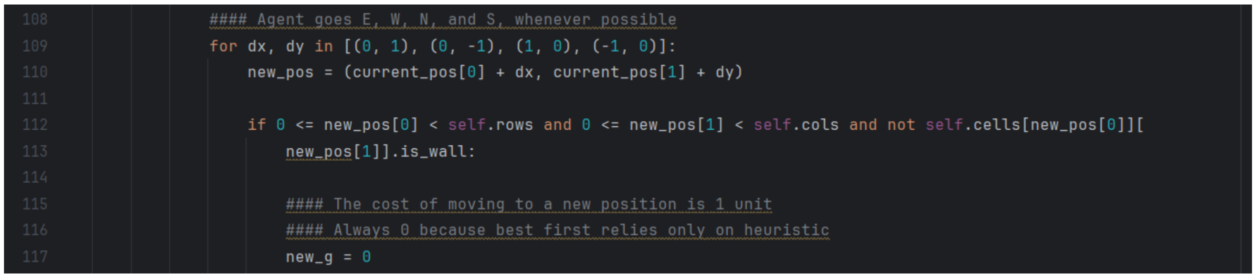
<u>Assignment 4 Technical Report</u>

Author: Jillian O’Connell

Purpose: The purpose of this assignment is to compare the differences between the A* algorithm and the Greedy Best First algorithm with an agent finding the optimum path through a maze. The goal is to then observe hoe calculating the heuristic differently and allowing different kinds of moves for the agent effect the two algorithms. Finally, the goal is to observe hoe changing the values of alpha and beta int eh evaluation function effect the path that the agent finds.

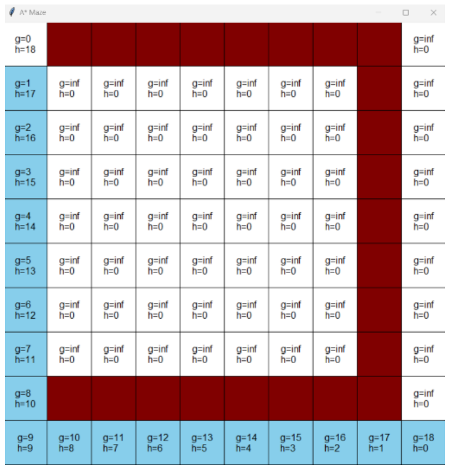
Problem 1:

In order to change the A* algorithm to act as a greedy best first algorithm, I changed the calculation of the path cost, g(n), to always equal zero. This way, the algorithm would only rely on the heuristics instead of the path cost. When calculating the actual cost via the evaluation function, g would always be zero thus forcing f(n) to always only equal the heuristic. This change is demonstrated in the file named “Problem1_BFS”as well as in Figure 1. The path produced by the A* algorithm is shown in Figure 2 whereas the path produced by the Greedy Best First algorithm is shown in Figure 3. The A* algorithm finds an overall shorter path because not only does it take into account the heuristic, but it also takes into account the actual cost. In this example, each move has an actual cost of 1. On the other hand, the Greedy Best First takes a longer path because it only relies on heuristic and ignores the actual path cost. The heuristic, in this example, is calculated using the Manhattan Distance. This is why the Greedy Best first moves all the way to the right before hitting a wall and having to change direction; moving towards the right gets closer to the goal but does not recognize that the path will have to eventually backtrack to get around the obstacle thus taking a longer total path.



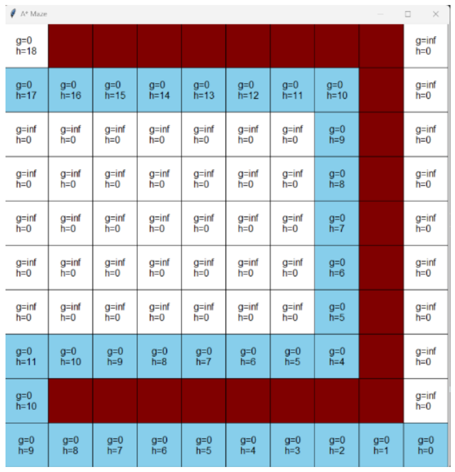
```
#### Agent goes E, W, N, and S, whenever possible
for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
    new_pos = (current_pos[0] + dx, current_pos[1] + dy)

    if 0 <= new_pos[0] < self.rows and 0 <= new_pos[1] < self.cols and not self.cells[new_pos[0]][
        new_pos[1]].is_wall:

        #### The cost of moving to a new position is 1 unit
        #### Always 0 because best first relies only on heuristic
        new_g = 0
```

Figure 1: The edited code to change the algorithm from A* to Greedy Best First



g=0 h=18									g=inf h=0
g=1 h=17	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0		g=inf h=0
g=8 h=10									g=inf h=0
g=9 h=9	g=10 h=8	g=11 h=7	g=12 h=6	g=13 h=5	g=14 h=4	g=15 h=3	g=16 h=2	g=17 h=1	g=18 h=0

Figure 2: Finding the shortest path through the maze using the A* algorithm

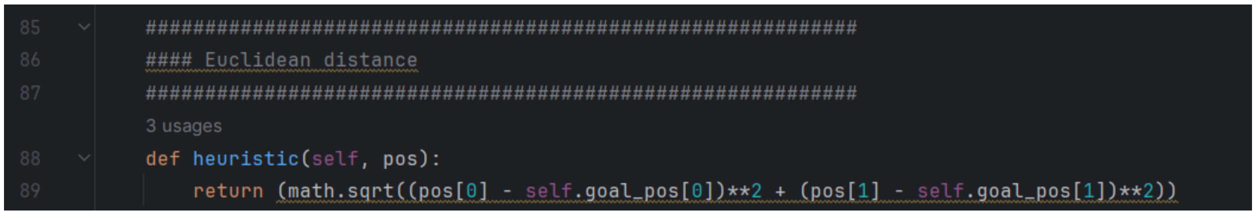


g=0 h=18									g=inf h=0
g=0 h=17	g=0 h=16	g=0 h=15	g=0 h=14	g=0 h=13	g=0 h=12	g=0 h=11	g=0 h=10		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=9		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=8		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=7		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=6		g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=5		g=inf h=0
g=0 h=11	g=0 h=10	g=0 h=9	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4		g=inf h=0
g=0 h=10									g=inf h=0
g=0 h=9	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4	g=0 h=3	g=0 h=2	g=0 h=1	g=0 h=0

Figure 3: Finding the shortest path through the maze using the Greedy Best First algorithm

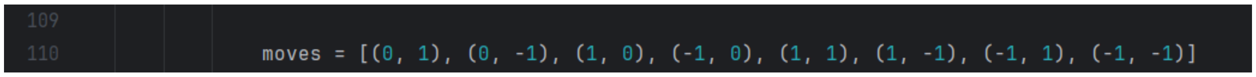
Problem 2:

For this problem, the heuristic was changed from being calculated through the Manhattan distance to being calculated through the Euclidean distance. This only required a minor change in the code within the method that calculates the heuristic; this change is illustrated in Figure 4. Additionally, this problem allows the agent to move in diagonal directions. This change was made when the agent first looks at which direction to move. In the previous application, this only allowed N, S, E, W thus one coordinate in the tuple was always changed by zero. To account for the agent being able to move diagonally as well, this section was edited to additionally have options where both coordinates in the tuple were edited at the same time. This change is reflected in Figure 5. Finally, this problem asks that the direction that the agent looks at for possible moves is calculated in a random order. To accomplish this, each time the agent picks a random direction from the list, and, as long as that direction has not already been looked at, checks that direction. This random selection repeats each time until the agent has picked the best direction to go in. This code change is demonstrated in Figure 6.



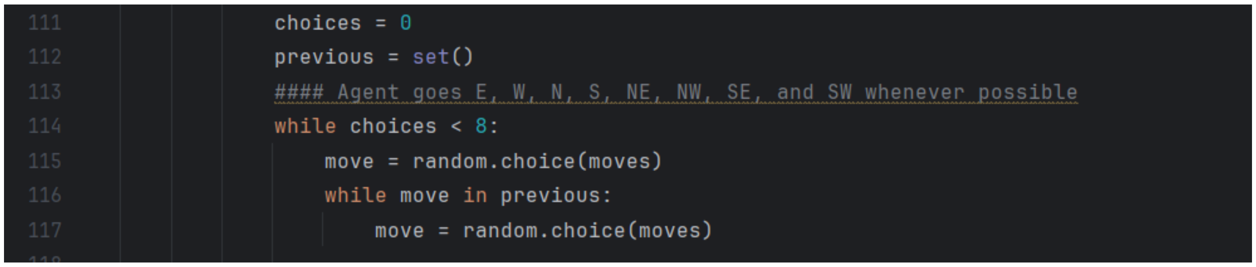
```
    ################################################################
    #### Euclidean distance
    ################################################################
    3 usages
    def heuristic(self, pos):
        return (math.sqrt((pos[0] - self.goal_pos[0])**2 + (pos[1] - self.goal_pos[1])**2))
```

Figure 4: The heuristic is now calculated using the Euclidean Distance instead of Manhattan Distance



```

            moves = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (1, -1), (-1, 1), (-1, -1)]
```

Figure 5: The moves allowed include all 8 directions: N, S, E, W, NW, NE, SW, SE.



```
            choices = 0
            previous = set()
            #### Agent goes E, W, N, S, NE, NW, SE, and SW whenever possible
            while choices < 8:
                move = random.choice(moves)
                while move in previous:
                    move = random.choice(moves)
```

Figure 6: The directions checked for possible move each time the agent makes a new move is chosen at a random order.

With these updated changes, both the A* algorithm and the Greedy Best First algorithm, were used to find the path through the maze. The path found by the A* algorithm is shown in Figure 7 whereas the path found by the Greedy Best First algorithm is shown in Figure 8. The A* algorithm utilizes the diagonal moves and originally moves diagonally towards the goal because the heuristic is estimated by the Euclidean distance and the quickest path would be diagonally. It begins to backtrack because of the actual path cost adding in to the evaluation function, and it determined a better way to reach the goal. On the other hand, the Greedy Best First algorithm only relies on heuristic, so it continues moving diagonally until it would block itself up against the wall and then fully backtracks. It takes a longer path than A* because it does not take into account the actual path cost and only relies on the heuristic.

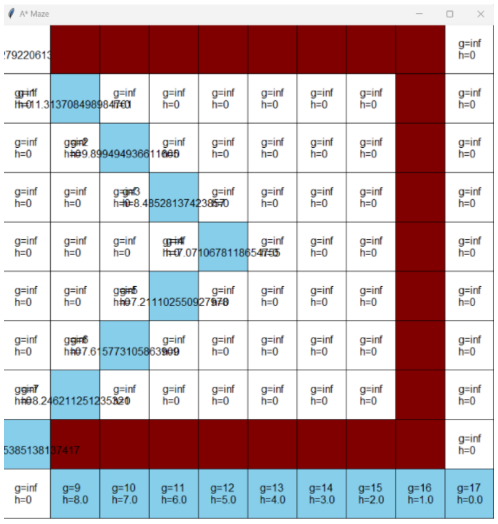


Figure 7: The path through the maze found by the A* algorithm when directions checked for possible moves were chosen by random, diagonal directions were allowed, and the heuristic was found using the Euclidean distance.

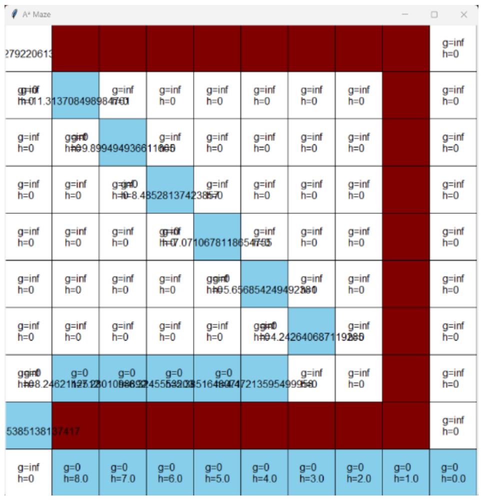


Figure 8: The path through the maze found by the Greedy Best First algorithm when directions checked for possible moves were chosen by random, diagonal directions were allowed, and the heuristic was found using the Euclidean distance.

Problem 3:

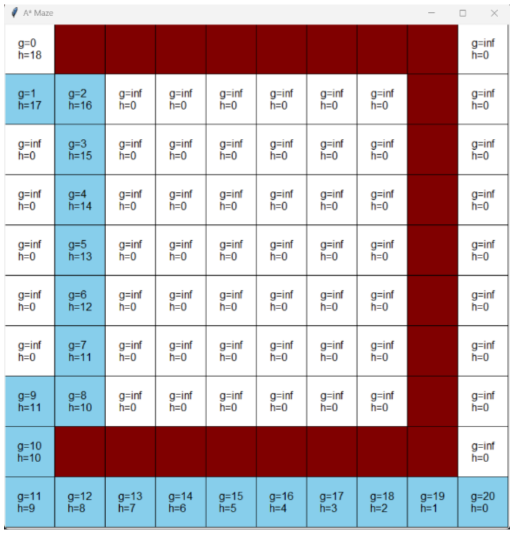
1.

alpha	beta	Observed behavior
1	2	The path changed to move over a row before moving back and continuing down.
2	1	Path did not change at all

2	2	Path did not change at all
3	2	Path did not change at all
2	3	Same changes as observed when alpha was 1 and beta was 2

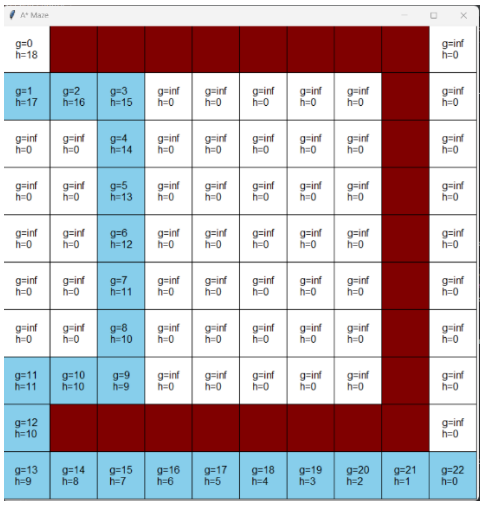
The changes in alpha did not change the path that was taken because the actual cost, g, is the same for each move. Since it is the same for each move, alpha just changes g by a factor thus changing the evaluation function by the same amount for each move. This keeps the decisions consistent with if alpha was kept at 1. When beta is changed, the path minorly changes because the heuristic was effected. With Manhattan distance, being scaled by a factor does not keep the heuristics consistent across moves. This causes the heuristics to effect which moves the agent takes thus effecting the path. When alpha and beta were equal, the evaluation function was increased by a constant factor thus not changing the path that the agent took just the total cost it accumulated.

2. As beta increases, the bias for states closer to the goal increases. The path is changed by moves to the right being favored early on until eventually going down whereas, when beta was 1, the path went down first. This is because, horizontally, the early states are closer to the goal in the paths produced from higher beta values. As beta increases, the number of moves to the right before beginning to move downward also increases. Once beta reaches a factor of 5, higher values of beta do not effect the path in anyways from the path produced with a beta of 5. The figures below show the different paths for different values of beta.



g=0 h=18	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	g=inf h=0
g=1 h=17	g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=9 h=11	g=8 h=10	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=10 h=10	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	g=inf h=0
g=11 h=9	g=12 h=8	g=13 h=7	g=14 h=6	g=15 h=5	g=16 h=4	g=17 h=3	g=18 h=2	g=19 h=1	g=20 h=0

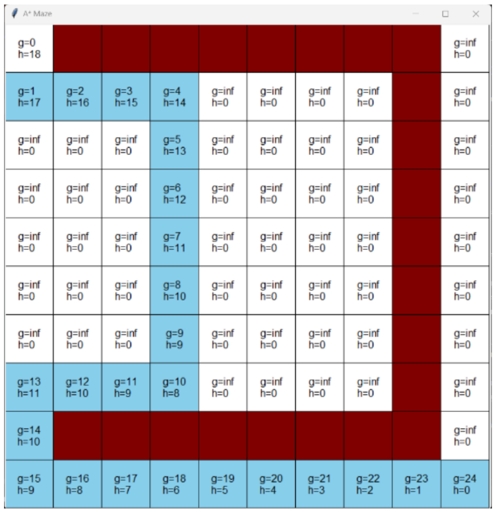
Beta = 2



A* Maze

g=0 h=18	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	g=inf h=0
g=1 h=17	g=2 h=16	g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=8 h=10	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=11 h=11	g=10 h=10	g=9 h=9	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=12 h=10	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	g=inf h=0
g=13 h=9	g=14 h=8	g=15 h=7	g=16 h=6	g=17 h=5	g=18 h=4	g=19 h=3	g=20 h=2	g=21 h=1	g=22 h=0

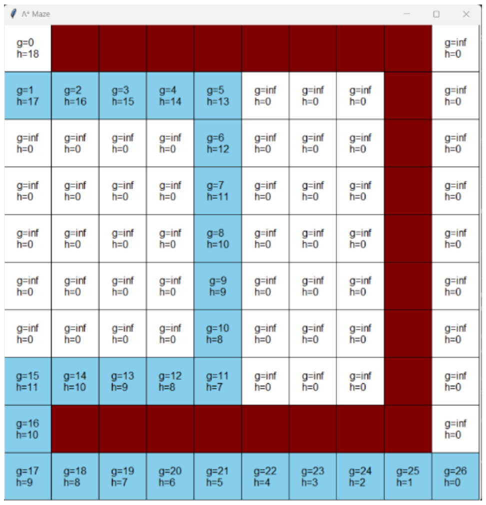
Beta = 3



A* Maze

g=0 h=18	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	g=inf h=0
g=1 h=17	g=2 h=16	g=3 h=15	g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=8 h=10	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=9 h=9	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=13 h=11	g=12 h=10	g=11 h=9	g=10 h=8	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=14 h=10	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	g=inf h=0
g=15 h=9	g=16 h=8	g=17 h=7	g=18 h=6	g=19 h=5	g=20 h=4	g=21 h=3	g=22 h=2	g=23 h=1	g=24 h=0

Beta = 4



A* Maze

g=0 h=18	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	g=inf h=0
g=1 h=17	g=2 h=16	g=3 h=15	g=4 h=14	g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=8 h=10	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=9 h=9	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=10 h=8	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=15 h=11	g=14 h=10	g=13 h=9	g=12 h=8	g=11 h=7	g=inf h=0	g=inf h=0	g=inf h=0	🟥	g=inf h=0
g=16 h=10	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	g=inf h=0
g=17 h=9	g=18 h=8	g=19 h=7	g=20 h=6	g=21 h=5	g=22 h=4	g=23 h=3	g=24 h=2	g=25 h=1	g=26 h=0

Beta = 6