LABREPORT

David van Balen, Joris van Gool, Daan van Laar
5513588 4270126 5518741

Exercise 1

a.

In the gameState is defined what the layout of the board is, where pacman, the ghosts and food are, aswell as the amount of food and ghosts.
Calling 'generateSuccesor' with an action returns a new gameState, which is what the game would look
like after completing said action. This is then passed as the new current gameState, when the agent decides to execute this action.

b.

In the agentState is stored whether this agent is pacman or a ghost, its speed, and
some variables neccesairy for choosing actions (ghosts for example have a 'scaredTimer').
The agent can choose between North, East, South and West and then updates the agentState itself.
For this, the agentState has options to copy, and to get some of the variables stored in the state.

c.

A III
B II
C I

d.

the east and west agents modify the costs of steps in the west, negative or
positively so they prefer squares in their own direction. This causes them to
generally go 2 diffrent ways instead of a normal searchagent which could be always
drawn to one side.

Exercise 2

In the exercised Q1 through Q4, the generic structure we used was the same. In each
algorith, we firstly pushed the starting state to the fringe, along with the moves
to get to that state. Then we would loop untill the fringe is empty. Inside the loop,
we pop a state out of the fringe, get a list with it's successors, return a path
if it leads to a goal state and check if any states have already been visited. Then
the states that are left are pushed onto the fringe

Exercise 3

a.

If we have a game tree with all possible game states from a starting space, and the
goal state is in this tree, then depth first search will always find the correct
solution to this problem. Depth first search doesn't skip any nodes in the tree, it
only defines in what order they are visited. Since the goal state is in the game tree,
dfs will reach it eventually

b.

Depth first search does not always give a least cost solution. It just tries to make
a path to the goal state and once it reaches the goal, dfs is done. however, this
path might not contain the shortest route, it's just the first route that has been
found. the openMaze map of this assignment gives a great example of this.

c.
The exploration order is exactly as we would have expected. Pac-Man does not visit all the states that have been expanded on his way to the goal. If the first path that the algorithm tries does not lead to a goal, a bunch of nodes have been expanded but Pac-Man won't visit them, simply because they don't lead to a goal

Exercise 4
a.
If we compose a game tree with all the possible states of a pacman game, and the goal state is in that tree, then bfs will find a path to this goal state. The only difference between bfs and dfs is the order in which states will be explored, but if need be, all states will be explored. That's why bfs is complete

b.
Granted that the costs of all the possible action in a game are the same, bfs will find the least cost solution. Simply because the node that has been expanded the least will be expanded next. If all the actions have the same cost, the nodes that have been expanded last have the same cost too. Because the path to the goal state will be returned as soon as the goal state is found, it should have the least cost. This does not hold up if the costs are not the same for each action

c.
Because the algorithm has been written in the most general fashion possible, it also works for the eightpuzzle. We tested it many times and it returns the correct solution

Exercise 5
a.
The first agent should just find the least cost solution to a pathfinding problem.
This is achieved by expanding the paths to other nodes with the least cost first.
In the case of a Pac-Man board, all the actions have the same cost and the ucs works the same as bfs.
The stayWestAgent should find a path through the west side of the maze, because all of the food pellets are located there. It does so by editing the costs so that nodes on the west have a lower cost than nodes on the east side of the board.
The stayEastAgent should go east as quick as possible, because there are ghosts in the west. The costs of nodes in the west are therefor very high which makes Pac-Man go east instantly

b.
The first agent finds a path of cost 68.
The second agent finds a path of cost 1.
The third agent finds a path of cost 68719479864.

Exercise 6
DFS:
dfs chooses a very swirly path to the goal which clearly isn't optimal. This is however one of the first paths that dfs chooses to calculate to see if it leads to the goal
BFS:
bfs chooses a path that is a least cost solution to the goal. It does explore a lot

of nodes, but it gets to the end as quick as it can
UCS:
this is the same as BFS, because UCS is exactly the same as BFS if all the costs are the same. Because the cost function is: lambda x: 1, all the costs are the same and ucs is bfs
A*:
computes the same least cost path to the goal as BFS and UCS, but because of the heuristics, almost a hundred less nodes have been expanded.

Exercise 7
In the state it is important to note the coordinates of Pac-Man. This is essential to compute the successor nodes. We also need to keep track of how many and which corners have been visited. This could be done with a Boolean for each corner, but that isn't too sophisticated. Instead, we chose to represent the state as a list of coordinates. The first coordinate is always the coordinate where Pac-Man is in that state. Any coordinates that follow this are the coordinates of the corners where pacman has been. This also allows Pac-Man to go to places where he's already been, after he has visited a corner. This is essential for computing any path at all.

Exercise 8
a.
If there are no unvisited corners, the heuristic is 0.
Our heuristic first looks for the farthest unvisited corner, then the farthest unvisited corner from the first corner. If there was only 1 unvisited corner, the heuristic is the distance between Pacman and this corner.
It then takes the distance between the two corners and
the distance between you and the second corner, and adds those up. This is the value it returns.
For the distance between two dots, we used a self-made function called 'manhattanWallDistance'.
It first checks if there is a horizontal wall inside the box closed in by the start dot and end dot.
If not, it just returns the standard manhattan distance between the two dots.
If there is, it finds the endpoint(s) of the wall, and returns the smallest distance of walking from the startdot to one of the endpoints of the wall to the end dot. This distance is again calculated using the manhattanWallDistance.

b.
The admissibility of this heuristic is obvious:
-The manhattanWallDistance is always smaller than or equal to the
    actual distance Pacman has to walk between the two points.
-The heuristic thus returns a value which is smaller than or equal to
    the distance Pacman has to walk to reach the 2 unvisited corners that we examined.

The constistency of this heuristic can also be proven:
-The wallDistance part is consistent because of the fact that we only look at horizontal walls.
    A step of cost c, will make the wallDistance between pacman and a dot fall by at most c:
        If the higher cost path is A, and the smaller cost path is B:
        If there is no wall in path A, there will obviously not be one in B. This means that it's the same as standard manhattan distance, so consistent.
        if there is a wall in path A, call the endpoints of this wall w1 and w2. Say A is the path from P to w1 to E.
        There are three options:

There is no wall in B. Simple drawings serve to see that there is no way a step of cost c will make B smaller then A-c in this case.

There is a different wall in B then in A. Again, simple drawings show the recursive call to manhattanWallDistance at the end points of this wall prove consistency.

B walks around w1 aswell. In this case the only difference between B and A is the distance to w1, which can clearly change at most c.

B walks around w2. This case is proven with the ASCII-art below. Since path A goes around w1, the value of path B + c can't be smaller than the value of path A. If this wouldn't be the case, path A would also go around w2.
From B+c>=A, it follows that B>=A-c.

```
                 ________A---c---B______
           -------                         -----
       w1 ########################### w2
           -------_______           _______-----
                         ----D----
```

[If we would also look at vertical walls using the same algorithm, the heuristic would either get stuck in an infinite loop at a corner of walls, or (if for example we ignore vertical walls when there is a horizontal wall) would turn inconsistant, because a small step can make a horizontal wall not run along the entire 'box' between pacman and corner anymore, which means the algorithm ignores this wall, which means it can suddenly detect a much larger vertical wall. This is the reason we decided to only look at horizontal walls.]

-The actual heuristic part is consistant because of the following reasoning:

The consistency of the two point method is easy to see, anytime a step is taken, there are two options, the furthest node doesn't change, in which case you get a maximum of the cost of the step closer to the second node, and the furthest node from the furthest node doesn’t change either so the maximal reduction in cost is exactly the cost of one step.

The second option is that the furthest node does change, in which case all the distances change but they never get smaller than the cost of the step taken. If a new node becomes the furthest away node, it will be equally far, or one further, away from the last furthest away node. The node furthest away from this node can never be more than 1 cost less away then it was from the last node. Because every node has a predefined value for the node furthest away, if a node is not in the boundary of the nodegroup it can't be the node furthest away from the player, because there is always one futher away, if it is, it has nodes that are equally far away from it as from the other furthest nodes, otherwise it wouldn't be a candidate for the furthest node to begin with.

c.
972

Exercise 9
a.
For the foodHeuristic we used exactly the same method as for the cornersHeuristic, except with food dots instead of unvisited corners.

b.
See 8b.

c.
1062

Exercise 10
a.
For finding the closest dot to pacman, we used the manhattan heuristics. Then we would simulate a problem where the state where Pac-Man is located at is the starting state and the closest dot to Pac-Man is the goal state. We used the search function bfs, because it assures the least cost path and it is a simple algorithm for this simple problem.

b.
```
. P.  .
```
In the example above, the closest dot search would move right first, then eat the dot(.) on the far left and finally send Pac-Man(P) to the node on the far right. However, it would be shorter if Pac-Man went left first and then go to the far right.

c.
The problem is that Pac-Man is very short sighted and doesn't take the shortest total path into account, just the shortest path to his goal at that point in time.

11
3/3 3/3 3/3 3/3 3/3 3/3 5/4 3/3   = 26/25