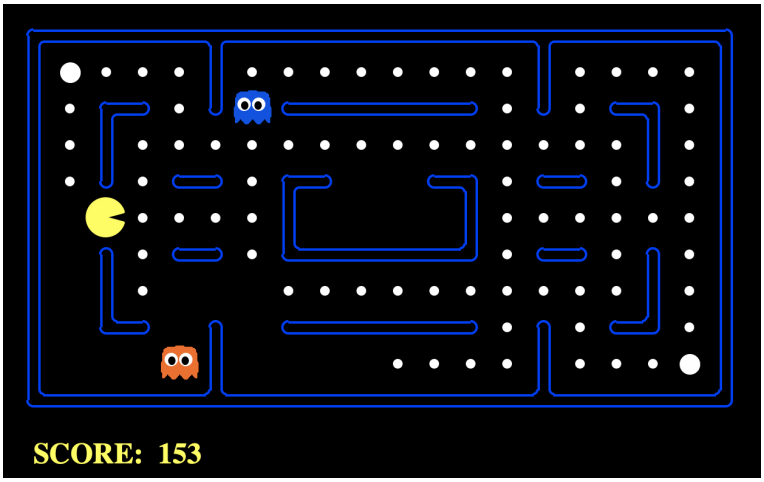
Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name: Chiș David, Cozma Victoria
Group: 30225
Email: victorya.cozma@gmail.com
chisdavid4321@gmail.com



Teaching Assistant: Roxana Szabo

Contents

1 Introducere **3**

2 Implementare **4**

3 Rezultate **7**

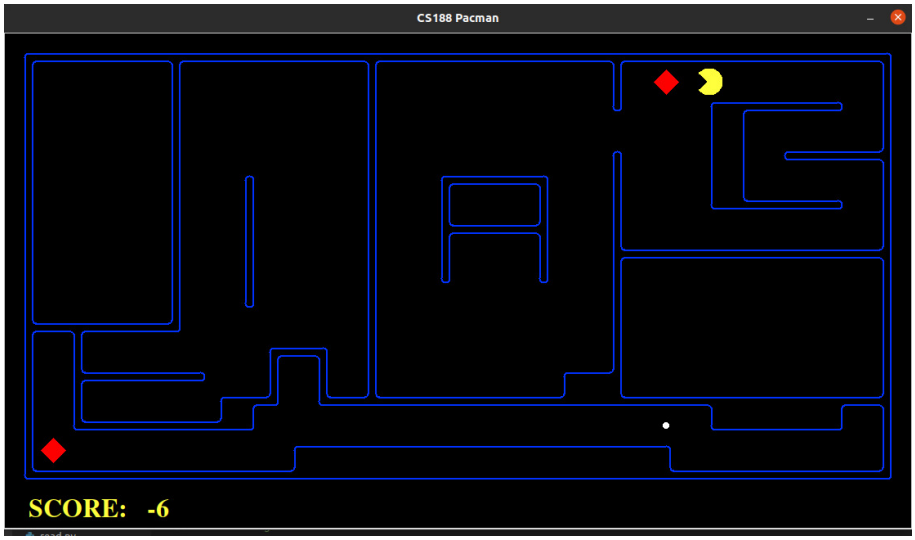
4 Concluzii **9**

5 Anexa **10**

Chapter 1

Introducere

Acest proiect are ca scop familiarizarea cu limbajul de programare Python si introducerea in Inteligenta Artificiala prin intermediul framework-ului Pacman. Am urmarit implementarea corecta si eficienta a algoritmilor de baza (BFS,DFS, A*, UCS), care rezolva diverse probleme de cautare (cel mai scurt drum pana la graunte, Corners Problem, teleportare, etc), implementarea altor strategii neinformate de cautare(Depth Limited Search, Iterative Deepening Search) si extinderea framework-ului cu noi functionalitati. Am optat pentru introducerea portalelor. Aceasta optiune permite schimbarea coordonatelor pacman-ului intre 2 puncte(portale) din grid. Teleportarea are loc in cazul in care distanta dintre pozitia pacman-ului si dot prin portal este mai mica decat drumul pe care ar merge acesta in absenta lor. Pentru rezolvarea cat mai eficienta a problemei, am optat pentru introducerea euristicii FoodHeuristic, care presupune calcularea celui mai scurt drum de la o pozitie data pana la GoalState. Aceasta eurirstica este una consistenta si presupune aplicarea algoritmului BFS, care returneaza cea mai scurta cale intre 2 noduri.



Chapter 2

Implementare

Vom prezenta pe scurt algoritmii implementati in proiectul nostru. Toti algoritmii urmarec acelasi scop: gasirea cat mai rapida a dot-urilor, diferenta dintre ei fiind modul de implementare a structurilor de stocare si gestionare a succesorilor (stiva, coada, coada de prioritate).

1. Depth Limited Search

```
def depthLimitedSearch(problem):
    begin = problem.getStartState()
    c = problem.getStartState()
    visited = []
    # visited.append(begin)
    stiva = util.Queue()
    limit=1000
    depth=0
    stiva.push((begin , []))
    while not stiva.isEmpty() :
        if depth<=limit:
            pozitie, final = stiva.pop()

            if problem.isGoalState(pozitie):
                return final
            visited.append(pozitie)
            vecini = problem.getSuccessors(pozitie)
            for i in vecini:
                nod = i[0]
                if not nod in visited:
                    c = i[0]
                    directie = i[1]
                    stiva.push((nod, final + [directie]))
        depth+=1
    return final + [directie]
```

2. Iterative Deepening First Search

```
def IterativeDeepeningDepthFirstSearch(problem):
    stiva=util.Stack()
    limit=1
    from game import Directions
    while True:
        visited=[]
        startState=problem.getStartState()
        stiva.push((startState,[],0))

        currentState , actions ,TotalCost = stiva.pop()
        visited.append(currentState)


        while not problem.isGoalState(currentState):
            successors = problem.getSuccessors(currentState)
            for newState,newAction,newCost in successors:
                if (not newState in visited and TotalCost+newCost<=limit):
                    stiva.push((newState,actions+ [newAction],TotalCost+newCost))
                    visited.append(newState)
            if stiva.isEmpty():
                break
            currentState , actions ,TotalCost = stiva.pop()
        if problem.isGoalState(newState):
            return actions

        limit+=1
```

Pentru a implementa ideea cu portalele, am decis sa cream un nou agent in fisierul "searchAgents.py" si sa-l numim sugestiv "OurAStarFoodSearchAgent"

```
class OurAStarFoodSearchAgent(SearchAgent):
    "A SearchAgent for FoodSearchProblem using A* and your foodHeuristic"
    def __init__(self):

        self.searchFunction = lambda prob: search.astar(prob,foodHeuristic)
        self.searchType = OurProblem
```

Problema "OurProblem" pe care am definit-o este asemanatoare cu "AnyFoodSearchProblem", doar ca permite schimbarea coordonatelor pacman-ului odata cu trecerea acestuia prin portal. Dupa cum se poate observa, am adaugat o noua directie "DIRECTIONS.TELEPORT" si am preluat coordonatele pacman-ului si ale portalelor. Cand aceste coordonate coincid, are loc teleportarea pacman-ului de la un portal la altul.

```
def getSuccessors(self, state):
    "Returns successor states, the actions they require, and a cost of 1."
    successors = []
    from layout import *
    self._expanded += 1 # DO NOT CHANGE
    for direction in [Directions.NORTH,Directions.WEST, Directions.SOUTH, Directions.EAST,  Directions.TELEPORTER]:
        x,y = state[0]
        x1=layout.portal[0][0]
        y1=layout.portal[0][1]
        x2=layout.portal[1][0]
        y2=layout.portal[1][1]
        if( (x ==x1 and y ==y1 )or (x==x2 and y==y2)):
            dx, dy = Actions.directionToVector(direction)
            nextx, nexty = int(x + dx), int(y + dy)
            if not self.walls[nextx][nexty]:
                nextFood = state[1].copy()
                nextFood[nextx][nexty] = False
                successors.append( ( ((nextx, nexty), nextFood), direction, 1) )
        else:
            if direction != Directions.TELEPORTER:
                dx, dy = Actions.directionToVector(direction)
                nextx, nexty = int(x + dx), int(y + dy)
                if not self.walls[nextx][nexty]:
                    nextFood = state[1].copy()
                    nextFood[nextx][nexty] = False
                    successors.append( ( ((nextx, nexty), nextFood), direction, 1) )

    return successors
```

Preluarea coordonatelor portalelor si ale pacman-ului a reprezentat cea mai imprtanta parte din proiect. Pentru a schimba pozitia pacman-ului pe grid, am adaugat o noua directie si in fisierul "game.py". Intrucat fisierul "layuot.py" importa date din "game.py", legatura inversa de import nu a fost posibila. Astfel, am fost nevoiti sa cream un fisier nou "read.py", in care preluam din linia de comanda maze-ul folosit si citim coordonatele portalelor. Noile metode xP() si yP() returneaza diferenta acestor coordonate, adica noua pozitie a pacman-ului.

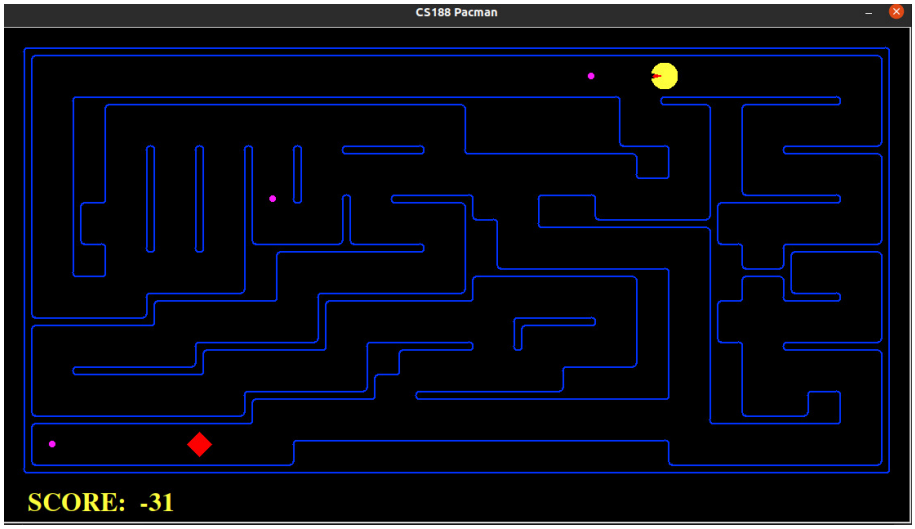
```
from read import Read
class Actions:
    def xP():

        f = Read(sys.argv[2])
        if f.portals == []:
            return 0
        print OurClass.pacmanX
        return f.portals[1][0]-f.portals[0][0]

    def yP():

        f = Read(sys.argv[2])
        if f.portals==[]:
            return 0
        return f.portals[1][1]-f.portals[0][1]

    _directions = {Directions.NORTH: (0, 1),
                   Directions.SOUTH: (0, -1),
                   Directions.EAST:  (1, 0),
                   Directions.WEST:  (-1, 0),
                   Directions.STOP:  (0, 0),
                #    Directions.TELEPORTER : (-21,-15)
                   Directions.TELEPORTER : (yP(),xP())
                   }

    _directionsAsList = _directions.items()
```

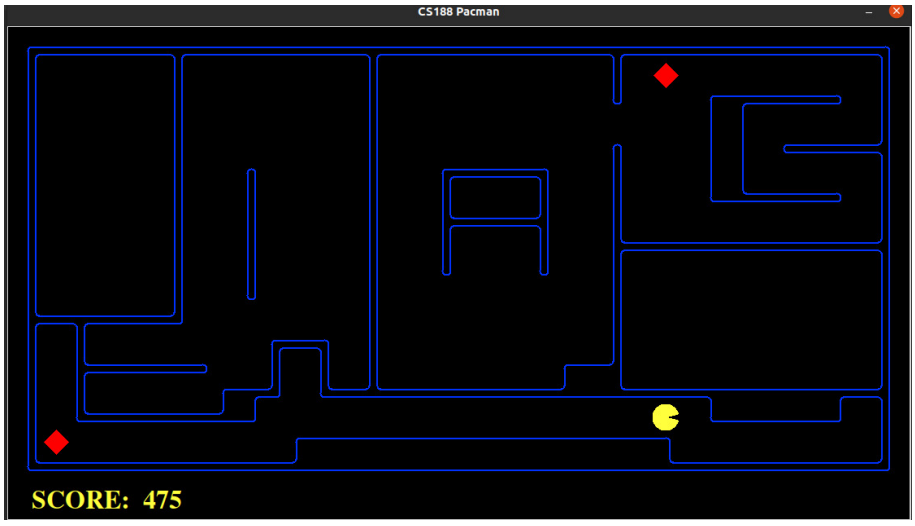
```
class Read:
    portals = []

    def __init__(self,file):
        self.file=file
        self.portals=[]
        self.citeste()

    def citeste(self):
        f = open("layouts/"+self.file+".lay","r")
        Lines = f.readlines()
        k=0
        nr_linii=0
        for line in Lines:
            nr_linii+=1
        # print nr_linii

        for line in Lines:
            for i in line:
                if i == 's':
                    self.portals.append((nr_linii-1-Lines.index(line),line.index(i)))
```

Chapter 3

Rezultate

Pentru a demonstra corectitudinea codului implementat, am creat 2 maze-uri, pe care am testat functionalitatea portalelor.





De asemenea, am facut o comparatie intre algoritmii implementati la laborator si a celor implementati de noi pe diferite griduri. Rezultatele pot fi urmarite in urmatorul tabel. Astfel, se poate observa ca pentru diferite configuratii ale gridului, algoritmii prezinta performante diferite. Se poate observa ca atunci cand maze-ul are mai multi pereti(MediumMaze), algoritmii implementati la laborator sunt mai eficienti. Pe de alta parte, exemplul gridului NewMaze2

demostreaza ca DLS si IDDFS expandeaza foarte putine noduri si sunt mai rapizi in cazul in care avem mai putini pereti. Desigur, numarul nodurilor expandate depinde si de pozitia initiala a pacman-ului si de pozitia boabelor de mancare.

MediumMaze

	Scor	Noduri Expandate	Timp (s)
BFS	442	245	0.2
DFS	442	245	0.2
DLS	442	241	0.0
A*	442	208	0.0
IDDFS	442	7657	0.1

NewMaze2

	Scor	Noduri Expandate	Timp (s)
BFS	442	245	1.7
DFS	442	245	1.8
DLS	442	241	1.7
A*	442	208	0.2
IDDFS	442	7657	0.1

Comenzi

```
python pacman.py -l newMaze1 -p OurAStarFoodSearchAgent
python pacman.py -l newMaze2 -p SearchAgent -a fn=iddfs
python pacman.py -l newMaze2 -p SearchAgent -a fn=dfs
python pacman.py -l newMaze2 -p SearchAgent -a fn=bfs
python pacman.py -l newMaze -p OurAStarFoodSearchAgent

python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
python pacman.py -l mediumMaze -p SearchAgent -a fn=dls
python pacman.py -l mediumMaze -p SearchAgent -a fn=iddfs
```

Chapter 4

Concluzii

Extinderea framework-ului Pacman realizata de noi se poate sumariza astfel:
• Am creat problema OurProblem, care foloseste directia "Direction.TELEPORT" pentru teleportarea pacmanului
• Am creat o clasa noua pentru preluarea coordonatelor portalelor din linia de comanda
• Am creat layout-uri noi pentru a ilustra cat mai bine ca pacman-ul gaseste cel mai scurt drum cu ajutorul teleportarii(daca aceasta este posibila)
• Am implementat alti 2 algoritmi de cautare(IDDFS si DLS) pentru a face comparatie intre eficienta algoritmilor

Acest proiect ne-a ajutat sa invatam sintaxa de baza a limbajului de programare Python, ne-am amintit de notiunile de baza de algoritmica si probleme de cautare. Framework-ul respectiv ne-a ajutat sa interactionam cu o interfata grafica si sa o modificam pe cat posibil.
Ca dezvoltare ulterioara a acestei idei de proiect se pot adauga mai multe portale, care sa ajute pacman-ul sa gaseasca cat mai repede mancarea. De asemenea, am putea adauga conditii suplimentare pentru a permite pacman-ului sa intre in portal (de exemplu, portalul nu se deschide decat dupa ce colecteaza cheia ce deschide portalul).

Chapter 5

Anexa

Cod propriu

```
def depthLimitedSearch(problem):
    begin = problem.getStartState()
    c = problem.getStartState()
    visited = []
    # visited.append(begin)
    stiva = util.Queue()
    limit=1000
    depth=0
    stiva.push((begin , []))
    while not stiva.isEmpty() :
        if depth<=limit:
            pozitie, final = stiva.pop()

            if problem.isGoalState(pozitie):
                return final
            visited.append(pozitie)
            vecini = problem.getSuccessors(pozitie)
            for i in vecini:
                nod = i[0]
                if not nod in visited:
                    c = i[0]
                    directie = i[1]
                    stiva.push((nod, final + [directie]))
        depth+=1
    return final + [directie]
```

```python
def IterativeDeepeningDepthFirstSearch(problem):
    stiva=util.Stack()
    limit=1
    from game import Directions
    while True:
        visited=[]
        startState=problem.getStartState()
        stiva.push((startState,[],0))

        currentState , actions ,TotalCost = stiva.pop()
        visited.append(currentState)


        while not problem.isGoalState(currentState):
            successors = problem.getSuccessors(currentState)
            for newState,newAction,newCost in successors:
                if (not newState in visited and TotalCost+newCost<=limit):
                    stiva.push((newState,actions+ [newAction],TotalCost+newCost))
                    visited.append(newState)
            if stiva.isEmpty():
                break
            currentState , actions ,TotalCost = stiva.pop()
        if problem.isGoalState(newState):
            return actions

        limit+=1
```

```python
def getSuccessors(self, state):
    "Returns successor states, the actions they require, and a cost of 1."
    successors = []
    from layout import *
    self._expanded += 1 # DO NOT CHANGE
    for direction in [Directions.NORTH,Directions.WEST, Directions.SOUTH, Directions.EAST,  Directions.TELEPORTER]:
        x,y = state[0]
        x1=layout.portal[0][0]
        y1=layout.portal[0][1]
        x2=layout.portal[1][0]
        y2=layout.portal[1][1]
        if( (x ==x1 and y ==y1 )or (x==x2 and y==y2)):
            dx, dy = Actions.directionToVector(direction)
            nextx, nexty = int(x + dx), int(y + dy)
            if not self.walls[nextx][nexty]:
                nextFood = state[1].copy()
                nextFood[nextx][nexty] = False
                successors.append( ( ((nextx, nexty), nextFood), direction, 1) )
        else:
            if direction != Directions.TELEPORTER:
                dx, dy = Actions.directionToVector(direction)
                nextx, nexty = int(x + dx), int(y + dy)
                if not self.walls[nextx][nexty]:
                    nextFood = state[1].copy()
                    nextFood[nextx][nexty] = False
                    successors.append( ( ((nextx, nexty), nextFood), direction, 1) )

    return successors
```

```
from read import Read
class Actions:
    def xP():

        f = Read(sys.argv[2])
        if f.portals == []:
            return 0
        print OurClass.pacmanX
        return f.portals[1][0]-f.portals[0][0]

    def yP():

        f = Read(sys.argv[2])
        if f.portals==[]:
            return 0
        return f.portals[1][1]-f.portals[0][1]

    _directions = {Directions.NORTH: (0, 1),
                   Directions.SOUTH: (0, -1),
                   Directions.EAST:  (1, 0),
                   Directions.WEST:  (-1, 0),
                   Directions.STOP:  (0, 0),
                #    Directions.TELEPORTER : (-21,-15)
                   Directions.TELEPORTER : (yP(),xP())
                   }

    _directionsAsList = _directions.items()
```

```
class Read:
    portals = []

    def __init__(self,file):
        self.file=file
        self.portals=[]
        self.citeste()

    def citeste(self):
        f = open("layouts/"+self.file+".lay","r")
        Lines = f.readlines()
        k=0
        nr_linii=0
        for line in Lines:
            nr_linii+=1
        # print nr_linii

        for line in Lines:
            for i in line:
                if i == 's':
                    self.portals.append((nr_linii-1-Lines.index(line),line.index(i)))
```

Bibliography

http://ai.berkeley.edu/projectoverview.html
http://ai.berkeley.edu/search.html
https://www.overleaf.com/
https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/