Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name:Galbaza Alexandru-Mihai, Apetrei Ioana-Amalia
Group:30432
Email:Galbaza.Fl.Alexandru@student.utcluj.ro, Apetrei.Pa.Ioana@student.utcluj.ro

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

1.1 Description

For this assignment, we have chosen to introduce a new mechanic to Pacman. We have chosen to call it: ”corruption”.

The simple idea is: pacman starts off as normal, being controlled by the player (human). However, due to some anomaly, after a random amount of time, Pacman goes rogue and starts seeking a ghost, ignoring input from the keyboard. If he fails to end himself before the corruption wears off, player control is restored and the cycle resumes.

This was added to add another layer of randomness to the usual, sometimes boring gameplay of Pacman.

`Without further ado, let’s roll in the code`

1.2 Updated agent

Due partly to laziness, an existing agent was MODIFIED to accommodate the new mechanic. Say hello to KeyboardAgent2. Some new parameters have been added in order to aid in our conquest and a couple of functions. (how much time he remains corrupted, how fast he gains corruption, from what percentage he starts etc.)

The first important task we had to pass was to somehow keep our agent up to date. He's very reliant on the clock (technically speaking on the frame) so we had to make him a method that gets called very often in the main loop of the game.

```python
    def update(self, delta_time, state):
        """
        Update the agent's state, including corruption mechanics and storing game state.
        """
        # Store the current state for use in moveTowardsGhost
        self.current_state = state

        # print(self.corruption)

        # Update corruption meter and possession status
        self.corruption += delta_time * self.corruption_rate

        if self.possessed:
            self.possession_time -= delta_time * 100 * self.corruption_degradation
            # print(self.possession_time)
            PacmanGraphics.setCorrupted(PacmanGraphics, True, self.possession_time)
            if self.possession_time <= 0:
                self.possessed = False
                self.corruption = 0
                PacmanGraphics.setCorrupted(PacmanGraphics, False, self.corruption)
        elif self.corruption >= 100:
            PacmanGraphics.setCorrupted(PacmanGraphics, True, self.possession_time)
            self.possessed = True
            self.possession_time = 100  # Pacman is possessed for 3-6 seconds
            # print(self.possessed)
        else:
            PacmanGraphics.setCorrupted(PacmanGraphics, False, self.corruption)
```

Listing 1.1: Staying updated

To make it pretty, we made sure that we’ll call our special update function only if we have our special agent as a ”client” to the framework

```python
            #...
            while not self.gameOver:
            # Fetch the next agent
            agent = self.agents[agentIndex]

            # Calculate delta time for corruption mechanics
            current_time = time.time()
            delta_time = current_time - last_frame_time
            last_frame_time = current_time

            # Update Pacman agent with corruption logic
            from keyboardAgents import KeyboardAgent2
            if isinstance(agent, KeyboardAgent2):
                agent.update(delta_time, self.state) # This is wrong, we should check if it’s our case or not

            move_time = 0
            skip_action = False
            # Generate an observation of the state
            if ’observationFunction’ in dir( agent ):
            # ...
```

Listing 1.2: Main function code game.py

Next up, the rest of the remade agent. We had to make him ignore keyboard input while rogue and follow some sort of auto-pilot. Here's the code for that:

```
        def getMove(self, legal):
            """
            Chooses the move for Pacman. If possessed, ignores keyboard input and seeks ghosts.
            Otherwise, listens to keyboard input.
            """
            if self.possessed:
                return self.moveTowardsGhost(self.current_state) # REPAIRED
            else:
                # If not possessed, return normal keyboard movement
                move = Directions.STOP
                if (self.WEST_KEY in self.keys) and Directions.WEST in legal:  move = Directions.WEST
                if (self.EAST_KEY in self.keys) and Directions.EAST in legal: move = Directions.EAST
                if (self.NORTH_KEY in self.keys) and Directions.NORTH in legal: move = Directions.NORTH
                if (self.SOUTH_KEY in self.keys) and Directions.SOUTH in legal: move = Directions.SOUTH
                return move

        def moveTowardsGhost(self, state):
        """
        Greedy method for moving towards the nearest ghost.
        Chooses a legal move that gets Pacman closer to the nearest ghost.
        """
        ghost_positions = state.getGhostPositions()
        pacman_position = state.getPacmanPosition()

        # Find the nearest ghost
        nearest_ghost = min(ghost_positions, key=lambda pos: self.getDistance(pacman_position, pos))

        # Get the legal move that brings Pacman closer to the nearest ghost
        best_move = Directions.STOP
        best_distance = float('inf')
        legal = state.getLegalPacmanActions()
        for action in legal:
            successor_pos = self.getSuccessorPosition(pacman_position, action)
            distance = self.getDistance(successor_pos, nearest_ghost)
            if distance < best_distance:
                best_move = action
                best_distance = distance

        return best_move
```

Listing 1.3: Moving Pacman

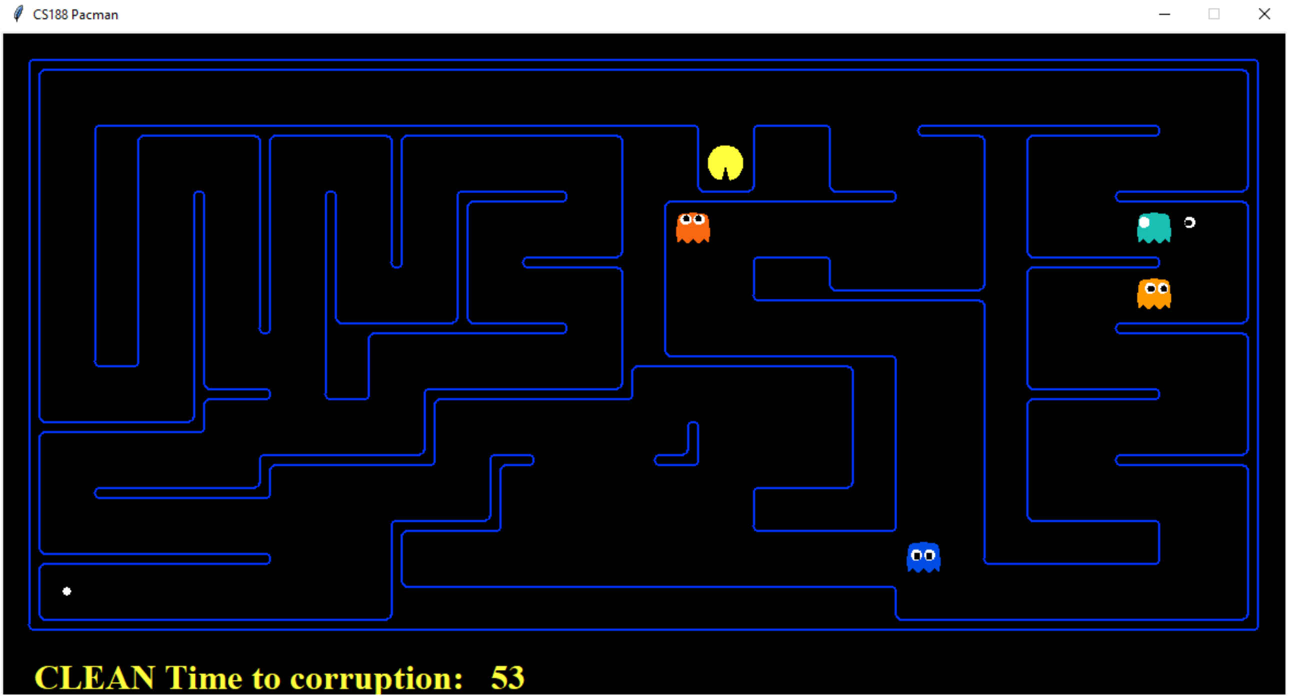


Figure 1.1: Strategy

I've chosen to guide the corrupted pacman (the one which doesn't respond to human input) with a simple, dumb, greedy algorithm. I've tried to run this with alpha-beta pruning, which is way more efficient at seeking ghosts, however this is its very flaw, the game is not fun to play anymore.

Take for example figure 1.1. We are at the very beginning of the game and PACMAN is to get possessed in a couple of moments. If we were to run alpha-beta pruning, the game would essentially be over since the orange ghost is too close. However, if we're running greedy, the player actually stands a chance at winning. He can place himself somewhere that will soft-lock the dumb, corrupted PACMAN.

Of course, if another ghost approaches from topside it's yet again game over for our boy in yellow.

With this decision I wanted the project to be actually fun to play.

I've tried to make him red when corrupted, but it's way more work than it seems. Unfortunately I had to abandon ship. There are some remnants of my attempts if you look at the source code.

1.3 Displaying the corruption status

```
        def update(self, newState):
        # print("Update called!")
        agentIndex = newState._agentMoved
        agentState = newState.agentStates[agentIndex]

        if self.agentImages[agentIndex][0].isPacman != agentState.isPacman: self.swapImages(agentIndex, agentState)
        prevState, prevImage = self.agentImages[agentIndex]
        if agentState.isPacman:
            self.animatePacman(agentState, prevState, prevImage)
        else:
            self.moveGhost(agentState, agentIndex, prevState, prevImage)
        self.agentImages[agentIndex] = (agentState, prevImage)

        if newState._foodEaten != None:
            self.removeFood(newState._foodEaten, self.food)
        if newState._capsuleEaten != None:
            self.removeCapsule(newState._capsuleEaten, self.capsules)
        #TODO: Print score if normal, corruption if that's the mode
        # self.infoPane.updateScore(newState.score)
        # print(self.corruptionLevel)

        self.infoPane.updateCorruption(self.corrupted, self.corruptionLevel)

        if 'ghostDistances' in dir(newState):
            self.infoPane.updateGhostDistances(newState.ghostDistances)
```

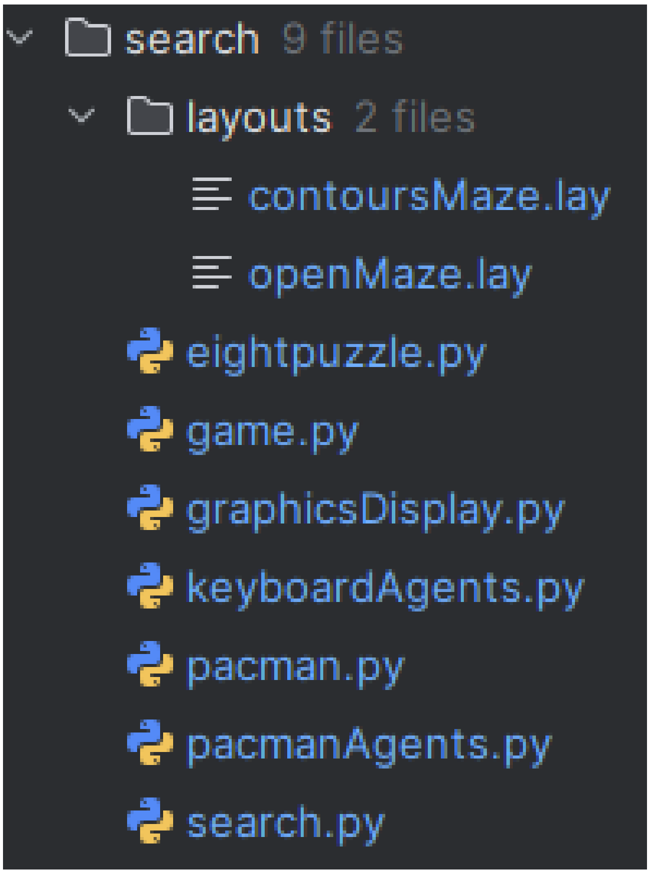
Listing 1.4: PacmanGraphics Class

In the "*graphicsDisplay.py*" file, in the "*PacmanGraphics*" class we have an "Update" function that is called very often in order to animate pacman, remove the consumed food and, fortunately for us, display the current score! We hijacked that function to display our corruption mechanics status.

```

    def updateCorruption(self, is_corrupted, corruptionLevel): #TODO: to complete
    '''
    Used to transfer the data from the PacmanGraphics class to the InfoPane class
    '''
        if is_corrupted:
            changeText(self.scoreText, "CORRUPTED,␣Percentage:␣%␣4d" % corruptionLevel)
        else:
            changeText(self.scoreText, "CLEAN␣Time␣to␣corruption:␣%␣4d" % corruptionLevel)
```

Listing 1.5: InfoPane Class



```
search 9 files
  layouts 2 files
    contoursMaze.lay
    openMaze.lay
  eightpuzzle.py
  game.py
  graphicsDisplay.py
  keyboardAgents.py
  pacman.py
  pacmanAgents.py
  search.py
```

1.4 Conclusion

That being said, our simple project is over.

A lot was learned from it, starting from actual AI notions such as agents, searches, heuristics and the likes to writing our own mind in someone else's shell (framework).

As a memorial of the work we've put in, here's a list of the changed files, straight from GIT.

I would really have liked to have this bloody image below the text, but I've spend half an hour trying to move it down and I just can't. Unfortunately it shall stay like this for now.

Chapter 2

A2: Logics

Chapter 3

A3: Planning

Bibliography

Appendix A

Your original code

Don't be a cheater! Cheating affects your colleagues, scholarships and a lot more. This section should contain only code developed by you, without any line re-used from other sources. This section helps me to correctly evaluate your amount of work and results obtained.

```python

class KeyboardAgent2(KeyboardAgent):
    """
    A second agent controlled by the keyboard, with a corruption mechanic.
    When the corruption meter hits 100%, Pacman becomes possessed and seeks the nearest ghost.
    """
    # Define keys for movement
    WEST_KEY = 'a'
    EAST_KEY = "d"
    NORTH_KEY = 'w'
    SOUTH_KEY = 's'
    STOP_KEY = 'q'

    def __init__(self):
        super().__init__()
        self.corruption = 30  # Start with 30% corruption
        self.corruption_rate = 100  # Bigger is fater
        self.corruption_degradation = 5 # Percent to drop the corruption (in Christ's own unit of measurement
        self.possessed = False  # Possession flag
        self.time_possessed = 0  # Time left. If it ain't broke, don't fix it
        self.possession_time=0

     # TODO: Be able to select between the 2 modes of moving from cmd param
    def getMove(self, legal):
        """
        Chooses the move for Pacman. If possessed, ignores keyboard input and seeks ghosts.
        Otherwise, listens to keyboard input.
        """
        # Check if Pacman is possessed
        if self.possessed:
            return self.moveTowardsGhost(self.current_state) # REPAIRED
        else:
            # If not possessed, return normal keyboard movement

        def moveTowardsGhost(self, state):
        """
        Greedy method for moving towards the nearest ghost.
        Chooses a legal move that gets Pacman closer to the nearest ghost.
        """
        ghost_positions = state.getGhostPositions()
        pacman_position = state.getPacmanPosition()

        # Find the nearest ghost
        # Line removed. It was retrieved from a stackOverflow page

        # Get the legal move that brings Pacman closer to the nearest ghost
        best_move = Directions.STOP
        best_distance = float('inf')
        legal = state.getLegalPacmanActions()
        for action in legal:
            successor_pos = self.getSuccessorPosition(pacman_position, action)
            distance = self.getDistance(successor_pos, nearest_ghost)
            if distance < best_distance:
                best_move = action
                best_distance = distance

        return best_move

        def getDistance(self, pos1, pos2):
        """
        Calculate Manhattan distance between two points.
        """
        return abs(pos1[0] - pos2[0]) + abs(pos1[1] - pos2[1])

        def getSuccessorPosition(self, position, action):
        """
        Get the position after taking the given action.
        """
        x, y = position
        dx, dy = Actions.directionToVector(action)
        return (int(x + dx), int(y + dy))

        def evaluateState(self, state): #In the class
        """
        Evaluation function to score a state based on Pacman's distance to the nearest ghost.
        """
        pacman_position = state.getPacmanPosition()
        ghost_positions = state.getGhostPositions()

        # Calculate the distance to the nearest ghost
        closest_ghost_distance = min([self.getDistance(pacman_position, ghost) for ghost in ghost_positions])

        # Return a higher score for closer distances (since Pacman wants to reach the ghosts)
        return -closest_ghost_distance  # Negative since closer distance is more "desirable" when possessed

        # The actual alpha beta pruning code was also retrieved from somewhere.
```

Listing 1: KeyboardAgents class

```

#TODO: Print score if normal, corruption if that's the mode
        # self.infoPane.updateScore(newState.score)
        # print(self.corruptionLevel)

        self.infoPane.updateCorruption(self.corrupted, self.corruptionLevel)

    corrupted = False
    corruptionLevel = 0

    @staticmethod
    def setCorrupted(cls, is_corrupted, corruptedLevel):
        cls.corrupted = is_corrupted
        print(is_corrupted, cls.corrupted)
        cls.corruptionLevel = corruptedLevel

        def updateCorruption(self, is_corrupted, corruptionLevel): #TODO: to complete
        '''
        Used to transfer the data from the PacmanGraphics class to the InfoPane class
        '''
        if is_corrupted:
            changeText(self.scoreText, "CORRUPTED,␣Percentage:␣%␣4d" % corruptionLevel)
        else:
            changeText(self.scoreText, "CLEAN␣Time␣to␣corruption:␣%␣4d" % corruptionLevel)

         # Calculate delta time for corruption mechanics
            current_time = time.time()
            delta_time = current_time - last_frame_time
            last_frame_time = current_time

            # Update Pacman agent with corruption logic
            from keyboardAgents import KeyboardAgent2
            if isinstance(agent, KeyboardAgent2):
```

```
            agent.update(delta_time, self.state) # This is wrong, we should check if it's our case or not
```

Listing 2: graphicsDisplay class

I could have included all the additional imports, the small complementary functions and that stuff, but I see no reason for it. This is all the code worth mentioning.