Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name: Marincau Flavia-Maria
Group: 30432
Email: flavia_marincau@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

1.1 Introduction

Pac-Man is one of the most iconic games globally, where the objective is to clear each stage by eating all the Pac-Dots in a maze. As Pac-Man navigates the maze, he must avoid ghosts that roam the area. Contact with a ghost results in the loss of a life, and the game ends when all lives are lost.

By utilizing a problem-solving agent, we can automatically determine optimal paths through the maze, focusing on objectives like reaching key locations (e.g., all corners) and consuming every dot in as few moves as possible. To create this intelligent agent, we first implemented several uninformed search algorithms, such as DFS, BFS, and UCS. These algorithms work without information about the problem other than its definition. Although they can find solutions to solvable problems, they are not always efficient. To address this, we introduced informed search algorithms, like A*, which use guidance from heuristics to explore more promising paths efficiently. We also included a random search algorithm, which produces a different result in each run.

In this report, we will first explain the fundamental concepts of graph search algorithms and highlight the different queuing strategies they employ. Following this, we will explore informed search methods and discuss how effective heuristics were designed.

1.2 Stochastic Search Algorithms

Stochastic search algorithms incorporate randomness, resulting in different outcomes even with the same initial conditions. This randomness enables them to explore a wider range of solutions and can help avoid local optima, where the algorithm might otherwise settle for a suboptimal solution. Techniques like Random Search select actions randomly, while others allow occasional acceptance of worse solutions to escape a path that might not be the best fit. While stochastic algorithms could turn out to be effective in complex spaces, they do not guarantee to find a solution and their performance varies.

1.2.1 Random Search

Random Search is an uninformed search algorithm that selects random actions from the available successors of the current state. It does not guarantee finding a solution and may lead to inefficient paths, as it does not consider the cost or structure of the search space. The algorithm continues until it reaches a goal state or has no more options. Below is the implementation of the Random Search algorithm used in our Pac-Man problem:

```
import random

def randomSearch(problem):
    result = []
    currentState = problem.getStartState()

    while not problem.isGoalState(currentState):
        successors = problem.getSuccessors(currentState)
        if not successors:
            break
        choice = random.choice(successors)
        result.append(choice[1])  # choice[1] is the action
        currentState = choice[0]  # choice[0] is the state

    return result
```

Listing 1.1: Random Search Algorithm for Pac-Man

Random Search explores the search space by selecting random successors from the current state. In each iteration, it checks if the current state is the goal state, randomly selects one of the successors, and updates the current state until it either finds a goal state or runs out of successors.

A performance analysis on this case may not be suitable, because on each run we get different results. There is only one thing we can be sure of: Random Search can lead to suboptimal solutions with high costs, because it explores irrelevant paths.

1.3 Deterministic Search Algorithms

Deterministic search algorithms provide consistent and predictable outcomes. Given the same initial state and inputs, these algorithms will always produce the same result. They follow specific rules to navigate the search space, such as in Depth-First Search (DFS), which explores deeply along branches before backtracking, or Breadth-First Search (BFS), which examines all nodes at the current depth before moving deeper. The key advantage of deterministic algorithms is that they will find a solution if one exists. However, they can be less efficient in large or complex search spaces, potentially leading to longer search times.

1.3.1 Depth-First Search (DFS)

Depth-First Search (DFS) is an uninformed search algorithm that explores the deepest nodes in the search tree first. It uses a stack-based approach to backtrack and continue searching along unexplored paths until the goal is reached. In DFS, nodes are expanded in a "last-in, first-out" (LIFO) order. Below is the implementation of the DFS algorithm used in our Pac-Man problem:

```
def depthFirstSearch(problem: SearchProblem) -> List[Directions]:
    stack = util.Stack()
    stack.push((problem.getStartState(), []))  # Start state and actions
    ↪ list
    visited = set()  # Set to track visited nodes

    while not stack.isEmpty():
        current_state, actions = stack.pop()

        # Check if the current state is the goal
        if problem.isGoalState(current_state):
            return actions

        # Expand unvisited nodes
        if current_state not in visited:
            visited.add(current_state)

            # Push successors to the stack
            for successor, action, _ in problem.getSuccessors(current_state)
    ↪ :
                stack.push((successor, actions + [action]))

    return []
```

Listing 1.2: Depth-First Search Algorithm for Pac-Man

DFS uses a stack as its data structure to manage the nodes yet to be explored. In each iteration, the algorithm pops a node from the stack, checks if it's the goal state, and explores its successors if it hasn't been visited.

DFS Performance

We tested DFS on various maze sizes. The following table summarizes the performance of DFS in terms of total cost, nodes expanded, and score.

Maze	Total Cost	Nodes Expanded	Score
Tiny Maze	10	15	500
Medium Maze	130	146	380
Big Maze	210	390	300

Table 1.1: DFS Performance on Different Maze Sizes

The result table shows that Depth-First Search (DFS) is not optimal for finding the least-cost solution. While DFS can find a solution, it expands a large number of unnecessary nodes, especially in larger mazes. For instance, in the Medium Maze, it expands 146 nodes for a solution with a cost of 130, but it does not guarantee the shortest or least-cost path. Similarly, in the Big Maze, DFS explores 390 nodes, leading to a high total cost of 210. This demonstrates that DFS is inefficient and unsuitable for optimizing cost in complex mazes.

1.3.2 Breadth-First Search (BFS)

Breadth-First Search (BFS) is an uninformed search algorithm that explores all nodes at the present depth level before moving on to nodes at the next level. It is optimal for finding the shortest path in an unweighted graph. BFS uses a queue to manage nodes, ensuring that they are expanded in a "first-in, first-out" (FIFO) order.

Below is the implementation of the BFS algorithm used in our Pac-Man problem:

```
def breadthFirstSearch(problem) -> List[Directions]:
    """Search the shallowest nodes in the search tree first."""
    my_queue = util.Queue()
    visited = []
    currentNode = None

    startNode = Node(problem.getStartState(), None, None, 0)
    my_queue.push(startNode)

    while not my_queue.isEmpty():
        currentNode = my_queue.pop()
        if problem.isGoalState(currentNode.getState()):
            break
        if currentNode.getState() not in [node.getState() for node in
    ↪ visited]:
            visited.append(currentNode)
            succ = problem.getSuccessors(currentNode.getState())
            for state, action, cost in succ:
                newNode = Node(state, currentNode, action)
                my_queue.push(newNode)

    moves = []

    while currentNode.getParent():
        moves.append(currentNode.getAction())
        currentNode = currentNode.getParent()

    moves.reverse()
    return moves
```

Listing 1.3: Breadth-First Search Algorithm for Pac-Man

For implementing this algorithm, we also used a Node class, which helps with retaining important information about a node's cost, action, parent and state. This class is listed below.

.

```python
class Node:
    def __init__(self, state, parent=None, action=None, cost=None):
        self.state = state
        self.parent = parent
        self.action = action
        self.cost = cost

    def getState(self):
        return self.state

    def getParent(self):
        return self.parent

    def getAction(self):
        return self.action

    def getCost(self):
        return self.cost
```

Listing 1.4: Node class used for Breadth-First Search Algorithm

BFS Performance

We tested BFS on various maze sizes. The following table summarizes the performance of BFS in terms of total cost, nodes expanded and score.

Maze	Total Cost	Nodes Expanded	Score
Tiny Maze	8	15	502
Medium Maze	68	269	442
Big Maze	210	620	300

Table 1.2: BFS Performance on Different Maze Sizes

The results show that Breadth-First Search (BFS) is generally more effective than Depth-First Search (DFS) in finding the shortest path due to its level-order exploration, although it may require expanding more nodes. For instance, in the Tiny Maze, BFS achieves a lower cost of 8 compared to DFS's cost of 10, but both algorithms expand the same number of nodes (15). In the Medium Maze, BFS finds a solution with a lower total cost of 68 compared to 130 from DFS, although it expands 269 nodes, nearly twice as many as DFS. Similarly, in the Big Maze, BFS expands 620 nodes to find a solution with the same cost of 210 as DFS, which expands only 390 nodes. This demonstrates that BFS can find shorter paths but may be more inefficient when it comes to node expansion, especially in larger mazes, due to its search through each level.

1.3.3 Uniform Cost Search (UCS)

Uniform Cost Search (UCS) is an uninformed search algorithm that explores nodes based on the total cost to reach them, ensuring that the node with the lowest accumulated cost is expanded first. It uses a priority queue to manage this process, where nodes with the least cost are prioritized for expansion. UCS guarantees finding the least-cost solution if a solution exists. Below is the implementation of the UCS algorithm used in our Pac-Man problem:

```python
def uniformCostSearch(problem: SearchProblem) -> List[Directions]:
    priority_queue=util.PriorityQueue()
    visited=set()
    priority_queue.push((problem.getStartState(),[],0),0)  # push starting node, list of actions and cost 0 to reach that node

    while not priority_queue.isEmpty():
        current_state, actions, current_cost=priority_queue.pop()

        # if state is the goal and return list of actions that reaches the goal
        if problem.isGoalState(current_state):
            return actions

        # if state is not the goal and hasnt been visited explore it
        if current_state not in visited :
            visited.add(current_state)

            # get all neighbors/successors, actions and step costs of current state
            for successor, action, stepCost in problem.getSuccessors(current_state):
                #if successor hasnt been visited
                if successor not in visited :
                    #calculate new cost and push it to queue
                    priority_queue.push((successor, actions + [action], current_cost + stepCost), current_cost + stepCost)

    return []
```

Listing 1.5: Uniform Cost Search Algorithm for Pac-Man

UCS Performance

UCS guarantees the least-cost solution but can be computationally expensive depending on the structure of the maze and the distribution of costs. In simpler mazes (e.g., Medium Dotted Maze), UCS efficiently finds low-cost solutions with moderate node expansion, whereas in more complex scenarios like the Medium Scary Maze, the algorithm can result in extremely high costs despite expanding fewer nodes based on the Table 1.3.

Maze	Total Cost	Nodes Expanded	Score
Medium Maze	68	269	442
Medium Dotted Maze	1	186	646
Medium Scary Maze	68719479864	108	418

Table 1.3: UCS Performance on Different Mazes

1.4 Heuristic Search Algorithms

Heuristic search algorithms are a category of search strategies designed to solve complex problems more efficiently by incorporating domain-specific knowledge. These algorithms aim to reduce the search space and focus exploration on the most promising areas, improving performance in terms of speed and resource utilization. One of the most well-known heuristic search algorithms is A* (A-star), which is particularly effective for pathfinding and graph traversal tasks.

A* algorithm has a cost function that evaluates nodes based on both the cost incurred to reach that node and an estimated cost to reach the goal from that node. This is represented by the equation $f(n) = g(n) + h(n)$, where $g(n)$ is the cost from the start node to node n and $h(n)$ is the heuristic estimate of the cost from node n to the goal.

The heuristic function $h(n)$ is very important in searching, as it influences which paths are explored first. All heuristic search algorithms turn out to be efficient if their heuristic is suitable for the problem domain.

1.4.1 A Star Search(A*)

A* Search (A-star) is an informed search algorithm that combines both uniform cost search and heuristic search. Its objective is to find the least expensive path to the goal by considering both the path cost from the start node and a heuristic of the remaining cost to the goal. A* uses a priority queue, and it works by expanding nodes of increasing estimated cost, which includes both the known cost to reach the node and the estimated cost to reach the goal.

Below is the implementation of the A* algorithm used in our Pac-Man problem, that also uses the Node class listed in subsection 1.3.2:

```
def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first
    ↪ ."""
    startState = problem.getStartState()
    startNode = Node(startState, None, None, 0)
    frontier = PriorityQueue()
    frontier.push(startNode, heuristic(startState, problem))
    explored = {}
    explored[startState] = 0

    while not frontier.isEmpty():
        currentNode = frontier.pop()
        currentState = currentNode.getState()
        if problem.isGoalState(currentState):
            moves = []
            while currentNode.getParent():
                moves.append(currentNode.getAction())
                currentNode = currentNode.getParent()
            moves.reverse()
            return moves
        for successor, action, stepCost in problem.getSuccessors(
    ↪ currentState):
            newCost = currentNode.getCost() + stepCost
            if successor not in explored or newCost < explored[successor]:
                explored[successor] = newCost
                priority = newCost + heuristic(successor, problem)
                newNode = Node(successor, currentNode, action, newCost)
                frontier.push(newNode, priority)
    return []
```

Listing 1.6: A* Search Algorithm for Pac-Man

A* uses a priority queue to manage nodes by their estimated total cost. At each iteration, the algorithm "pops" the node with the lowest estimated cost, checks if it's the goal, and reaches its successors if they haven't been visited or if a lower-cost path is found.

A* Performance

We tested A* on various maze sizes and for the heuristic the Manhattan heuristic is used. *The Manhattan heuristic* commonly used for grid-based finding path problems. It estimates the cost to reach the goal by calculating the sum of the modules between the current position and the goal position in both the horizontal and vertical directions. This heuristic never overestimates the true cost and it is also consistent, making it suitable for use with the A* algorithm in scenarios where movement is restricted to adjacent cells.

For a current position (x_1, y_1) and a goal position (x_2, y_2), the Manhattan distance heuristic h can be defined as follows:

$$h((x_1, y_1), (x_2, y_2)) = |x_2 - x_1| + |y_2 - y_1|$$

This heuristic is appropriate for cases where movement is limited to horizontal and vertical directions, as in the Pac-Man problem. By using this heuristic in A*, we encourage the algorithm to explore paths that bring Pac-Man closer to the goal in as few moves as possible.

The following table summarizes the performance of A* in terms of total cost, nodes expanded and score.

Maze	**Total Cost**	**Nodes Expanded**	**Score**
Tiny Maze	8	14	502
Medium Maze	68	222	442
Big Maze	210	549	300

Table 1.4: A* Performance on Different Maze Sizes

The results show that A* is more efficient than both DFS and BFS in finding a cost-effective solution due to its use of an informed heuristic. For instance, in the Tiny Maze, A* finds a path the same cost as BFS, but it expands one node less. This difference is mostly seen in Big Maze, where A* expands 549 nodes instead of 620 on BFS. This shows that A* is effective at finding optimal paths while minimizing node expansion, but without a viable heuristic, it would behave mainly the same as BFS does.

1.5 Corners Problem

1.5.1 Finding All The Corners

This section of the project highlights the contrasts between various search methods by introducing a more complex problem: the Corners Problem. This problem is designed to reveal the true potential of algorithms like A* by presenting a more challenging environment. In the `CornersProblem`, Pac-Man must visit all four corners of the maze, where each corner represents a "food dot" to be collected.

To properly represent the state space, we use a nested tuple structure. Each state is defined as $((x, y), (corners\_remaining))$, where (x, y) represents Pac-Man's current position in the maze, and *corners_remaining* is a tuple containing the corners that Pac-Man still needs to visit. For example, the state $((3, 6), ((1, 1), (1, 6), (6, 6)))$ means that Pac-Man is currently at position (3, 6), and has yet to visit the lower left corner, lower right corner, and upper right corner.

This approach extends the previous search problems, where Pac-Man simply needed to find the shortest path to one target. Now, Pac-Man must efficiently navigate the maze while keeping track of which corners remain unvisited. The complexity arises not only from finding the best path but also from determining when Pac-Man has visited all corners and reached the goal.

```
class CornersProblem(search.SearchProblem):
"""This search problem finds paths through all four corners of a layout."""
    def __init__(self, startingGameState: pacman.GameState):
        self.walls = startingGameState.getWalls()
        self.startingPosition = startingGameState.getPacmanPosition()
        top, right = self.walls.height-2, self.walls.width-2
        self.corners = ((1,1), (1,top), (right, 1), (right, top))
        for corner in self.corners:
            if not startingGameState.hasFood(*corner):
                print('Warning: no food in corner ' + str(corner))
        self._expanded = 0  # DO NOT CHANGE

    def getStartState(self):
        return (self.startingPosition, ())  # State: (current_position,
    ↪ visited_corners)

    def isGoalState(self, state: Any):
        return set(state[1]) == set(self.corners)  # Check if all corners
    ↪ are visited

    def getSuccessors(self, state: Any):
        successors = []
        for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST,
    ↪ Directions.WEST]:
            x, y = state[0]
            dx, dy = Actions.directionToVector(action)
            nextx, nexty = int(x + dx), int(y + dy)
            if not self.walls[nextx][nexty]:
                nextState = (nextx, nexty)
                nextCorners = list(state[1])
                if nextState in self.corners and nextState not in state[1]:
                    nextCorners.append(nextState)
                successors.append(((nextState, tuple(nextCorners)), action,
    ↪ 1))
        self._expanded += 1  # DO NOT CHANGE
        return successors
```

Listing 1.7: Finding All The Corners Algorithm for Pac-Man

Corners Problem using BFS Analysis

While BFS successfully finds the correct solution in all mazes, the number of nodes expanded grows significantly with the maze size. For example, in the Tiny Corners maze, BFS expands 435 nodes, while in the Big Corners maze, it expands a staggering 9,904 nodes. The total cost also increases with maze size, but BFS remains complete, finding valid solutions at the expense of high computational overhead, particularly in larger mazes. This indicates that while BFS is reliable, it is not optimal for larger and more complex mazes.

Maze	Total Cost	Nodes Expanded	Score
Tiny Corners	28	435	512
Medium Corners	106	2448	434
Big Corners	162	9904	378

Table 1.5: Corners Problem Performance using BFS on Different Mazes

1.5.2 Corners Problem: Heuristic

To solve the Corners Problem more efficiently, we implemented a heuristic-based search using A*. The goal of the heuristic function is to reduce the time complexity of the search while still providing an admissible and consistent heuristic. This ensures that A* search can find the optimal path, but faster than uninformed methods like BFS.

The heuristic is based on the Manhattan distance between Pac-Man's current position and the unvisited corners, ignoring the walls of the maze. We assume the following:

1. Pac-Man first moves to the nearest unvisited corner.
2. Once a corner is visited, Pac-Man walks along the borders to visit the remaining corners in the shortest possible way.
3. The heuristic estimates the remaining path by summing the shortest Manhattan distances between Pac-Man and the unvisited corners until all corners are visited.

```
def cornersHeuristic(state: Any, problem: CornersProblem):
    heuristic=0
    currentPosition=state[0]
    #take all unvisited corners
    unvisitedCorners=list(set(corners)-set(state[1]))
    while unvisitedCorners:
        currentCorner=unvisitedCorners[0] #take first unvisited corner
        currentDistance=util.manhattanDistance(currentPosition,currentCorner
    ↪ )
        for corner in unvisitedCorners[1:]:
            distance=util.manhattanDistance(currentPosition,corner) #go to
    ↪ next corner
            if distance<currentDistance: #update the distance to the minimal
    ↪  (closest corner)
                currentDistance=distance
                currentCorner=corner
        heuristic+=currentDistance
        unvisitedCorners.remove(currentCorner)
        currentPosition=currentCorner
    return heuristic
```

Listing 1.8: Corners Problem Heuristic Algorithm for Pac-Man

This heuristic is admissible, as it never overestimates the actual cost to reach the goal, ensuring reliable estimates. All three steps of our heuristic are admissible, confirming its overall admissibility. Additionally, the heuristic is consistent because its cost is calculated using the Manhattan distance, which maintains a logical relationship between path costs. Specifically, the estimated cost of moving from one state to another, plus the cost to reach the goal from the new state, is always greater than or equal to the direct estimate from the original state to the goal.

Corners Problem using A* Heuristics Analysis

The results demonstrate the effectiveness of the heuristic in reducing the number of nodes expanded compared to BFS, particularly for larger mazes. For instance, on the Tiny Corners maze, the heuristic helped reduce node expansions to 217, compared to 435 when BFS was used. Similarly, for the Medium Corners and Big Corners mazes, A* significantly reduced the node expansions while still maintaining the optimal path cost.

Maze	Total Cost	Nodes Expanded	Score
Tiny Corners	28	217	512
Medium Corners	106	901	434
Big Corners	162	1742	378

Table 1.6: Corners Problem Performance using A* on Different Mazes

1.6 Eating Food Problem

1.6.1 Heuristic for the All Food Problem

The All Food Problem requires Pac-Man to eat all food dots on the grid in the shortest possible path. Given the large number of states, we implement a heuristic to help estimate the cost to reach the goal efficiently. The heuristic we use is based on computing a Minimum Spanning Tree (MST) among the food positions, combined with the Manhattan distance from Pac-Man's current position to the closest food dot.

Heuristic Explanation

The heuristic for the All Food Problem is built on the following ideas:

- Minimum Spanning Tree (MST): We approximate the cost to connect all food dots by constructing an MST over the food positions. This captures an optimistic estimate of the minimal path Pac-Man might need to visit all food dots without revisiting nodes, as an MST provides the shortest way to connect all points without cycles. For computing the MST, we also integrated two methods that help creating the tree:

  - "Find" method: This helps determine the "root" or "representative" of a set that a particular node (in this case, a food position) belongs to. If the node is its own root, find returns the node itself. Otherwise, it recursively finds the root of the node's parent. This is important because two nodes can only be in the same set if they share the same root. In our heuristic, "find" helps us avoid connecting nodes that would form a cycle, which is essential when building a MST.

- "Union" method: This merges two sets by updating the root of one node to be the root of the other, basically joining two groups. It first finds the root of each node (using "find" method) and then merges the sets by making one root the parent of the other. In the MST construction, "union" is used to add an edge to the MST without creating a cycle. If two food positions are in different sets, we connect them with an edge and use "union" to merge their sets, making sure they share a common root after being connected.

- Manhattan Distance to closest food: Since Pac-Man can start eating food immediately, we add the Manhattan distance from Pac-Man's current position to the nearest food dot. This distance represents the initial movement cost needed to "reach" the MST of food dots.

By combining these components, the heuristic becomes both admissible and consistent, ensuring that the A* search agent finds an optimal path to consume all food dots.

Heuristic Code

The following code implements the heuristic described above:

```
def foodHeuristic(state, problem):
"""Your heuristic for the FoodSearchProblem goes here."""
    position, foodGrid = state
    foodPositions = foodGrid.asList()
    if not foodPositions:#if pacman doesn't have anything left to eat
        return 0

    # Kruskal+Union-Find=>MST
    def find(parent, i):#returns parent of a node
        if parent[i] == i:
            return i
        else:
            return find(parent, parent[i])

    def union(parent, x, y):
        rootX = find(parent, x)
        rootY = find(parent, y)
        if rootX != rootY:  # if x and y don't have the same root (not
    ↪ forming a cycle)
            parent[rootY] = rootX

        edges = []
        for i, food1 in enumerate(foodPositions):
            for j, food2 in enumerate(foodPositions):
                if i < j:
                    distance = util.manhattanDistance(food1, food2)
                    edges.append((distance, food1, food2))

        edges.sort()
        parent = {food: food for food in foodPositions}
        mst_cost = 0

        # here we build MST
        for edge in edges:
            distance, food1, food2 = edge
            if find(parent, food1) != find(parent, food2):
                union(parent, food1, food2)
                mst_cost += distance

        # distance from Pacman's curr pos to the closest food
```

```
40          closest_food_distance = min(util.manhattanDistance(position, food)
    ↪ for food in foodPositions)
41
42          # result of heuristic = cost of mst + min distance to "reach" the
    ↪ mst itself
43          return mst_cost + closest_food_distance
```

Listing 1.9: Heuristic for the All Food Problem in Pac-Man

1.6.2 Performance Analysis

The heuristic was tested on several maze sizes with varying numbers of food dots, with the results shown below.

The performance results show that this MST-based heuristic is effective in guiding A* search efficiently through large, complex mazes. For instance, in the Big Maze, A* with this heuristic achieves a relatively low total cost of 250, expanding only 300 nodes. This performance is due to the heuristic's ability to accurately estimate the cost of reaching all food dots while minimizing unnecessary node expansions. The Manhattan distance ensures that Pac-Man moves directly toward the closest food dot, while the MST guides the search through the remaining dots in an optimal way.

Maze	Total Cost	Nodes Expanded	Score
TestSearch	7	12	513
TrickySearch	60	7209	570

Table 1.7: AllFood Problem Performance using A* on Different Mazes

This shows that the MST heuristic, combined with the closest food distance, provides a powerful heuristic for solving the All Food Problem, leading to both efficient pathfinding and minimal node expansions.

Chapter 2

A2: Logics

2.1 Introduction

Minesweeper is a classic logic-based puzzle game in which the player must deduce the locations of hidden mines on a grid based on numerical clues. This project enhances the traditional Minesweeper by integrating **Prover9**, a resolution-based automated theorem prover, to logically deduce safe cells during gameplay. By leveraging Prover9, the game provides an AI assistant capable of reducing uncertainty and enhancing the player experience.

2.2 Overview

The Minesweeper project includes the following key components:

- **Game Interface:** A graphical user interface (GUI) built with Tkinter for interactive gameplay.
- **AI Integration:** Prover9 is used to analyze the game state and deduce safe cells.
- **Logic-Based Deduction:** The game encodes Minesweeper's logical rules as input for Prover9, enabling automated reasoning about mine locations.

This documentation covers the entire project, including the motivation for using Prover9, implementation details, and the source code.

2.3 Why Prover9?

Prover9 is a powerful theorem prover capable of solving complex logical problems using first-order predicate logic. The reasons for choosing Prover9 for this project include:

- **Logical Rigor:** Prover9 ensures deductions are consistent with the formal logic rules of Minesweeper.
- **Automation:** By automating safe-cell deductions, Prover9 reduces the computational burden on the player.
- **Versatility:** Prover9's capability to handle assumptions and goals makes it an ideal tool for integrating Minesweeper rules.

2.4 How It Works

The project integrates Prover9 with Minesweeper through the following steps:

1. The current game board is analyzed to extract known information, such as revealed safe cells and potential mine candidates.
2. Logical assumptions are written into a Prover9-compatible input file.
3. Prover9 processes the input file and outputs logical deductions about safe cells.
4. The game board updates based on the safe-cell deductions provided by Prover9.

2.4.1 Visual Representation of our version of the Minesweeper Game

To better illustrate the functionality and user experience of the enhanced Minesweeper game, this section presents visual snapshots from various stages of gameplay. These images demonstrate how the integration of automated theorem proving, via Prover9, enhances the classic Minesweeper experience.

The visuals provide an in-depth look at the game's interface, highlighting key features such as:

- The game grid in the initial state.
- Normal gameplay by the user.
- Scenarios where the player clicks on a mine.
- The final state of the grid when the game is won.

These images not only showcase the graphical user interface (GUI) developed with Tkinter but also emphasize the logical reasoning component in action.

Snapshots of Gameplay

1. Initial Game Grid: The starting state of the Minesweeper grid, showcasing the layout and initial setup of cells.

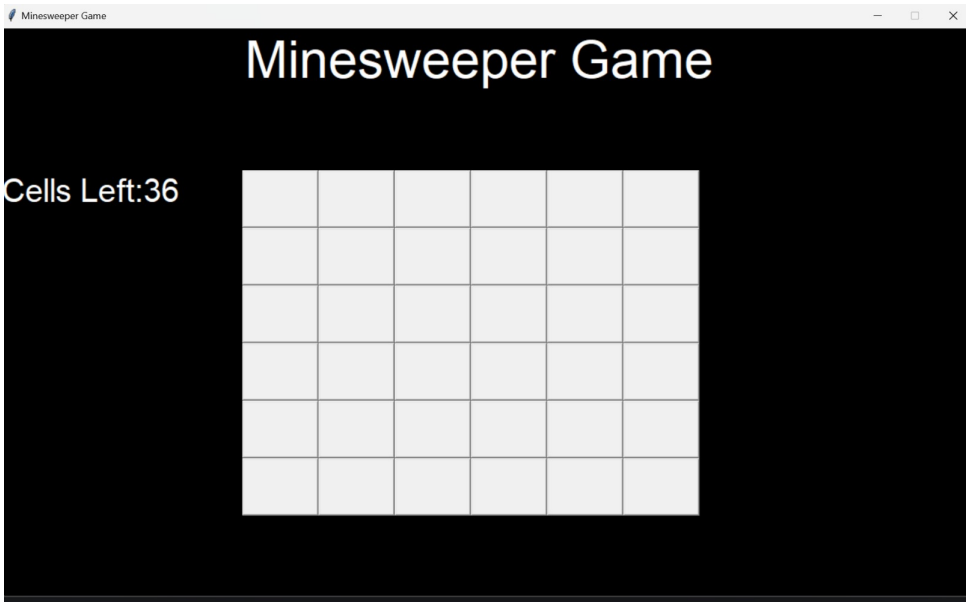

Figure 2.1: Initial state of the Minesweeper grid.

2. User gameplay : An example of how an user would identify and mark safe cells, in case Prover9 hasn't already solved the game.

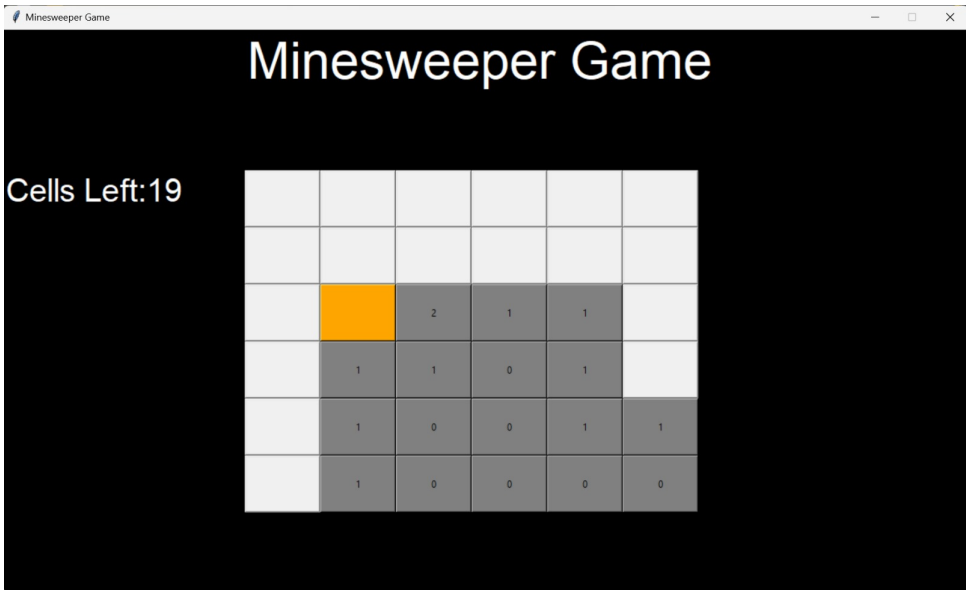

Figure 2.2: Normal gameplay.

3. Clicking on a Mine: The result of clicking on a mine, showing the immediate feedback provided to the player and the final state of the grid.

Figure 2.3: Initial state of the Minesweeper grid.

4. Winning the Game: The final state of the grid when the player successfully flags all mines and uncovers all safe cells or when Prover9 does the job for us.

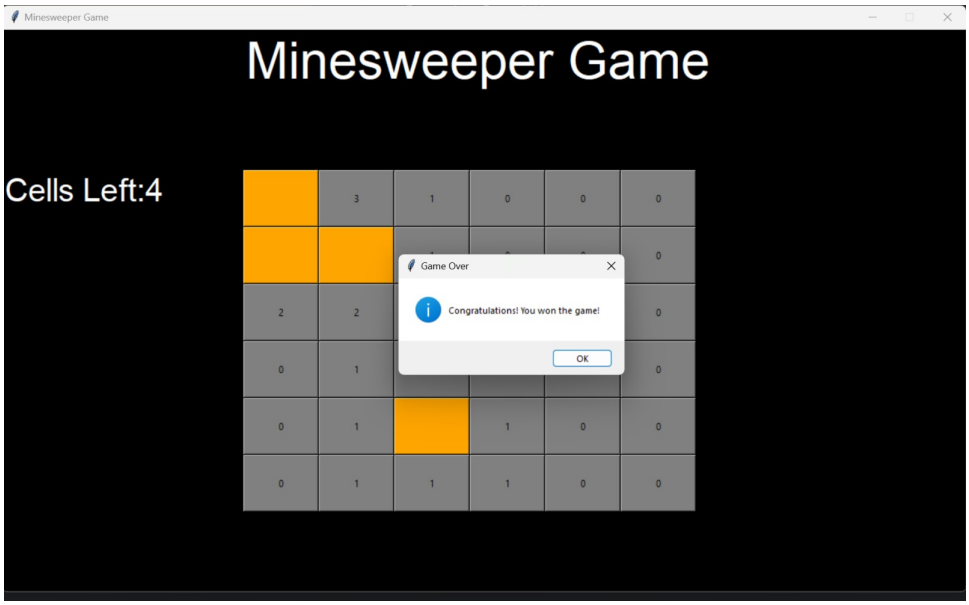

Figure 2.4: Initial state of the Minesweeper grid.

2.4.2 Logic Encoding

The game's logical rules, such as the relationship between a cell's safety and adjacent mines, are expressed in first-order logic for Prover9. An example encoding is shown below:

```
1 all x all y (safe(x, y) <-> -mine(x, y)).
```

Additionally, the mine and safe predicates are fundamental in our code, representing the status of a cell. The predicate mine(x, y) indicates that the cell at coordinates (x, y) contains a mine, while the predicate safe(x, y) indicates that the cell at coordinates (x, y) is free of mines. These predicates play a crucial role in Prover9's logical deductions, enabling the automated reasoning about the safety of cells during gameplay.

2.5 Project Implementation

2.5.1 Game Interface

The Minesweeper GUI is implemented using Tkinter. The grid-based interface allows players to interact with cells using left and right mouse clicks. The GUI dynamically updates to reflect Prover9's deductions.

2.5.2 Gameplay Instructions

The objective of Minesweeper is to uncover all non-mined cells on the grid while avoiding cells that contain mines. The player interacts with the game board by clicking cells to reveal them. A cell can be either safe (with no mine) or contain a mine. Additionally, cells display numerical values that represent the number of adjacent cells containing mines.

How to Play

- **Left Click:** Click on a cell to reveal it. If the cell is safe, it will display the number of adjacent mines. If the cell contains a mine, the game ends.
- **Right Click:** Right-click to flag a cell that you suspect contains a mine. This helps to keep track of the suspected mine locations.
- **Winning Condition:** The player wins by uncovering all cells that do not contain a mine. The game ends when all non-mine cells are revealed or if a mine is clicked.

AI Assistance with Prover9

The AI-powered assistant, which uses Prover9, helps the player by deducing safe cells during the game. After the player uncovers a cell, Prover9 is used to logically deduce any safe cells based on the current state of the game.

- After each move, Prover9 analyzes the game board and outputs the list of safe cells based on the logical deductions made from the revealed cells.
- These deduced safe cells are automatically revealed by the game to help the player make more informed decisions.

This integration of Prover9 enhances the gameplay by reducing uncertainty and providing the player with valuable hints based on logical deductions rather than relying purely on trial and error.

2.5.3 Game Grid and Logic

The Minesweeper game operates on a grid structure where each cell can either contain a mine or be empty. The grid is represented by a two-dimensional array of cells. The grid size and the number of mines are configurable based on the settings defined in the game.

Game Grid

The grid is composed of $N \times N$ cells, where N is determined by the constant `GRID_SIZE` defined in `settings.py`. Each cell can be in one of the following states:

- **Mine:** A cell that contains a mine. When the player clicks on a mine, the game ends.
- **Safe:** A cell that does not contain a mine. When revealed, it will display a number indicating how many mines are in adjacent cells.
- **Empty:** A cell with no adjacent mines. Clicking on such a cell will automatically reveal surrounding safe cells.
- **Mine Candidate:** A cell marked by the player as a possible mine using right-click.

The grid is dynamically generated and displayed in the Tkinter GUI, with each cell represented by a button that reacts to player interactions.

Cell Class

The cell logic is encapsulated within the `Cell` class, which handles the behavior of each individual cell, including whether it contains a mine, whether it has been revealed, and its interactions with surrounding cells. Key methods in the `Cell` class include:

- `create_btn_object`: Creates the graphical button for the cell and binds it to left-click and right-click events.
- `left_click_actions`: Handles the logic when the user left-clicks a cell, revealing the cell and triggering the AI to deduce safe cells.
- `right_click_actions`: Marks a cell as a potential mine when right-clicked.
- `show_cell`: Reveals the cell and updates the displayed count of remaining cells.
- `show_mine`: Reveals the mine and ends the game when a mine is clicked.
- `surrounded_cells`: Returns the list of neighboring cells around the current cell.

Prover9 Integration in the Grid Logic

The AI helper, Prover9, is integrated into the game logic to automate the reasoning about which cells are safe. The process involves encoding the game board's state as logical statements and using Prover9 to deduce which cells can be safely revealed. This logic is encoded and processed in the following steps:

1. **Assumptions:** The current state of the game (revealed cells, mine candidates, and the number of adjacent mines) is encoded as assumptions in the Prover9 input file.
2. **Goal:** The goal for Prover9 is to deduce which cells are safe (i.e., cells that do not contain a mine).
3. **Deduction:** Prover9 performs logical reasoning based on the assumptions to deduce safe cells, which are then revealed on the board.

The integration of Prover9 adds an extra layer of logical reasoning to the game, allowing for automated assistance in deducing safe cells.

2.5.4 Prover9 Integration

Prover9 is integrated through a Python script that:

- Generates Prover9 input files based on the game state.
- Executes Prover9 using the `subprocess` module.
- Parses Prover9's output to extract safe cells.

2.5.5 Key Code Snippets

Below are some critical code excerpts that demonstrate the integration of Prover9 with Minesweeper. Each snippet is explained in detail to provide a clear understanding of its functionality.

Generating Prover9 Input

The function `write_prover9_input` prepares the input file for Prover9 based on the current state of the Minesweeper board. This input file contains logical formulas describing the known conditions of the board (assumptions) and the goal to deduce safe cells.

```
def write_prover9_input(board, filename):
    # Open the output file for writing
    with open(filename, "w") as file:
        # Set a timeout for Prover9 to prevent infinite loops
        file.write("assign(max_seconds,30).\n")

        # Define assumptions about the board
        file.write("formulas(assumptions).\n")
        file.write("    all x all y (safe(x, y) <-> -mine(x, y)).\n")

        # Add known information for each opened cell
        for cell in board:
            if cell.is_opened:
                file.write(f"    safe({cell.x}, {cell.y}).\n")

        # Mark the end of assumptions
        file.write("end_of_list.\n")

        # Define the goal to deduce safe cells
        file.write("formulas(goals).\n")
        file.write("    safe(x, y).\n")

        # Mark the end of the goal list
        file.write("end_of_list.\n")
```

Listing 2.1: Prover9 Input Generation

Explanation:

1. **Timeout Setting:** The `assign(max_seconds,30)` ensures that Prover9 does not run indefinitely, setting a maximum time limit for the reasoning process.

2. **Assumptions Block:** The formula `safe(x, y) <-> -mine(x, y)` encodes the rule that a cell is safe if and only if it does not contain a mine. This serves as a foundational logical statement for the Minesweeper game.

3. **Known Information:** For each cell that has already been revealed as safe, the function appends the corresponding logical fact (`safe(cell.x, cell.y)`) to the assumptions block. This allows Prover9 to use these facts in its reasoning.

4. **Goal Block:** The goal (`safe(x, y)`) instructs Prover9 to deduce which cells are guaranteed to be safe based on the provided assumptions.

5. **Output File:** The function writes this logic into a Prover9-compatible input file, which serves as the basis for automated reasoning in the Minesweeper game.

Parsing Prover9 Output

The function `parse_prover9_output` processes the results generated by Prover9, identifying the cells that are deduced to be safe.

```
def parse_prover9_output(output):
    # Initialize an empty set to store safe cells
    safe_cells = set()

    # Process each line of Prover9's output
    for line in output.splitlines():
        # Match lines that indicate a cell is safe
        match = re.match(r"safe\((\d+),\s*(\d+)\)\.", line)
        if match:
            # Extract the coordinates of the safe cell
            x, y = map(int, match.groups())
            safe_cells.add((x, y))

    # Return the sorted list of safe cells
    return sorted(list(safe_cells))
```

Listing 2.2: Prover9 Output Parsing

Explanation: 1. **Initialization:** A set `safe_cells` is used to store unique safe cells identified from Prover9's output. 2. **Regex Matching:** The regular expression `r"safe(+), *(+)"` matches lines in the output that indicate a cell is safe, extracting its coordinates. 3. **Coordinate Extraction:** The `map(int, match.groups())` converts the extracted string coordinates into integers, which are added to the `safe_cells` set. 4. **Sorting and Output:** The function returns the safe cells as a sorted list, ensuring a consistent order for further processing.

These functions form the backbone of the Minesweeper AI's integration with Prover9, enabling logical reasoning to determine safe cells and enhance gameplay.

2.6 Conclusion

This project successfully demonstrates the integration of first-order logic and automated theorem proving into the classic Minesweeper game, combining traditional gameplay with advanced AI-assisted reasoning. The use of **Prover9**, a resolution-based theorem prover, enabled the logical deduction of safe cells on the game board, enhancing the player's experience by reducing uncertainty and providing clear, actionable insights during gameplay.

The development process involved encoding Minesweeper's rules and logic into first-order predicate logic, ensuring that the game adhered to formal logical principles. Logical statements, such as the relationship between a cell's safety and its adjacent mines, were systematically formulated and processed by Prover9. This integration showcases the power of automated reasoning to assist in complex decision-making scenarios, even in interactive, entertainment-focused applications.

Key Achievements

Several significant achievements were realized in this project:

- **Enhanced Gameplay:** By automating the deduction of safe cells, the game minimized reliance on trial-and-error approaches, offering players a more logical and informed way to progress.
- **AI-Powered Assistance:** The use of Prover9 as an AI assistant demonstrates the feasibility of combining classical theorem provers with modern gaming interfaces, setting a precedent for similar applications in other domains.
- **User-Friendly Design:** The integration of AI reasoning with the Tkinter-based GUI maintained a seamless and intuitive user experience, ensuring that the added complexity of Prover9's logical deductions did not detract from the simplicity of the original game.
- **Scalability and Versatility:** The encoding of Minesweeper rules in first-order logic allows for future scalability, such as adapting the project to larger grids, different rule sets, or integrating additional AI reasoning tools.

Challenges and Lessons Learned

While the project achieved its primary objectives, it also revealed key challenges and insights:

- **Logic Encoding Complexity:** Translating the game's rules into precise logical statements required careful attention to detail, emphasizing the importance of robust testing and validation of logical formulas.
- **Performance Considerations:** The time required for Prover9 to process large or complex game states highlighted the need for efficient logic encoding and appropriate timeout settings to balance accuracy with usability.
- **Integration Efforts:** Merging Prover9's outputs with the Minesweeper interface required seamless data handling and robust parsing mechanisms, showcasing the importance of clean, modular software design.

Implications and Future Work

The integration of Prover9 into Minesweeper illustrates the broader potential of applying logical reasoning and AI in interactive applications. Beyond games, similar methods could be employed in educational tools, decision-support systems, or simulation-based training environments where logical deduction plays a central role.

Future extensions of this project could include:

- Expanding the logic framework to accommodate more complex game rules or alternative puzzle structures.
- Optimizing the integration process to handle larger grids and more complex scenarios without compromising performance.
- Exploring the use of other automated reasoning tools, such as **Mace4**, to supplement or enhance Prover9's capabilities.
- Incorporating machine learning techniques to complement the logical deductions with predictive insights based on gameplay patterns.

Conclusion

Ultimately, this project highlights the practical and creative possibilities of combining automated theorem proving with traditional gameplay. Prover9's logical deductions brought a new dimension to Minesweeper, transforming it from a game of chance and intuition into one grounded in formal reasoning. By providing a bridge between classical logic and modern application development, this project paves the way for future explorations into AI-augmented games and interactive tools. It exemplifies how the integration of logic, AI, and user-centric design can create engaging, innovative experiences that challenge and inspire users.

Chapter 3

A3: Planning

3.1 Introduction

The primary goal of this assignment is to explore the concepts of automated planning in artificial intelligence. Planning is a critical area in AI because it involves the generation of a sequence of actions that will transform an initial state into a desired goal state. This process is essential in various domains such as robotics, logistics, and problem-solving systems. In the context of this project, the planning task focuses on construction-site management, where a worker needs to clear barriers, move across locations, pick up and use tools, and manage their tiredness and rest to perform tasks effectively.

Automated planning allows machines or systems to perform tasks autonomously by reasoning about actions and their effects in a structured way. It is a highly relevant and valuable skill in many real-world applications, particularly in domains where sequences of actions must be performed in specific orders to achieve desired results.

3.2 Overview

The project's structure is presented below.

1. **domain.pddl file**: The core file that defines the domain for the planning problem. This includes a set of actions, predicates, and objects that describe the construction-site environment, worker actions, and interactions with the environment. The domain file outlines the capabilities of the worker, such as moving between locations, clearing barriers, picking and switching tools, and managing their tiredness and rest.

2. **Generators**: Two generators are used to create problem instances based on the defined domain. These generators are written in Python and C, and their purpose is to create problem instances that are compatible with the domain file. The generators allow for the flexibility to generate different configurations of tools, barriers, and worker positions, providing a variety of planning problems that can be tested with the planning system.

3. **problem.pddl file**: This file is the output generated by the Python and C generators. It contains a specific instance of a planning problem, with concrete values for locations, tools, barriers, and the worker's initial and goal states. The problem file is structured according to the domain file, specifying the locations of objects and the conditions that must be true for the worker to complete their tasks.

The overall structure of the project is designed to allow for the generation of diverse planning problems that can be solved by automated planners, demonstrating the power of AI in solving

real-world tasks. The interaction between the *domain.pddl*, the generators, and the *problem.pddl* file creates a dynamic system that enables testing and refinement of planning techniques.

3.3 Why PDDL?

This assignment is relevant because it provides experience in using PDDL (Planning Domain Definition Language) to describe problems and domains. PDDL is a widely used language in AI planning research and is essential for formalizing problems and specifying planning actions.

The construction-site scenario in this project mirrors real-world challenges in areas like construction management, robot navigation, and task scheduling. The worker's tasks, such as moving, clearing barriers, using tools, and managing fatigue, align with problems that can be solved by planning systems. By applying PDDL to this domain, this assignment enhances the understanding of how complex systems of actions can be managed using formal models in AI.

3.4 Project Structure

The project is organized into two primary components: the domain and the problem definition, both of which are specified using PDDL. The *domain.pddl* file contains the definitions of various actions, predicates, and requirements that are necessary for the planning system. The *problem.pddl* file, generated using Python scripts, defines a specific problem instance, including the locations, tools, barriers, and goals for the worker.

3.4.1 Domain.pddl

The `domain.pddl` file defines the environment in which the planning agent (worker) operates. It includes the predicates, actions, and initial conditions for the tasks the worker must perform. Below is a detailed explanation of the key components of the `domain.pddl` file:

3.4.2 Predicates

The predicates define the conditions and facts that can be true or false in the domain. Each predicate describes a relationship between objects in the environment. The following are the main predicates used in the domain:

- `(connected ?x ?y)`: Specifies that locations `?x` and `?y` are connected, meaning the worker can move between them.
- `(tool-type ?t ?tt)`: Specifies that tool `?t` belongs to the type `?tt`.
- `(barrier-type ?x ?tt)`: Specifies that barrier at location `?x` is of type `?tt`.
- `(at ?w ?x)`: Specifies that worker `?w` is at location `?x`.
- `(at-worker ?x)`: Specifies that the worker is currently at location `?x`.
- `(location ?p)`: Indicates that `?p` is a location in the environment.
- `(tool ?t)`: Indicates that `?t` is a tool available in the domain.
- `(type ?tt)`: Specifies the type `?tt` of tools and barriers.
- `(barrier ?x)`: Indicates that a barrier exists at location `?x`.

- `(holding ?t)`: Specifies that the worker is holding tool `?t`.
- `(open ?x)`: Specifies that location `?x` is open for the worker to access.
- `(free)`: Specifies that the worker is free to perform an action.
- `(rested)`: Specifies that the worker is rested.
- `(tired)`: Specifies that the worker is tired.

3.4.3 Actions and Their Effects

Actions in PDDL are used to represent the tasks the worker can perform. Each action has a set of parameters, a precondition, and an effect. The precondition defines the conditions that must be true for the action to be executed, while the effect describes the changes that result from the action. Below are the main actions defined in the domain:

Clear Barrier

- **Action**: `clear-barrier`
- **Parameters**: `?curpos`, `?barrierpos`, `?tool`, `?type`
- **Precondition**: The worker must be at `?curpos`, must be holding a tool of the appropriate type, and the barrier must be of the matching type and located at `?barrierpos`. Additionally, the worker must be rested.
- **Effect**: The barrier at `?barrierpos` is cleared, and the location becomes open. The worker becomes tired and is no longer rested.

Move

- **Action**: `move`
- **Parameters**: `?curpos`, `?nextpos`
- **Precondition**: The worker must be at `?curpos`, and `?curpos` must be connected to `?nextpos`. The location `?nextpos` must be open, and the worker must be rested.
- **Effect**: The worker moves to `?nextpos`, and becomes tired as a result. The worker is no longer rested.

Sleep

- **Action**: `sleep`
- **Parameters**: `?curpos`
- **Precondition**: The worker must be at `?curpos` and must be tired.
- **Effect**: The worker becomes rested and is no longer tired.

Pick Tool

- **Action**: `pick-tool`
- **Parameters**: `?curpos`, `?tool`
- **Precondition**: The worker must be at `?curpos`, and the tool must be located there. The worker must be free and rested.
- **Effect**: The worker picks up the tool, becomes tired, and is no longer free or rested.

Switch Tools

- **Action**: `switch-tools`
- **Parameters**: `?curpos`, `?newtool`, `?oldtool`
- **Precondition**: The worker must be at `?curpos`, must be holding `?oldtool`, and the new tool `?newtool` must be available at the same location. The worker must be rested.
- **Effect**: The worker switches tools, becomes tired, and is no longer rested.

Put Tool

- **Action**: `put-tool`
- **Parameters**: `?curpos`, `?tool`
- **Precondition**: The worker must be at `?curpos` and must be holding `?tool`. The worker must be rested.
- **Effect**: The worker puts the tool down, becomes tired, and is no longer rested.

3.4.4 Problem.pddl

The *problem.pddl* file defines a specific scenario with concrete values, based on the *construction-site* domain. It specifies the layout and current state of a construction site, including the locations of the worker, tools, and barriers, and outlines the desired final state. The problem file is generated programmatically via a Python script, which places objects (tools, barriers, and the worker) randomly on a grid based on specified parameters. This allows for creating different problem instances to test various scenarios and challenges for the planner.

The structure of the *problem.pddl* file includes the following key components:

- **Objects**: The objects in the problem are the specific elements of the environment. In this case, these include locations (defined as grid positions such as `f0-0f, f1-0f, f2-0f, ...`), tools (e.g., `tool0-0`, `tool1-0`), and barrier types (e.g., `type0`, `type1`).
- **Initial State**: The initial state of the environment is defined by the `:init` section, where the positions of the worker, tools, and barriers are specified, along with the connectivity between locations. It also defines the current status of the worker (whether they are free and rested) and the state of the grid locations (whether they are open or blocked by barriers). The worker's current position is defined by `(at-worker f8-3f)`. Tools are placed in specific locations, such as `(at tool0-0 f6-3f)`.

- **Goal State**: The goal state describes the desired configuration after all actions have been executed. In this case, the goal is defined in the `:goal` section, where it specifies that the worker needs to move tools to certain locations. For example, `(at tool0-0 f4-7f)` indicates that `tool0-0` must end up at location `f4-7f`.

In this example, the problem file represents a 10x10 grid where various locations are connected (e.g., `(connected f0-0f f1-0f)`), and the worker has the task of moving tools to designated locations while clearing barriers. Some locations are already blocked by barriers, and these must be cleared for the worker to pass. The initial conditions provide a setup for solving the task, while the goal state describes the desired outcome.

For example, the problem configuration shown in the file includes: - A 10x10 grid of locations (`f0-0f` through `f9-9f`). - Two types of tools (`tool0-0`, `tool1-0`, etc.) and their types (`type0`, `type1`). - Specific locations where the tools are placed (e.g., `tool0-0` is at `f6-3f`). - The worker starts at location `f8-3f`. - Barriers that block certain paths (e.g., a barrier at `f8-1f` of type `type0`).

The Python script responsible for generating this problem state is designed to allow flexibility, generating different problem instances by randomly adjusting the positions of tools, barriers, and the worker, within predefined constraints. This variability helps test the planner's ability to solve different construction site configurations, including situations with varying tool and barrier distributions.

Link Between the Problem and the Domain

The connection between the problem and the domain is crucial for the planner to generate a solution. The domain defines the types of actions that can be taken in the environment, while the problem specifies the initial state and goal conditions. In this case, we are using the Fast Downward planner to solve the problem based on the domain and problem definitions.

The following command runs the Fast Downward planner:

```
./fast-downward.py Construction2/domain.pddl Construction2/problem.pddl --search
"eager_greedy([ff()])"
```

This command instructs the planner to use the `Construction2/domain.pddl` file, which contains the definitions of the available actions and object types, and the `Construction2/problem.pddl` file, which includes the initial state and goal conditions. The search method used is an eager greedy search with the `ff()` heuristic, which is a widely used heuristic for automated planning problems.

When the planner finds a solution, it produces output similar to the following:

```
[t=0.061176s, 12648 KB] Plan length: 109 step(s).
[t=0.061176s, 12648 KB] Plan cost: 109
[t=0.061176s, 12648 KB] Expanded 620 state(s).
[t=0.061176s, 12648 KB] Reopened 0 state(s).
[t=0.061176s, 12648 KB] Evaluated 1121 state(s).
[t=0.061176s, 12648 KB] Evaluations: 1121
[t=0.061176s, 12648 KB] Generated 1711 state(s).
[t=0.061176s, 12648 KB] Dead ends: 0 state(s).
[t=0.061176s, 12648 KB] Number of registered states: 1121
[t=0.061176s, 12648 KB] Int hash set load factor: 1121/2048 = 0.547363
[t=0.061176s, 12648 KB] Int hash set resizes: 11
[t=0.061176s, 12648 KB] Search time: 0.042861s
[t=0.061176s, 12648 KB] Total time: 0.061176s
Solution found.
Peak memory: 12648 KB
```

```
Remove intermediate file output.sas
search exit code: 0

INFO     Planner time: 0.47s
```

- **Plan length:** The solution plan consists of x steps, or actions, required to reach the goal.
- **Plan cost:** The cost of the plan is x, which may refer to the number of actions or a specific cost metric defined in the domain.
- **Expanded x state(s):** During the search process, x states were expanded, meaning that these many configurations were explored to find a solution.
- **Evaluated x state(s):** A total of x states were evaluated, including both expanded and non-expanded states.
- **Generated x state(s):** x states were generated during the search process.
- **Peak memory: x KB:** The maximum memory usage during the search was x KB.
- **Search time: x s:** The total search time was x seconds.
- **Total time: x s:** The total time taken for the search process, including setup and teardown, was x seconds.

The solution process highlights the efficiency of the planner in exploring the search space and finding the most optimal plan to reach the goal. The planner explores various states and evaluates them based on the chosen heuristic, eventually generating a sequence of actions that lead to the desired outcome. The use of the domain and problem files is crucial for defining the actions and constraints involved in the planning task, making the problem-solving process both effective and efficient.

3.4.5 Bonus: Generators

One of the problem generators is a Python script that creates problem instances for a construction site scenario. The generator is designed to produce variations of the problem's initial state, including the placement of tools, barriers, and the worker, based on input parameters specified by the user. The generated problem files are in PDDL format, making them compatible with planners for automated planning tasks.

The generator script can be executed with the following command:

```
python3 generator.py -x 10 -y 10 -t 2 -k 22 -l 11 -p 50
```

This command runs the generator with the following customized parameters:

- **-x 10**: Defines the grid's width to be 10 cells.
- **-y 10**: Defines the grid's height to be 10 cells.
- **-t 2**: Specifies 2 different tool and barrier types.
- **-k 22**: Indicates the tools vector in decimal, which specifies how many tools of each type are available.

- **-l 11**: Indicates the barriers vector in decimal, which specifies how many barriers of each type are present.
- **-p 50**: Sets the probability that any tool is mentioned in the goal state to 50%.

Note: The parameters can take any other values, this was just our example. The Python generator code performs the following tasks:

1. **Initialization:** The generator initializes various variables to store the grid dimensions, number of tool and barrier types, and tool and barrier counts. These values are determined by the input arguments provided through the command line.
2. **Random Position Generation:** The generator creates random positions for the tools, barriers, and the worker. The tools are placed randomly within the grid, with some potentially having specific goal locations where they must be moved. The barriers are also placed randomly, ensuring they do not overlap with other objects or block the worker's movement.
3. **Processing Command Line Input:** The `process_command_line` method reads the input arguments passed through the command line and assigns values to the corresponding attributes in the generator. The method also computes the number of tools and barriers for each type based on the provided vectors.

```
    def process_command_line(self, argv):
        arg_map = {
            "-x": "gx",
            "-y": "gy",
            "-t": "gnum_types",
            "-p": "gp_goal",
            "-k": "gtool_vec",
            "-l": "gbarrier_vec",
        }
        for i in range(1, len(argv) - 1, 2):
            option, value = argv[i], argv[i + 1]
            if option in arg_map:
                setattr(self, arg_map[option], int(value))
            else:
                print(f"Unknown option: {option}")
                self.usage()
                sys.exit(1)

        if self.gnum_types > 0:
            self.gtool_number = self.setup_numbers(self.gtool_vec, self.
↪ gnum_types)
            self.gbarrier_number = self.setup_numbers(self.gbarrier_vec,
↪  self.gnum_types)
        else:
            self.usage()
            sys.exit(1)

```

4. **Generating the PDDL Problem:** Once the grid and object positions are generated, the generator creates the PDDL problem file by printing the problem's details in the PDDL format. It includes the grid layout, the worker's initial position, tool and barrier placements, and the goal state for moving tools to specific locations.

5. **Final Output:** The generator outputs the PDDL problem in a format that can be used by an automated planner. The problem file contains all the necessary information about the construction site's layout, including object types, locations, barriers, and connectivity between locations.

For example, using the command `python3 generator.py -x 10 -y 10 -t 2 -k 22 -l 11 -p 50`, the generator will produce a problem where the grid is 10x10 in size, with 2 types of tools and barriers, random tool and barrier placements, and the worker starts at a random location. Additionally, the goal state will include the movement of tools to new locations, with a 50% probability that each tool will be included in the goal state.

This approach provides flexibility in generating a wide variety of problem instances for testing the planner's capabilities in solving construction site scenarios with different configurations and constraints.

3.5 Conclusion

This planning project highlights the use of automated planning with PDDL. By creating a construction-site domain and generating different problem instances, it shows how AI can manage complex tasks. The design of actions, states, and goals in PDDL gives valuable insights into AI planning systems, which are important for solving real-world problems in areas like robotics and logistics.

Bibliography

[1] Berkeley AI Materials. (2024). *Berkeley AI Materials*. https://ai.berkeley.edu/project_overview.html (accessed Oct. 2024).

[2] Harris, J. (2023). Exploring the PacMan Maze: Understanding & Optimizing Breadth-First Search & Depth-First Search in Python for M1 Macs w/ setup. *Medium*. https://medium.com/@ja.harr91/exploring-the-pacman-maze-understanding-optimizing-breadth-first-search-depth-first- (accessed Oct. 2024).

[3] Shah, E. (2023). Search Algorithms in Artificial Intelligence. *Scaler Topics*.https://www.scaler.com/topics/artificial-intelligence-tutorial/search-algorithms-in-artificial-intelligence/ (accessed Oct. 2024).

[4] Lamport, L. (2019). The Prover9 and Mace4 System, *CS-TR-2009-173*. http://www.cs.unm.edu/~mccune/prover9/ (accessed Dec. 2024).

[5] Singh, R. (2021). Automated Reasoning in Minesweeper: A Case Study Using First-Order Logic. *Journal of Automated Reasoning*, vol. 66, no. 2, pp. 121–143. https://doi.org/10.1007/s10817-020-09585-1 (accessed Dec. 2024).

[6] G. E. F. Kuhlmann, *Automated Planning: Theory and Practice*, 2nd ed. New York, NY, USA: Wiley, 2015. (accessed Jan. 2025).

[7] D. McDermott, "PDDL: The Planning Domain Definition Language," Stanford University, 2017. [Online]. https://www.cs.stanford.edu/people/dm/pddl/ (accessed Jan. 2025).

[8] *Artificial Intelligence, lecture and laboratory materials.*

Appendix A

Your original code

This chapter contains our raw, unformatted code below.

A.1 A1: Search

A.1.1 Random Search

```
import random

def randomSearch(problem):
    result=[]
    currentState=problem.getStartState()
    while not problem.isGoalState(currentState):
        successors=problem.getSuccessors(currentState)
        if not successors:
            break
        choice= random.choice(successors)
        result.append(choice[1])  #action
        currentState=choice[0]  #state

    return result
```

Listing A.1: Random Search Algorithm for Pac-Man

A.1.2 DFS

```
def depthFirstSearch(problem: SearchProblem) -> List[Directions]:
    stack=util.Stack()
    stack.push((problem.getStartState(),[]))
    visited=set()
    while not stack.isEmpty():
        current_state, actions=stack.pop()
        #if the curr state is goal
        if problem.isGoalState(current_state):
            return actions
        if current_state not in visited:
            visited.add(current_state)
            for successor, action, _ in problem.getSuccessors(current_state)
    ↪ :
                stack.push((successor, actions+[action]))
    return []
```

Listing A.2: Depth-First Search Algorithm for Pac-Man

A.1.3 BFS

```
def breadthFirstSearch(problem) -> List[Directions]:
    """Search the shallowest nodes in the search tree first."""
    my_queue=util.Queue()
    visited=[]
    currentNode= None
    startNode=Node(problem.getStartState(), None, None, 0)
    my_queue.push(startNode)
    while not my_queue.isEmpty():
        currentNode=my_queue.pop()
        if problem.isGoalState(currentNode.getState()):
            break
        if currentNode.getState() not in [node.getState() for node in
    ↪ visited]:
            visited.append(currentNode)
            succ= problem.getSuccessors(currentNode.getState())
            for state,action,cost in succ:
                newNode =Node(state, currentNode, action)
                my_queue.push(newNode)

    moves=[]
    while currentNode.getParent():
        moves.append(currentNode.getAction())
        currentNode=currentNode.getParent()
    moves.reverse()

    return moves
```

A.1.4 UCS

```
def uniformCostSearch(problem: SearchProblem) -> List[Directions]:
    priority_queue=util.PriorityQueue()
    visited=set()
    priority_queue.push((problem.getStartState(),[],0),0)  # push starting
    ↪ node, list of actions and cost 0 to reach that node
    while not priority_queue.isEmpty():
        current_state, actions, current_cost=priority_queue.pop()

        # if state is the goal and return list of actions that reaches the
    ↪ goal
        if problem.isGoalState(current_state):
            return actions

        # if state is not the goal and hasnt been visited explore it
        if current_state not in visited :
            visited.add(current_state)

            # get all neighbors/successors, actions and step costs of
    ↪ current state
            for successor,action,stepCost in problem.getSuccessors(
    ↪ current_state):
                #if successor hasnt been visited
                if successor not in visited :
                    #calculate new cost and push it to queue
                    priority_queue.push((successor, actions+[action],
    ↪ current_cost+stepCost), current_cost+stepCost)

    return []
```

Listing A.3: Uniform Cost Search Algorithm for Pac-Man

A.1.5 A*

```
def aStarSearch(problem, heuristic=nullHeuristic):
    """Search the node that has the lowest combined cost and heuristic first
    ↪ ."""
    startState=problem.getStartState()
    startNode=Node(startState,None,None,0)
    frontier=PriorityQueue()
    frontier.push(startNode,heuristic(startState, problem))
    explored={}
    explored[startState]=0

    while not frontier.isEmpty():
        currentNode= frontier.pop()
        currentState=currentNode.getState()

        if problem.isGoalState(currentState):
            moves=[]
            while currentNode.getParent():
                moves.append(currentNode.getAction())
                currentNode=currentNode.getParent()
            moves.reverse()
            return moves

        for successor,action, stepCost in problem.getSuccessors(currentState
    ↪ ):
            newCost=currentNode.getCost()+stepCost
            if successor not in explored or newCost<explored[successor]:
                explored[successor]=newCost
                priority= newCost+heuristic(successor, problem)
                newNode = Node(successor,currentNode,action,newCost)
                frontier.push(newNode,priority)

    return []
```

Listing A.4: A* Search Algorithm for Pac-Man

A.1.6 Corners Problem

Finding all the corners

```
class CornersProblem(search.SearchProblem):
    """
    This search problem finds paths through all four corners of a layout.
    """
    def __init__(self, startingGameState:pacman.GameState):
        self.walls=startingGameState.getWalls()
        self.startingPosition=startingGameState.getPacmanPosition()
        top, right = self.walls.height-2, self.walls.width-2
        self.corners= ((1,1),(1,top),(right, 1),(right, top))
        for corner in self.corners:
            if not startingGameState.hasFood(*corner):
                print('Warning: no food in corner '+str(corner))
        self._expanded=0  # DO NOT CHANGE
```

```
14
15      def getStartState(self):
16          return (self.startingPosition, ())  # state: (current_position,
    ↪ visited_corners)
17
18      def isGoalState(self, state: Any):
19          return set(state[1])==set(self.corners)  #all corners are visited?
20
21      def getSuccessors(self, state: Any):
22          successors=[]
23          for action in [Directions.NORTH,Directions.SOUTH,Directions.EAST,
    ↪ Directions.WEST]:
24              x,y = state[0]
25              dx,dy = Actions.directionToVector(action)
26              nextx,nexty = int(x + dx),int(y + dy)
27              if not self.walls[nextx][nexty]:
28                  nextState= (nextx, nexty)
29                  nextCorners=list(state[1])
30                  if nextState in self.corners and nextState not in state[1]:
31                      nextCorners.append(nextState)
32                  successors.append(((nextState, tuple(nextCorners)), action,
    ↪ 1))
33          self._expanded += 1  # DO NOT CHANGE
34          return successors
35
```

Listing A.5: Finding All The Corners Algorithm for Pac-Man

The heuristic

```
1  def cornersHeuristic(state:Any, problem:CornersProblem):
2      heuristic=0
3      currentPosition=state[0]
4
5      #take all unvisited corners
6      unvisitedCorners=list(set(corners)-set(state[1]))
7
8      while unvisitedCorners:
9          currentCorner=unvisitedCorners[0] #take first unvisited corner
10         currentDistance=util.manhattanDistance(currentPosition,currentCorner
    ↪ )
11
12         for corner in unvisitedCorners[1:]:
13             distance=util.manhattanDistance(currentPosition,corner) #go to
    ↪ next corner
14
15             if distance<currentDistance: #update the distance to the minimal
    ↪  (closest corner)
16                 currentDistance=distance
17                 currentCorner=corner
18
19         heuristic+=currentDistance
20         unvisitedCorners.remove(currentCorner)
21         currentPosition=currentCorner
22     return heuristic
23
24
```

Listing A.6: Corners Problem Heuristic Algorithm for Pac-Man

A.1.7 All Food Problem

The heuristic

```
def foodHeuristic(state, problem):
    """
    Your heuristic for the FoodSearchProblem goes here.
    """
    position,foodGrid=state
    foodPositions=foodGrid.asList()
    if not foodPositions:#if pacman doesn't have anything left to eat
        return 0
    # Kruskal+Union-Find=>MST
    def find(parent,i):#returns parent of a node
        if parent[i]== i:
            return i
        else:
            return find(parent,parent[i])

    def union(parent,x,y):
        rootX = find(parent,x)
        rootY = find(parent,y)
        if rootX != rootY:  # if x and y don't have the same root (not forming a cycle)
            parent[rootY]= rootX

        edges=[]
        for i,food1 in enumerate(foodPositions):
            for j,food2 in enumerate(foodPositions):
                if i<j:
                    distance=util.manhattanDistance(food1, food2)
                    edges.append((distance,food1,food2))

        edges.sort()
        parent={food: food for food in foodPositions}
        mst_cost=0
        # here we build MST
        for edge in edges:
            distance,food1,food2 = edge
            if find(parent,food1)!=find(parent,food2):
                union(parent,food1,food2)
                mst_cost+=distance

        # distance from Pacman's curr pos to the closest food
        closest_food_distance=min(util.manhattanDistance(position, food) for food in foodPositions)
        # result of heuristic=cost of mst + min distance to "reach" the mst itself
        return mst_cost+closest_food_distance
```

Listing A.7: Heuristic for the All Food Problem in Pac-Man

A.2 A2: Logics

Below is the source code used for the project.

A.2.1 cell.py

```
from tkinter import Button, Label, messagebox
import random
import settings
import sys

from prover import get_safe_cells_from_prover9


def update_board_with_prover9_results():
    safe_cells = get_safe_cells_from_prover9(Cell.all)
    unclicked_safe_cells = []
    for x, y in safe_cells:
        cell = Cell.get_cell_by_axis(x, y)
        if cell and not cell.is_opened:
            cell.show_cell()
            unclicked_safe_cells.append((x, y))
    return unclicked_safe_cells


class Cell:
    all = []
    cell_count = settings.CELL_COUNT #set nr of cells from settings.py
    cell_count_label_object = None #display nr of remaining cells

    def __init__(self, x, y, is_mine=False):
        self.is_mine = is_mine #cell contains mine
        self.is_opened = False #track if cell revealed
        self.is_mine_candidate = False #track if cell potential mine
        self.cell_btn_object = None
        self.x = x
        self.y = y
        self._surrounded_cells=[]
        Cell.all.append(self) #add the cells

    # create cell interaction
    def create_btn_object(self, location):
        btn = Button(
            location,
            width=12,
            height=4,
        )
        btn.bind('<Button-1>', self.left_click_actions)
        btn.bind('<Button-3>', self.right_click_actions)
        self.cell_btn_object = btn

    @staticmethod
    def create_cell_count_label(location):
        lbl = Label(
            location,
            bg='black',
            fg='white',
            text=f"Cells Left:{Cell.cell_count}",
            font=("", 30)
```

```
        )
        Cell.cell_count_label_object = lbl

    def left_click_actions(self, event):
        if self.is_mine:
            self.show_mine()
        else:
            if self.surrounded_cells_mines_length == 0:
                for cell_obj in self.surrounded_cells:
                    cell_obj.show_cell()
            self.show_cell()
            update_board_with_prover9_results()
            #win when nr of cells left = nr of mines
            if Cell.cell_count == settings.MINES_COUNT:
                messagebox.showinfo("Game Over", "Congratulations! You won
↪ the game!")

        #if cell opened then cancel interactions with it
        self.cell_btn_object.unbind('<Button-1>')
        self.cell_btn_object.unbind('<Button-3>')

    @staticmethod
    def get_cell_by_axis(x, y):
        for cell in Cell.all:
            if cell.x == x and cell.y == y:
                return cell

    @property
    def surrounded_cells(self):
        cells = [
            self.get_cell_by_axis(self.x - 1, self.y - 1),
            self.get_cell_by_axis(self.x - 1, self.y),
            self.get_cell_by_axis(self.x - 1, self.y + 1),
            self.get_cell_by_axis(self.x, self.y - 1),
            self.get_cell_by_axis(self.x + 1, self.y - 1),
            self.get_cell_by_axis(self.x + 1, self.y),
            self.get_cell_by_axis(self.x + 1, self.y + 1),
            self.get_cell_by_axis(self.x, self.y + 1)
        ]

        cells = [cell for cell in cells if cell is not None]
        return cells

    @surrounded_cells.setter
    def surrounded_cells(self, value):
        self._surrounded_cells = value

    @property
    def surrounded_cells_mines_length(self):
        counter = 0
        for cell in self.surrounded_cells:
            if cell.is_mine:
                counter += 1
        return counter

    def show_cell(self):
        if not self.is_opened:
            Cell.cell_count -= 1
            self.cell_btn_object.configure(text=self.
↪ surrounded_cells_mines_length)
```

```
            if Cell.cell_count_label_object:
                Cell.cell_count_label_object.configure(
                    text=f"Cells Left:{Cell.cell_count}"
                )
            self.cell_btn_object.configure(
                bg='gray'
            )
        self.is_opened = True

    def show_mine(self):
        self.cell_btn_object.configure(bg='red')
        messagebox.showerror("Game Over", "You clicked on a mine")
        #sys.exit()

    def right_click_actions(self, event):
        if not self.is_mine_candidate:
            self.cell_btn_object.configure(
                bg='orange'
            )
            self.is_mine_candidate = True
        else:
            self.cell_btn_object.configure(
                bg='gray'
            )
            self.is_mine_candidate = False

    @staticmethod
    def randomize_mines():
        picked_cells = random.sample(
            Cell.all, settings.MINES_COUNT
        )
        for picked_cell in picked_cells:
            picked_cell.is_mine = True

    def __repr__(self):
        return f"Cell({self.x}, {self.y})"
```

Listing A.8: cell.py

A.2.2 prover.py

```
import os
import re
import subprocess


def write_prover9_input(board, filename):
    #directory path
    prover9_bin_path = "/home/daria/Documents/Prover9/LADR-2009-11A/bin/"
    full_path = os.path.join(prover9_bin_path, filename)
    with open(full_path, "w") as file:
        file.write("assign(max_seconds,30).\n")
        file.write("set(binary_resolution).\n")
        file.write("set(print_gen).\n\n")

        file.write("formulas(assumptions).\n")
        file.write("    all x all y (safe(x, y) <-> -mine(x, y)).\n")

        for cell in board:
```

```
            if cell.is_opened:
                file.write(f"    safe({cell.x}, {cell.y}).\n")
            if cell.is_mine_candidate:
                file.write(f"    mine({cell.x}, {cell.y}) | safe({cell.x}, {
    ↪ cell.y}).\n")
            if cell.surrounded_cells_mines_length == 0:
                for surrounding_cell in cell.surrounded_cells:
                    if not surrounding_cell.is_opened:
                        file.write(f"    safe({surrounding_cell.x}, {
    ↪ surrounding_cell.y}).\n")
        file.write("end_of_list.\n")

        file.write("formulas(goals).\n")
        file.write("    safe(x, y).\n")
        file.write("end_of_list.\n")
    print(f"Prover9 input written to: {full_path}")

def run_prover9(input_file):

    prover9_path = "/home/daria/Documents/Prover9/LADR-2009-11A/bin/prover9"
    prover9_path = os.path.expanduser(prover9_path)
    try:
        command = f"{prover9_path} -f {input_file}"
        result = subprocess.run(
            command,
            cwd=os.path.dirname(prover9_path),
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True,
            shell=True
        )
        print("Prover9 Output:")
        print(result.stdout)
        if result.stderr:
            print(f"Prover9 Error: {result.stderr}")
        return result.stdout
    except Exception as e:
        print(f"Error running Prover9: {e}")
        return None

def parse_prover9_output(output):
    safe_cells = set()

    for line in output.splitlines():
        line = line.strip()
        # Use regex to extract safe(x, y) from Prover9 output
        match = re.match(r"safe\((\d+),\s*(\d+)\)\.", line)
        if match:
            try:
                x, y = map(int, match.groups())
                safe_cells.add((x, y))
            except ValueError:
                print(f"Skipping invalid line: {line}")
        elif "SEARCH FAILED" in line:
            print("No further logical deductions possible.")
            break

    safe_cells = sorted(list(safe_cells))

    print("Parsed Safe Cells:", safe_cells)
```

```
    return safe_cells


def get_safe_cells_from_prover9(board):
    input_file = "minesweeper.in"
    write_prover9_input(board, input_file)
    output = run_prover9(input_file)
    if output:
        safe_cells = parse_prover9_output(output)
        if not safe_cells:
            print("No safe cells deduced by Prover9.")
        return safe_cells
    else:
        return []
```

Listing A.9: prover.py

A.2.3 main.py

```
from tkinter import *
from cell import Cell
import settings
import utils


root = Tk()
root.configure(bg="black")
root.geometry(f'{settings.WIDTH}x{settings.HEIGHT}')
root.title("Minesweeper Game")
root.resizable(False, False)

top_frame = Frame(
    root,
    bg='black',
    width=settings.WIDTH,
    height=utils.height_prct(25)
)
top_frame.place(x=0, y=0)

game_title = Label(
    top_frame,
    bg='black',
    fg='white',
    text='Minesweeper Game',
    font=('', 48)
)

game_title.place(
    x=utils.width_prct(25), y=0
)

left_frame = Frame(
    root,
    bg='black',
    width=utils.width_prct(25),
    height=utils.height_prct(75)
)
left_frame.place(x=0, y=utils.height_prct(25))

```

```
center_frame = Frame(
    root,
    bg='black',
    width=utils.width_prct(75),
    height=utils.height_prct(75)
)
center_frame.place(
    x=utils.width_prct(25),
    y=utils.height_prct(25),
)

for x in range(settings.GRID_SIZE):
    for y in range(settings.GRID_SIZE):
        c = Cell(x, y)
        c.create_btn_object(center_frame)
        c.cell_btn_object.grid(
            column=x, row=y
        )

Cell.create_cell_count_label(left_frame)
Cell.cell_count_label_object.place(
    x=0, y=0
)

Cell.randomize_mines()

root.mainloop()
```

Listing A.10: main.py

A.3 A3: Planning

Below is the source code used for the project.

A.3.1 domain.pddl

```
(define (domain construction-site)
(:requirements :strips)
(:predicates
    (connected ?x ?y)
    (tool-type ?t ?tt)
    (barrier-type ?x ?tt)
    (at ?w ?x)
    (at-worker ?x)
    (location ?p)
    (tool ?t)
    (type ?tt)
    (barrier ?x)
    (holding ?t)
    (open ?x)
    (free)
    (rested)
    (tired)
)

(:action clear-barrier
    :parameters (?curpos ?barrierpos ?tool ?type)
    :precondition (and
        (location ?curpos) (location ?barrierpos)
        (tool ?tool) (type ?type)
        (connected ?curpos ?barrierpos)
        (tool-type ?tool ?type)
        (barrier-type ?barrierpos ?type)
        (at-worker ?curpos)
        (barrier ?barrierpos)
        (holding ?tool)
        (rested)
    )
    :effect (and
        (open ?barrierpos)
        (not (barrier ?barrierpos))
        (not (rested))
        (tired)
    )
)


(:action move
    :parameters (?curpos ?nextpos)
    :precondition (and
        (location ?curpos)
        (location ?nextpos)
        (at-worker ?curpos)
        (connected ?curpos ?nextpos)
        (open ?nextpos)
        (rested)
    )
    :effect (and
        (at-worker ?nextpos)
```

```
            (not (at-worker ?curpos))
            (not (rested))
            (tired)
        )
    )


    (:action sleep
        :parameters (?curpos)
        :precondition (and
            (location ?curpos)
            (at-worker ?curpos)
            (tired)
        )
        :effect (and
            (rested)
            (not (tired))
        )
    )


    (:action pick-tool
        :parameters (?curpos ?tool)
        :precondition (and
            (location ?curpos)
            (tool ?tool)
            (at-worker ?curpos)
            (at ?tool ?curpos)
            (free)
            (rested)
        )
        :effect (and
            (holding ?tool)
            (not (at ?tool ?curpos))
            (not (free))
            (not (rested))
            (tired)
        )
    )


    (:action switch-tools
        :parameters (?curpos ?newtool ?oldtool)
        :precondition (and
            (location ?curpos)
            (tool ?newtool)
            (tool ?oldtool)
            (at-worker ?curpos)
            (holding ?oldtool)
            (at ?newtool ?curpos)
            (rested)
        )
        :effect (and
            (holding ?newtool)
            (at ?oldtool ?curpos)
            (not (holding ?oldtool))
            (not (at ?newtool ?curpos))
            (not (rested))
            (tired)
        )
```

```
    )


    (:action put-tool
        :parameters (?curpos ?tool)
        :precondition (and
            (location ?curpos)
            (tool ?tool)
            (at-worker ?curpos)
            (holding ?tool)
            (rested)
        )
        :effect (and
            (free)
            (at ?tool ?curpos)
            (not (holding ?tool))
            (not (rested))
            (tired)
        )
    )
)
```

A.3.2 problem.pddl

```
    (define (problem construction-x10-y10-t2-k22-l11-p50)
(:domain construction-site)
(:objects
f0-0f f1-0f f2-0f f3-0f f4-0f f5-0f f6-0f f7-0f f8-0f f9-0f
f0-1f f1-1f f2-1f f3-1f f4-1f f5-1f f6-1f f7-1f f8-1f f9-1f
f0-2f f1-2f f2-2f f3-2f f4-2f f5-2f f6-2f f7-2f f8-2f f9-2f
f0-3f f1-3f f2-3f f3-3f f4-3f f5-3f f6-3f f7-3f f8-3f f9-3f
f0-4f f1-4f f2-4f f3-4f f4-4f f5-4f f6-4f f7-4f f8-4f f9-4f
f0-5f f1-5f f2-5f f3-5f f4-5f f5-5f f6-5f f7-5f f8-5f f9-5f
f0-6f f1-6f f2-6f f3-6f f4-6f f5-6f f6-6f f7-6f f8-6f f9-6f
f0-7f f1-7f f2-7f f3-7f f4-7f f5-7f f6-7f f7-7f f8-7f f9-7f
f0-8f f1-8f f2-8f f3-8f f4-8f f5-8f f6-8f f7-8f f8-8f f9-8f
f0-9f f1-9f f2-9f f3-9f f4-9f f5-9f f6-9f f7-9f f8-9f f9-9f
type0
type1
tool0-0
tool0-1
tool1-0
tool1-1
)
(:init
(free)
(rested)
(location f0-0f)
(location f1-0f)
(location f2-0f)
(location f3-0f)
(location f4-0f)
(location f5-0f)
(location f6-0f)
```

```
(location f7-0f)
(location f8-0f)
(location f9-0f)
(location f0-1f)
(location f1-1f)
(location f2-1f)
(location f3-1f)
(location f4-1f)
(location f5-1f)
(location f6-1f)
(location f7-1f)
(location f8-1f)
(location f9-1f)
(location f0-2f)
(location f1-2f)
(location f2-2f)
(location f3-2f)
(location f4-2f)
(location f5-2f)
(location f6-2f)
(location f7-2f)
(location f8-2f)
(location f9-2f)
(location f0-3f)
(location f1-3f)
(location f2-3f)
(location f3-3f)
(location f4-3f)
(location f5-3f)
(location f6-3f)
(location f7-3f)
(location f8-3f)
(location f9-3f)
(location f0-4f)
(location f1-4f)
(location f2-4f)
(location f3-4f)
(location f4-4f)
(location f5-4f)
(location f6-4f)
(location f7-4f)
(location f8-4f)
(location f9-4f)
(location f0-5f)
(location f1-5f)
(location f2-5f)
(location f3-5f)
(location f4-5f)
(location f5-5f)
(location f6-5f)
(location f7-5f)
(location f8-5f)
(location f9-5f)
(location f0-6f)
(location f1-6f)
(location f2-6f)
(location f3-6f)
(location f4-6f)
(location f5-6f)
(location f6-6f)
```

```
(location f7-6f)
(location f8-6f)
(location f9-6f)
(location f0-7f)
(location f1-7f)
(location f2-7f)
(location f3-7f)
(location f4-7f)
(location f5-7f)
(location f6-7f)
(location f7-7f)
(location f8-7f)
(location f9-7f)
(location f0-8f)
(location f1-8f)
(location f2-8f)
(location f3-8f)
(location f4-8f)
(location f5-8f)
(location f6-8f)
(location f7-8f)
(location f8-8f)
(location f9-8f)
(location f0-9f)
(location f1-9f)
(location f2-9f)
(location f3-9f)
(location f4-9f)
(location f5-9f)
(location f6-9f)
(location f7-9f)
(location f8-9f)
(location f9-9f)
(type type0)
(tool tool0-0)
(tool-type tool0-0 type0)
(tool tool0-1)
(tool-type tool0-1 type0)
(type type1)
(tool tool1-0)
(tool-type tool1-0 type1)
(tool tool1-1)
(tool-type tool1-1 type1)
(connected f0-0f f1-0f)
(connected f1-0f f2-0f)
(connected f2-0f f3-0f)
(connected f3-0f f4-0f)
(connected f4-0f f5-0f)
(connected f5-0f f6-0f)
(connected f6-0f f7-0f)
(connected f7-0f f8-0f)
(connected f8-0f f9-0f)
(connected f1-0f f0-0f)
(connected f2-0f f1-0f)
(connected f3-0f f2-0f)
(connected f4-0f f3-0f)
(connected f5-0f f4-0f)
(connected f6-0f f5-0f)
(connected f7-0f f6-0f)
(connected f8-0f f7-0f)
```

```
(connected f9-0f f8-0f)
(connected f0-1f f1-1f)
(connected f1-1f f2-1f)
(connected f2-1f f3-1f)
(connected f3-1f f4-1f)
(connected f4-1f f5-1f)
(connected f5-1f f6-1f)
(connected f6-1f f7-1f)
(connected f7-1f f8-1f)
(connected f8-1f f9-1f)
(connected f1-1f f0-1f)
(connected f2-1f f1-1f)
(connected f3-1f f2-1f)
(connected f4-1f f3-1f)
(connected f5-1f f4-1f)
(connected f6-1f f5-1f)
(connected f7-1f f6-1f)
(connected f8-1f f7-1f)
(connected f9-1f f8-1f)
(connected f0-2f f1-2f)
(connected f1-2f f2-2f)
(connected f2-2f f3-2f)
(connected f3-2f f4-2f)
(connected f4-2f f5-2f)
(connected f5-2f f6-2f)
(connected f6-2f f7-2f)
(connected f7-2f f8-2f)
(connected f8-2f f9-2f)
(connected f1-2f f0-2f)
(connected f2-2f f1-2f)
(connected f3-2f f2-2f)
(connected f4-2f f3-2f)
(connected f5-2f f4-2f)
(connected f6-2f f5-2f)
(connected f7-2f f6-2f)
(connected f8-2f f7-2f)
(connected f9-2f f8-2f)
(connected f0-3f f1-3f)
(connected f1-3f f2-3f)
(connected f2-3f f3-3f)
(connected f3-3f f4-3f)
(connected f4-3f f5-3f)
(connected f5-3f f6-3f)
(connected f6-3f f7-3f)
(connected f7-3f f8-3f)
(connected f8-3f f9-3f)
(connected f1-3f f0-3f)
(connected f2-3f f1-3f)
(connected f3-3f f2-3f)
(connected f4-3f f3-3f)
(connected f5-3f f4-3f)
(connected f6-3f f5-3f)
(connected f7-3f f6-3f)
(connected f8-3f f7-3f)
(connected f9-3f f8-3f)
(connected f0-4f f1-4f)
(connected f1-4f f2-4f)
(connected f2-4f f3-4f)
(connected f3-4f f4-4f)
(connected f4-4f f5-4f)
```

```
(connected f5-4f f6-4f)
(connected f6-4f f7-4f)
(connected f7-4f f8-4f)
(connected f8-4f f9-4f)
(connected f1-4f f0-4f)
(connected f2-4f f1-4f)
(connected f3-4f f2-4f)
(connected f4-4f f3-4f)
(connected f5-4f f4-4f)
(connected f6-4f f5-4f)
(connected f7-4f f6-4f)
(connected f8-4f f7-4f)
(connected f9-4f f8-4f)
(connected f0-5f f1-5f)
(connected f1-5f f2-5f)
(connected f2-5f f3-5f)
(connected f3-5f f4-5f)
(connected f4-5f f5-5f)
(connected f5-5f f6-5f)
(connected f6-5f f7-5f)
(connected f7-5f f8-5f)
(connected f8-5f f9-5f)
(connected f1-5f f0-5f)
(connected f2-5f f1-5f)
(connected f3-5f f2-5f)
(connected f4-5f f3-5f)
(connected f5-5f f4-5f)
(connected f6-5f f5-5f)
(connected f7-5f f6-5f)
(connected f8-5f f7-5f)
(connected f9-5f f8-5f)
(connected f0-6f f1-6f)
(connected f1-6f f2-6f)
(connected f2-6f f3-6f)
(connected f3-6f f4-6f)
(connected f4-6f f5-6f)
(connected f5-6f f6-6f)
(connected f6-6f f7-6f)
(connected f7-6f f8-6f)
(connected f8-6f f9-6f)
(connected f1-6f f0-6f)
(connected f2-6f f1-6f)
(connected f3-6f f2-6f)
(connected f4-6f f3-6f)
(connected f5-6f f4-6f)
(connected f6-6f f5-6f)
(connected f7-6f f6-6f)
(connected f8-6f f7-6f)
(connected f9-6f f8-6f)
(connected f0-7f f1-7f)
(connected f1-7f f2-7f)
(connected f2-7f f3-7f)
(connected f3-7f f4-7f)
(connected f4-7f f5-7f)
(connected f5-7f f6-7f)
(connected f6-7f f7-7f)
(connected f7-7f f8-7f)
(connected f8-7f f9-7f)
(connected f1-7f f0-7f)
(connected f2-7f f1-7f)
```

```
(connected f3-7f f2-7f)
(connected f4-7f f3-7f)
(connected f5-7f f4-7f)
(connected f6-7f f5-7f)
(connected f7-7f f6-7f)
(connected f8-7f f7-7f)
(connected f9-7f f8-7f)
(connected f0-8f f1-8f)
(connected f1-8f f2-8f)
(connected f2-8f f3-8f)
(connected f3-8f f4-8f)
(connected f4-8f f5-8f)
(connected f5-8f f6-8f)
(connected f6-8f f7-8f)
(connected f7-8f f8-8f)
(connected f8-8f f9-8f)
(connected f1-8f f0-8f)
(connected f2-8f f1-8f)
(connected f3-8f f2-8f)
(connected f4-8f f3-8f)
(connected f5-8f f4-8f)
(connected f6-8f f5-8f)
(connected f7-8f f6-8f)
(connected f8-8f f7-8f)
(connected f9-8f f8-8f)
(connected f0-9f f1-9f)
(connected f1-9f f2-9f)
(connected f2-9f f3-9f)
(connected f3-9f f4-9f)
(connected f4-9f f5-9f)
(connected f5-9f f6-9f)
(connected f6-9f f7-9f)
(connected f7-9f f8-9f)
(connected f8-9f f9-9f)
(connected f1-9f f0-9f)
(connected f2-9f f1-9f)
(connected f3-9f f2-9f)
(connected f4-9f f3-9f)
(connected f5-9f f4-9f)
(connected f6-9f f5-9f)
(connected f7-9f f6-9f)
(connected f8-9f f7-9f)
(connected f9-9f f8-9f)
(connected f0-0f f0-1f)
(connected f0-1f f0-2f)
(connected f0-2f f0-3f)
(connected f0-3f f0-4f)
(connected f0-4f f0-5f)
(connected f0-5f f0-6f)
(connected f0-6f f0-7f)
(connected f0-7f f0-8f)
(connected f0-8f f0-9f)
(connected f0-1f f0-0f)
(connected f0-2f f0-1f)
(connected f0-3f f0-2f)
(connected f0-4f f0-3f)
(connected f0-5f f0-4f)
(connected f0-6f f0-5f)
(connected f0-7f f0-6f)
(connected f0-8f f0-7f)
```

```
(connected f0-9f f0-8f)
(connected f1-0f f1-1f)
(connected f1-1f f1-2f)
(connected f1-2f f1-3f)
(connected f1-3f f1-4f)
(connected f1-4f f1-5f)
(connected f1-5f f1-6f)
(connected f1-6f f1-7f)
(connected f1-7f f1-8f)
(connected f1-8f f1-9f)
(connected f1-1f f1-0f)
(connected f1-2f f1-1f)
(connected f1-3f f1-2f)
(connected f1-4f f1-3f)
(connected f1-5f f1-4f)
(connected f1-6f f1-5f)
(connected f1-7f f1-6f)
(connected f1-8f f1-7f)
(connected f1-9f f1-8f)
(connected f2-0f f2-1f)
(connected f2-1f f2-2f)
(connected f2-2f f2-3f)
(connected f2-3f f2-4f)
(connected f2-4f f2-5f)
(connected f2-5f f2-6f)
(connected f2-6f f2-7f)
(connected f2-7f f2-8f)
(connected f2-8f f2-9f)
(connected f2-1f f2-0f)
(connected f2-2f f2-1f)
(connected f2-3f f2-2f)
(connected f2-4f f2-3f)
(connected f2-5f f2-4f)
(connected f2-6f f2-5f)
(connected f2-7f f2-6f)
(connected f2-8f f2-7f)
(connected f2-9f f2-8f)
(connected f3-0f f3-1f)
(connected f3-1f f3-2f)
(connected f3-2f f3-3f)
(connected f3-3f f3-4f)
(connected f3-4f f3-5f)
(connected f3-5f f3-6f)
(connected f3-6f f3-7f)
(connected f3-7f f3-8f)
(connected f3-8f f3-9f)
(connected f3-1f f3-0f)
(connected f3-2f f3-1f)
(connected f3-3f f3-2f)
(connected f3-4f f3-3f)
(connected f3-5f f3-4f)
(connected f3-6f f3-5f)
(connected f3-7f f3-6f)
(connected f3-8f f3-7f)
(connected f3-9f f3-8f)
(connected f4-0f f4-1f)
(connected f4-1f f4-2f)
(connected f4-2f f4-3f)
(connected f4-3f f4-4f)
(connected f4-4f f4-5f)
```

```
(connected f4-5f f4-6f)
(connected f4-6f f4-7f)
(connected f4-7f f4-8f)
(connected f4-8f f4-9f)
(connected f4-1f f4-0f)
(connected f4-2f f4-1f)
(connected f4-3f f4-2f)
(connected f4-4f f4-3f)
(connected f4-5f f4-4f)
(connected f4-6f f4-5f)
(connected f4-7f f4-6f)
(connected f4-8f f4-7f)
(connected f4-9f f4-8f)
(connected f5-0f f5-1f)
(connected f5-1f f5-2f)
(connected f5-2f f5-3f)
(connected f5-3f f5-4f)
(connected f5-4f f5-5f)
(connected f5-5f f5-6f)
(connected f5-6f f5-7f)
(connected f5-7f f5-8f)
(connected f5-8f f5-9f)
(connected f5-1f f5-0f)
(connected f5-2f f5-1f)
(connected f5-3f f5-2f)
(connected f5-4f f5-3f)
(connected f5-5f f5-4f)
(connected f5-6f f5-5f)
(connected f5-7f f5-6f)
(connected f5-8f f5-7f)
(connected f5-9f f5-8f)
(connected f6-0f f6-1f)
(connected f6-1f f6-2f)
(connected f6-2f f6-3f)
(connected f6-3f f6-4f)
(connected f6-4f f6-5f)
(connected f6-5f f6-6f)
(connected f6-6f f6-7f)
(connected f6-7f f6-8f)
(connected f6-8f f6-9f)
(connected f6-1f f6-0f)
(connected f6-2f f6-1f)
(connected f6-3f f6-2f)
(connected f6-4f f6-3f)
(connected f6-5f f6-4f)
(connected f6-6f f6-5f)
(connected f6-7f f6-6f)
(connected f6-8f f6-7f)
(connected f6-9f f6-8f)
(connected f7-0f f7-1f)
(connected f7-1f f7-2f)
(connected f7-2f f7-3f)
(connected f7-3f f7-4f)
(connected f7-4f f7-5f)
(connected f7-5f f7-6f)
(connected f7-6f f7-7f)
(connected f7-7f f7-8f)
(connected f7-8f f7-9f)
(connected f7-1f f7-0f)
(connected f7-2f f7-1f)
```

```
(connected f7-3f f7-2f)
(connected f7-4f f7-3f)
(connected f7-5f f7-4f)
(connected f7-6f f7-5f)
(connected f7-7f f7-6f)
(connected f7-8f f7-7f)
(connected f7-9f f7-8f)
(connected f8-0f f8-1f)
(connected f8-1f f8-2f)
(connected f8-2f f8-3f)
(connected f8-3f f8-4f)
(connected f8-4f f8-5f)
(connected f8-5f f8-6f)
(connected f8-6f f8-7f)
(connected f8-7f f8-8f)
(connected f8-8f f8-9f)
(connected f8-1f f8-0f)
(connected f8-2f f8-1f)
(connected f8-3f f8-2f)
(connected f8-4f f8-3f)
(connected f8-5f f8-4f)
(connected f8-6f f8-5f)
(connected f8-7f f8-6f)
(connected f8-8f f8-7f)
(connected f8-9f f8-8f)
(connected f9-0f f9-1f)
(connected f9-1f f9-2f)
(connected f9-2f f9-3f)
(connected f9-3f f9-4f)
(connected f9-4f f9-5f)
(connected f9-5f f9-6f)
(connected f9-6f f9-7f)
(connected f9-7f f9-8f)
(connected f9-8f f9-9f)
(connected f9-1f f9-0f)
(connected f9-2f f9-1f)
(connected f9-3f f9-2f)
(connected f9-4f f9-3f)
(connected f9-5f f9-4f)
(connected f9-6f f9-5f)
(connected f9-7f f9-6f)
(connected f9-8f f9-7f)
(connected f9-9f f9-8f)
(open f0-0f)
(open f1-0f)
(open f2-0f)
(open f3-0f)
(open f4-0f)
(open f5-0f)
(open f6-0f)
(open f7-0f)
(open f8-0f)
(open f9-0f)
(open f0-1f)
(open f1-1f)
(open f2-1f)
(open f3-1f)
(open f4-1f)
(open f5-1f)
(open f6-1f)
```

```
(open f7-1f)
(open f9-1f)
(open f0-2f)
(open f1-2f)
(open f2-2f)
(open f3-2f)
(open f4-2f)
(open f5-2f)
(open f6-2f)
(open f7-2f)
(open f8-2f)
(open f9-2f)
(open f0-3f)
(open f1-3f)
(open f2-3f)
(open f3-3f)
(open f4-3f)
(open f5-3f)
(open f6-3f)
(open f7-3f)
(open f8-3f)
(open f9-3f)
(open f0-4f)
(open f1-4f)
(open f2-4f)
(open f3-4f)
(open f4-4f)
(open f5-4f)
(open f6-4f)
(open f7-4f)
(open f8-4f)
(open f9-4f)
(open f0-5f)
(open f1-5f)
(open f2-5f)
(open f3-5f)
(open f4-5f)
(open f5-5f)
(open f6-5f)
(open f7-5f)
(open f8-5f)
(open f9-5f)
(open f0-6f)
(open f2-6f)
(open f3-6f)
(open f4-6f)
(open f5-6f)
(open f6-6f)
(open f7-6f)
(open f8-6f)
(open f9-6f)
(open f0-7f)
(open f1-7f)
(open f2-7f)
(open f3-7f)
(open f4-7f)
(open f5-7f)
(open f6-7f)
(open f7-7f)
(open f8-7f)
```

```
(open f9-7f)
(open f0-8f)
(open f1-8f)
(open f2-8f)
(open f3-8f)
(open f4-8f)
(open f5-8f)
(open f6-8f)
(open f7-8f)
(open f8-8f)
(open f9-8f)
(open f0-9f)
(open f1-9f)
(open f2-9f)
(open f3-9f)
(open f4-9f)
(open f5-9f)
(open f6-9f)
(open f7-9f)
(open f8-9f)
(open f9-9f)
(barrier f8-1f)
(barrier-type f8-1f type0)
(barrier f1-6f)
(barrier-type f1-6f type1)
(at tool0-0 f6-3f)
(at tool0-1 f7-6f)
(at tool1-0 f6-1f)
(at tool1-1 f8-0f)
(at-worker f8-3f)
)
(:goal
(and
(at tool0-0 f4-7f)
(at tool1-0 f4-7f)
(at tool1-1 f7-5f)
))
)
```

A.3.3 Python generator

```
    import random
import sys


class ProblemGenerator:
    def __init__(self):
        self.gx = -1
        self.gy = -1
        self.gnum_types = -1
        self.gtool_vec = -1
        self.gbarrier_vec = -1
        self.gp_goal = 100
        self.gtool_number = []
        self.gbarrier_number = []

        self.gx_pos = -1
        self.gy_pos = -1
        self.gx_tool_pos = []
```

```
        self.gy_tool_pos = []
        self.gx_barrier_pos = []
        self.gy_barrier_pos = []
        self.gx_tool_goal_pos = []
        self.gy_tool_goal_pos = []
        self.gbarrier = []

    def create_random_positions(self):
        max_value = max(max(self.gtool_number, default=0), max(self.
    ↪ gbarrier_number, default=0))
        self.gx_tool_pos = [[0] * max_value for _ in range(self.gnum_types)]
        self.gy_tool_pos = [[0] * max_value for _ in range(self.gnum_types)]
        self.gx_barrier_pos = [[0] * max_value for _ in range(self.
    ↪ gnum_types)]
        self.gy_barrier_pos = [[0] * max_value for _ in range(self.
    ↪ gnum_types)]
        self.gx_tool_goal_pos = [[-1] * max_value for _ in range(self.
    ↪ gnum_types)]
        self.gy_tool_goal_pos = [[-1] * max_value for _ in range(self.
    ↪ gnum_types)]

        self.gbarrier = [[False] * self.gy for _ in range(self.gx)]

        for i in range(self.gnum_types):
            for j in range(self.gbarrier_number[i]):
                while True:
                    rx, ry = random.randint(0, self.gx - 1), random.randint
    ↪ (0, self.gy - 1)
                    if not self.gbarrier[rx][ry]:
                        self.gbarrier[rx][ry] = True
                        self.gx_barrier_pos[i][j] = rx
                        self.gy_barrier_pos[i][j] = ry
                        break

            for j in range(self.gtool_number[i]):
                self.gx_tool_pos[i][j] = random.randint(0, self.gx - 1)
                self.gy_tool_pos[i][j] = random.randint(0, self.gy - 1)

                if random.randint(0, 99) < self.gp_goal:
                    self.gx_tool_goal_pos[i][j] = random.randint(0, self.gx
    ↪ - 1)
                    self.gy_tool_goal_pos[i][j] = random.randint(0, self.gy
    ↪ - 1)

        while True:
            rx, ry = random.randint(0, self.gx - 1), random.randint(0, self.
    ↪ gy - 1)
            if not self.gbarrier[rx][ry]:
                self.gx_pos = rx
                self.gy_pos = ry
                break

    def process_command_line(self, argv):
        arg_map = {
            "-x": "gx",
            "-y": "gy",
            "-t": "gnum_types",
            "-p": "gp_goal",
            "-k": "gtool_vec",
            "-l": "gbarrier_vec",
```

```
        }
        for i in range(1, len(argv) - 1, 2):
            option, value = argv[i], argv[i + 1]
            if option in arg_map:
                setattr(self, arg_map[option], int(value))
            else:
                print(f"Unknown option: {option}")
                self.usage()
                sys.exit(1)

        if self.gnum_types > 0:
            self.gtool_number = self.setup_numbers(self.gtool_vec, self.
    ↪ gnum_types)
            self.gbarrier_number = self.setup_numbers(self.gbarrier_vec,
    ↪ self.gnum_types)
        else:
            self.usage()
            sys.exit(1)

    def setup_numbers(self, vec, num_types):
        numbers = []
        for _ in range(num_types):
            numbers.insert(0, vec % 10)
            vec //= 10
        return numbers

    def usage(self):
        print("Usage:")
        print("OPTIONS   DESCRIPTIONS")
        print("-x <num>    x scale (minimal 1)")
        print("-y <num>    y scale (minimal 1)")
        print("-t <num>    num different tool+barrier types (minimal 1)")
        print("-k <num>    number tools vector (decimal)")
        print("-l <num>    number barriers vector (decimal)")
        print("-p <num>    probability of any tool being mentioned in the
    ↪ goal (default: 100)")

    def print_problem(self):
        print(f"(define (problem construction-x{self.gx}-y{self.gy}-t{self.
    ↪ gnum_types}-"
              f"k{self.gtool_vec}-l{self.gbarrier_vec}-p{self.gp_goal})")
        print("(:domain construction-site)")
        print("(:objects")

        for y in range(self.gy):
            print(" ".join(f"f{x}-{y}f" for x in range(self.gx)))
        for i in range(self.gnum_types):
            print(f"type{i}")
        for i, count in enumerate(self.gtool_number):
            for j in range(count):
                print(f"tool{i}-{j}")
        print(")")

        print("(:init")
        print("(free)")
        print("(rested)")  # Worker starts in the rested state
        for y in range(self.gy):
            for x in range(self.gx):
                print(f"(location f{x}-{y}f)")
        for i in range(self.gnum_types):
```

```
            print(f"(type type{i})")
            for j in range(self.gtool_number[i]):
                print(f"(tool tool{i}-{j})")
                print(f"(tool-type tool{i}-{j} type{i})")
        self.print_connections()
        self.print_open_barriers()
        self.print_tool_positions()
        print(f"(at-worker f{self.gx_pos}-{self.gy_pos}f)")
        print(")")

        print("(:goal")
        print("(and")
        self.print_tool_goal_positions()
        print(")")

    def print_connections(self):
        for y in range(self.gy):
            for x in range(self.gx - 1):
                print(f"(connected f{x}-{y}f f{x + 1}-{y}f)")
            for x in range(1, self.gx):
                print(f"(connected f{x}-{y}f f{x - 1}-{y}f)")
        for x in range(self.gx):
            for y in range(self.gy - 1):
                print(f"(connected f{x}-{y}f f{x}-{y + 1}f)")
            for y in range(1, self.gy):
                print(f"(connected f{x}-{y}f f{x}-{y - 1}f)")

    def print_open_barriers(self):
        for y in range(self.gy):
            for x in range(self.gx):
                if not self.gbarrier[x][y]:
                    print(f"(open f{x}-{y}f)")
        for i in range(self.gnum_types):
            for j in range(self.gbarrier_number[i]):
                print(f"(barrier f{self.gx_barrier_pos[i][j]}-{self.
    ↪ gy_barrier_pos[i][j]}f)")
                print(f"(barrier-type f{self.gx_barrier_pos[i][j]}-{self.
    ↪ gy_barrier_pos[i][j]}f type{i})")

    def print_tool_positions(self):
        for i in range(self.gnum_types):
            for j in range(self.gtool_number[i]):
                print(f"(at tool{i}-{j} f{self.gx_tool_pos[i][j]}-{self.
    ↪ gy_tool_pos[i][j]}f)")

    def print_tool_goal_positions(self):
        for i in range(self.gnum_types):
            for j in range(self.gtool_number[i]):
                if self.gx_tool_goal_pos[i][j] != -1:
                    print(f"(at tool{i}-{j} f{self.gx_tool_goal_pos[i][j]}-{
    ↪ self.gy_tool_goal_pos[i][j]}f)")


if __name__ == "__main__":
    generator = ProblemGenerator()
    generator.process_command_line(sys.argv)
    generator.create_random_positions()
    generator.print_problem()
```

A.3.4 C generator

```
    #include <stdlib.h>
#include <stdio.h>
#include <sys/timeb.h>

typedef unsigned char Bool;
#define TRUE 1
#define FALSE 0

/* helpers */
void create_random_positions(void);
void print_open_barriered(void);
void print_tool_positions(void);
void print_tool_goal_positions(void);

/* command line */
void usage(void);
Bool process_command_line(int argc, char *argv[]);
Bool setup_tool_numbers(int vec);
Bool setup_barrier_numbers(int vec);

/* globals */
/* command line params */
int gx, gy, gnum_tooltypes, gtool_vec, *gtool_number, gbarrier_vec, *
    ↪ gbarrier_number, gp_goal;

/* random values */
int gx_pos, gy_pos, **gx_tool_pos, **gy_tool_pos, **gx_barrier_pos, **
    ↪ gy_barrier_pos;
int **gx_tool_goal_pos, **gy_tool_goal_pos;

/* helper */
Bool **gbarrier;

int main(int argc, char *argv[]) {
    int x, y, i, j;

    /* seed the random() function */
    struct timeb tp;
    ftime(&tp);
    srandom(tp.millitm);

    /* command line treatment, first preset values */
    gx = -1;
    gy = -1;
    gnum_tooltypes = -1;
    gtool_vec = -1;
    gbarrier_vec = -1;
    gp_goal = 100;

    if (argc == 1 || (argc == 2 && *++argv[0] == '?')) {
        usage();
        exit(1);
    }
    if (!process_command_line(argc, argv)) {
        usage();
        exit(1);
    }

```

```
57      create_random_positions();
58
59      /* now output problem in PDDL syntax */
60      printf("\n\n\n");
61
62      /* header */
63      printf("(define (problem construction-x%d-y%d-t%d-k%d-l%d-p%d)",
64             gx, gy, gnum_tooltypes, gtool_vec, gbarrier_vec, gp_goal);
65      printf("\n(:domain construction-site)");
66
67      printf("\n(:objects ");
68      for (y = 0; y < gy; y++) {
69          printf("\n          ");
70          for (x = 0; x < gx; x++) {
71              printf("f%d-%df ", x, y);
72          }
73      }
74      printf("\n          ");
75      for (i = 0; i < gnum_tooltypes; i++) {
76          printf("type%d ", i);
77      }
78      for (i = 0; i < gnum_tooltypes; i++) {
79          if (gtool_number[i] == 0)
80              continue;
81          printf("\n          ");
82          for (j = 0; j < gtool_number[i]; j++) {
83              printf("tool%d-%d ", i, j);
84          }
85      }
86      printf("\n)");
87
88      printf("\n(:init");
89      printf("\n(free)");
90      for (y = 0; y < gy; y++) {
91          for (x = 0; x < gx; x++) {
92              printf("\n(location f%d-%df)", x, y);
93          }
94      }
95      for (i = 0; i < gnum_tooltypes; i++) {
96          printf("\n(type type%d)", i);
97      }
98      for (i = 0; i < gnum_tooltypes; i++) {
99          for (j = 0; j < gtool_number[i]; j++) {
100             printf("\n(tool tool%d-%d)", i, j);
101             printf("\n(tool-type tool%d-%d type%d)", i, j, i);
102         }
103     }
104     for (y = 0; y < gy; y++) {
105         for (x = 0; x < gx - 1; x++) {
106             printf("\n(connected f%d-%df f%d-%df)", x, y, x + 1, y);
107         }
108     }
109     for (y = 0; y < gy - 1; y++) {
110         for (x = 0; x < gx; x++) {
111             printf("\n(connected f%d-%df f%d-%df)", x, y, x, y + 1);
112         }
113     }
114     for (y = 0; y < gy; y++) {
115         for (x = 1; x < gx; x++) {
116             printf("\n(connected f%d-%df f%d-%df)", x, y, x - 1, y);
```

```
        }
    }
    for (y = 1; y < gy; y++) {
        for (x = 0; x < gx; x++) {
            printf("\n(connected f%d-%df f%d-%df)", x, y, x, y - 1);
        }
    }
    print_open_barriered();
    print_tool_positions();
    printf("\n(at-worker f%d-%df)", gx_pos, gy_pos);
    printf("\n)");

    printf("\n(:goal");
    printf("\n(and");
    print_tool_goal_positions();
    printf("\n)");
    printf("\n)");

    printf("\n)");

    printf("\n\n\n");

    exit(0);
}

/* random problem generation functions */

void create_random_positions(void) {
    int MAX;
    int i, j, rx, ry, r;

    MAX = -1;
    for (i = 0; i < gnum_tooltypes; i++) {
        if (MAX == -1 || gtool_number[i] > MAX) {
            MAX = gtool_number[i];
        }
        if (gbarrier_number[i] > MAX) {
            MAX = gbarrier_number[i];
        }
    }
    gx_tool_pos = (int **)calloc(gnum_tooltypes, sizeof(int *));
    gy_tool_pos = (int **)calloc(gnum_tooltypes, sizeof(int *));
    gx_barrier_pos = (int **)calloc(gnum_tooltypes, sizeof(int *));
    gy_barrier_pos = (int **)calloc(gnum_tooltypes, sizeof(int *));
    gx_tool_goal_pos = (int **)calloc(gnum_tooltypes, sizeof(int *));
    gy_tool_goal_pos = (int **)calloc(gnum_tooltypes, sizeof(int *));
    for (i = 0; i < gnum_tooltypes; i++) {
        gx_tool_pos[i] = (int *)calloc(MAX, sizeof(int));
        gy_tool_pos[i] = (int *)calloc(MAX, sizeof(int));
        gx_barrier_pos[i] = (int *)calloc(MAX, sizeof(int));
        gy_barrier_pos[i] = (int *)calloc(MAX, sizeof(int));
        gx_tool_goal_pos[i] = (int *)calloc(MAX, sizeof(int));
        gy_tool_goal_pos[i] = (int *)calloc(MAX, sizeof(int));
    }
    gbarrier = (Bool **)calloc(gx, sizeof(Bool *));
    for (i = 0; i < gx; i++) {
        gbarrier[i] = (Bool *)calloc(gy, sizeof(Bool));
        for (j = 0; j < gy; j++) {
            gbarrier[i][j] = FALSE;
        }
```

```
177     }
178
179     for (i = 0; i < gnum_tooltypes; i++) {
180         for (j = 0; j < gbarrier_number[i]; j++) {
181             while (TRUE) {
182                 rx = random() % gx;
183                 ry = random() % gy;
184                 if (!gbarrier[rx][ry])
185                     break;
186             }
187             gbarrier[rx][ry] = TRUE;
188             gx_barrier_pos[i][j] = rx;
189             gy_barrier_pos[i][j] = ry;
190         }
191     }
192     for (i = 0; i < gnum_tooltypes; i++) {
193         for (j = 0; j < gtool_number[i]; j++) {
194             rx = random() % gx;
195             ry = random() % gy;
196             gx_tool_pos[i][j] = rx;
197             gy_tool_pos[i][j] = ry;
198         }
199     }
200     for (i = 0; i < gnum_tooltypes; i++) {
201         for (j = 0; j < gtool_number[i]; j++) {
202             r = random() % 100;
203             if (r >= gp_goal) {
204                 gx_tool_goal_pos[i][j] = -1;
205                 gy_tool_goal_pos[i][j] = -1;
206                 continue;
207             }
208             rx = random() % gx;
209             ry = random() % gy;
210             gx_tool_goal_pos[i][j] = rx;
211             gy_tool_goal_pos[i][j] = ry;
212         }
213     }
214     while (TRUE) {
215         rx = random() % gx;
216         ry = random() % gy;
217         if (!gbarrier[rx][ry])
218             break;
219     }
220     gx_pos = rx;
221     gy_pos = ry;
222 }
223
224 /* printing functions */
225
226 void print_open_barriered(void) {
227     int x, y, i, j;
228
229     for (y = 0; y < gy; y++) {
230         for (x = 0; x < gx; x++) {
231             if (!gbarrier[x][y]) {
232                 printf("\n(open f%d-%df)", x, y);
233             }
234         }
235     }
236
```

```
237     for (i = 0; i < gnum_tooltypes; i++) {
238         for (j = 0; j < gbarrier_number[i]; j++) {
239             printf("\n(barrier f%d-%df)",
240                    gx_barrier_pos[i][j], gy_barrier_pos[i][j]);
241             printf("\n(barrier-type f%d-%df type%d)",
242                    gx_barrier_pos[i][j], gy_barrier_pos[i][j], i);
243         }
244     }
245 }
246 
247 void print_tool_positions(void) {
248     int i, j;
249 
250     for (i = 0; i < gnum_tooltypes; i++) {
251         for (j = 0; j < gtool_number[i]; j++) {
252             printf("\n(at tool%d-%d f%d-%df)", i, j,
253                    gx_tool_pos[i][j], gy_tool_pos[i][j]);
254         }
255     }
256 }
257 
258 void print_tool_goal_positions(void) {
259     int i, j;
260 
261     for (i = 0; i < gnum_tooltypes; i++) {
262         for (j = 0; j < gtool_number[i]; j++) {
263             if (gx_tool_goal_pos[i][j] == -1)
264                 continue;
265             printf("\n(at tool%d-%d f%d-%df)", i, j,
266                    gx_tool_goal_pos[i][j], gy_tool_goal_pos[i][j]);
267         }
268     }
269 }
270 
271 /* command line functions */
272 
273 void usage(void) {
274     printf("\nusage:\n");
275 
276     printf("\nOPTIONS   DESCRIPTIONS\n\n");
277     printf("-x <num>    x scale (minimal 1)\n");
278     printf("-y <num>    y scale (minimal 1)\n\n");
279     printf("-t <num>    num different tool+barrier types (minimal 1)\n\n");
280     printf("-k <num>    number tools vector (decimal)\n");
281     printf("-l <num>    number barriers vector (decimal)\n\n");
282     printf("-p <num>    probability of any tool being mentioned in the goal
    ↪ (preset: %d)\n\n",
283            gp_goal);
284 }
285 
286 Bool process_command_line(int argc, char *argv[]) {
287     char option;
288 
289     while (--argc && ++argv) {
290         if (*argv[0] != '-' || strlen(*argv) != 2) {
291             return FALSE;
292         }
293         option = *++argv[0];
294         switch (option) {
295         default:
```

```
296             if (--argc && ++argv) {
297                 switch (option) {
298                 case 'x':
299                     sscanf(*argv, "%d", &gx);
300                     break;
301                 case 'y':
302                     sscanf(*argv, "%d", &gy);
303                     break;
304                 case 't':
305                     sscanf(*argv, "%d", &gnum_tooltypes);
306                     break;
307                 case 'p':
308                     sscanf(*argv, "%d", &gp_goal);
309                     break;
310                 case 'k':
311                     sscanf(*argv, "%d", &gtool_vec);
312                     if (gnum_tooltypes == -1) {
313                         break;
314                     } else {
315                         if (setup_tool_numbers(gtool_vec)) {
316                             break;
317                         } else {
318                             printf("\n\ncannot interpret tool number vector
    ↪ .\n\n");
319                             exit(1);
320                         }
321                     }
322                     break;
323                 case 'l':
324                     sscanf(*argv, "%d", &gbarrier_vec);
325                     if (gnum_tooltypes == -1) {
326                         break;
327                     } else {
328                         if (setup_barrier_numbers(gbarrier_vec)) {
329                             break;
330                         } else {
331                             printf("\n\ncannot interpret barrier number
    ↪ vector.\n\n");
332                             exit(1);
333                         }
334                     }
335                     break;
336                 default:
337                     printf("\n\nunknown option: %c entered\n\n", option);
338                     return FALSE;
339                 }
340             } else {
341                 return FALSE;
342             }
343         }
344     }
345 
346     if (gx < 1 || gy < 1 || gnum_tooltypes < 1) {
347         return FALSE;
348     }
349 
350     return TRUE;
351 }
352 
353 Bool setup_tool_numbers(int vec) {
```

```
354     int current, i;
355
356     if (gnum_tooltypes < 1)
357         return FALSE;
358
359     gtool_number = (int *)calloc(gnum_tooltypes, sizeof(int));
360
361     current = vec;
362
363     for (i = gnum_tooltypes - 1; i >= 0; i--) {
364         gtool_number[i] = (int)(current % 10);
365         if (gtool_number[i] < 0)
366             return FALSE;
367         current = (int)(current / 10);
368     }
369
370     return TRUE;
371 }
372
373 Bool setup_barrier_numbers(int vec) {
374     int current, i;
375
376     if (gnum_tooltypes < 1)
377         return FALSE;
378
379     gbarrier_number = (int *)calloc(gnum_tooltypes, sizeof(int));
380
381     current = vec;
382
383     for (i = gnum_tooltypes - 1; i >= 0; i--) {
384         gbarrier_number[i] = (int)(current % 10);
385         if (gbarrier_number[i] < 0)
386             return FALSE;
387         current = (int)(current / 10);
388     }
389
390     return TRUE;
391 }
```

Appendix B

Additional documentation

B.1 A1: Search

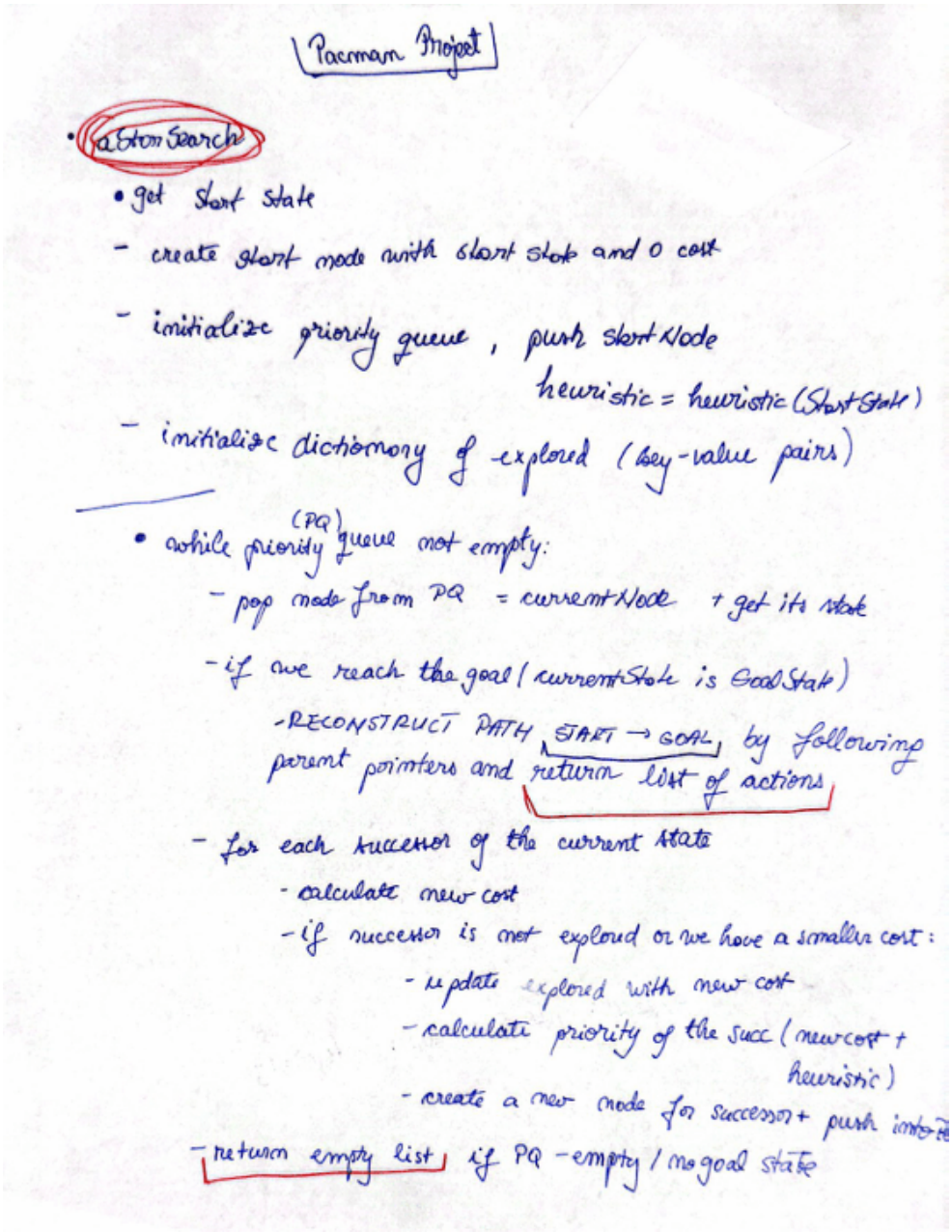

Figure B.1: A* Search, page 1

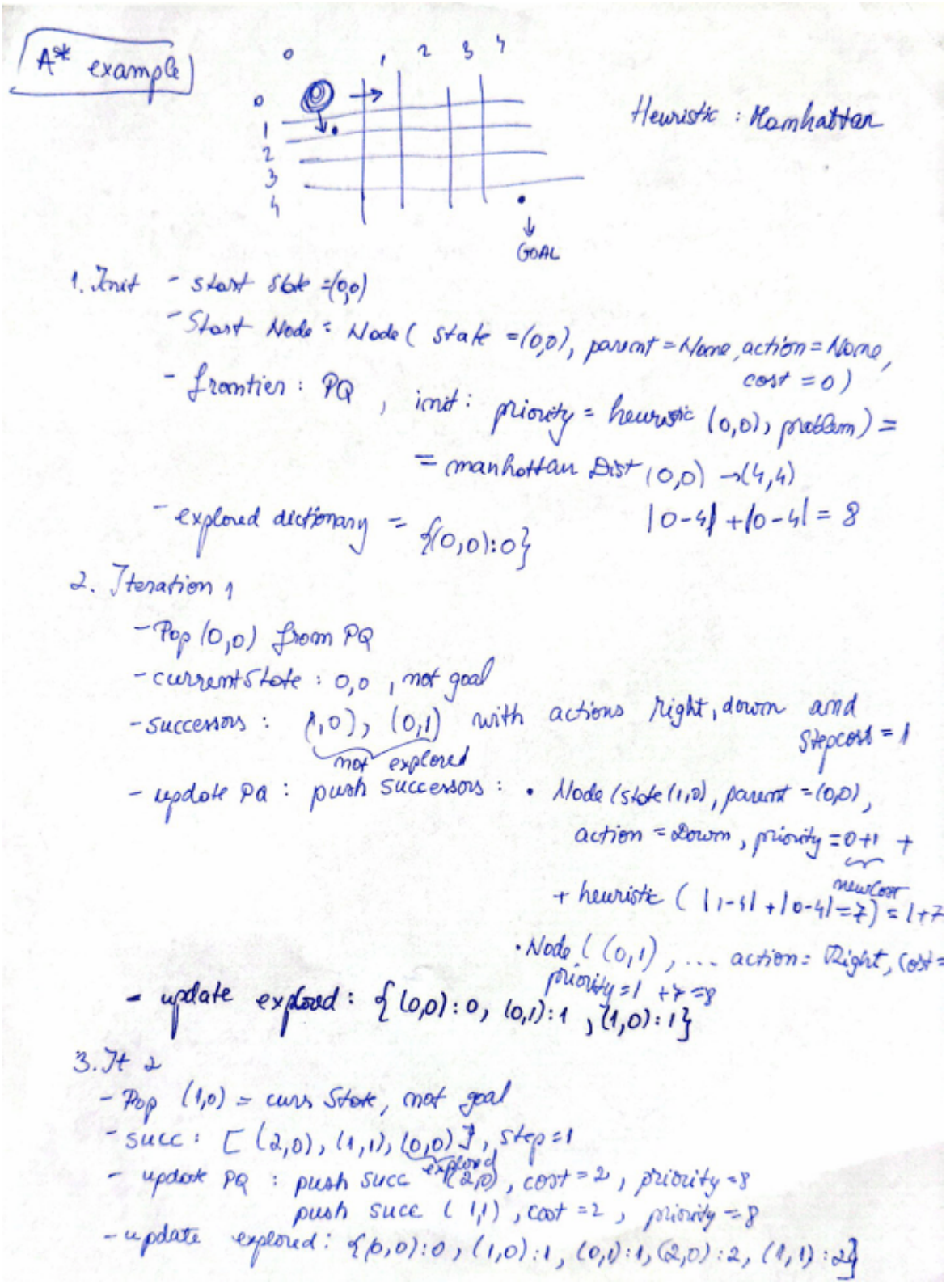

Figure B.2: A* Search, page 2

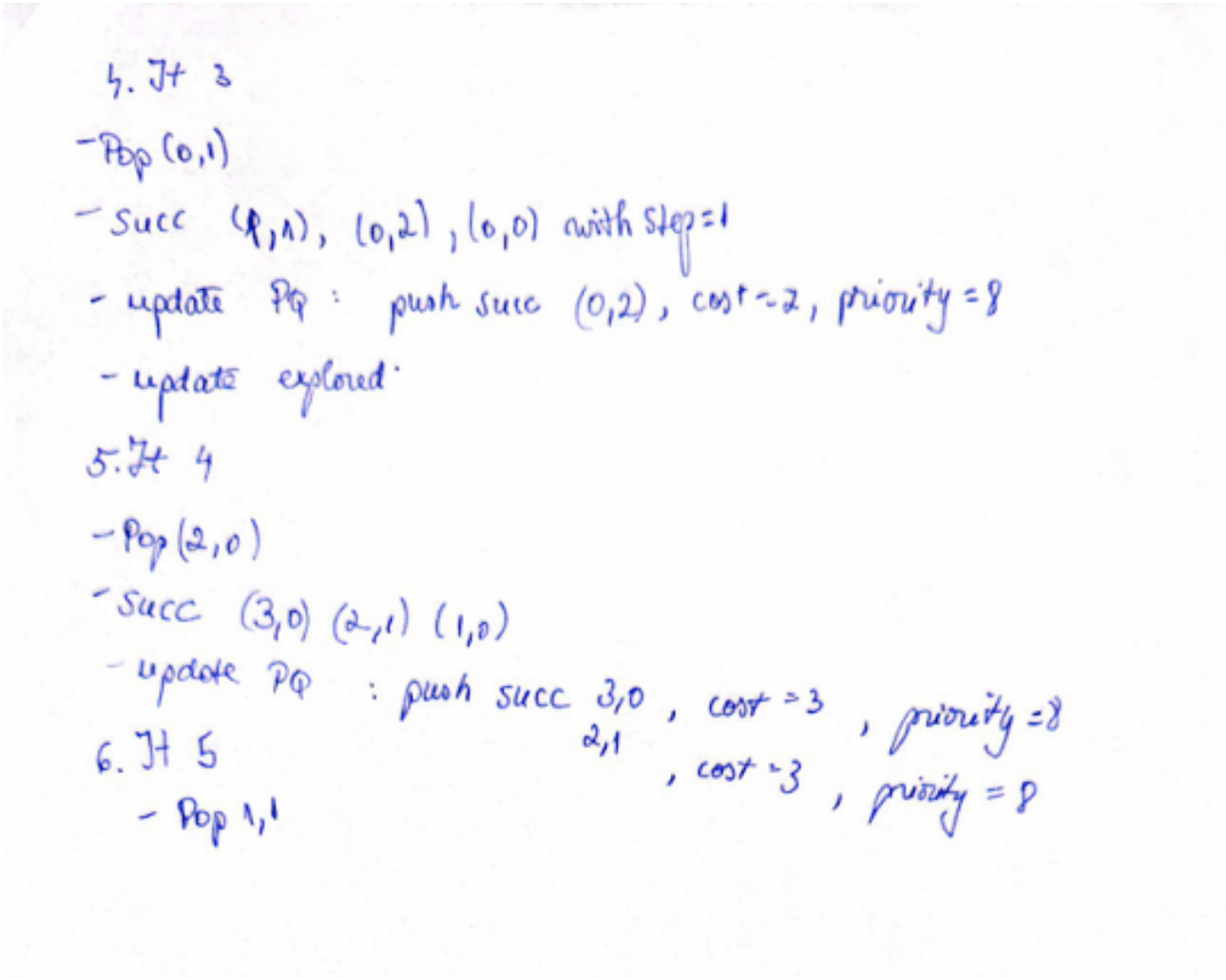

Figure B.3: A* Search, page 3

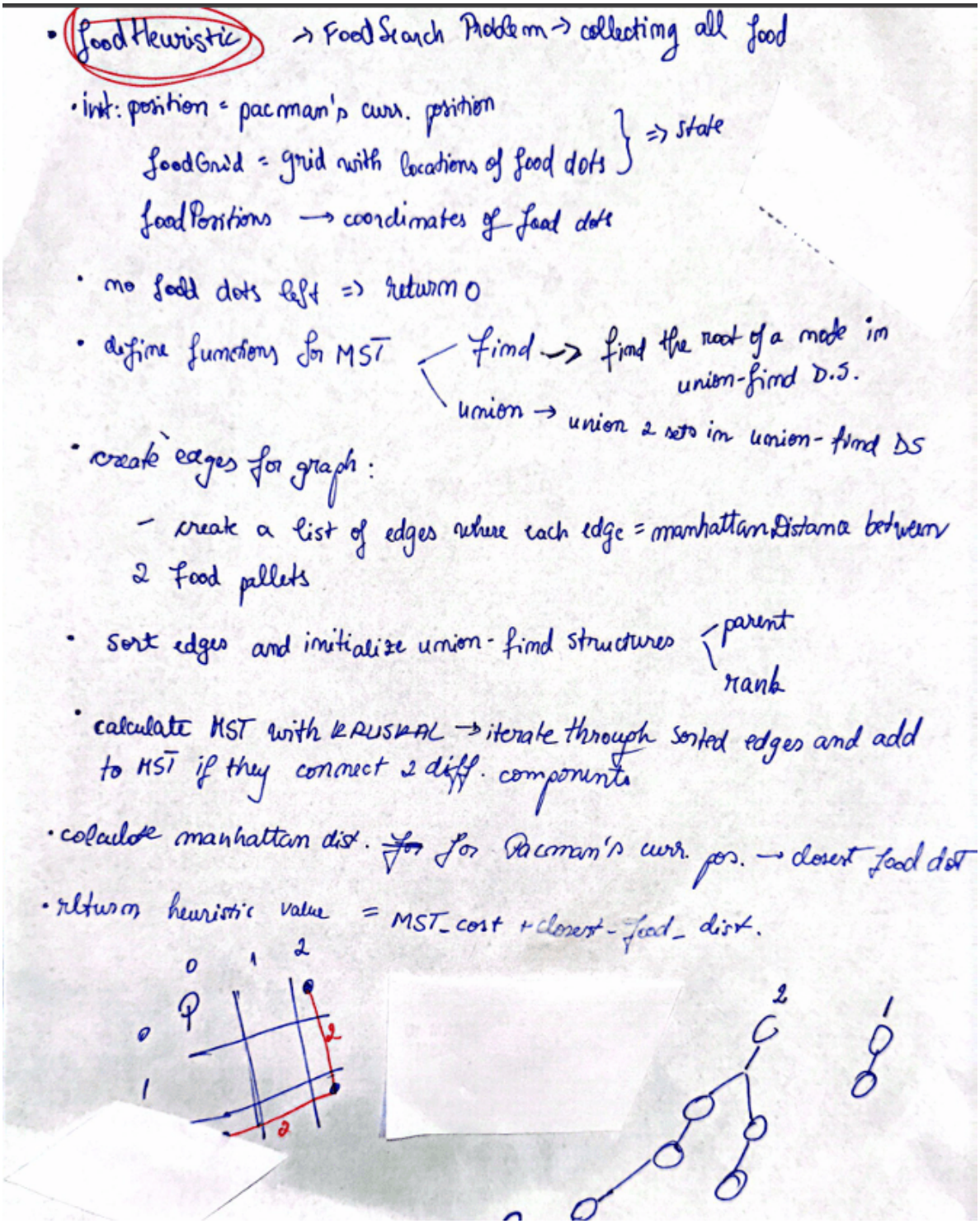

Figure B.4: All Food Heuristic, page 1