CS7637 - Mini-Project 1: Sheep Wolves

Apurva Gandhi
agandhi301@gatech.edu

1 AGENT DESCRIPTION

1.1 Introduction

The agent has a single method, solve, that receives the initial number of sheep and wolves as integers and returns a list of 2-tuples that represent the moves required to get all the sheep and wolves to the right side of the river. If it is impossible to move the animals over according to the rules of the problem, the method returns an empty list of moves.

1.2 State Class

The solve method creates a State class, which is used to represent the current state of the game. The State class has several methods. Goal_state method is used to check if the current state of the object is a goal state. A goal state is defined as having all the sheep and wolves on the right bank of the river, with the boat also on the right bank. Furthermore, is_valid_state method checks if the current state of the game is valid or not.

1.3 Successors method

The solve method also defines a successors function that generates a list of possible next states (successors) that can be reached from the current state. The function takes in a current state object and loops through the possible moves (2 sheep, 2 wolves, 1 sheep + 1 Wolfe, 1 Wolfe only, or 1 sheep only) and for each move, it creates a new state object by subtracting the values of the move from the left side sheep and wolves, and adding the values of the move to the right side sheep and wolves, and sets the boat position to the right side. If the new state is valid, it is added to the 'successor' list and the 'parent' attribute of the new state object is set to the current state. If the boat is on the right side, the loop is the same but this time it subtracts values from the right side sheep and wolves and adds them to the left side sheep and wolves, and sets the boat position to the left side.

1.4 Breadth-first search (BFS)

The solve method then uses a Breadth-first search algorithm to search for the goal state, starting from the initial state. It uses a queue data structure (implemented as a deque from the collections module) to keep track of the states to be explored. It starts by adding the initial state to the queue and then repeatedly pops the first state from the queue, checks if it is a goal state and if not, generates its successors and adds them to the queue. The search continues until the goal state is found or the queue is empty. If the goal state is found, it constructs the list of moves required to reach the goal state by following the parent pointers of the states starting from the goal state.

2 AGENT PERFORMANCE AND EFFICIENCY

Several tests are done to decrease the amount of alternative moves that the agent will try. Because non-possible moves are excluded before the agent tries them, this improves the agent's productivity and efficiency. To start, it is necessary to evaluate whether the move is legal: there cannot be negative numbers of animals, and the number of sheep or wolves cannot exceed those of the original state. Once the illegal moves are rejected, the agent must determine if the move is a failure and so violates the rules, implying that the wolves outweigh the sheep on both sides of the river. Furthermore, implementation of BFS algorithm should avoids the agent to loop over the same solutions over and over again, being unproductive. If the solution is not possible after executing BFS algorithm, agent simply returns an empty list indicating that solution is not possible for the given sheep and wolves. Because the agent is made up of a smart tester, it does not struggle with any specific scenario or with a rising number of beginning sheep and wolves. It prevents a computational explosion by eliminating unlawful, ineffective, and invalid moves before pursuing them. This helps in arriving at a solution in a more efficient and timely manner. The agent uses a deque from the collections library to implement the breadth-first search. The algorithmic performance of the code will depend on the size of the search space, the branching factor of the problem, and the number of goal states in the problem. In general, the time complexity of a breadth-first search algorithm is $O(b\hat{d})$ where b is the branching factor and d is the depth of the solution, and the space complexity is $O(b\hat{d})$. The branching factor is the number of possible moves from any given state which is 5 and the depth is the number of moves required to reach the goal state, which is not fixed and depends on the initial state. Therefore, the performance of the algorithm is exponential in the depth of the search. Apart from above mentioned algorithm and concepts, agent does not do anything clever and out of the box.

3 HUMAN COMPARISON

No, implemented agent would not solve the problem as an average human would solve. The agent uses BFS algorithm which would compute to see if there is a possible solution. An average human would probably try to move and take different steps in some logical way to get to the goal state. However, an average human would not follow a particular pattern and it would not be able to solve complex problems in efficient way. There are some similarities between an agent and how a human would think. An average human would think of a move, check if it is valid, move the side if it is valid, and check if it a goal state. If not, it would probably continue to move and asses until he or she would find a solution or a path to solution. While agent might not completely think like a human would think, I think it solves problem with more accuracy and efficiently compared to an average human approach at solving similar problem.