Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name: Fucă Răzvan-Ionuț, Goina Radu
Group: 33432
Email: razvanfuca@gmail.com, r.goina@yahoo.com

Teaching Assistant: Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Table 1: Lab scheduling

Activity	**Deadline**
Searching agents, Linux, Latex, Python, Pacman	W_1
Uninformed search	W_2
Informed Search	W_3
Adversarial search	W_4
Propositional logic	W_5
First order logic	W_6
Inference in first order logic	W_7
Knowledge representation in first order logic	W_8
Classical planning	W_9
Contingent, conformant and probabilistic planning	W_{10}
Multi-agent planing	W_{11}
Modelling planning domains	W_{12}
Planning with event calculus	W_{14}

Lab organisation.

1. Laboratory work is 25% from the final grade.
2. There are three deliverables in total: 1. Search, 2. Logic, 3. Planning.
3. Before each deadline, you have to send your work (latex documentation/code) at moodle.cs.utcluj.ro
4. We use Linux and Latex
5. Plagiarism: Don't be a cheater! Cheating affects your colleagues, scholarships and a lot more.

Chapter 1

A1: Search

This chapter covers the search problem and uses the classic arcade game of Pacman as the backdrop in order to explore multiple common search algorithms.

We have implemented both optimal(A*) and non-optimal(DFS) searching algorithms.

Searching can be applied in different ways depending on the game state and the definition of the goal state. In this project, we've used these algorithms to tackle the following tasks:

- Finding a single position in a maze
- Visiting every corner of in the maze
- Eating all the food in the maze
- All of the above but with the added twist of using warp tunnels

List of implemented algorithms:

- Depth First Search
- Breadth First Search
- Uniform Cost Search
- A*

Chapter 2

A2: Logics

Chapter 3

A3: Planning

Bibliography

Appendix A

Your original code

search.py :

```
1
2 def depthFirstSearch(problem: SearchProblem):
3
4     frontier = util.Stack()
5     visited = set()
6     actions = []
7     frontier.push( (problem.getStartState(), actions) )
8     while not frontier.isEmpty():
9         currentNode, actions = frontier.pop()
10
11        if problem.isGoalState(currentNode):
12            return actions
13        if currentNode not in visited:
14            visited.add(currentNode)
15            for node, action, _ in problem.getSuccessors(
                  currentNode):
16                new_path = actions.copy()
17                new_path.append(action)
18                frontier.push((node, new_path))
19
20    return []
21
22
23 def breadthFirstSearch(problem: SearchProblem):
24    visited=set()
25
26    states=util.Queue()
27    dir_taken=util.Queue()
28
29    visited.add(problem.getStartState())
30    states.push(problem.getStartState())
31    curr_path=[]
32
33    while not states.isEmpty():
34        curr_state=states.pop()
35        if not dir_taken.isEmpty():
```

```
36              curr_path=dir_taken.pop()
37
38          if(problem.isGoalState(curr_state)):
39              return curr_path
40
41          for succesor in problem.getSuccessors(curr_state):
42              if succesor[0] not in visited:
43                  states.push(succesor[0])
44                  visited.add(succesor[0])
45
46                  new_path = curr_path[:]
47                  new_path.append(succesor[1])
48
49                  dir_taken.push(new_path)
50      return None
51
52  def uniformCostSearch(problem: SearchProblem):
53      visited = set()
54      states_directions = util.PriorityQueue()
55
56      start_state = problem.getStartState()
57      states_directions.push((start_state, [], 0), 0)
58
59      while not states_directions.isEmpty():
60          curr_state, curr_path, curr_cost = states_directions.pop
                ()
61
62          if problem.isGoalState(curr_state):
63              return curr_path
64
65          if curr_state not in visited:
66              visited.add(curr_state)
67              for state, direction, cost in problem.getSuccessors(
                    curr_state):
68                  if state not in visited:
69                      new_path = curr_path + [direction]
70                      new_cost = problem.getCostOfActions(new_path)
71                      states_directions.push((state, new_path,
                            new_cost), new_cost)
72
73      return []
74
75  def aStarSearch(problem: SearchProblem, heuristic=nullHeuristic):
76      visited = set()
77      states_directions = util.PriorityQueue()
78
79      start_state = problem.getStartState()
80      states_directions.push((start_state, [], 0), 0)
81
82      while not states_directions.isEmpty():
```

```
        curr_state, curr_path, curr_cost = states_directions.pop()

        if problem.isGoalState(curr_state):
            return curr_path

        if curr_state not in visited:
            visited.add(curr_state)
            for state, direction, cost in problem.getSuccessors(curr_state):
                if state not in visited:
                    new_path = curr_path + [direction]
                    new_cost = problem.getCostOfActions(new_path) + heuristic(state,problem)
                    states_directions.push((state, new_path, new_cost), new_cost)

    return []
```

searchAgents.py:

```
class CornersProblem:
    def getSuccessors(self, state: Any):

        successors = []
        for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:
            x,y = state[0]
            notVisited=state[1]
            dx, dy = Actions.directionToVector(action)
            nextx, nexty = int(x + dx), int(y + dy)
            hitsWall = self.walls[nextx][nexty]
            if not hitsWall:
                nextCoordonate=(nextx,nexty)
                if nextCoordonate in self.corners and nextCoordonate in notVisited:
                    element_to_remove = nextCoordonate
                    notVisited = tuple(item for item in notVisited if item != element_to_remove)

                if self.portals[nextx][nexty]!=0:
                    for portalCoord,portalType in self.portals.asListNotNull():
                        if portalCoord != nextCoordonate and portalType == self.portals[nextx][nexty]:
                            successors.append(((portalCoord, notVisited),action,1))
                else:
                    successors.append(((nextCoordonate, notVisited),action,1))
```

```
23
24              self._expanded += 1 # DO NOT CHANGE
25              return successors
26
27
28          def cornersHeuristic(state: Any, problem: CornersProblem)
                :
29          corners = problem.corners # These are the corner
               coordinates
30          walls = problem.walls # These are the walls of the maze,
               as a Grid (game.py)
31
32          position, notVisited = state
33
34          min = 999999
35
36          for corner in corners:
37              if corner in notVisited:
38                  manhattanHeuristicVal = util.manhattanDistance(
                       corner, position)
39                  wallCountingVal = wallCounting(position, corner,
                       walls)
40
41                  if manhattanHeuristicVal < wallCountingVal:
42                      distanceAprox = manhattanHeuristicVal +
                           wallCountingVal
43                  else:
44                      distanceAprox = manhattanHeuristicVal
45
46                  if distanceAprox < min:
47                      min = distanceAprox
48
49          return min if min != 999999 else 0
```

Changes to layouts.py to support warp tunnels:

```
1           def processLayoutChar(self, x, y, layoutChar):
2           if layoutChar == '%':
3               self.walls[x][y] = True
4           elif layoutChar == '.':
5               self.food[x][y] = True       # Added a new Layout for
                   Red and Blue portals.
6           elif layoutChar == 'B':          # 0 -> no portal
7               self.portals[x][y] = 1       # 1 -> Blue portal
8           elif layoutChar == 'R':          # 2 -> Red portal
9               self.portals[x][y] = 2
10          elif layoutChar == 'o':
11              self.capsules.append((x, y))
12          elif layoutChar == 'P':
13              self.agentPositions.append( (0, (x, y) ) )
14          elif layoutChar in ['G']:
15              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
```

Changes to graphicsDisplay.py to support warp tunnels:

```
    def drawPortal(self, portals):
    portalImages={}
    for portalCoord,type in portals.asListNotNull():
        (screen_x, screen_y) = self.to_screen(portalCoord)
        dot = circle( (screen_x, screen_y),
                      PORTAL_SIZE * self.gridSize,
                      outlineColor = PORTAL_COLORS[type
                         -1],
                      fillColor = PORTAL_COLORS[type-1],
                      width = 1)
        sdot = circle( (screen_x, screen_y),
                      PORTAL_SIZE* 0.65 * self.gridSize,
                      outlineColor = PORTAL_COLORS[type
                         +1],
                      fillColor = PORTAL_COLORS[type+1],
                      width = 1)
        portalImages[portalCoord]=dot
    return portalImages
```

Changes to game.py to support warp tunnels:

```
class Configuration:
    def generateSuccessor(self, vector,PortalUsed=False):
        if PortalUsed == False:
            x, y= self.pos
            dx, dy = vector
            direction = Actions.vectorToDirection(vector)
            if direction == Directions.STOP:
                direction = self.direction # There is no stop
                    direction
            return Configuration((x + dx, y+dy), direction)
        else:
            direction = Directions.WEST
            return Configuration(vector,direction)
```

Changes to pacman.py to support warp tunnels:

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

    pacmanState = state.data.agentStates[0]
    (pacmanx, pacmany)=pacmanState.getPosition()
    type =state.data.layout.portals[pacmanx][pacmany]
    # Update Configuration
    if type!=0:
```

```
            twinPortalPosition=None
            for portal,types in state.data.layout.portals.
               asListNotNull():
                if types == type and portal!=(pacmanx, pacmany):
                    twinPortalPosition=portal
            print(twinPortalPosition)
            pacmanState.configuration=pacmanState.configuration.
               generateSuccessor(twinPortalPosition,PortalUsed=
               True)

        vector = Actions.directionToVector( action, PacmanRules.
           PACMAN_SPEED )
        pacmanState.configuration = pacmanState.configuration.
           generateSuccessor( vector )

        # Eat
        next = pacmanState.configuration.getPosition()
        nearest = nearestPoint( next )
        if manhattanDistance( nearest, next ) <= 0.5 :
            # Remove food
            PacmanRules.consume( nearest, state )
    applyAction = staticmethod( applyAction )
```