare, trecând de obstacole sub forma unor obiecte verticale cu o porțiune lipsă ce face posibilă trecerea caracterului.

1.3 Mecanica jocului

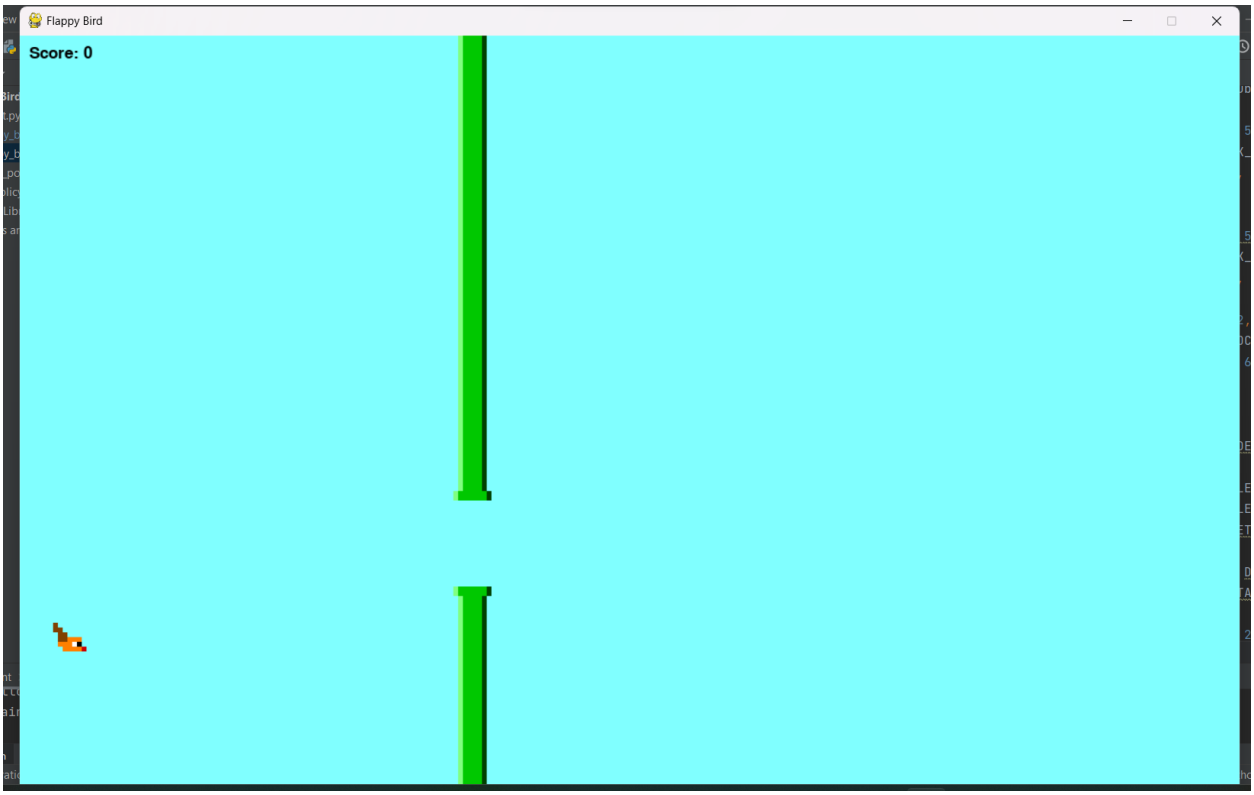
Mișcarea caracterului se realizează în mod automat, înaintând către următorul obstacol. Pasărea cade în mod constant către partea de jos a ecranului. Jucătorul poate controla pasărea apăsând pe ecran, astfel modificând poziția păsării în sus. Scopul este ca jucătorul să poziționeze pasărea cât mai bine încât aceasta să treacă printre obstacole, obținând un scor cât mai mare.

1.4 Obstacole

Obstacolele din joc sunt reprezentate de niste tuburi de culoare verde, poziționate pe verticală și care prezintă o porțiune lipsă prin care se poate face trecerea pasării.

1.5 Punctaj & Sfârșitul jocului

Câte un punct este acumulat pentru fiecare obstacol depășit. Daca unul din obstacole este atins, jocul se încheie și punctajul redevine 0.



2. Implementarea jocului folosind pygame

Pentru implementarea grafică a jocului am optat pentru utilizarea modulului *pygame* ce ne oferă facilitatea de creare a graficii 2D in cadrul limbajului python.

Pentru a ne asigura că proiectul rulează, folosiți comanda '**pip install pygame'** la prima rulare a proiectului în cazul în care modului pygame lipseste din environment.

Comenzile jocului:

- Săgeată sus: pasărea se mișcă în sus
- Săgeată jos: pasărea se mișcă în jos (față de jocul original am adăugat și optiunea de a te deplasa în jos la apăsarea unei taste)

La începerea jocului, pasărea se află în mijlocul ecranului pe verticală, în partea stângă,iar obiectele apar din partea dreaptă, având porțiunea lipsă la înălțime aleatoare. Trecerea prin fiecare obiect îți acordă +1 punct, iar în cazul unei coliziuni jocul se termină.

Pentru implementare am ales să structurăm codul după cum urmează:

1. Am definit codul RGB al culorilor pe care le folosim precum și constantele BLOCK_SIZE și DETAILED_BLOCK_SIZE care ne ajuta la randarea caracterului și a obstacolelor.

```
# RGB Colors
WHITE = (255, 255, 255)
RED = (200, 0, 0)
BLACK = (0, 0, 0)
BLUE = (128, 255, 255)
GREEN = (0, 200, 0)
LIGHT_GREEN = (128, 255, 128)
DARK_GREEN = (0, 64, 0)
ORANGE = (255, 128, 0)
BROWN = (128, 64, 0)

BLOCK_SIZE = 10
DETAILED_BLOCK_SIZE = 5
SPEED = 20
```

2. Clasa FlappyBirdGame care înglobează toată logica jocului și a randării obiectelor în scenă. În constructorul acestei clase generam valorile de început de joc precum, dimensiunile ferestrei, poziția inițială a caracterului, scorul, timerele pentru fiecare acțiune și pentru apariția obstacolelor și poziția acestora.

```python
class FlappyBirdGameAI:

    def __init__(self, w=1280, h=800):
        self.w = w
        self.h = h
        self.display = pygame.display.set_mode((self.w, self.h))
        pygame.display.set_caption('Flappy Bird')
        self.clock = pygame.time.Clock()

        self.reset()


    def reset(self):
        self.bird = Point(50, self.h // 2)
        self.score = 0
        self.tubes = []
        self.tube_timer = 0
        self.rise_timer = 0
        self.fall_timer = 0
        self.frame_count = 0
```

3. Metodele spaw_tube, move_tubes, remove_passed_tubes gestionează apariția și mișcarea obstacolelor și dispariția acestora atunci când ies din scenă.

```python
def spaw_tube(self):
    tube = Point(self.w, random.randint(BLOCK_SIZE * 4, self.h - 4 * BLOCK_SIZE))
    self.tubes.append(tube)



def move_tubes(self):
    for i, tube in enumerate(self.tubes):
        self.tubes[i] = tube._replace(x=tube.x - SPEED)



def remove_passed_tubes(self):
    counter = 0
    for i, tube in enumerate(self.tubes):
        if tube.x < 0:
            self.tubes.pop(i)
            counter += 1

    self.score += counter

    return counter
```

4. Metoda draw_tubes se ocupă de randarea în scenă a obstacolelor.

```python
def draw_tubes(self):
    for tube in self.tubes:
        pygame.draw.rect(self.display, LIGHT_GREEN, (tube.x, 0, BLOCK_SIZE // 2, tube.y - BLOCK_SIZE * 5))
        pygame.draw.rect(self.display, GREEN, (tube.x + BLOCK_SIZE // 2, 0, BLOCK_SIZE * 2, tube.y - BLOCK_SIZE * 5))
        pygame.draw.rect(self.display, DARK_GREEN, (tube.x + BLOCK_SIZE * 2.5, 0, BLOCK_SIZE // 2, tube.y - BLOCK_SIZE * 5))

        pygame.draw.rect(self.display, LIGHT_GREEN, (tube.x - BLOCK_SIZE // 2, tube.y - BLOCK_SIZE * 5, BLOCK_SIZE // 2, BLOCK_SIZE))
        pygame.draw.rect(self.display, GREEN, (tube.x, tube.y - BLOCK_SIZE * 5, BLOCK_SIZE * 3, BLOCK_SIZE))
        pygame.draw.rect(self.display, DARK_GREEN, (tube.x + BLOCK_SIZE * 3, tube.y - BLOCK_SIZE * 5, BLOCK_SIZE // 2, BLOCK_SIZE))


        pygame.draw.rect(self.display, LIGHT_GREEN, (tube.x - BLOCK_SIZE // 2, tube.y + BLOCK_SIZE * 5, BLOCK_SIZE // 2, BLOCK_SIZE))
        pygame.draw.rect(self.display, GREEN, (tube.x, tube.y + BLOCK_SIZE * 5, BLOCK_SIZE * 3, BLOCK_SIZE))
        pygame.draw.rect(self.display, DARK_GREEN, (tube.x + BLOCK_SIZE * 3, tube.y + BLOCK_SIZE * 5, BLOCK_SIZE // 2, BLOCK_SIZE))

        pygame.draw.rect(self.display, LIGHT_GREEN, (tube.x, tube.y + BLOCK_SIZE * 6, BLOCK_SIZE // 2, self.h))
        pygame.draw.rect(self.display, GREEN, (tube.x + BLOCK_SIZE // 2, tube.y + BLOCK_SIZE * 6, BLOCK_SIZE * 2, self.h))
        pygame.draw.rect(self.display, DARK_GREEN, (tube.x + BLOCK_SIZE * 2.5, tube.y + BLOCK_SIZE * 6, BLOCK_SIZE // 2, self.h))
```

5. Metoda draw_bird se ocupă de randarea în scenă a păsării, în funcție de acțiunea pe care jucătorul o alege. Avem următoarele cazuri posibile: simple, up și down.

```
def draw_bird(self, mode="simple"):
    if mode == "simple":
        pygame.draw.rect(self.display, ORANGE, (self.bird.x, self.bird.y - DETAILED_BLOCK_SIZE, 3 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, ORANGE, (self.bird.x, self.bird.y, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, WHITE, (self.bird.x + DETAILED_BLOCK_SIZE, self.bird.y, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, BLACK, (self.bird.x + 2 * DETAILED_BLOCK_SIZE, self.bird.y, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, ORANGE, (self.bird.x - 2 * DETAILED_BLOCK_SIZE, self.bird.y + DETAILED_BLOCK_SIZE, 5 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, RED, (self.bird.x + 3 * DETAILED_BLOCK_SIZE, self.bird.y + DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, ORANGE, (self.bird.x - 3 * DETAILED_BLOCK_SIZE, self.bird.y + 2 * DETAILED_BLOCK_SIZE, 6 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, ORANGE, (self.bird.x - 2 * DETAILED_BLOCK_SIZE, self.bird.y + 3 * DETAILED_BLOCK_SIZE, 4 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, ORANGE, (self.bird.x - DETAILED_BLOCK_SIZE, self.bird.y + 4 * DETAILED_BLOCK_SIZE, 2 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
```

```
    if mode == "up":
        pygame.draw.rect(self.display, ORANGE, (self.bird.x, self.bird.y - DETAILED_BLOCK_SIZE, 3 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, BROWN, (self.bird.x - 3 * DETAILED_BLOCK_SIZE, self.bird.y - DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, ORANGE, (self.bird.x, self.bird.y, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, WHITE, (self.bird.x + DETAILED_BLOCK_SIZE, self.bird.y, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, BLACK, (self.bird.x + 2 * DETAILED_BLOCK_SIZE, self.bird.y, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, BROWN, (self.bird.x - 3 * DETAILED_BLOCK_SIZE, self.bird.y, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, BROWN, (self.bird.x - 3 * DETAILED_BLOCK_SIZE, self.bird.y + DETAILED_BLOCK_SIZE, 2 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, ORANGE, (self.bird.x, self.bird.y + DETAILED_BLOCK_SIZE, 3 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, RED, (self.bird.x + 3 * DETAILED_BLOCK_SIZE, self.bird.y + DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, BROWN, (self.bird.x - 3 * DETAILED_BLOCK_SIZE, self.bird.y + 2 * DETAILED_BLOCK_SIZE, 3 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, ORANGE, (self.bird.x, self.bird.y + 2 * DETAILED_BLOCK_SIZE, 3 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, BROWN, (self.bird.x - 2 * DETAILED_BLOCK_SIZE, self.bird.y + 3 * DETAILED_BLOCK_SIZE, 2 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, ORANGE, (self.bird.x, self.bird.y + 3 * DETAILED_BLOCK_SIZE, 2 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, ORANGE, (self.bird.x - DETAILED_BLOCK_SIZE, self.bird.y + 4 * DETAILED_BLOCK_SIZE, 2 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
```

```
    if mode == "down":
        pygame.draw.rect(self.display, BROWN, (self.bird.x - 3 * DETAILED_BLOCK_SIZE, self.bird.y - DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, BROWN, (self.bird.x - 3 * DETAILED_BLOCK_SIZE, self.bird.y, 2 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, BROWN, (self.bird.x - 2 * DETAILED_BLOCK_SIZE, self.bird.y + DETAILED_BLOCK_SIZE, 2 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, BROWN, (self.bird.x - 2 * DETAILED_BLOCK_SIZE, self.bird.y + 2 * DETAILED_BLOCK_SIZE, 2 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, ORANGE, (self.bird.x, self.bird.y + 2 * DETAILED_BLOCK_SIZE, 3 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, ORANGE, (self.bird.x - 2 * DETAILED_BLOCK_SIZE, self.bird.y + 3 * DETAILED_BLOCK_SIZE, 3 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, WHITE, (self.bird.x + DETAILED_BLOCK_SIZE, self.bird.y + 3 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, BLACK, (self.bird.x + 2 * DETAILED_BLOCK_SIZE, self.bird.y + 3 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))

        pygame.draw.rect(self.display, ORANGE, (self.bird.x - 1 * DETAILED_BLOCK_SIZE, self.bird.y + 4 * DETAILED_BLOCK_SIZE, 4 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
        pygame.draw.rect(self.display, RED, (self.bird.x + 3 * DETAILED_BLOCK_SIZE, self.bird.y + 4 * DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE, DETAILED_BLOCK_SIZE))
```

6. Metoda update_ui folosește cele doua funcții anterioare pentru a randa fiecare scenă la fiecare frame.

```python
def _update_ui(self, mode="simple"):
    self.display.fill(BLUE)

    self.draw_bird(mode=mode)

    self.draw_tubes()

    text = "Score: " + str(self.score)

    label = font.render(text, 1, BLACK)
    self.display.blit(label, (10, 10))
```

7. Metoda check_collision verifică condiția de sfârșit a jocului, mai exact dacă pasărea atinge cel mai apropiat obstacol.

```python
def check_collision(self):
    if len(self.tubes) == 0:
        return False

    closest_tube = self.tubes[0]

    if self.bird.x + BLOCK_SIZE >= closest_tube.x and self.bird.x <= closest_tube.x + BLOCK_SIZE * 3:
        if self.bird.y <= closest_tube.y - BLOCK_SIZE * 5 or self.bird.y + BLOCK_SIZE >= closest_tube.y + BLOCK_SIZE * 5:
            return True

    if not 0 <= self.bird.y <= self.h - BLOCK_SIZE:
        return True

    return False
```

8. Metoda _move_bird modifica pozitia pasarii in functie de parametrul *action*, stabilit de choose_action in agent.py.

```
def _move_bird(self, action):
    #print("ACTION" + str(action))
    if action == Action.JUMP.value:
        #print("JUMP")
        self.bird = self.bird._replace(y= self.bird.y - BLOCK_SIZE)
        self.rise_timer = 6
        self.fall_timer = 0
    elif action == Action.DIVE.value:
        #print("DIVE")
        self.bird = self.bird._replace(y= self.bird.y + BLOCK_SIZE)
        self.fall_timer = 6
        self.rise_timer = 0

    #print("NO ACTION")
    if self.rise_timer > 0:
        self.bird = self.bird._replace(y= self.bird.y - BLOCK_SIZE)
    elif self.fall_timer > 0:
        self.bird = self.bird._replace(y= self.bird.y + BLOCK_SIZE)
    else:
        self.bird = self.bird._replace(y= self.bird.y + BLOCK_SIZE // 2)
```

9. Metoda play updatează timerele și folosește funcțiile de mai sus pentru a implementa logica jocului în funcție de acțiunea primita din agent.py.

```python
def play(self, action):
    self.frame_count += 1
    self.tube_timer += 1
    self.rise_timer -= 1
    self.fall_timer -= 1

    if self.tube_timer == 50:
        self.spaw_tube()
        self.tube_timer = 0

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()

    self._move_bird(action)

    reward = 0
    game_over = False
    if self.check_collision():
        game_over = True
        reward = -10
        return reward, self.score, game_over
```

```
        self.move_tubes()
        reward = 10 * self.remove_passed_tubes()

        self.display.fill(BLACK)

        mode = "simple"

        if self.rise_timer > 0:
            mode = "up"
        elif self.fall_timer > 0:
            mode = "down"

        self._update_ui(mode=mode)
        self.clock.tick(SPEED * 2)
        pygame.display.update()
        return reward, self.score, game_over
```

3)Agent.py

Acest cod implementează un agent de învățare reinforcement learning folosind algoritmul SARSA si TD pentru a juca jocul Flappy Bird. Agentul învață o politică pentru a lua decizii în funcție de starea curentă a jocului. Mai jos sunt explicate functiile:

- Constructor (__init__): configurează setările de învățare ale agentului, inclusiv politica, tabelul Q (valori de acțiune de stare) și parametrii de învățare precum alpha (rata de învățare) și gamma (factor de reducere).
- get_dictionary_for_tuple_to_index_conversion: creează un dicționar cu scopul de a mapa tupluri de stare la indici din tabelul Q
- save_policy and load_policy: Salvați și încărcați tabelul Q învățat într-un fișier folosind funcțiile save_policy și load_policy.
- get_state: Returnează indexul de stare și tuplul de stare după extragerea caracteristicilor de stare ale jocului (înălțimea păsărilor, locațiile tuburilor etc.).
- build_policy: Folosind valorile Q învățate, politica este construită.

- choose_action : Alegeți o acțiune aplicând politica epsilon-greedy. Se investighează o acțiune aleatorie cu epsilon de probabilitate; altfel, profită de politica existentă.
- temporal_difference_update: Această funcție aplică regula de actualizare a diferenței temporale pentru a actualiza valoarea Q.
- temporal_difference_train: folosește diferența temporală pentru a antrena agentul-
- update_q_sarsa: Actualizează valoarea Q folosind regula SARSA de actualizare.
- sarsa_train: foloseste SARSA. Asemanator cu învățarea prin diferențe temporale, dar cu actualizări SARSA.
- test_policy: evaluează cunoștințele politice jucând jocuri și adunând dovezi.
- Bloc principal:

  Creează o instanță a jocului Flappy Bird (joc) și o instanță a agentului (agent).

  Realizeaza TD și salvarea politicilor.

  Realizeaza SARSA, salvează politica învățată, o reîncarcă, elaborează politica și apoi o testează pe mai multe episoade.

Algoritmul SARSA:

- funcția update_q_sarsa:

Această funcție actualizează valorile Q pe baza regulii de actualizare SARSA.

Ia ca intrare starea curentă, acțiunea curentă, recompensa obținută, starea următoare și acțiunea următoare.

current_estimate este estimarea curentă a valorii Q pentru starea și acțiunea curentă.

next_estimate este estimarea valorii Q pentru următoarea stare și următoarea acțiune.

Valoarea Q pentru perechea stare-acțiune actuală este actualizată folosind formula SARSA:

Q[stare][acțiune] += alfa * (recompensă + gamma * estimarea_următoarea - estimarea_actuală).

alfa este rata de învățare, iar gamma este factorul de reducere.

- Funcția sarsa_train:

Această funcție realizează instruirea agentului folosind SARSA.

Este nevoie de mediul de joc, numărul de episoade și parametrii opționali.

epsilon este parametrul de compromis între explorare și exploatare, care începe sus și se degradează pe parcursul episoadelor.

Inițializează o listă goală de scoruri pentru a ține evidența scorurilor obținute în fiecare episod.

Folosește matplotlib pentru vizualizarea opțională a progresului antrenamentului dacă verbose este setat la True.

Valoarea epsilon este diminuată în timpul episoadelor pentru a reduce explorarea pe măsură ce agentul învață.

Dacă verbose este activat, se actualizează și grafice scorul în timpul antrenamentului.

For loop:

Bucla de antrenament rulează pentru numărul specificat de episoade.

În fiecare episod, jocul este resetat, iar agentul interacționează cu mediul.

Valorile Q sunt actualizate folosind SARSA după fiecare actiune.

Scorul episodului este printat și adăugat la lista de scoruri.

Epsilon este degradat dacă este încă peste valoarea minimă specificată.

Dacă verbose este setat la True, codul folosește matplotlib pentru a reprezenta dinamic scorurile în timpul antrenamentului.

Algoritmul TD

- funcția temporal_difference_update:

Această funcție actualizează valorile Q utilizând regula de actualizare a Diferenței temporale.

Ia starea curentă, acțiunea curentă, recompensa obținută și următoarea stare ca intrare.

Valoarea Q pentru perechea stare-acțiune curentă este actualizată folosind formula TD:

Q[stare, acțiune] += alfa * (recompensă + gamma * max(Q[stare_următoare, :]) - Q[stare, acțiune]).

alfa este rata de învățare, iar gamma este factorul de reducere.

Utilizează np.max al lui NumPy pentru a găsi valoarea Q maximă pentru următoarea stare.

- funcția temporal_difference_train:

Această funcție realizează antrenamentul agentului utilizând învățarea Diferenței temporale.

Este nevoie de mediul de joc, numărul de episoade și parametrii opționali

epsilon este parametrul de compromis între explorare și exploatare, care începe sus și se degradează pe parcursul episoadelor.

Inițializează o listă goală de scoruri pentru a ține evidența scorurilor obținute în fiecare episod.

Folosește matplotlib pentru vizualizarea opțională a progresului antrenamentului dacă verbose este setat la True.

For loop:

> În cadrul fiecărui episod, agentul alege acțiuni, joacă jocul, actualizează valorile Q folosind TD și urmărește scorul.
>
> Valoarea epsilon este diminuată în timpul episoadelor pentru a reduce explorarea pe măsură ce agentul învață.
>
> Dacă verbose este activat, se actualizează și grafice scorul în timpul antrenamentului.
>
> Bucla de antrenament:
>
> Bucla de antrenament rulează pentru numărul specificat de episoade.
>
> În fiecare episod, jocul este resetat, iar agentul interacționează cu mediul.
>
> Valorile Q sunt actualizate folosind Diferența temporală după fiecare acțiune întreprinsă în mediu.
>
> Scorul episodului este tipărit și adăugat la lista de scoruri.
>
> Epsilon este degradat dacă este încă peste valoarea minimă specificată.

Dacă verbose este setat la True, codul folosește matplotlib pentru a reprezenta dinamic scorurile în timpul antrenamentului.

4) AI Environment

Pentru a crea mediul in care sa joace agentul am folosit o clasa separata, FlappyBirdGameAI, care este in mare parte similara cu varianta care poate fi jucata de om.

Printre similaritati sunt randarea grafica a jocului (pasarea, tuburile, backgroundul, afisarea scorului, etc) care se realizeaza prin aceleasi metode:

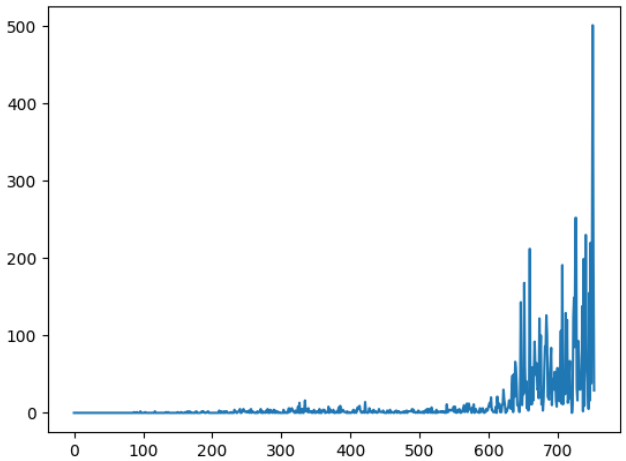
- Spawn_tube, move_tubes, draw_tubes, remove_passed_tubes la fel ca in cealalta varianta misca tuburile si creeaza in timp tuburi noi in timp ce cele trecute sunt de-spawnate cand ajung in stanga ecranului
- Draw_bird afiseaza pasarea in sine in functie daca sare (up), coboara (down) sau nu face nimic (simple)
- Update_ui afiseaza efectiv ce se intampla in starea curenta a jocului (deseneaza pasarea, tuburile, backgroundul) si afiseaza scorul

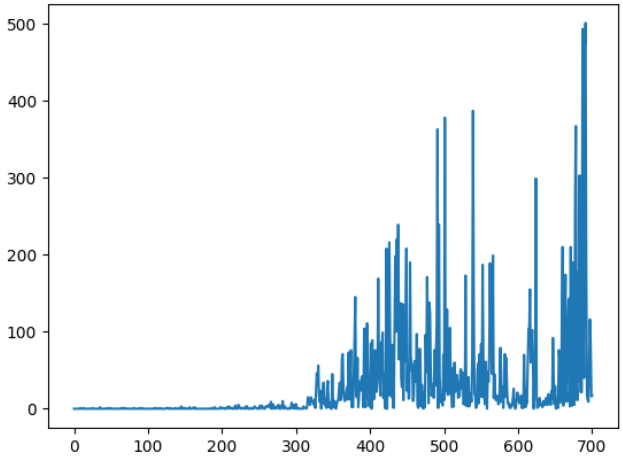
Pe langa aceste metode, in varianta jucata de AI mai este adaugata metoda _move_bird pe care o acceseaza agentul si ii permite sa miste pasarea ca atunci cand omul ar apasa tastele sus/jos.

Agentul este antrenat separat in clasa Agent prin SARSA si Temporal Difference, folosindu-ne de faptul ca in joc exista un numar finit de stari posibile care este direct proportional cu cat de mare sau mica este discretizarea distantei fata de urmatorul tub. Prin testare, am gasit ca cel mai bun compromis intre precizie si performanta ar fi 20.

Deci, in cazul nostru, spatiul total de stari este de 3 * 4 * 20 = 240 de stari posibile ale pasarii fata de urmatorul tub. 3-ul reprezinta 3 posibilitati, daca pasarea se va lovi in partea de sus a tubului, in partea de jos sau daca va trece ok, iar 4-ul reprezinta inaltimea fata de tub la care se afla, discretizata in 4 blocuri – deasupra, dedesubt, iar pentru mijloc, e impartit in doua blocuri, upper si lower.

Agentul este antrenat cu un gamma foarte aproape de 1, 0.995, punand un accent cat mai mare pe reward pe termen lung. Learning rate-ul este moderat spre mare, 0.1, care ajuta sa invete mai repede, dar poate crea fluctuatii destul de mari si frecvente in procesul de invatare.





Tabelele arata performanta pentru metodele SARSA si Temporal Difference – numarul generatiei pe Ox si scorul atins pe Oy.

Pentru primele cateva sute de generatii, scorul atins era sub 50, cu fluctuatii aproape aleatoare, incepand abia de la generatia 600 pentru SARSA, respectiv 300 pentru TD si mai departe sa creasca mult in scor, dar chiar si asa, cu tot cu fluctuatii mari, dupa cum se vede in grafic.

In mare, acesti parametri, desi poate nu 100% optimi, au fost mai mult decat suficient de buni, dat fiind ca agentul, in ambele metode, de la generatia 700 incolo, ceea ce ia cateva secunde de antrenare pe o placa video performanta (fara interfata grafica), ajunge la un scor de peste 500, invingand performanta average a omului cu mult, acesta fiind scopul nostru initial.