Rabbit and Carrot Game: Explanation and Report

Introduction

The Rabbit and Carrot Game is a simple game in which the goal is to find the shortest route for a rabbit to collect all of the carrots on a two-dimensional grid. The rabbit begins at a random location on the grid, and the carrots are distributed among some of the cells. The objective of this project is to implement three different algorithms to solve the game: Breadth-First Search (BFS), Depth-First Search (DFS), and A* search, and to display the game using the console in Python.

Experimentation Explained

The first step in the code is to initialize the game board randomly using the initialize_board() function. The size of the board is randomly generated between 3x3 and 5x5, and a random number of cells (between 3 and 5) are selected to contain carrots. The rabbit is then placed at a random position on the board.

The show_board() function is used to display the board on the console. This function converts the board into a numpy array and iterates over the rows and columns to display the contents of each cell in the board.

BFS Algorithm

The BFS algorithm is implemented using the bfs() function. The function finds the initial position of the rabbit on the board and initializes a queue, a set to keep track of visited cells, and a list to store the positions of all the carrots found. The function then enters a loop that continues until all the carrots have been found. In each iteration, the function dequeues the next position and actions from the queue, checks if the current position contains a carrot, adds it to the list of carrots found if it does, and checks the neighbouring cells of the current position and adds them to the queue if they have not been visited before.

DFS Algorithm

The DFS algorithm is implemented using the dfs() and dfs_search() functions. The dfs() function takes a board, a position, a set of visited cells, and a list of actions as input, and

implements the DFS algorithm to find the shortest path for the rabbit to collect all the carrots. The dfs_search() function initializes the starting position of the rabbit on the board, an empty set to keep track of visited cells, and an empty list to store the actions taken. It then calls the dfs() function on the initial position of the rabbit, passing in the board, the starting position, the set of visited cells, and the list of actions.

A* Search

The A* search algorithm is implemented using the heuristic(), find_carrots(), a_star(), and a_star_search() functions. The heuristic() function calculates the Manhattan distance between two points on the board, which is used as the heuristic to estimate the cost of reaching the goal. The find_carrots() function returns the positions of all the carrots on the board. The a_star() function implements the A* search algorithm using a priority queue to keep track of the nodes to be visited and a dictionary to keep track of the cost of reaching each node. The a_star_search() function initializes the starting position of the rabbit on the board and calls the a_star() function to find the shortest path to the carrots.

Comparing Algorithms

ALGORITHM	COMPLETENESS	OPTIMALITY	TIME COMPLEXITY	SPACE COMPLEXITY
BFS	Yes	Yes	O(b^d)	O(b^d)
DFS	No	No	O(b^m)	O(bm)
A* SEARCH	Yes	Yes	O(b^d)	O(b^d)

BFS, DFS, A* Search are implemented to search for the rabbit on the game board. The time and space complexities of both BFS, DFS and A* Search are shown in the table above, where b is the branching factor (i.e., the number of possible moves from each state), d is the depth of the shallowest goal node, and m is the maximum depth of the search tree.

BFS has a time complexity of O(b^d) and a space complexity of O(b^d) as it expands all the nodes at each level of the search tree before moving to the next level. This guarantees that BFS

will find the optimal solution, and it is also complete, i.e., it is guaranteed to find a solution if one exists.

DFS, on the other hand, has a time complexity of O(b^m) and a space complexity of O(bm), where m is the maximum depth of the search tree. Unlike BFS, DFS does not guarantee an optimal solution or completeness. It can get stuck in an infinite loop if there is a cycle in the search tree.

Finally, A* search's time and space complexity is the same as BFS, i.e., O(b^d). A* search uses an admissible heuristic function to guide the search towards the goal state, making it more efficient than BFS in terms of the number of nodes expanded. Like BFS, A* search is also complete and optimal.

Conclusion

In conclusion, this project successfully implemented three different algorithms to solve the Rabbit and Carrot Game: Breadth-First Search (BFS), Depth-First Search (DFS), and A* search. The game was displayed using the console in Python, and the algorithms were explained and tested. Each algorithm was able to find the shortest path for the rabbit to collect all of the carrots on the board. This project demonstrates how different algorithms can be used to solve a simple problem and can be used as a starting point for more complex games and applications.