COMP 424 Final Project Game: *Colosseum Survival!*

Course Instructor: Jackie Cheung and Bogdan Mazoure
Project TAs: Koustuv Sinha (koustuv.sinha@mail.mcgill.ca) **and Shuhao Zheng** (shuhao.zheng@mail.mcgill.ca)
Due Date: April 12th, 2022, 11:59PM EST
Team Members: Vishvak Raghavan and Mark Chen

1. Agent Explanation and Motivation

The agent that we put forward utilizes a mixture of two ideas using a heuristic that prioritizes proximity to the opponent agent, and the minimax algorithm. To begin, we analyzed that the game of Colosseum Survival had a static, observable, and discrete environment with deterministic actions. In addition to having the same environment characteristics, Colosseum Survival is similar to games like Tic-Tac-Toe or Dots and Boxes that feature a grid and look to draw out sections. From this, the complete and optimal properties of the minimax algorithm look well suited to take advantage of the static, observable, and discrete environment and deterministic actions of the game, being able to analyze board states, plan win conditions, and predict opponent actions.

However, the minimax algorithm alone is not sufficient to yield an optimal agent as Colosseum Survival differs from the other mentioned games in regards to a few aspects. Firstly, the 12x12 board and 6 move space means that an agent utilizing minimax algorithm has too many branches to compute feasibly, especially when accounting for increasing depth and opponent moves, due to the exponential nature of the algorithm.

Furthermore, Colosseum Survival is, as a game, more complex than Tic-Tac-Toe or Dots and Boxes. Moves in the aforementioned games occur with minimal restrictions, typically limited by whether or not a barrier or spot is already taken. Otherwise, an agent typically has access to the entire board to enact their move and shares a very similar action space with the opponent. This is not the case as the agent in Colosseum Survival is limited by the position of its avatar, its movement and actions restricted by pre existing barriers and the presence of the opponent's avatar.

Given the more complex rules, larger potential board states, and the new interplay of relational positioning between the agents, calculating the utility of a given board state has become more difficult. There are now cases where an agent can completely reverse a "winning" position by taking advantage of the barriers placed by the opponent's barriers or instances where agents can trap the opponent in subsections to restrict their movement. Overall, short of an "obvious" winning condition, minimax would be too expensive in terms of both memory and time for an agent to rely wholly on.

To compensate for the expensive processes of the minimax algorithm, it was determined that the agent also requires an heuristic in the case where minimax is unable to determine a win condition within a given depth. Ultimately, it was settled that the agent would use a heuristic that prioritizes proximity to the opponent. With this, if the minimax algorithm is unable to determine winning or losing game moves, the agent looks to cover the maximum distance required to be adjacent to the opponent.

This heuristic was decided as, in contrast to the minimax algorithm, it balances out many features of Colosseum Survival that the minimax algorithm neglects. In Colosseum Survival, proximity to the other agent entails a similar action space as both agents move the same distance. It follows that this heuristic, by reducing the complexity associated with the interplay of the positioning between agents, allows for the minimax algorithm to be much more effective from multiple perspectives.

Offensively, with the heuristic ensuring proximity to the opponent agent, the minimax algorithm is primed to take advantage of flaws in the opponent agent's strategy, especially if the opponent agents are unable to react effectively and account to a reactionary strategy. If the opponent attempts to make a matching-winning move, the provided agent will take the opportunity to react and close out any potential areas before, effectively reversing the opponent agent's strategy. This also carries over when it comes to the agent's self preservation as the heuristic will, when the opponent looks to close out an area, allow the agent to stay within the same region as the opponent.

Together, the minimax algorithm and the proximity heuristic is able to take advantage of the nature of Colosseum Survival's rules and game characteristics. The minimax algorithm ensures that the agent always takes optimal moves whenever viable and computationally feasible. The proximity heuristic allows for the agent to take advantage of the strategies of the opponent agents, effectively countering agents that do not adequately account for their opponent's actions, as well as balancing out the expensive computation cost of the minimax algorithm

2. Theoretical Basis

Before determining what algorithm we would use to implement our agent, we first decided on the general strategy our agent will use. We decided that the best course of strategy in coliseum survival would be to try to be as close to your opponent as possible. This is because if you are close to your opponent, it is generally easier to box them in as well as easier to counter any moves they make to trap you. Therefore, our agent uses a heuristic that minimizes the distance to the opponent. The first step in our algorithm is to find all the neighbouring tiles of the opponent's position that do not have a barrier in between. From each neighbouring tile we run a breadth first search that continues until it reaches a tile that is a valid move of our agent. Once a valid tile is reached, the search will terminate since this tile is the closest tile to the source tile of the search that is a valid move for our agent (follows the heuristic).

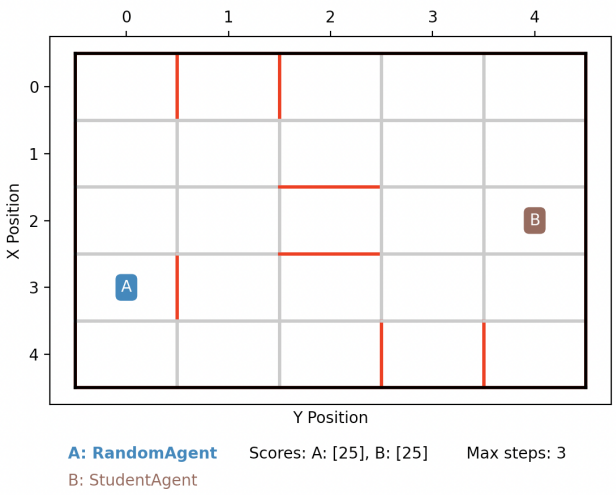

Figure 1: The opponent is placed at tile (3,0). The tiles (2,0) and (4,0) are tiles neighbouring the opponent's position and thus will be the start of the respective breadth first searches. Although the position (3,1) is a neighbouring tile, there is a barrier in between (3,0) and (3,1). Therefore no search will start from that tile.

BFS will only give us the position of the tile, but will not determine the possible barrier placement. Thus after BFS, for each possible tile position, we generate all the possible barrier options. Pseudocode for the barrier generation is provided below.

```
Moves = []
For tile in possible_moves:
        For i in range(0,4):
                X,y = tile
                Dir  = dir
                Move = ((x,y), dir)
                Moves.append(move)
```

From here we run a modified minimax algorithm on all the generated moves to choose thes best possible option. The minimax only runs to depth 2 (our possible move and the opponents theoretical response). The utility function in minimax calculates if it is a win, loss, tie, or none of the above (similar to how the check_endgame function works in World.py). Instead of working in a bottom up approach like a typical minimax would, our algorithm works in a top down approach where it calculates the utility of our possible move and then the utility of the opponent's move. Though our minimax function does not involve alpha-beta pruning, there is some pruning involved. First off, if the utility function determines one of the potential moves will win us the game, all subsequent moves are pruned, minimax terminates and the winning move is returned and played by the agent. We also effectively have pruned some of the move options before running minimax by running BFS to get a subset of all possible moves. If a potential move results in either a loss or a tie then we save the move as a worst case scenario, but we do not continue running minimax on these moves. Only moves that do not terminate the

game will run to the second depth. After the algorithm finishes running the best move is returned and played by the agent.

3. Advantages, Disadvantages, etc.

There are two aspects of the agent—the proximity heuristic and the minimax algorithm—each contribute and detract from the agent's effectiveness when considered individually and collectively.

To begin, minimax is a complete and optimal search which can absolutely deduce a winning and losing board state, giving it powerful capabilities when it comes to the most fundamental qualities of winning the game and preventing the opponent from winning for the game Colosseum Survival. However, such an approach is expensive computationally, especially if it is desired for the agent to be able to predict and account for future situations and opponent moves. This limits the usage of the algorithm, meaning that the agent can be only absolutely sure of the utility of its move for immediate and obvious situations.

On the other hand, the usage of the proximity heuristic prioritizes the leeway required by the minimax algorithm, allowing for the agent to be able to utilize the minimax algorithm whenever possible due to the similar action space between the agents. The issue with using the proximity heuristic is that its effectiveness is dependent on the opponent's agent to some degree. In some cases, if the opponent agent has no strategy at all, the proximity heuristic is unable to bring the agent closer to winning the game as it cannot react in a meaningful way. On the other end of the spectrum, if the opponent's agent has brilliant predictive abilities, it would be able to set up a strategy that, by the time the provided agent is able to react with the minimax algorithm, it is already in a losing situation.

Together, the proximity heuristic and the minimax algorithm are complementary when facing most agents. If the opponent's agent is not complex enough to react, the provided agent is poised to take advantage of the opponent's strategies. However, their integration relies on some strong premises and presumptions. Minimax requires that the opponent plays optimally. In this case, the agent's "optimal" would be the adversary following the same strategy as the provided agent (minimizing distance to the opponent). However, not all opponent agents necessarily act in this manner, reducing the agent's search optimality.

4. Other Approaches

In the initial attempt, the agent was programmed with the simple gimmick of approaching and placing the barrier closest to the opponent agent as fast as possible. This tactic was meant to attempt to trap the opponent agent in their own area. Against a random agent, it yielded very acceptable results but only primarily due to the fact that the random agent would end up sealing off the area and therefore losing the game by itself. Against itself, it became apparent that this

initial agent was flawed as the copies would yield many incomplete areas with one or two barriers, unable to cohesively close out a given area. This issue compounded when it was put up against a human agent where it would be unable to recognize other obvious win conditions and oblivious when the human agent would begin to set up an area to trap it in.

In comparison, the final version of the provided agent utilizes the minimax algorithm so that win conditions are capitalized on and moves that yield a loss are recognized and avoided. When playing against itself, matches with the final version of the agent take significantly longer as the agent notices and avoids losing moves. Additionally, the final agent actively recognizes and takes into account the presence of barriers as a restriction of its movement space, allowing it to path more efficiently toward the opponent agent.

5. Improvements

While the provided agent is very effective in many cases, it stands to benefit from refinement regarding its ability to account for edge cases as well as its overall search efficiency.

While this agent was implemented with the standard minimax algorithm, more development and time spent would allow for better pruning (Alpha-Beta or otherwise) of the potential moves, allowing for greater efficiency. Such gains would yield a better game performance as the more efficient usage of memory and time resources would allow minimax to compute more states to a greater depth.

Additionally, we could have given minimax more moves to traverse through by modifying the BFS. BFS could be modified to not terminate once a valid move is reached. Rather it could continue until three or five valid moves are reached. Even though some of those moves will not be the closest to the opponent, sometimes the shortest path is not always the best option.

Another potential improvement would be a foray into determining if the Monte Carlo Tree Search yields predictive results more accurately and efficiently than minimax. It's possible that, by not using a heuristic, Monte Carlo Tree Search is able to perform consistently regardless of the opponent agent's strategy. Furthermore, by focussing on more promising trees and board states, Monte Carlo Tree Search may be able to expend more time and memory resources on traveling greater depths and yielding an better performance overall.

A final potential addition that may result in an improved agent is the breakdown and simplification of the board state into smaller board states. Pre-computing and using some of the allocated memory to remember specific smaller board states that yield win conditions may prove to be computationally efficient if, as the game goes on, the agents find themselves fighting in a smaller board as areas are blocked off by barriers.