Snake

Zaharia Robert Jan

November 6, 2022

1 Introduction

1.1 Context

In cadrul acestui proiect se doreste implementarea unui Ai pentru jocul "Snake". Snake este un joc simplist in care tot ce trebuie sa faci este sa mergi in 4 directii si mananci punctele rosii de pe harta, crescand in lungime. Modurile de a pierde sunt fie te lovesti de perete fie de propriul corp.

1.2 Specificatii

Pentru a crea acest AI vom folosi 2 algoritmi de cautare: A* si DFS, care vor face sarpele sa ajung la obiectivul sau , marul, fara a iesi de pe harta sau a se lovi de el insasi. Pentru a ilustra jocul vom folosi un framework care va folosi pygame pentru grafica iar ca si limbaj Python.

1.3 Rulare

Pentru A* se va deschide terminal in fisierul cu proiect si se va rula in comanda python3 agent2.py . Iar pentru DFS se va rula comanda python3 agent.py

2 Algoritmii

Inainte de a aplica ambii algoritmi trebuie mai intai sa cream 3 functii: IsgoalState, getSuccesors,getStartstate

2.1 A*

A* este un algoritm de cautare care se bazează pe diferite euristici pentru a obține optimalitate și pentru a completa problema. Când un algoritm de căutare este optim, înseamnă că este garantată găsirea celei mai bune soluții posibile, în cazul nostru cea mai scurtă cale către marul rosul al Snake-ului. Cea mai buna solutie posibila este aleasa prin maparea intregii harti cu o valoare aleasa in functie de distanta fata de pozitia de start a sarpelui cat si pozitia marului. In cadrul A*-ului implementat am folosit euristica manhattan pentru a determina distanta de la pozitia snake-ului pana la mar.

2.2 DFS

DFS este un algoritm pentru parcurgerea sau căutarea structurilor de date arborescente sau grafice. Algoritmul începe de la nodul rădăcină (selectând un nod arbitrar ca nod rădăcină în cazul unui grafic) și explorează cât mai departe posibil de-a lungul fiecărei ramuri înainte de a reveni. In cadrul acestui algortim este necesară o stiva pentru a ține evidența nodurilor descoperite până acum de-a lungul unei ramuri specificate, ceea ce ajută la întoarcerea graficului. Din pacate DFS este un algoritm inferior A*.

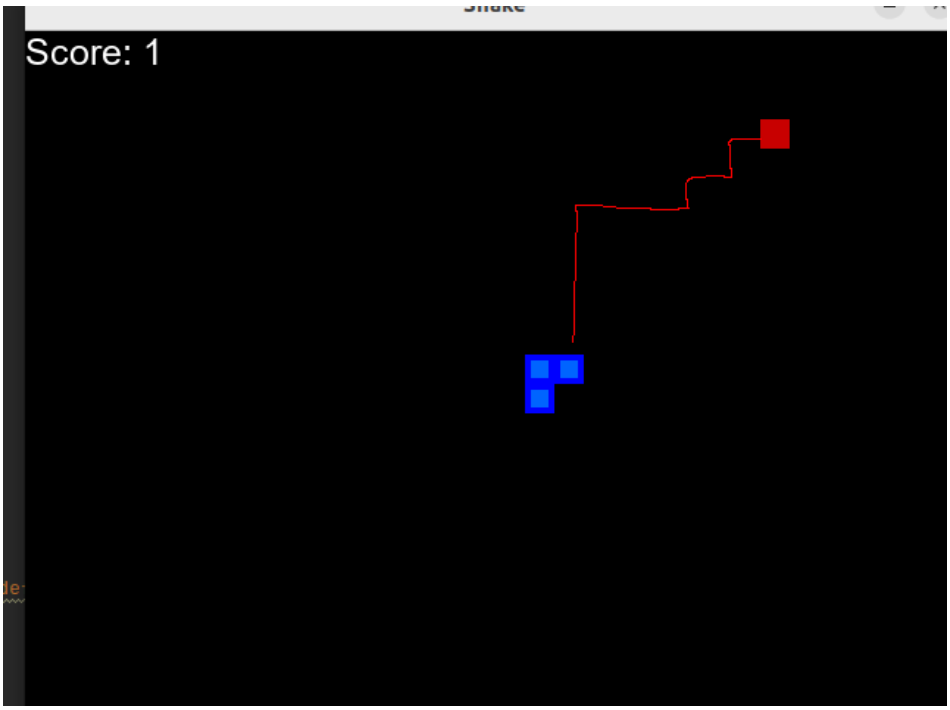

Figure 1: Traseul snake-ului folosind A*.

3 Testare

Mai intai vom testa A*. A* are un stil destul de simplu intrucat acesta merge cat mai dreapt pana cand se apropie de mar apoi incepe sa mearga pe "diagonala" pana ajunge la mar. Cand sarpele ajunge la o marime considerabil incepe sa mearga cat mai aproape de corpul sau lasand in schimba sansa destul de mare sa fie de propria coada.

Scorul maxim atins de Snake folosind A* este 74.

Dfs este arata putin mai ciudat. in loc sa incerce sa mearga oX sau pe oY pana cand vede intr-o directie dreapta marul el alege sa tot mearga pe "diagonala" pana isi atinge scopul. Desi este mai simplu de implementat si cu mai putin cod DFS a obtinut un scor destul de bun problema sa principala fiind ca reusea foarte des sa se blocheze.

Totusi DFS a reusit sa obtina un scor de 57

4 Comparatia algoritmilor

In cadrul capitolului anterior se pot vedea diferente subtile intre A* si DFS dar din pacate A* pare sa fie mai eficient prin implementarea sa. Insa pentru rezolvarea acestui Joc de snake se pot folosi multiplii algoritmi diferiti A* nefiind in toate cazurile cel mai bun. Prima varianta de rezolvare alternativa ar fi realizarea unui ciclu hamiltonian. Aceasta solutie desi plictisitoare este cea mai sigura si cea are asigura victoria jucatorului. Un mare dezavantaj al acestui algoritm este timpul mare de rezolvare al problemei. In schimb daca nu iei timpul de rezolvare este cea mai buna solutie

O alta incercare pe care am intalnit-o este realizarea unui Snake care reuseste sa invete singur prin Deep Q learning. Multa lume ar considera acesta un adevarat Ai deoarece invata sa joace singur. Aceasta metoda este una destul de eficienta dar putin mai inceata deoarece Snake-ul trebuie sa invete multe miscari pana incepe sa acumuleze puncte

5 Concluzie

Snake pare un joc simplist insa crearea unui Ai pentru el sigur nu este usor. Cea mai mare provocare este incercarea de al face sa nu se mai loveasca de propria coada insa cu sacrificarea eficientei sau poate

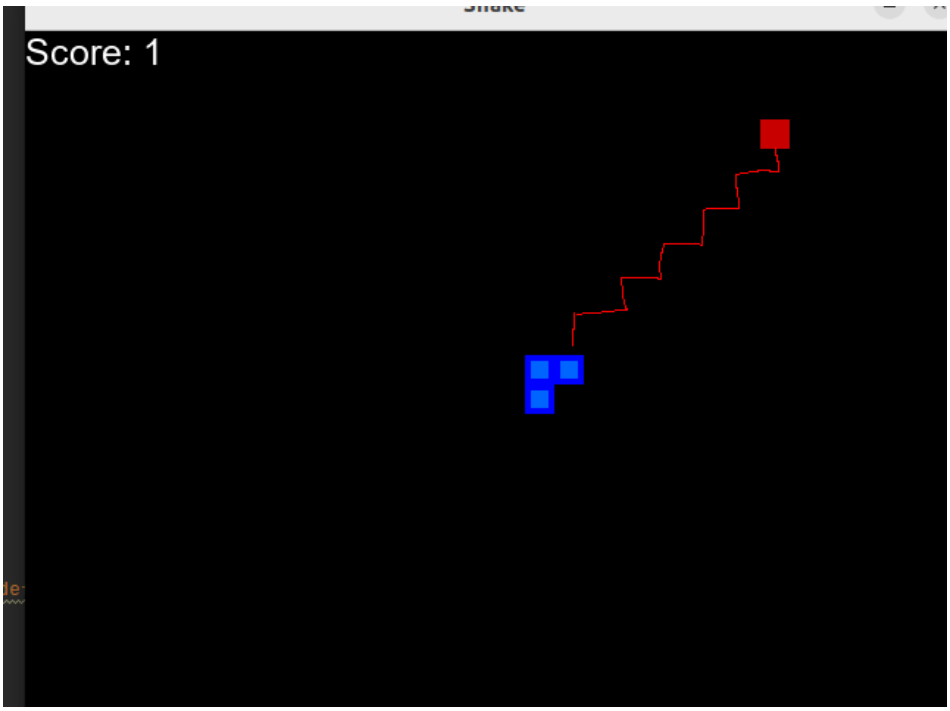

Figure 2: Traseul snake-ului folosind Dfs.

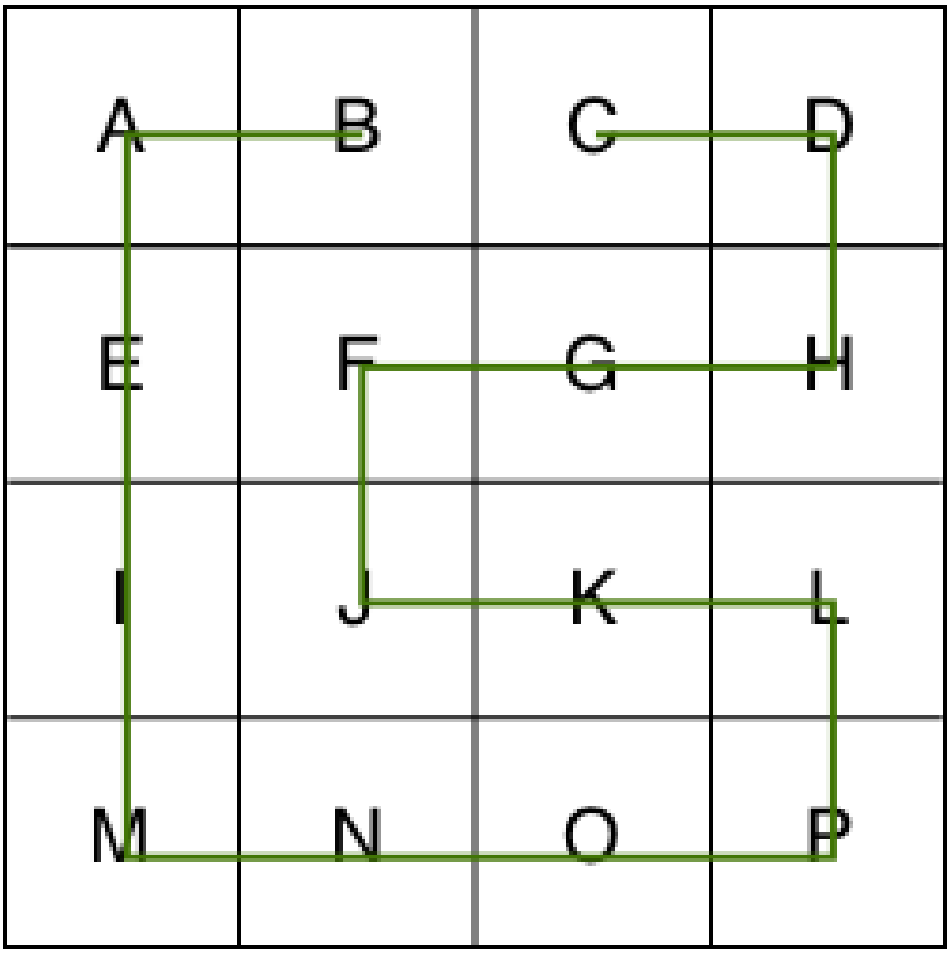

Figure 3: Traseul snake-ului Cu un ciclu hamiltonian.

```
def getSuccesors(self,current_pos):
        cost=0
        successors=[]
        possible_moves=[+BLOCK_SIZE,-BLOCK_SIZE]

        for move in possible_moves:
            nextx=current_pos[0]+move
            nexty=current_pos[1]
            point=Point(nextx,nexty)
            if point not in self.snake[1:] and point[0]<640 and point[0]>0:
                nextState=point
                if nextState!=current_pos:
                    if nextState not in successors:
                        dirr="Right"
                        if move == +BLOCK_SIZE:
                            dirr="Right"
                            cost= (self.euclideanCost(current_pos, self.food) / 2)
                        elif move == -BLOCK_SIZE:
                            dirr="Left"
                            cost= (self.euclideanCost(current_pos, self.food) / 2)
                        successors.append((nextState,dirr,cost))

        for move in possible_moves:
            nextx=current_pos[0]
            nexty=current_pos[1]+move
            point=Point(nextx,nexty)
            if point not in self.snake[1:] and point[1]<480 and point[1]>0:
                nextState=point
                if nextState!=current_pos:
                    if nextState not in successors:
                        dirr = "Up"
                        if move == +BLOCK_SIZE:
                            dirr = "Up"
                            cost= (self.euclideanCost(current_pos, self.food) / 2)
                        elif move == -BLOCK_SIZE:
                            dirr = "Down"
                            cost= (self.euclideanCost(current_pos, self.food) / 2)
                        successors.append((nextState,dirr,cost))
```

Figure 4: .

cu folosirea unui algoritm mai eficient se poate crea un AI care sa joace perfect snake

6 Cod

6.1 getSuccesor

FUnctie de get succesors returneaza toate pozitiile pe care le poate lua snake-ul tinand cont atat de pozitia snake-ului. Aceasta returneaza urmatoarea locatie, directia si costul acestei miscari. (fig. 4)

6.2 getStartState

Aceasta functii returneaza coordonatele capului snake-ului. (fig. 5)

```
def getStartState(self):
    return self.head
```

Figure 5: .

```
def isGoalState(self,current):
    if current[0]== self.food[0] and current[1]==self.food[1]:
        #print(current[0], current[1])
        return True
    else:
        return False
```

Figure 6: .

6.3 isGoalState

Aceasta functiie verifica daca capul snake-ului este pe mar(fig. 6)