Departamentul de Automatică și Calculatoare
Universitatea Tehnică din Cluj Napoca

Compararea algoritmilor de căutare

Activitate de Laborator

Nume: Calin Cătălin și Ionaș Alex
Grupă: 30232
Email: calincatalin99@gmail.com și ionas.alex319@gmail.com

Asistent de laborator: Katona Áron
Email: katona.io.aron@student.utcluj.ro

Contents

Chapter 1

A1: Introducere

Scopul acestui prim proiect este de a compara algoritmii care îl ajută pe Pacman să găsească drumul spre mâncare. Pentru a realiza acest lucru am implementat, pe lângă algoritmii discutați la laborator (Breadth-First Search, Depth-First Search, A* și Uniform Cost Search), următorii algoritmi:

- Weighted A*
- Bidirectional A*
- Iterative Deepening A*

De asemenea, pentru o vizualizare a acestor algoritmi, am adăugat posibilitatea de a rula, cu ajutorul comenzii

```
python pacman.py -l Maze -p SearchAgent -a fn=dfs -j
```

o instanță de joc, cu mai mulți Pacmani, în care, pornind de la aceeași locație, cu propriul algoritm de căutare, fiecare Pacman încearcă să ajungă primul la premiul cel mare: o bucățică de mâncare.

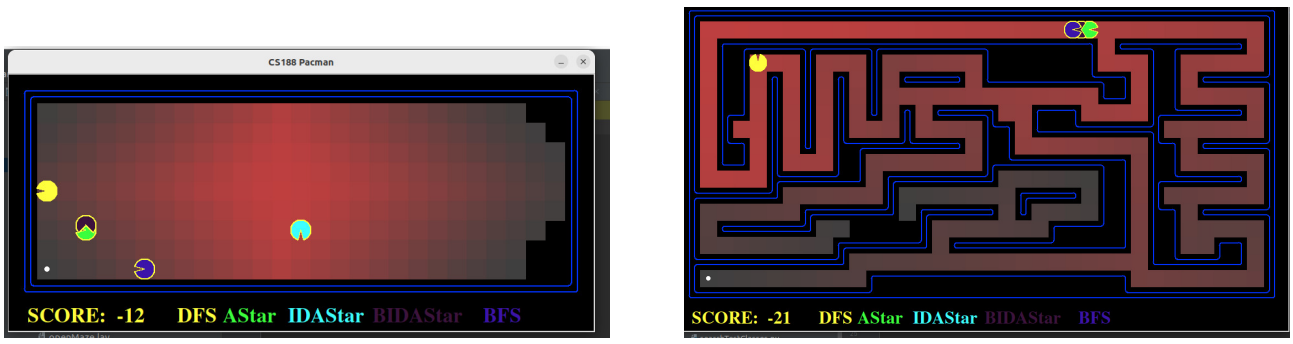

Figure 1.1: Concurs Pacman

Chapter 2

A2: Algoritmi de căutare

2.1 Weighted A* Search

Weighted A* Search este un algoritm asemănător cu A*, diferența constând în faptul că funcția "f(n)" este calculată într-un mod diferit, și anume:

$$\mathbf{f(n) = g(n) + w * h(n),}$$

unde funcția "g" reprezintă costul acțiunii, funcția "h" reprezintă euristica, iar parametrul "w" reprezintă "greutatea".
Dacă acest parametru este w = 1, atunci algoritmul se comportă exact la fel cu A*. În caz contrar, cu cât w este mai mare, cu atat funcția de căutare este mai rapidă.

2.1.1 Implementare

În implementare, folosim o coadă de priorități căreia îi asociem funcția **f** discutată mai sus cu un parametru w = 10 și o euristică euclidiană. Pornind din nodul de start, adăugăm, adăugăm vecinii nodului curent folosindu-ne de funcția de calcul, urmând ca la pasul următor să luăm din stivă nodul din vârf, procedând în mod identic până când găsim mâncarea ascunsă în labirint, sau stiva rămâne goală.
Codul sursă al algoritmului se găsește în secțiunea 4.

2.2 Bidirectional A* Search

Acest algoritm găsește cea mai scurtă rută dintre un punct de start și unul de final. Efectuează două căutări simultane: una înainte din starea inițială și una înapoi de la obiectiv, oprindu-se atunci când cei doi se întâlnesc. Fiecare dintre două căutări au complexitatea $O(b^{d/2})$,iar suma acestor doi timpi de căutare este mult mai mică decât complexitatea $O(b^d)$ care ar rezulta dintr-o singură căutare de la început până la target.

2.2.1 Implementare

În implementare am folosit două structuri de date de tip coadă de prioritate, una pentru direcția înainte și una pentru direcția înapoi. Începând din nodurile de început și final, adăugăm în listele corespunzătoare nodurile vizitate. După ce iterăm prin lista 'înapoi', verificăm dacă există un nod comun în cele două liste, și în caz pozitiv, returnăm drumul curent împreună cu drumul 'înapoi' în ordine inversă.

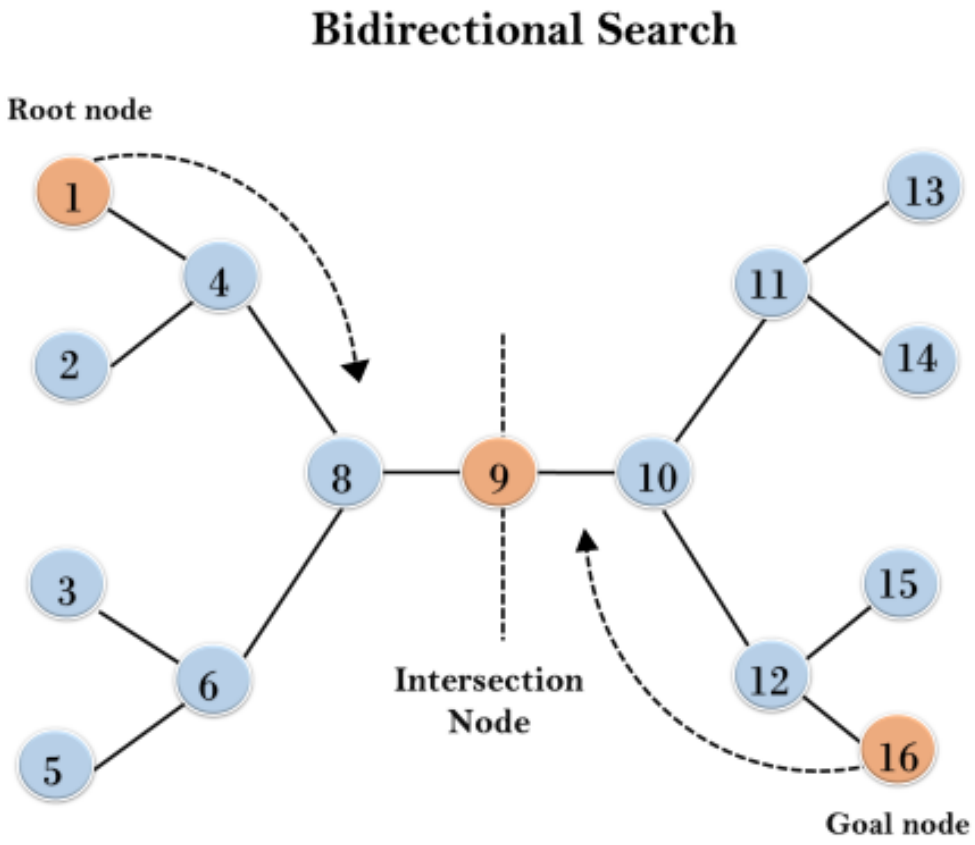

Figure 2.1: Reprezentare vizuală a Bidirectional A* Search

2.2.2 Iterative Deepening A* Search

Iterative deepening depth-first search

Acest algoritm reprezintă o strategie de căutare într-un graf, în care o versiune a DFS cu limită de adâncime este rulată de mai multe ori, de fiecare dată cu incrementarea adâncimii, până se ajunge la nodul obiectiv.

Algoritmul propriu-zis

Iterative Deepening A*(IDA*) este un algoritm de traversare a grafului și de căutare a căilor, care poate găsi cea mai scurtă cale între un nod de pornire desemnat și un nod obiectiv într-un graf. Este o variantă de căutare a iterative deepening depth-first search care împrumută ideea de a utiliza o funcție euristică pentru a evalua costul rămas pentru a ajunge la obiectivul din algoritmul de căutare A*. Deoarece este un algoritm de căutare în profunzime, utilizarea memoriei sale este mai mică decât în A *, dar spre deosebire de iterative deepening depth-first search, se concentrează pe explorarea celor mai promițătoare noduri și, prin urmare, nu merge la aceeași adâncime peste tot în arborele de căutare. Spre deosebire de A*, IDA* nu utilizează programarea dinamică și, prin urmare, de multe ori ajunge să exploreze aceleași noduri de mai multe ori.

2.2.3 Implementare

În implementare folosim un set în care stocăm nodurile vizitate și un set pentru cale.

Folosim o funcție 'search' definită de noi, pe care o apelăm până când valoarea returnata este o valoare mare predefinită (9999 în codul nostru) sau -1. Dacă niciuna dintre aceste valori nu este returnată, atunci vom incrementa limita de adâncime. Valoarea inițială pentru limita de adâncime, este rezultatul sumei dintre cost si euristică.

Chapter 3

A3: Testare și rezultate

După implementarea algoritmilor, a urmat testarea acestora. Aceasta s-a realizat automat, folosind comanda:

```
python pacman.py -l Maze -p SearchAgent -a fn=dfs -n 6
```

La executarea comenzii, după finalizarea rulării algoritmilor, apare numărul de noduri expandate și scorul pentru fiecare algoritm. Rezultatele se pot vedea în tabelele următore:

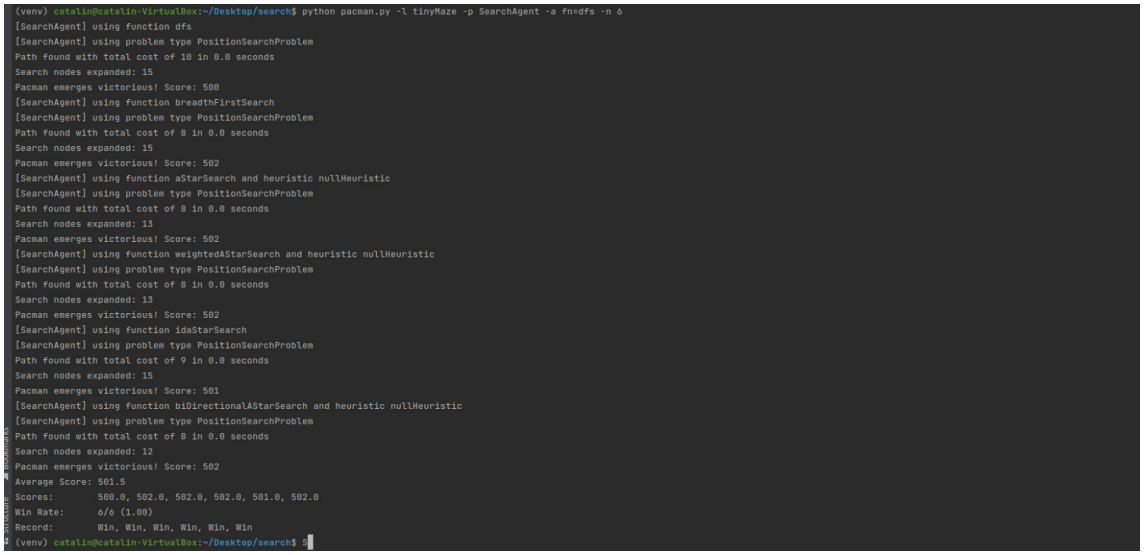

Figure 3.1: Rezultatul rulării comenzii

Noduri expandate	BFS	DFS	A*	UCS	BDA*	IDA*	WA*
Small Maze	92	59	39	92	44	93	54
Medium Maze	269	177	137	269	196	273	162
Complicated Maze	549	427	132	549	356	498	134
Big Maze	620	390	471	620	561	477	488
Custom Maze	603	229	556	603	444	258	556

Table 3.1: Table număr de noduri expandate în functie de hartă.

Noduri expandate	BFS	DFS	A*	UCS	BDA*	IDA*	WA*
Small Maze	491	461	481	491	481	473	481
Medium Maze	429	357	839	429	429	438	389
Complicated Maze	444	372	436	444	444	229	436
Big Maze	380	300	300	300	300	299	436
Custom Maze	383	381	383	383	383	320	383

Table 3.2: Tabel cu scor în functie de hartă.

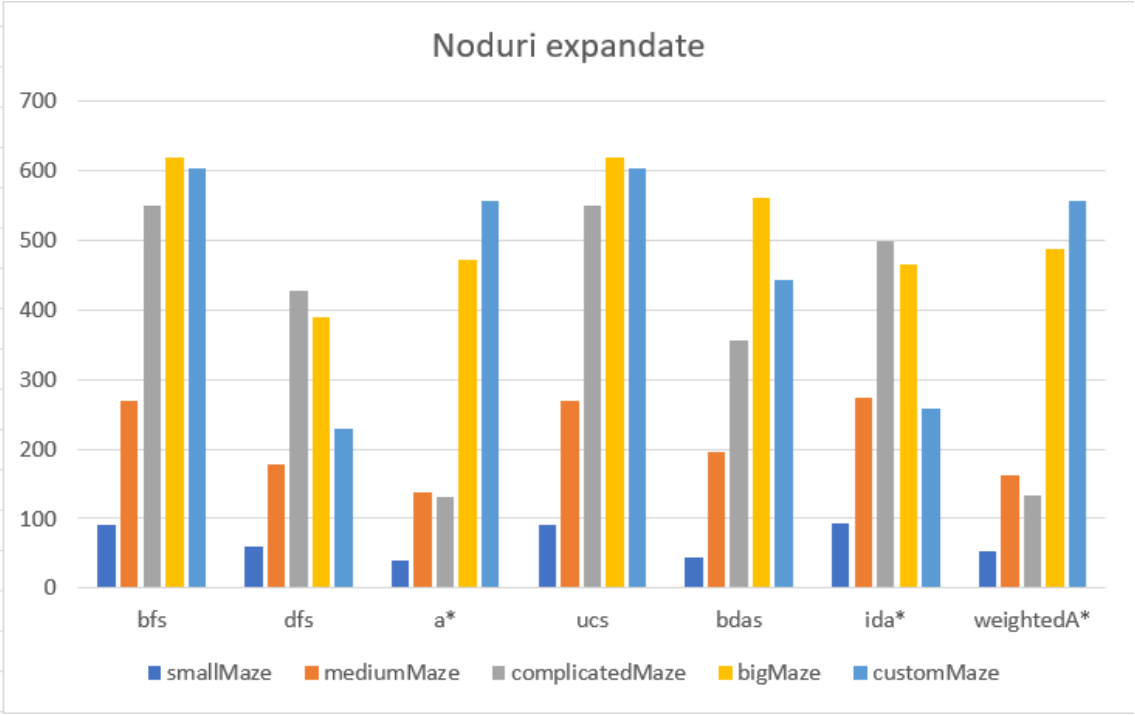

Figure 3.2: Comparație vizuală a algoritmilor

Chapter 4

A4: Codul original

4.1 DFS- Depth First Search

```
def depthFirstSearch(problem):
    if (problem.isGoalState(problem.getStartState())):
        return []
    start = problem.getStartState()
    queue = util.Stack()
    nodes = []
    path = []
    queue.push((start, path))
    while not queue.isEmpty():
        (node, path) = queue.pop()
        if node not in nodes:
            nodes.append(node)
            if problem.isGoalState(node):
                return path
            succesors = problem.getSuccessors(node)
            for p in succesors:
                nod, action, cost = p
                queue.push((nod, path + [action]))

    return []
    util.raiseNotDefined()
```

4.2 BFS- Breath First Search

```
def breadthFirstSearch(problem):

    if (problem.isGoalState(problem.getStartState())):
        return []
    start = problem.getStartState()
    queue = util.Queue()
    nodes = []
    path = []
    queue.push((start, path))
    while not queue.isEmpty():
```

```
        (node, path) = queue.pop()
        if node not in nodes:
            nodes.append(node)
            if problem.isGoalState(node):
                return path
            succesors = problem.getSuccessors(node)
            for p in succesors:
                nod, action, cost = p
                queue.push((nod, path + [action]))

    return []
    util.raiseNotDefined()
```

4.3 UCS- Uniform Cost Search

```
def uniformCostSearch(problem):
    """Search the node of least total cost first."""
    "* YOUR CODE HERE *"
    if (problem.isGoalState(problem.getStartState())):
        return []
    start = problem.getStartState()
    queue = util.PriorityQueueWithFunction(f)
    nodes = []
    path = []
    queue.push((start, path, 0))
    while not queue.isEmpty():
        (node, path, cost) = queue.pop()
        if node not in nodes:
            nodes.append(node)
            if problem.isGoalState(node):
                return path
            succesors = problem.getSuccessors(node)
            for p in succesors:
                nod, action, cost1 = p
                queue.push((nod, path + [action], cost + cost1))

    return []
    util.raiseNotDefined()
```

4.4 A*- A Star Search

```
def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    "*** YOUR CODE HERE ***"
    (x, y) = problem.getGoal()

    if (problem.isGoalState(problem.getStartState())):
        return []
    start = problem.getStartState()
```

```
queue = util.PriorityQueueWithFunction(lambda ((x1, y1), b, c):c + math.sqrt((x
    - x1)**2 + (y - y1)**2))
nodes = []
path = []
queue.push((start, path, 0))
while not queue.isEmpty():
    (node, path, cost) = queue.pop()
    if node not in nodes:
        nodes.append(node)
        if problem.isGoalState(node):
            return path
        succesors = problem.getSuccessors(node)
        for p in succesors:
            nod, action, cost1 = p
            queue.push((nod, path + [action], cost + cost1))

return []
util.raiseNotDefined()
```

4.5 WA*- Weighted A Star Search

```
def weightedAStarSearch(problem, heuristic=nullHeuristic):
"""Search the node that has the lowest combined cost and heuristic first."""
"*** YOUR CODE HERE ***"
(x, y) = problem.getGoal()

if (problem.isGoalState(problem.getStartState())):
    return []
start = problem.getStartState()

queue = util.PriorityQueueWithFunction(lambda ((x1, y1), b, c):10*(c +
    math.sqrt((x - x1)**2 + (y - y1)**2)))
nodes = []
path = []
queue.push((start, path, 0))
while not queue.isEmpty():
    (node, path, cost) = queue.pop()
    if node not in nodes:
        nodes.append(node)
        if problem.isGoalState(node):
            return path
        succesors = problem.getSuccessors(node)
        for p in succesors:
            nod, action, cost1 = p
            queue.push((nod, path + [action], cost + cost1))

return []
util.raiseNotDefined()
```

4.6 IDA*- Iterative Deepining A Star Search

```
#Iterative deepenig A*
def h(node):
    (nod, action, cost) = node
    (x,y) = nod
    return abs(1-x + 1- y )
    #return ((1-x)**2 + (1-y)**2)**0.5
    #return 0

def search(problem,path, g, bound, posPath):
    node = path[-1]
    (nod, action, cost) = node
    f = g + h(node)
    if f > bound:
        return f
    if problem.isGoalState(nod):
        return -1
    min = 9999
    succesors = problem.getSuccessors(nod)
    for p in succesors:
        (x , y, z) = p
        if x not in posPath:
            path.append(p)
            posPath.append(x)
            t = search(problem, path, g + z, bound+2, posPath)
            if t == -1:
                return -1
            if t < min:
                min = t
                posPath.pop(-1)
            path.pop(-1)

    return min

def idaStarSearch(problem):
    start = problem.getStartState()
    finalPath = []
    path = []
    posPath = [start]
    path.append((start, Directions.STOP , 0))
    bound = h((start, Directions.STOP, 0))

    while True:
        t = search(problem, path, 0, bound, posPath)
        if t == -1:
            for (nod, action, cost) in path:
                finalPath.append(action)
            return finalPath
        if t == 9999 :
            return []
        bound = t
#end of Iterative deepenig A*
```

4.7 BIDA*- Bidirectional A* Search

```
# Bidirectional A*
def biDirectionalAStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first."""
    "*** YOUR CODE HERE ***"
    goal = problem.getGoal()
    (x,y) = goal
    if (problem.isGoalState(problem.getStartState())):
        return []
    start = problem.getStartState()

    queueStart = util.PriorityQueueWithFunction(lambda ((x1, y1), b, c):c +
        math.sqrt((x - x1)**2 + (y - y1)**2))
    queueEnd = util.PriorityQueueWithFunction(lambda ((x1, y1), b, c):c +
        math.sqrt((x - x1)**2 + (y - y1)**2))
    nodesStart = []
    nodesEnd = []
    pathStart = []
    pathEnd = []
    queueStart.push((start, pathStart, 0))
    queueEnd.push((goal, pathEnd, 0))
    flagStart = True
    flagEnd = True
    nodeToFind = None
    finalPath = []
    while not queueEnd.isEmpty() and not queueStart.isEmpty():
        (nodeStart, pathStart, cost) = queueStart.pop()
        (nodeEnd, pathEnd, cost1) = queueEnd.pop()

        path1 = checkNode(queueStart, nodeEnd)
        path2 = checkNode(queueEnd, nodeStart)

        if not path1.__len__() == 0:
            flagStart = False
            finalPath = path1 + reverseActions(pathEnd)

        if not path2.__len__() == 0:
            flagStart = False
            finalPath = pathStart + reverseActions(path2)

        if not flagStart:
            problem.isGoalState(goal)
            return finalPath

        if flagStart:
            funcAStar(nodeStart, nodesStart, problem, queueStart, pathStart, cost)
            funcAStar(nodeEnd, nodesEnd, problem, queueEnd, pathEnd, cost1)
```

```
    return []

def checkNode(queue, node):
    for (a, b, c) in queue.getList():
        (nod,path, cost) = c
        if node == nod:
            return path
    return []

def reverseActions(path):
    resultPath = []
    n = len(path) - 1
    newPath = []
    while n > -1:
        newPath.append(path[n])
        n -= 1
    for p in newPath:
        if p == Directions.WEST:
            resultPath.append(Directions.EAST)
        if p == Directions.EAST:
            resultPath.append(Directions.WEST)
        if p == Directions.NORTH:
            resultPath.append(Directions.SOUTH)
        if p == Directions.SOUTH:
            resultPath.append(Directions.NORTH)
    return resultPath

def funcAStar(node, nodes, problem, queue, path, cost):
    if node not in nodes:
        nodes.append(node)
        succesors = problem.getSuccessors(node)
        for p in succesors:
            nod, action, cost1 = p
            queue.push((nod, path + [action], cost + cost1))
```

4.8 Rularea algoritmilor de cautare implementați pe rând

Pentru a realiza această funcționalitate am schimbat la fiecare nouă rulare tipul Pacman-ului.

4.8.1 Pacman.py

```
def runGames(flag, layout, pacman, ghosts, display, numGames, record,
    numTraining=0, catchExceptions=False, timeout=30):
    import __main__
    __main__.__dict__['_display'] = display

    rules = ClassicGameRules(timeout)
    games = []

    for i in range(numGames):
        beQuiet = i < numTraining
```

```
        if beQuiet:
            # Suppress output and graphics
            import textDisplay
            gameDisplay = textDisplay.NullGraphics()
            rules.quiet = True
        else:
            gameDisplay = display
            rules.quiet = False
            #Cod adugat
            if not flag:
                if i==1:
                    pacman = SearchAgent("breadthFirstSearch")
                if i == 2:
                    pacman = SearchAgent("aStarSearch")
                if i == 3:
                    pacman = SearchAgent("weightedAStarSearch")
                if i == 4:
                    pacman = SearchAgent("idaStarSearch")
                if i == 5:
                    pacman = SearchAgent("biDirectionalAStarSearch")
                if i == 6:
                    pacman = SearchAgent("uniformCostSearch")
             #final
        game = rules.newGame(flag, layout, pacman, ghosts, gameDisplay, beQuiet,
            catchExceptions)
        game.run()

        if not beQuiet: games.append(game)

        if record:
            import time, cPickle
            fname = ('recorded-game-%d' % (i + 1)) + '-'.join([str(t) for t in
                time.localtime()[1:6]])
            f = file(fname, 'w')
            components = {'layout': layout, 'actions': game.moveHistory}
            cPickle.dump(components, f)
            f.close()

    if (numGames - numTraining) > 0:
        scores = [game.state.getScore() for game in games]
        wins = [game.state.isWin() for game in games]
        winRate = wins.count(True) / float(len(wins))
        print 'Average Score:', sum(scores) / float(len(scores))
        print 'Scores:      ', ', '.join([str(score) for score in scores])
        print 'Win Rate:     %d/%d (%.2f)' % (wins.count(True), len(wins), winRate)
        print 'Record:      ', ', '.join([['Loss', 'Win'][int(w)] for w in wins])

    return games
```

4.9 Rularea algoritmilor implementați în același timp

Pentru a realiza aceasta s-au făcut mai multe modificări, prezentate mai jos.

4.9.1 Pacman.py

```
 def getLegalActions(self, agentIndex):

        #       GameState.explored.add(self)
        if self.isWin() or self.isLose(): return []
# Conditia anterioara agentIndex == 0, am modificat-o pentru ca acum avem mai multi
    agenti Pacman
        if agentIndex < self.data.layout.getNumPacman(): # Pacman is moving
            return PacmanRules.getLegalActions(self)
        else:
            return GhostRules.getLegalActions(self, agentIndex)

    def generateSuccessor(self, agentIndex, action):

        if self.isWin() or self.isLose(): raise Exception('Can\'t generate a
            successor of a terminal state.')

        state = GameState(self)

# Conditia anterioara agentIndex == 0, am modificat-o pentru ca acum avem mai multi
    agenti Pacman
        if agentIndex < self.data.layout.getNumPacman(): # Pacman is moving
            state.data._eaten = [False for i in range(state.getNumAgents())]
            PacmanRules.applyAction(state, action, agentIndex)
        else:
            GhostRules.applyAction(state, action, agentIndex)

# Conditia anterioara agentIndex == 0, am modificat-o pentru ca acum avem mai multi
    agenti Pacman
        if agentIndex < self.data.layout.getNumPacman():
            state.data.scoreChange += -TIME_PENALTY # Penalty for waiting around
        else:
            GhostRules.decrementTimer(state.data.agentStates[agentIndex])


        GhostRules.checkDeath(state, agentIndex)


        state.data._agentMoved = agentIndex
        if agentIndex == 0:
            state.data.score += state.data.scoreChange
        GameState.explored.add(self)
        GameState.explored.add(state)
        return state
#Am adugant un index functiei pentru a stii pentru care Pacman se apeleaza functia
    def getLegalPacmanActions(self, agentIndex):
        return self.getLegalActions(agentIndex)
#Am adugant un index functiei pentru a stii pentru care Pacman se apeleaza functia
    def generatePacmanSuccessor(self, action, agentIndex):

        return self.generateSuccessor(agentIndex, action)
```

```
#Am adugant un index functiei pentru a stii pentru care Pacman se apeleaza functia
    def getPacmanState(self, agentIndex):

        return self.data.agentStates[agentIndex].copy()
#Am adugant un index functiei pentru a stii pentru care Pacman se apeleaza functia
    def getPacmanPosition(self, agentIndex):
        return self.data.agentStates[agentIndex].getPosition()

#Am adugat un parametru flag care determin cand se realizeaza rularea normala sau
    toti deodata
  def newGame(self, flag, layout, pacmanAgent, ghostAgents, display, quiet=False,
      catchExceptions=False):
        if flag:
        # Array-ul de agents atunci cand ruleaza toti algoritmi
            agents = [pacmanAgent, SearchAgent("aStarSearch"),
                SearchAgent("idaStarSearch"),
                      SearchAgent("biDirectionalAStarSearch"), SearchAgent("bfs")] +
                          ghostAgents[:layout.getNumGhosts()]
        else: #Rulare in regim normal
            agents = [pacmanAgent] + ghostAgents[:layout.getNumGhosts()]

        initState = GameState()
        initState.initialize(layout, len(ghostAgents))

        game = Game(agents, display, self, catchExceptions=catchExceptions)
        game.state = initState
        self.initialState = initState.deepCopy()
        self.quiet = quiet
        return game

class PacmanRules:

    PACMAN_SPEED = 1
    #Index pentru Pacman
    def getLegalActions(state, index):

        return
            Actions.getPossibleActions(state.getPacmanState(index).configuration,
            state.data.layout.walls)

    getLegalActions = staticmethod(getLegalActions)
#Index pentru Pacman
    def applyAction(state, action, index):

        legal = PacmanRules.getLegalActions(state, index)
        if action not in legal:
            raise Exception("Illegal action " + str(action))

        pacmanState = state.data.agentStates[index]

def collide(state, ghostState, agentIndex):
        if ghostState.scaredTimer > 0:
            state.data.scoreChange += 200
```

```
            GhostRules.placeGhost(state, ghostState)
            ghostState.scaredTimer = 0
            # Added for first-person
            state.data._eaten[agentIndex] = True
        else:
            if not state.data._win:
                state.data.scoreChange -=
                #Am setat state-ul de lose pe False, pentru ca jocul sa continue,
                    atunci cand doi Pacmani se intalnesc
                state.data._lose = False

    collide = staticmethod(collide)


#Am adugat comanda -j, pentru a determina cand se va realiza functionalitatea.
    def readCommand(argv):

    from optparse import OptionParser
    usageStr = """
    USAGE:      python pacman.py <options>
    EXAMPLES:   (1) python pacman.py
                    - starts an interactive game
                (2) python pacman.py --layout smallClassic --zoom 2
                OR  python pacman.py -l smallClassic -z 2
                    - starts an interactive game on a smaller board, zoomed in
    """
    parser = OptionParser(usageStr)
#Comanda adugata
    parser.add_option('-j', '--flag', dest='flag', action='store_true',
                      help=default('Run all search algorithms'), default=False)

    parser.add_option('-n', '--numGames', dest='numGames', type='int',
                      help=default('the number of GAMES to play'), metavar='GAMES',
                          default=1)
    parser.add_option('-l', '--layout', dest='layout',
                      help=default('the LAYOUT_FILE from which to load the map
                          layout'),
                      metavar='LAYOUT_FILE', default='mediumClassic')
    parser.add_option('-p', '--pacman', dest='pacman',
                      help=default('the agent TYPE in the pacmanAgents module to
                          use'),
                      metavar='TYPE', default='KeyboardAgent')
    parser.add_option('-t', '--textGraphics', action='store_true',
        dest='textGraphics',
                      help='Display output as text only', default=False)
    parser.add_option('-q', '--quietTextGraphics', action='store_true',
        dest='quietGraphics',
                      help='Generate minimal output and no graphics', default=False)
    parser.add_option('-g', '--ghosts', dest='ghost',
                      help=default('the ghost agent TYPE in the ghostAgents module to
                          use'),
                      metavar='TYPE', default='RandomGhost')
    parser.add_option('-k', '--numghosts', type='int', dest='numGhosts',
```

```
                  help=default('The maximum number of ghosts to use'), default=4)
parser.add_option('-z', '--zoom', type='float', dest='zoom',
                  help=default('Zoom the size of the graphics window'),
                      default=1.0)
parser.add_option('-f', '--fixRandomSeed', action='store_true',
    dest='fixRandomSeed',
                  help='Fixes the random seed to always play the same game',
                      default=False)
parser.add_option('-r', '--recordActions', action='store_true', dest='record',
                  help='Writes game histories to a file (named by the time they
                      were played)', default=False)
parser.add_option('--replay', dest='gameToReplay',
                  help='A recorded game file (pickle) to replay', default=None)
parser.add_option('-a', '--agentArgs', dest='agentArgs',
                  help='Comma separated values sent to agent. e.g.
                      "opt1=val1,opt2,opt3=val3"')
parser.add_option('-x', '--numTraining', dest='numTraining', type='int',
                  help=default('How many episodes are training (suppresses
                      output)'), default=0)
parser.add_option('--frameTime', dest='frameTime', type='float',
                  help=default('Time to delay between frames; <0 means
                      keyboard'), default=0.1)
parser.add_option('-c', '--catchExceptions', action='store_true',
    dest='catchExceptions',
                  help='Turns on exception handling and timeouts during games',
                      default=False)
parser.add_option('--timeout', dest='timeout', type='int',
                  help=default('Maximum length of time an agent can spend
                      computing in a single game'), default=30)

options, otherjunk = parser.parse_args(argv)
if len(otherjunk) != 0:
    raise Exception('Command line input not understood: ' + str(otherjunk))
args = dict()

#Noul argument
args['flag'] = options.flag


if options.fixRandomSeed: random.seed('cs188')

#Flagul e trimis si la Layout
layout1 = layout.getLayout(options.layout, options.flag)
args['layout'] = layout1
if args['layout'] == None: raise Exception("The layout " + options.layout + "
    cannot be found")


noKeyboard = options.gameToReplay == None and (options.textGraphics or
    options.quietGraphics)
pacmanType = loadAgent(options.pacman, noKeyboard)
agentOpts = parseAgentArgs(options.agentArgs)
if options.numTraining > 0:
```

```
        args['numTraining'] = options.numTraining
        if 'numTraining' not in agentOpts: agentOpts['numTraining'] =
            options.numTraining

    pacman = pacmanType(**agentOpts) # Instantiate Pacman with agentArgs
    args['pacman'] = pacman


    if 'numTrain' in agentOpts:
        options.numQuiet = int(agentOpts['numTrain'])
        options.numIgnore = int(agentOpts['numTrain'])

    ghostType = loadAgent(options.ghost, noKeyboard)
    args['ghosts'] = [ghostType(i + 1) for i in range(options.numGhosts)]


    if options.quietGraphics:
        import textDisplay
        args['display'] = textDisplay.NullGraphics()
    elif options.textGraphics:
        import textDisplay
        textDisplay.SLEEP_TIME = options.frameTime
        args['display'] = textDisplay.PacmanGraphics()
    else:
        import graphicsDisplay
        #Flagul este trimis si la InfoPane prin PacmanGraphics
        args['display'] = graphicsDisplay.PacmanGraphics(options.zoom,
            options.flag, frameTime=options.frameTime)
    args['numGames'] = options.numGames
    args['record'] = options.record
    args['catchExceptions'] = options.catchExceptions
    args['timeout'] = options.timeout


    if options.gameToReplay != None:
        print 'Replaying recorded game %s.' % options.gameToReplay
        import cPickle
        f = open(options.gameToReplay)
        try:
            recorded = cPickle.load(f)
        finally:
            f.close()
        recorded['display'] = args['display']
        replayGame(**recorded)
        sys.exit(0)

    return args
```

4.9.2 Layout.py

```
class Layout:
```

```
#Am modificat constructorul pentru a derermina cand realizez functionaliatea,
    adugand un flag de tip bool
    def __init__(self, layoutText, flag):
        self.width = len(layoutText[0])
        self.height= len(layoutText)
        self.walls = Grid(self.width, self.height, False)
        self.food = Grid(self.width, self.height, False)
        self.capsules = []
        self.agentPositions = []
        self.numGhosts = 0
        #am adugat un atribut pentru a retine numarul de Pacmani generati
        self.numPacmans = 0
        self.allPacmanFlag = flag
        self.processLayoutText(layoutText)
        self.layoutText = layoutText
        self.totalFood = len(self.food.asList())
        # self.initializeVisibilityMatrix()
#Getter pt numarul de Pacmani
    def getNumPacman(self):
        return self.numPacmans

    def processLayoutChar(self, x, y, layoutChar):
        if layoutChar == '%':
            self.walls[x][y] = True
        elif layoutChar == '.':
            self.food[x][y] = True
        elif layoutChar == 'o':
            self.capsules.append((x, y))
        elif layoutChar == 'P':
        #Generez mai multi Pacmnai la aceasi pozitie
            if self.allPacmanFlag:
                self.agentPositions.append( (0, (x, y) ) )
                self.agentPositions.append((0, (x, y)))
                self.agentPositions.append((0, (x, y)))
                self.agentPositions.append((0, (x, y)))
                self.numPacmans += 4
            self.agentPositions.append((0, (x, y)))
            self.numPacmans += 1
            #-
        elif layoutChar in ['G']:
            self.agentPositions.append( (1, (x, y) ) )
            self.numGhosts += 1
        elif layoutChar in ['1', '2', '3', '4']:
            self.agentPositions.append( (int(layoutChar), (x,y)))
            self.numGhosts += 1
    #Am adugat flag-ul in urmatorele trei functii
def getLayout(name, flag,back = 2):
    if name.endswith('.lay'):
        layout = tryToLoad('layouts/' + name)
        if layout == None: layout = tryToLoad(name, flag)
    else:
        layout = tryToLoad('layouts/' + name + '.lay', flag)
        if layout == None: layout = tryToLoad(name + '.lay', flag)
```

```
    if layout == None and back >= 0:
        curdir = os.path.abspath('.')
        os.chdir('..')
        layout = getLayout(name, flag, back -1)
        os.chdir(curdir)
    return layout

def tryToLoad(fullname, flag):
    if(not os.path.exists(fullname)): return None
    f = open(fullname)
    try: return Layout([line.strip() for line in f], flag)
    finally: f.close()
```

4.9.3 GraphicDisplay.py

```
#Am adugat un array de culorii pentru ca fiecare Pacman sa fie de culaore difireta
PACMAN_COLORS = [formatColor(255.0/255.0, 255.0/255.0,61.0/255),
    formatColor(60.0/255.0, 255.0/255.0,61.0/255),
                formatColor(60.0/255.0, 255.0/255.0,255.0/255),
                    formatColor(60.0/255.0, 20.0/255.0,61.0/255),
                formatColor(60.0/255.0, 20.0/255.0,170.0/255)]


class InfoPane:
    def __init__(self, layout, gridSize):
        self.gridSize = gridSize
        self.width = (layout.width) * gridSize
        self.base = (layout.height + 1) * gridSize
        self.height = INFO_PANE_HEIGHT
        self.fontSize = 24
        self.itsDisplay = False
        self.textColor = PACMAN_COLOR
        #Atribut adugat pt a determina daca se va desena sau nu in InfoPane
            denumirile algoritmilor care ruleaza.
        self.drawPacmans = False
        self.drawPane()

     def drawPane(self):
        if not self.itsDisplay:
            self.itsDisplay = True
            self.scoreText = text( self.toScreen(0, 0 ), self.textColor, "SCORE: 0",
                "Times", self.fontSize, "bold")
            #Adaug in InfoPane denumirile algoritmilor care ruleaza fiecare avand
                culeara Pacman-ului pe care il reprezinta pentru ai indentifica pe
                mapa
        if self.drawPacmans:
            self.dfsText = text( self.toScreen(230,0 ), PACMAN_COLORS[0],
                "DFS","Times", self.fontSize, "bold")
            self.dfsText = text( self.toScreen(300,0 ), PACMAN_COLORS[1],
                "AStar","Times", self.fontSize, "bold")
            self.dfsText = text( self.toScreen(400,0 ), PACMAN_COLORS[2],
                "IDAStar","Times", self.fontSize, "bold")
```

```
            self.dfsText = text( self.toScreen(530,0 ), PACMAN_COLORS[3],
                "BIDAStar","Times", self.fontSize, "bold")
            self.dfsText = text( self.toScreen(700,0 ), PACMAN_COLORS[4],
                "BFS","Times", self.fontSize, "bold")

class PacmanGraphics:
#Am adugat flagul in constructor pentru a-l seta pe cel din InfoPane
    def __init__(self, zoom=1.0, flag=False, frameTime=0.0, capture=False):
        self.have_window = 0
        self.currentGhostImages = {}
        self.pacmanImage = None
        #Adugat si ca atribut aic
        self.flag = flag
        self.zoom = zoom
        self.gridSize = DEFAULT_GRID_SIZE * zoom
        self.capture = capture
        self.frameTime = frameTime
#fuctia care seteaza in InfoPane flagul
    def setFlag(self):
        self.infoPane.drawPacmans = self.flag

    def initialize(self, state, isBlue = False):
        self.isBlue = isBlue
        self.startGraphics(state)
       #Setez flag-ul
        self.setFlag()
        #Reactualize Infopane
        self.infoPane.drawPane()


        self.distributionImages = None # Initialized lazily
        self.drawStaticObjects(state)
        self.drawAgentObjects(state)
        self.previousState = state

#Am adugat indexul pentru a desena Pacmani diferiti
  def drawPacman(self, pacman, index):
        position = self.getPosition(pacman)
        screen_point = self.to_screen(position)
        endpoints = self.getEndpoints(self.getDirection(pacman))

        width = PACMAN_OUTLINE_WIDTH
        outlineColor = PACMAN_COLOR
        fillColor = PACMAN_COLORS[index]

        if self.capture:
            outlineColor = TEAM_COLORS[index % 2]
            fillColor = GHOST_COLORS[index]
            width = PACMAN_CAPTURE_OUTLINE_WIDTH

        return [circle(screen_point, PACMAN_SCALE * self.gridSize,
                       fillColor = fillColor, outlineColor = outlineColor,
                       endpoints = endpoints,
```

```
width = width)]
```

4.10 Complicated Maze

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  %                                    P    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
%          %%         %%                     %
%%%%%%%  % %%%% %%    %%%% %%%%% %%%%%%%     %
%%       %%%%%          %   %%               %
%%%%%%     %          %%    %   %%%%%        %
%   %        %%%%%     %%%%%%      %%%%%%    %
%%%%%%%   %%%               %          %%    %
%          % %%%            %          %     %
%%%%%%%%               %%%%      %%%%%       %
%        %%%%%  %%   %%%          %%         %
%  %%%%%               %%%%        %         %
%         %%%%                 %%%%          %
% %   % %%   %           %%%%%%%             %
%%%%% %   %%%%       %%%%%                   %
%   % %   %       %%%%     %%%%%%            %
%   % %   %      %              %%%%%        %
%.    %                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

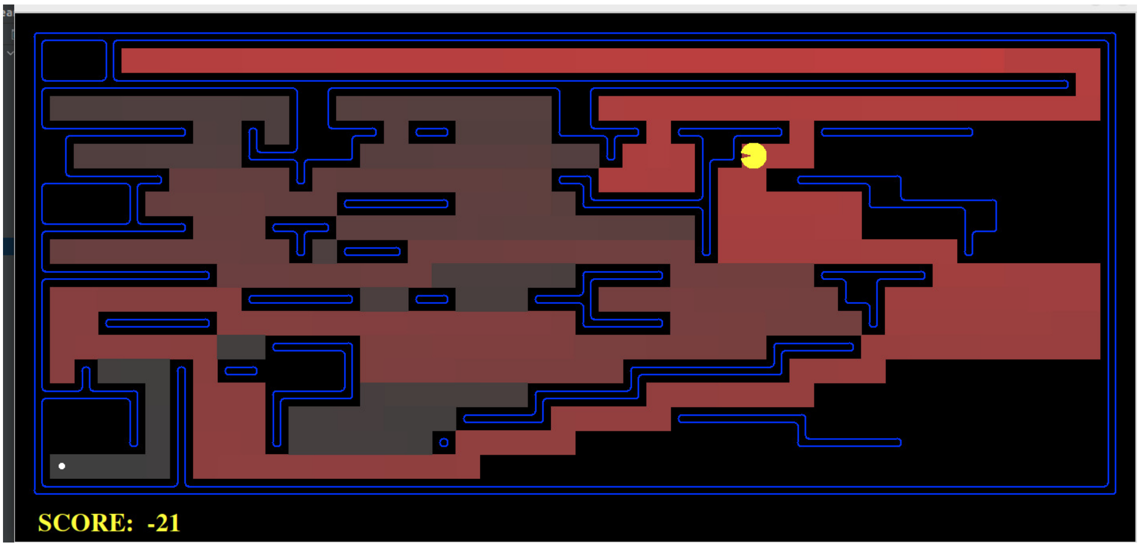

Figure 4.1: Complicated Maze

4.11 Custom Maze

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%P                    %                                                 %
% %%%%%%%%%%%%%% %%%%%%%%%%%%%% %%%%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                %                  %                                   %
% %%%%%%%%%%%% %%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
% %            % %              % %                           %%% %
% % %          % % % %%%%%%%% % % %%%%%%%%%%%%%%%%%%%%%%%% %%% %
% % %          % % % %          % % %                       %     %
% % %%%%%%%%%% % %%%%%%%%%%%%%% % %%%%%%%% %       % %%   %
% %            % %                %     %          %       % %% %%
% %%%         % %%%%% %%%%%%%%%%%%%% %%%%%%%%%%%%%% %% %%
%    %         %    %                    %                    %%   %
%%% % %%%%%%%%%%%%% %%%%%%%%%%%%%%%%% %%%%%%%%%%%%% %%% %
% % %                                  %  %           %  % %%% %
% % %%%%%%%%%% %%%%%%%%%%%%%%%%%%%% % %%    %    %  % %%  %
% % %            %            %       % % %      %       % %% %%
% % %         %%%%%%%      %%%%    % % %  %%%%%%%%%%% %% %%
%                                    % % %                  %% %%
% %%%%%%%%                        %  % % %  % %%        %%% %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%.                                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

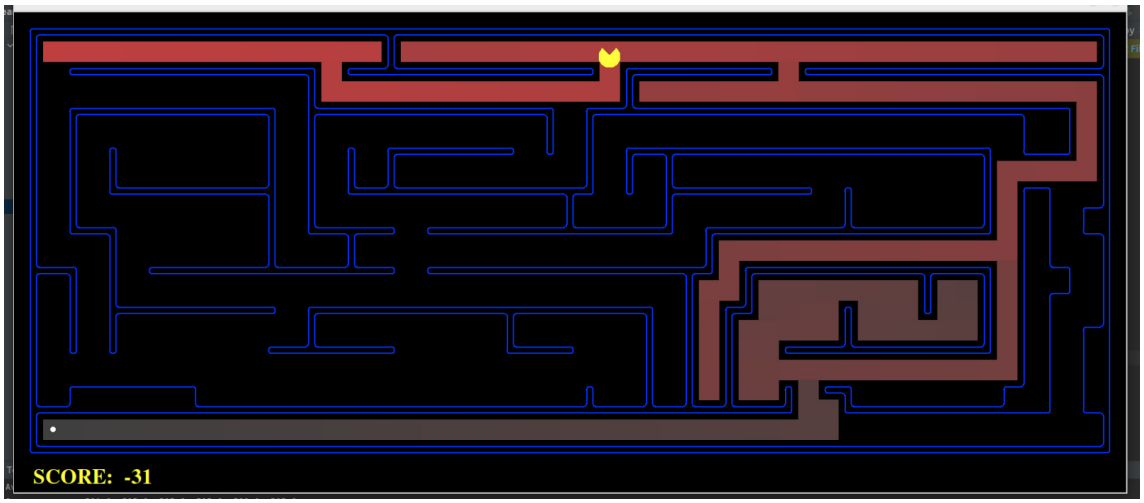

Figure 4.2: Custom Maze

Bibliography

https://en.wikipedia.org/wiki/Iterative_deepening_A*
https://www.geeksforgeeks.org/bidirectional-search/