SNAKE GAME

LIPSA DAS- (22051173)
AASTHA SINGH- (22051653)
UMME SALEH- (22051733)

INTRODUCTION:

This report details the optimization efforts undertaken to enhance the AI-driven Snake game, focusing on refining search algorithms, improving food creation, and ensuring smooth gameplay at higher difficulty levels.

CHANGES IN SNAKE.PY:

(a). We added one generate food function to avoid creation of food on obstacles.

```
45  # Function to generate food avoiding obstacles
46  def generate_food(rows, cols, snake_pos, obstacles):
47      while True:
48          food = (random.randint(0, rows - 1), random.randint(0, cols - 1))
49          if food not in obstacles and food != tuple(snake_pos):
50              return list(food)  # Convert tuple to list to match existing code
51
52  # Food position
53  food_pos = generate_food(ROWS, COLS, snake_pos, set())
54
```

This will check the condition whether the food is not generated in obstacles and on the snake itself.
For food creation we call generate_food function and it will create food at the right position.

(b). We check the exact position of food and snake rather than checking the position of its object created.

```
    # Check if AI reaches food (Increase Score, Relocate Food)
    if tuple(snake_pos) == tuple(food_pos):
        score += 1  # Increase score
        food_pos = generate_food(ROWS, COLS, snake_pos, obstacles)
        path = []  # Reset path to calculate new one
```

SEARCH_ALGORITHM.PY:

1. Greedy Best First Search Optimization

- **Heuristic Function:**
  - Greedy Best First Search is optimized by using a heuristic function (Manhattan distance), which prioritizes nodes based on their estimated distance to the goal. The heuristic guides the search to move toward the goal efficiently, which reduces the number of nodes explored.
- **Priority Queue:**
  - The algorithm uses a priority queue (heapq) to always process the node with the lowest heuristic value first. This ensures that the search is focused on the most promising paths.
- **Visited Set:**
  - A visited set is used to prevent revisiting nodes, optimizing the search by eliminating redundant work.
- **Early Termination:**
  - The algorithm terminates immediately once the goal is reached, preventing further exploration.

```
PS C:\Users\KIIT\Desktop\6th sem\22051653\AI Lab\src> python snake.py level3 greedy_bfs
pygame 2.6.1 (SDL 2.28.4, Python 3.10.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
Greedy BFS Time Taken: 0.0000000 seconds
Greedy BFS Time Taken: 0.0000000 seconds
Greedy BFS Time Taken: 0.0000000 seconds
Greedy BFS Time Taken: 0.0000000 seconds
Greedy BFS Time Taken: 0.0000000 seconds
Greedy BFS Time Taken: 0.0000000 seconds
Greedy BFS Time Taken: 0.0000000 seconds
Greedy BFS Time Taken: 0.0000000 seconds
Greedy BFS Time Taken: 0.0000000 seconds
Greedy BFS Time Taken: 0.0000000 seconds
Greedy BFS Time Taken: 0.0000000 seconds
Score: 10
PS C:\Users\KIIT\Desktop\6th sem\22051653\AI Lab\src>
```

We got the highest score in this algorithm .

2. Uniform Cost Search (UCS) Optimization

- **Priority Queue**:
  - UCS uses a **priority queue (heapq)**, which ensures that the node with the lowest cost is processed first. This reduces unnecessary exploration and optimizes the pathfinding process by always expanding the least costly nodes first.
- **Cost Tracking**:

    - The algorithm tracks the cost of reaching each node (cost_so_far). This ensures that once a node is reached with a lower cost, the algorithm updates the priority queue to reflect this improvement, ensuring that the optimal path is always prioritized.
- **Visited Set**:
    - Similar to other algorithms, UCS uses a visited set to avoid revisiting already processed nodes.
- **Early Termination**:
    - UCS stops as soon as the goal is found, avoiding any unnecessary exploration after reaching the goal.

```
Hello from the pygame community. https://www.pygame.org/contribute.html
Usage: python snake.py <level> <search_algorithm>
PS C:\Users\KIIT\Desktop\6th sem\22051653\AI Lab\src> python snake.py level3 ucs
pygame 2.6.1 (SDL 2.28.4, Python 3.10.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
UCS Time Taken: 0.0000000 seconds
UCS Time Taken: 0.0040736 seconds
UCS Time Taken: 0.0029857 seconds
UCS Time Taken: 0.0009284 seconds
UCS Time Taken: 0.0019720 seconds
UCS Time Taken: 0.0000000 seconds
UCS Time Taken: 0.0010223 seconds
UCS Time Taken: 0.0000000 seconds
UCS Time Taken: 0.0000000 seconds
UCS Time Taken: 0.0032387 seconds
Score: 9
```

3. Breadth First Search (BFS) Optimization

- **Queue for BFS**: The BFS uses a **queue (deque)**, which operates in O(1) time for both enqueue and dequeue operations, making it an efficient way to explore the grid level by level.
- **Visited Set**: A visited set is used to track the positions already explored. This prevents revisiting nodes, which reduces unnecessary processing and speeds up the algorithm.
- **Path Reconstruction**: BFS not only explores but also tracks the path taken to reach the goal by appending the direction taken at each step. This provides a clear path when the goal is reached.
- **Early Termination**: As soon as the goal is reached, BFS returns the path, avoiding further unnecessary exploration.

```
PS C:\Users\KIIT\Desktop\6th sem\22051653\AI Lab\src> python snake.py level3 bfs
pygame 2.6.1 (SDL 2.28.4, Python 3.10.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
BFS Time Taken: 0.0010102 seconds
BFS Time Taken: 0.0000000 seconds
BFS Time Taken: 0.0020812 seconds
BFS Time Taken: 0.0020316 seconds
BFS Time Taken: 0.0010176 seconds
BFS Time Taken: 0.0010328 seconds
BFS Time Taken: 0.0010211 seconds
BFS Time Taken: 0.0000000 seconds
BFS Time Taken: 0.0000000 seconds
Score: 8
```

4. A* Search Optimization

- **Heuristic + Cost:**
  - A* combines cost and heuristic to guide its search. It uses the sum of the cost so far (new_cost) and the heuristic value (heuristic(new_pos, goal)) to prioritize nodes, ensuring both the shortest path and the most promising path are considered. This makes A* an optimal and efficient algorithm.
- **Priority Queue:**
  - Like UCS and Greedy BFS, A* uses a priority queue (heapq), which ensures that nodes with the lowest combined cost and heuristic are expanded first.
- **Visited Set:**
  - A visited set prevents revisiting nodes and helps optimize the search by removing redundant operations.
- **Early Termination:**
  - As soon as the goal is reached, the algorithm terminates, returning the path and avoiding further unnecessary exploration.

```
Score: 8
PS C:\Users\KIIT\Desktop\6th sem\22051653\AI Lab\src> python snake.py level3 a*
pygame 2.6.1 (SDL 2.28.4, Python 3.10.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
A* Time Taken: 0.0000000 seconds
A* Time Taken: 0.0009861 seconds
A* Time Taken: 0.0000000 seconds
A* Time Taken: 0.0008576 seconds
A* Time Taken: 0.0010200 seconds
A* Time Taken: 0.0000000 seconds
A* Time Taken: 0.0009208 seconds
A* Time Taken: 0.0010214 seconds
Score: 7
```

MAJOR BREAKTHROUGH OPTIMIZATION: IDS GIVEN BELOW

5. Iterative Deepening Search (IDS) Optimization

Key Optimizations in IDS:

The **IDS** algorithm has been significantly optimized for both performance and functionality:

- **Depth-Limited Search (DLS)**:
  - The algorithm uses **depth-limited search** (dls) to progressively explore deeper levels of the grid until a solution is found or the maximum depth is reached.
  - Each depth iteration is independent, allowing the algorithm to adaptively search at different levels while still adhering to the depth limit, ensuring it doesn't go too deep and waste time exploring irrelevant nodes.
- **Time Limit Check**:
  - The algorithm is optimized by enforcing a **time limit** (time_limit = 5 seconds) on each search iteration. This ensures that the algorithm doesn't run indefinitely and provides feedback if it's taking too long. It returns Time Limit Exceeded when the algorithm runs past the defined time, which helps in avoiding unnecessary delays.
- **Early Termination**:
  - The dls function terminates early if the goal is found, immediately returning the path without further unnecessary exploration.
- **Visited Set**:
  - A visited set is used within the depth-limited search to avoid revisiting the same positions, improving the efficiency and preventing cycles.
- **Adaptive Depth Expansion**:
  - The depth starts at zero and increases progressively (depth += 1). The algorithm adjusts its search based on the goal's proximity and the available time, which makes it adaptive to different scenarios.
- **Efficient Result Construction**:
  - The path is built recursively by appending directions at each level, allowing for a clean and efficient path reconstruction when the goal is found.

```
PS C:\Users\KIIT\Desktop\6th sem\22051653\AI Lab\src> python snake.py level3 ids
pygame 2.6.1 (SDL 2.28.4, Python 3.10.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
IDS Time Taken: 0.0050404 seconds
IDS Time Taken: 0.0070117 seconds
IDS Time Taken: 0.0047028 seconds
IDS Time Taken: 0.0085032 seconds
IDS Time Taken: 0.0121996 seconds
IDS Time Taken: 0.0018604 seconds
IDS Time Taken: 0.0035772 seconds
Score: 6
```

6. Random Move Algorithm Optimization

- **Limited Number of Moves**: The algorithm limits the search to 1000 steps (for _ in range(1000)). This ensures that the algorithm won't run indefinitely in the event that no valid path is found, which is crucial for performance in a potentially large or blocked grid.

- **Efficient Movement Check**: Each time a random move is chosen, the algorithm checks if the new position is within bounds.

```
    if 0 <= new_pos[0] < rows and 0 <= new_pos[1] < cols and new_pos not in obstacles:
        path.append(direction)
        current = new_pos
```

  This ensures that the algorithm doesn't perform unnecessary work by checking invalid or blocked positions.

- **Early Termination**: The algorithm checks if the current position matches the goal and terminates early, providing the path if found within the limited moves.

```
PS C:\Users\KIIT\Desktop\6th sem\22051653\AI Lab\src> python snake.py level3 random
pygame 2.6.1 (SDL 2.28.4, Python 3.10.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
Random Move Time Taken: 0.0000000 seconds
Random Move Time Taken: 0.0020282 seconds
Random Move Time Taken: 0.0030074 seconds
Random Move Time Taken: 0.0020051 seconds
Random Move Time Taken: 0.0019412 seconds
Random Move Time Taken: 0.0019522 seconds
Random Move Time Taken: 0.0000000 seconds
Random Move Time Taken: 0.0018551 seconds
Random Move Time Taken: 0.0025475 seconds
Random Move Time Taken: 0.0021324 seconds
Random Move Time Taken: 0.0027454 seconds
Random Move Time Taken: 0.0034394 seconds
Random Move Time Taken: 0.0020211 seconds
Random Move Time Taken: 0.0036016 seconds
Random Move Time Taken: 0.0022161 seconds
Random Move Time Taken: 0.0019429 seconds
Score: 2
```

7. Depth First Search (DFS) Optimization

- **Stack for DFS**: The algorithm uses a **stack**, ensuring that the last explored position is visited first, which is the hallmark of DFS. The stack operations (`pop` and `append`) are optimized by using Python's built-in list structure, which allows efficient pop operations from the end of the list.
- **Visited Set**: The use of a `visited` set ensures that the algorithm doesn't revisit nodes and avoid unnecessary cycles.
- **Early Termination**: As with BFS, the algorithm stops immediately once the goal is found, reducing unnecessary steps.
- **Path Construction**: DFS maintains a path stack and adds directions at each step. When the goal is reached, the complete path from start to goal is returned.

```
PS C:\Users\KIIT\Desktop\6th sem\22051653\AI Lab\src> python snake.py level3 dfs
pygame 2.6.1 (SDL 2.28.4, Python 3.10.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
DFS Time Taken: 0.0000000 seconds
DFS Time Taken: 0.0019243 seconds
Score: 1
```

PERFORMANCE:

ALGORITHM	LEVEL0 SCORE	LEVEL1 SCORE	LEVEL2 SCORE	LEVEL3 SCORE
Random	0	1	0	2
BFS	8	9	8	8
DFS	1	2	2	1
UCS	8	7	9	9
IDS	6	3	7	6
A*	7	8	6	7
Greedy BFS	6	14	8	10