Avery Nelson --- 0 late days used

010920903

amn018@uark.edu

Artificial Intelligence Assignment 3 Report

Description of Implementation

The goal of this assignment is to build on the concepts used in assignment two to lead Pacman to collect all his food. We are required to implement searching agents to find all four corners of the map, implement a heuristic for the corners, lead Pacman to all food, and implement a greedy algorithm that prioritizes Pacman to collect his closest food. We were given files that are used to render the Pacman game: graphicsDisplay.py, graphicsUtils.py, textDisplay.py, ghostDisplay.py, keyboardAgents.py, and layout.py. game.py defines how the game is played, describing AgentState, Agent, Direction, and Grid. util.py consists of useful data structures. pacman.py is the main program that runs the game. search.py provides searching algorithms implemented in the second assignment. searchAgents.py includes search agent problem and is the main file to edit. Completing the tasks required implementing the provided CornersProblem, cornersHeuristic, and foodHeuristic classes, findPathToClosestDot method, and isGoalState in the AnyFoodSearchProblem class in searchAgents.py. The challenging part was dealing with the corner problem.

For each of the tasks, Pacman needs to hit the walls at least one time and no more than two times (at max two times). For example, if Pacman reaches the destination (or goal state) without hitting the wall, you lose the game. Or if Pacman reaches the destination by hitting the wall three times, you also lose the game.

For the CornersProblem, I defined the state as a tuple with three components: position (current x and y coordinates of Pacman), visited corners (tuple of Boolean values tracking which corners have been visited), and hit walls (counter tracking how many walls Pacman has hit). The key functions are getStartState, isGoalState, isWall, and getSuccessors. getStartState initializes Pacman at the starting position with no corners visited and zero wall hits. isGoalState marks that the goal state is reached when all four corners have been visited, and the wall has been hit 1-2 times. isWall is a helper function that checks if a given position is a wall. getSuccessors generates valid successor states by checking all four possible movements, checking if the next position is in bounds, determining if the next position is a wall and if so, updating the hit counter, updating the visited corners list if a corner is reached, and adding valid successors to the result list.

For cornersHeuristic, my heuristic is admissible because it never overestimates the actual cost (determined by three things). First, it uses Manhattan distance which is always less than or equal to the actual path distance. Second, it provides a lower bound on the cost needed to visit all corners by taking the maximum distance to any unvisited corner. And finally, it handles wall-hitting constraints by assigning appropriate costs. It assigns infinite cost to paths that have already hit more than two walls and to goal states that have not hit any walls. It then assigns a

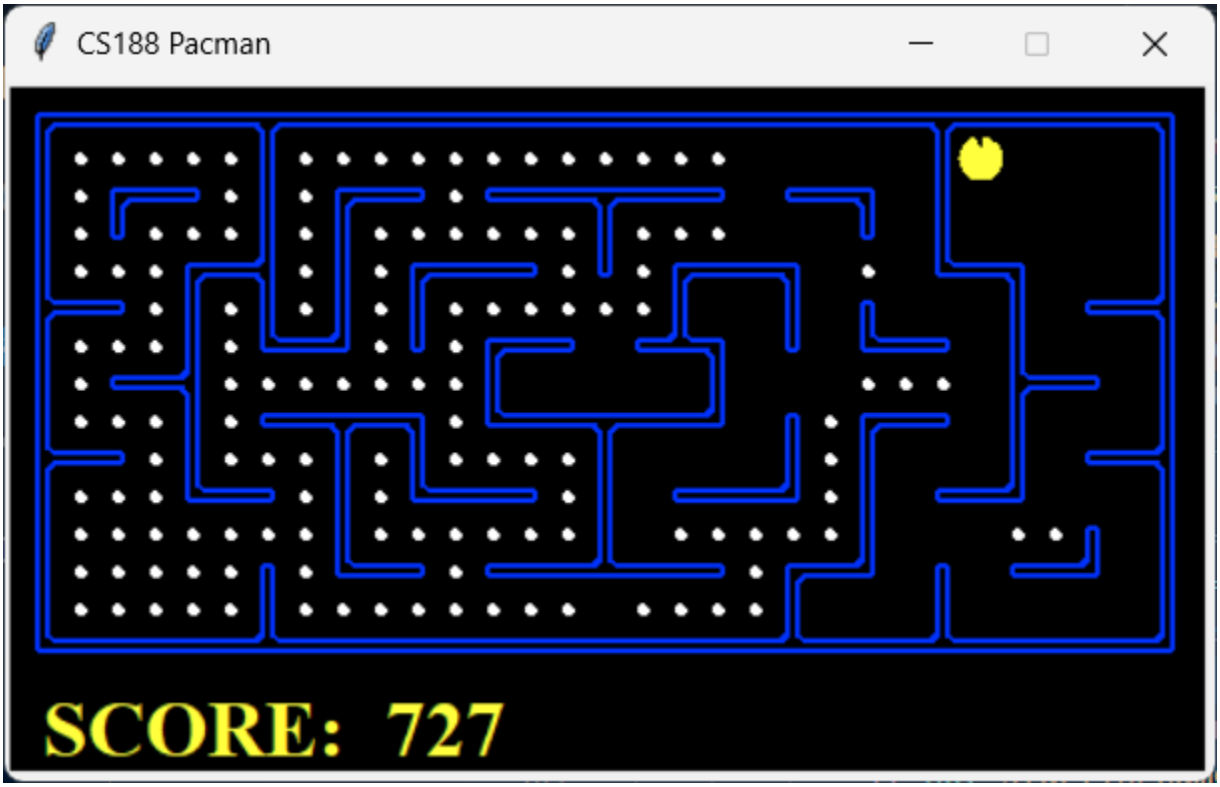
minimum cost to make sure at least one wall hit occurs. Additionally, it assigns the maximum Manhattan distance to any unvisited corner.

foodHeuristic is more simplistic than either of the above tasks. It works similarly to the corners heuristic. It returns zero if no food is left, calculates Manhattan distances to all remaining food dots, and returns the maximum distance as a lower bound on the cost to collect all food. This heuristic is admissible because Manhattan distance never overestimates the true distance, and to collect all food dots, Pacman must travel at least as far as the farthest food dot.

The greedy approach problem creates an AnyFoodSearchProblem which had an updated goal state definition. Rather than returning self.food[x][y], it returns the coordinates of the state in self.food.asList(). This implementation uses BFS to find the shortest path to the closest food and tracks wall hits across multiple searches by maintaining a counter.

Results

Overall, my implementations of each of the problems appears to be successful. All of them determine victory given the test codes provided. For the corners problem, Pacman visits all four corners with 1-2 wall hits. The A* search with the corner heuristic finds efficient paths. The food heuristic guides Pacman to collect all food dots in complex mazes and the closest dot search algorithm provides a greedy but effective approach for large mazes. Thus, the project successfully completes the required tasks given the test cases and parameters provided.



```
PS C:\Users\avery\Documents\GitHub\AI-sp25\HW3\search> python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
readCommand argv {argv}
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 18 in 0.2 seconds
Search nodes expanded: 1051
Number of Hitting Walls: 2
Pacman emerges victorious! Score: 522
Average Score: 522.0
Scores:        522.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\avery\Documents\GitHub\AI-sp25\HW3\search> python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
readCommand argv {argv}
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 77 in 5.4 seconds
Search nodes expanded: 6584
Number of Hitting Walls: 2
Pacman emerges victorious! Score: 463
Average Score: 463.0
Scores:        463.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\avery\Documents\GitHub\AI-sp25\HW3\search> python pacman.py -l mediumCorners -p SearchAgent -a fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic
readCommand argv {argv}
[SearchAgent] using function aStarSearch and heuristic cornersHeuristic
[SearchAgent] using problem type CornersProblem
Path found with total cost of 77 in 0.1 seconds
Search nodes expanded: 3102
Number of Hitting Walls: 2
Pacman emerges victorious! Score: 463
Average Score: 463.0
Scores:        463.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
PS C:\Users\avery\Documents\GitHub\AI-sp25\HW3\search> python pacman.py -l trickySearch -p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic
readCommand argv {argv}
[SearchAgent] using function astar and heuristic foodHeuristic
[SearchAgent] using problem type FoodSearchProblem
Path found with total cost of 34 in 15.0 seconds
Search nodes expanded: 23449
Number of Hitting Walls: 2
Pacman emerges victorious! Score: 596
Average Score: 596.0
Scores:        596.0
Win Rate:      1/1 (1.00)
Record:        Win
PS C:\Users\avery\Documents\GitHub\AI-sp25\HW3\search> python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
readCommand argv {argv}
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 348.
Number of Hitting Walls: 2
Pacman emerges victorious! Score: 2362
Average Score: 2362.0
Scores:        2362.0
Win Rate:      1/1 (1.00)
Record:        Win
```