Artificial Intelligent Isolation Game-Playing Agent

Anas Abdulahi
aaabdulahi@stcloudstate.edu

Nadika Bandara
jnbandara@stcloudstate.edu

ABSTRACT

The Isolation game playing agent was created to mimic intelligent behavior while playing a two player strategy game. This was done by implementing a minimax tree that searches through values created by a static evaluation function. The program chooses the best moves for itself in the next play and also in future possible moves. A static evaluation function determines the value of the state by the available legal moves and the locations on the board.

The main test subjects are the developers of the game: 2 players with no prior isolation game knowledge, and one player with basic isolation game knowledge. In conclusion, the program exhibited intelligent behavior by playing against all opponents and showing the capacity to defeat humans.

Keywords

Minimax search, Static Evaluation Function, Game tree, Strategy two player game , Isolation game, Game state

INTRODUCTION

In artificial intelligence, games of strategy in which two players oppose each other enlisting a range of cognitive abilities, are well defined, and yet are complicated enough to be challenging, constitute an important sub area of Artificial intelligence.

Isolation board game is a two-player abstract strategy game of perfect information played on a 7x7 board. The game is especially valuable in this application of AI as it has a definite goal, well defined rules, and is fairly easy to explain to new players.

The game of isolation will be introduced in depth, together with the game's rules, our variation of the game, and how to play the game. The structure and design choices will also be discussed as well the implementation of the AI agent.

1. THE GAME: ISOLATION

1.1. Background

Isolation board game is a two-player "don't get stranded" abstract strategy game of perfect information played on a 7x7 board. The board game is one of a series of 2 player strategy games by Lakeside from the 1970s. Players alternate turns moving a single piece from one cell to another on a board. The board is initially filled with squares except the starting points of both players. The first player with no remaining legal moves loses, and the opponent is declared winner.

1.2 Rules of the Game

Several versions of the game exist. Our version takes place on a 7x7 board. Each of two players has a single piece on that board. Either player may play first. Play begins with each player placing his/her piece anywhere on the board, as this picture indicates:

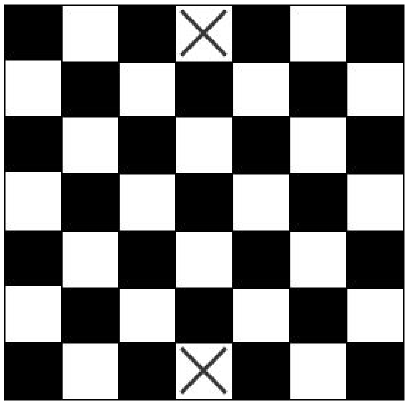

Figure 1: A representation of the 7x7 board use to play the Isolation game with 2 pieces inplace

1.3 How to Play

The players take turns moving their pieces. Moves imitate those of the knight in chess. They are L-shaped: two

spaces vertically and one horizontally, or two spaces horizontally and one vertically.
The goal is to prevent one's opponent from being able to move at all. The first player with no remaining legal moves loses, and the opponent wins.

The user interface indicates the player's turns at the top using their respective icons. refer to figure 3 where it is currently the white unicorn's turn.

Every ilegal square, except the players' current position, is highlighted red. When a player moves to a different square, their previous square becomes an ilegal square.

2. STRUCTURE AND DESIGN

The Structure and the design of the game is broken down into the following components: The board representation and design, and the implementation of the game rules. Since the purpose of the game implementation is testing the AI agent, the design of the AI agent count as a major component of the program as well.

The overall user interface of the program is built using the React.JS, a JavaScript framework which generates web based frontend HTML components that are bound to JavaScript classes. React.JS is especially selected because of the ease of user interface design, program state management and fast rendering.

2.1 Board Representation and Design

Since the React.js framework is inplace, to render a single square in the Isolation game board a <*button*> HTML attribute is used and modified with CSS to be displayed as a square.

The program iterates through a 2-dimensional array and pushes each <*button*> attribute to each of the 7X7 board squares. As a result, the 2-dimensional array renders as a board on a web page with 7X7 clickable squares.

With the above UI components, the current state of the game is declared in the program by a 2 dimensional array with the values displayed in each HTML square. The values are as follows. A boolean value that indicates the player's turn, player 1 location on the board and player 2 location on the board. All of the elements of the 7x7 2-dimensional array are initially Null. Once player 1 places their piece anywhere on the board, the location of the 2-dimensional array becomes "X" and for the player 2 it becomes "O". The UI re-renders with the new state of the board automatically with the use of the React framework. Once a player moves to a different location, the previous location becomes "@" on the 2D array and the new location updates accordingly and re-renders the board. With each of these updates, the player 1 and player 2 locations on the state updates to the latests locations accordingly and the player's turn boolean alternates to update turn rights.

```
[  ["X", null, null, null, null, null, null],
   [null, null, null, null, null, null, null],
   [null, "@", "O", null, null, null, null],
   [null, null, null, null, null, null, null],
   [null, null, null, "@", null, null, null],
   [null, null, null, null, null, null, null],
   [null, null, null, null, null, null, null]  ]
```

Figure 2: A sample representation of a 2-dimensional array that contains the game state

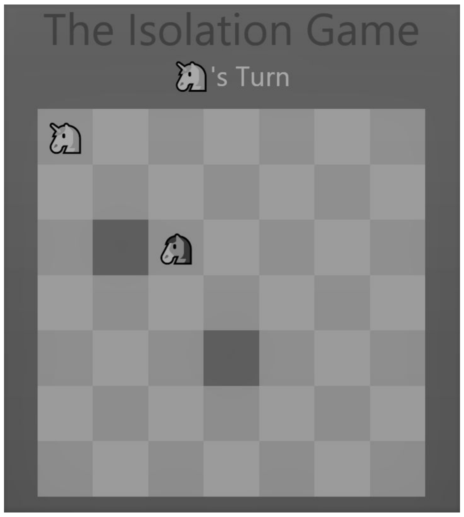

Figure 3: A sample representation of the user interface of the game rendered using the 2 dimensional array which contains the game state

2.2 Implementation of the Game Rules

The implemented isolation game variation has a unique set of rules. There are three major rules by which each player has to comply with. The rule to pick the starting locations, check if a move is legal and the rule to choose a winner.

When the game start, the player chooses whether the program plays first or the player. The other player then plays their turn and then both players alternate placing their piece on any square of the board.

To make a move, the player clicks on a desired legal square and the 2D array updates with the player 1 location using the “X” symbol. If the program had this turn, a random number generator chooses an element from [0][0] to [6][6] and updates the game state.

In this particular Isolation game implementation, the players can only choose L-shaped moves. Which means either moving two spaces vertically and one horizontally or two horizontally and one vertically. All the moves are required to be bound to the borders of the board as well. This rule is applied to the game by calling a function named *checkIfMoveLegal* to each move which takes the moving location of the player and returning if it is legal or not. If the move isn’t legal, the player has to make a different move as the piece will not move to the selected square. This function iterates through each possible move for the player and match it with the players intended move to decide if the move is legal or not.

Finally the decision of choosing the winner. If a player has no legal moves left, the opponent wins the game. To check this, the *calculateWinner* function is called whenever there’s a change in the game state. This function counts all possible legal moves of each player and pass false if both players has at least one legal move left or pass the opponent players identification if a player has no legal moves left.

3. ARTIFICIAL INTELLIGENT AGENT

Artificial Intelligence (AI) is the area of understanding the computation of intelligence. There are many sub areas under the AI umbrella. The Isolation game occupies the game playing study of AI and the focus is on two player strategy board games. An AI agent capable of understanding the game rules and make moves based on intelligent predictions is to be developed.. “Agents are viewed as black boxes that take in knowledge, past experiences, goals/values, and observations and output actions”(Poole, David Lynton, Alan K. Mackworth, and Randy Goebel) is a simple definition of an AI agent. The AI agent is implemented using a blind search mechanism which chooses a move on each game state with the agent’s turn.

3.1 Game Tree

Each problem has a search space of solutions. Particularly, a board game has a search space which can be structured as tree. The root node contains the current board configuration or the current state of the game. Each child of the root contains a possible states of the game after a legal move by the player. The children of these children nodes represents the possible states of the game after the opponent's move. Likewise the tree goes on until the leaf nodes which represents the state where the game is over.

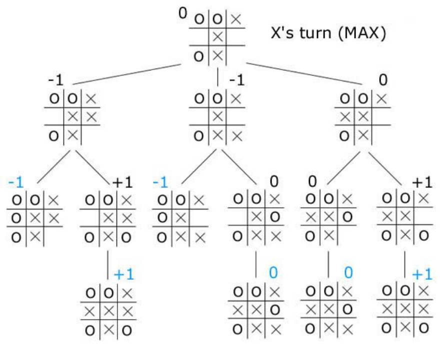

Figure 4: The game tree of the two player strategy board game tic tac toe (Lin, Y).

For the Isolation game, root node of the game tree at any given game state will have a maximum of 8 children because the maximum possible amount of legal moves for a player is 8. These apply for any parent nodes as well. When the tree grows deeper, the amount of children decreases because the amount of used spaces increase. Another reason to have less children would be to locate the piece at a corner of the board.

3.2 Minimax Search

Since the game’s search space is structured as a tree, there should be an effective algorithm to traverse the game tree in order to find the best move to make. This is when the minimax search comes into the play. Minimax is a search algorithm which search through the game tree until a specified depth to find out the best path to move towards on the search space. Essentially this is a hill climbing search algorithm.

```
static minimax(currentState, depth, maximizingPlayer) {

  if (depth == 0 || this.checkGameOver(currentState))
    return this.SEF(currentState);

  const children = this.getChildren(currentState);
  if (maximizingPlayer) {
    let maxEvaluation = -Infinity;

    children.forEach(child => {
          const evaluation = this.minimax(child, depth - 1,
false);
              maxEvaluation = Math.max(maxEvaluation,
evaluation);
    });
    return maxEvaluation;
  } else {
    let minEvaluation = Infinity;

    children.forEach(child => {
        const evaluation = this.minimax(child, depth - 1,
true);
      minEvaluation = (minEvaluation, evaluation);
    });
    return minEvaluation;
  }
 }
}
```

Pseudocode 1: Pseudocode for the minimax search algorithm

According to the above pseudocode, the minimax function takes the current state of the game board, depth to search and a boolean value indicating if the player wants to maximise(if it is the program's turn) or minimize(if it is the player's turn). Since this is a recursive function, the first if condition acts as the base case. If the depth is zero, which means the search has iterated till the desired depth or if the search iterated until the leaf node which the *gameOver* function returns, the static evaluation of the current game state is return. More about the static evaluation function implementation is discussed further in this paper.

If it is not the base case, all possible children of the current state are searched. Then if it is the programs turn, the algorithm loops through each child node and get the maximum evaluation by calling the minimax function recursively. If it is the player's turn, the children are looped through the recursive call to get the minimum evaluation.

This search mechanism trevors the best path to make by the program accordingly to the given depth. This path is decided based on the evaluation, which the static evaluation function is responsible for.

3.2 Static Evaluation Function

The static evaluation of a game state represents how good is the current state when compared with the goal state. Static evaluation function is the core intelligence in an AI agent of this sort. The decision on which moves are good and which moves are bad completely depends on the evaluation done by the static evaluation function. For the Isolation game, the static evaluation counts all possible legal moves for each player. Then the total available moves for the AI agent gets subtracted by the total available moves for the human player. If the resulting value is 0, the game state is neither good or bad for both player. If the value if more that 0, this is a good game state for the program and bad for the human player. If the value is smaller than 0 or a negative integer, the game state is bad for the program and good for the human player. These values get traversed until the root state using the minimax tree so the program can choose a path that can lead to a beneficial outcome.

3.3 Performance

In an attempt to validate the performance of the game, we've tested it against the game developers, 2 players with no prior isolation game knowledge, and one player with basic isolation game knowledge.

According to Herbert Simon, learning denotes changes in a system that enable a system to do the same task more efficiently the next time.

Because of the fact that the agent does not learn from previous game plays, the agent's decision making is linear throughout the tests. It is able to make decisions fast enough before the turn is switched to the opponent.

CONCLUSION

in conclusion, he Isolation game playing agent was created to mimic intelligent behavior while playing a two player strategy game. This was done by implementing a minimax tree that searches through values created by a static evaluation function. The program chooses the best moves for itself in the next play and also in future possible moves. A static evaluation function determines the value of the state by the available legal moves and the locations on the board. Because of the fact that the agent does not learn from previous game plays, the agent's decision making is linear and does not get better overtime.

REFERENCES

[1] Poole, David Lynton, Alan K. Mackworth, and Randy Goebel. *Computational intelligence: a logical approach*. Vol. 1. New York: Oxford University Press, 1998.

[2] Lin, Y. *Computer Science Game Trees*. [online] Ocf.berkeley.edu. Available at: https://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html, 2019

[3] Herbert Simon. *Neural Networks for Perception: Computation, Learning, and Architectures*. Retrieved March 1, 2019 from https://www.tutorialride.com/artificial-intelligence/learning-and-expert-system-in-ai.htm