Project Report
on
POLARIZED LADDER GAME

Submitted to
Dr. Nora Houari (Ph.D)
Concordia University
For COMP 472
April 12, 2016

By

Jordan Orsini
26471196
&
Zachary Bergeron
26593925

Jordan Orsini – 26471196
Zachary Bergeron – 26593925

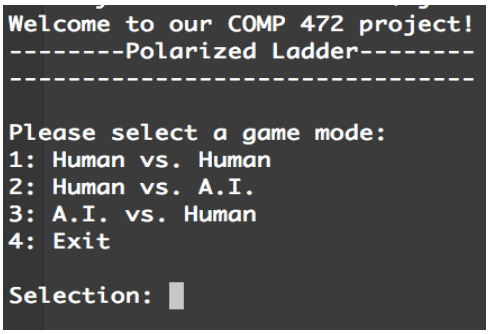
COMP 472 PROJECT
FINAL DELIVERABLE

12/04/2016

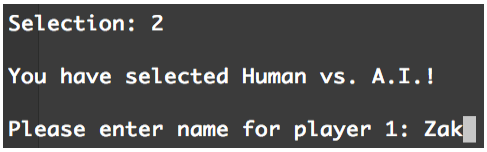
Polarized Ladder Game

Game play:

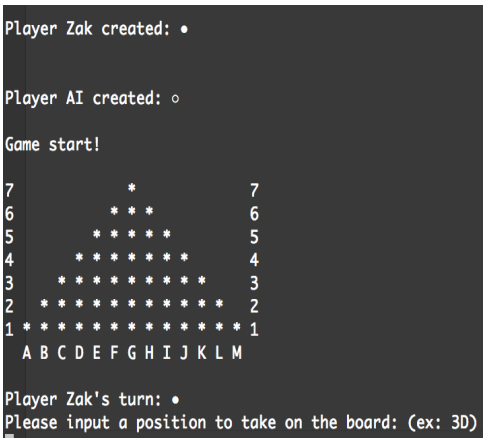
- This polarized ladder game was written purely in java using Java 8. The game is played using the console either through running the main method in the driver class or running the JAR file through the command prompt.



- The game starts by welcoming the player and asking them the type of game they would like to play providing three (3) choices.

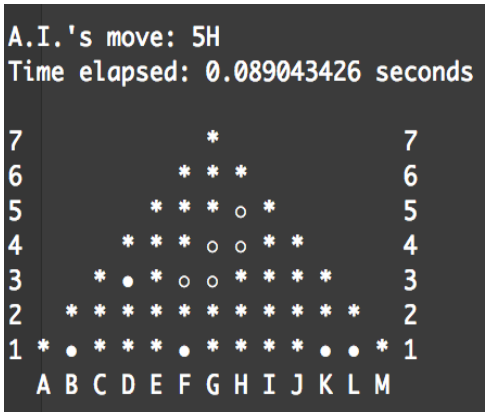


- After selecting a choice, the player is prompted to type their name into the console. If the player is playing another player the system will prompt for the second player's name, otherwise the other player will be named "AI".



- If the player chooses the first choice, he will start a player versus player game where each player takes their turn placing a token on the board attempting to build a ladder and stopping the other player from building his own.

- If the player chooses the second or third choice the player will be playing against the AI, the AI will attempt to build up its ladder while blocking the human player from building up his.



- The game will end once one of the players or AI successfully creates a polarized ladder successfully within the constraints of the game and that player will win. An alternative is if the entire game grid is entirely taken up without a single player able to construct a ladder the game ends in a tie.

How the game works:

- When the game starts it stores all the available possible points a player can choose in an array. Every time a player enters a point it uses the convertInput[1] method to convert the users alphanumerical input into a 2-dimensional array coordinate which are checked against that array to see if that point is available. If it is, the point it taken from the array and added to the users array of personal points.

- After placing a point, the game scans a list of 10 patterns that would form a ladder and uses the last placed point as its point of origin. This uses the checkForLadder[2] method (see below) to detect if the player who placed a point has a completed ladder.

- If a player has a completed ladder it must test the two conditions 1-polarized and 2-blocked:
    - 1-Polarized: this checks to see if 2 of the points in the completed ladder are touching the base using the polarizedAtBase[3] method, if they are it is an automatic win, however if they aren't then
    - 2-Blocked: this checks to see if there is a point from the opposite player on either side of the ladder that would neutralize the ladder and ensure that the move is not a win using the checkBlocked[4] method.
    - The previous method also ensures the condition that if three (3) points of the ladder are touching the sides of the game grid it is an automatic win; it ensures this because the opposite player would never be able to own a point outside of the scope of the game due to the array list of positions only containing legal points.

- If playing in a Human versus Human game, the players each take their turn selecting a point while the system checks the above conditions to test for a win.

- If playing versus an AI player however the system is more complex; whenever it is the AI's turn to play it must be able to select the appropriate point where it can block the human player from winning or building up a ladder in order to win. In order to do so it uses the minimax[5] method to determine which point has the highest priority and chooses it.

- Once a human player or an AI player has placed his tokens in a legal ladder formation and that the above mentioned conditions have all been met the user is declared the winner of that game. If no ladder is ever formed and all the points on the game grid are taken up then the game results in a tie.

Main methods:

[1] convertInput:

- The convertInput method is a simple method taking in an alphanumerical input by the user such as 3D (case sensitive) and converting it into a 2-dimensional array point. The numerical value is reversed as the standard 2-dimensional array goes up to down but the game grid goes down to up. Therefore, an input of 6 is actually a value on 1 in the backend.
- The alphabetical value is converted into its equivalent number in the alphabet such as B equaling 2.
- ex) 3E is equal to [5][4] because in the game we are asking for the numerical (vertical) number first and then the alphabetical (horizontal) value second. A 2-dimensional array works [horizontal][vertical].

[2] checkForLadder:

- The checkForLadder function does exactly what it's name implies in that it checks to see if a ladder is formed. It does this after every input, whereby the game scans a 2-dimenssional array. Each first dimension of the array contains a series of five (5) points that would form a ladder (see below). The last placed point either by a player or AI is set as the point of origin (0,0) in the array.
- Each point is checked against the player's owned physical points, and for every owned physical point the player gets a numerical point.
- For example, if the player has four (4) of the five (5) physical points necessary to complete a ladder he is scored 4 numerical points. The system continues verifying the entire 2-dimensional array to verify if the player has more points with any other pattern at which point the highest amount of points is sent back to be used by the minimax[5] later.
- Once a player has completely matched a pattern and is awarded five (5) points, the game proceeds to check the polarizedAtBase[3] method and the checkBlocked[4] method to verify the polarity of the ladder and determine if it is a win.

```
Point patterns[][] =
{
    //(0,0) as first point in right and left ladder respectfully
    {new Point(0,0), new Point(1,0), new Point (1,1), new Point (2,1) , new Point (2,2)}, //0
    {new Point(0,0), new Point(-1,0), new Point (-1,1), new Point (-2,1) , new Point (-2,2)}, //1

    //(0,0) as second point in right and left ladder respectfully
    {new Point(-1,0), new Point(0,0), new Point (0,1), new Point (1,1) , new Point (1,2)}, //2
    {new Point(1,0), new Point(0,0), new Point (0,1), new Point (-1,1) , new Point (-1,2)}, //3

    //(0,0) as middle point in right and left ladder respectfully
    {new Point (-1,-1), new Point (0,-1), new Point(0,0), new Point(1,0), new Point (1,1)}, //4
    {new Point (1,-1), new Point (0,-1), new Point(0,0), new Point(-1,0), new Point (-1,1)}, //5

    //(0,0) as fourth point in right and left ladder respectfully
    {new Point (-2,-1), new Point (-1,-1), new Point (-1,0), new Point(0,0), new Point(0,1)}, //6
    {new Point (2,-1), new Point (1,-1), new Point (1,0), new Point(0,0), new Point(0,1)}, //7

    //(0,0) as fifth point in right and left ladder respectfully
    {new Point (-2,-2), new Point (-1,-2), new Point (-1,-1), new Point(0,-1), new Point(0,0)},
    {new Point (2,-2),  new Point (1,-2), new Point (1,-1), new Point(0,-1), new Point(0,0)}, //9

    //RIGHT LADDER PATTERN INDEXES (0, 2, 4, 6, 8)
    //LEFT LADDER PATTERN INDEXES (1, 3, 5, 7, 9)

};
```

- An example of a conversion by point of origin, if the last point selected was (3,4), is as follows:
    - {(0,0), (1,0), (1,1), (2,1), (2,2)} = {(3,4), (4,4), (4,5), (5,5), (5,6)}

[3] polarizedAtBase:

- The polarizedAtBase is checked after the checkForLadder[2] method successfully identifies a five (5) numerical point ladder that has matched the win pattern. This win pattern is then stored in an array where it is checked against an array of points that constitute the base of the triangular game grid.
- If any two points in the winning array consist of points in the base array, then the player has won as his ladder is polarized and whether or not the opposing player has neutralized it is of no relevance.
- However if there is not two points in the winning array that are part of the base array then the game proceeds to check if the player's ladder has been neutralized through the checkBlocked[4] method.

[4] checkBlocked:

- The checkBlocked method follows the checkLadder[2] method and proceeds only if the polarizedAtBase[3] method has not returned a true value indicating the ladder has been polarized at the base.
- The checkBlocked methods takes the winning array and checks in which direction is is going; bottom-left to top-right or bottom-right to top-left.
- If the ladder is going from bottom-left to top-right, then the method checks to see if the ladder is blocked using the center point and adding a 1 x and y coordinates to verify if its blocked on the right as well as subtracting 1 from the x and y coordinates to verify if it blocked from the left.
- If the ladder is going from bottom-right to top-left, then the method checks to see if the ladder is blocked using the center point and adding a 1 to the x-coordinate and subtracting 1 from the y-coordinate to verify if its blocked on the right as well as subtracting 1 from the x-coordinate and adding 1 to the y-coordinate to verify if it blocked from the left.
- If the ladder has been successfully blocked on the left and right it has been neutralized and the ladder is not a win and the game continues.
- If the ladder has not been successfully blocked the the final condition has been met and the player with the ladder has won.
- Please note that as previously mentioned, this method also verifies if the ladder is polarized by three (3) points on the sides of the triangular game grid as it will not be able to successfully verify if is is blocked on that same side as the point will be out of scope of the game grid.

[5] minimax + heuristic:

- The minimax method is located in PlayGame.java and takes the current player number and depth as inputs.
- Whenever we call the minimax we always call it using depth 0 and the current player. Each time the method recurs we alternate the player number and increase the depth by 1.

- We have set a max depth of 3 in order to abide to the 3-second compute time maximum as outlined in the project description. With max depth set at 3 we get an output in less than 1 second from our minimax method. If set to a max depth of 4, computations to calculate a move for the A.I. player take approximately 19 seconds, placing it well outside of the time restriction.
- The method works by iterating through all available moves for the current player at the current depth and accumulating score according to our heuristic. Once all available moves for the current player have been looked at, we alternate the player, increase the depth by one and recur in order for the method to check and modify the score depending on the opposite player. This process continues until we reach the max depth or one of 2 stopping conditions, causing the method to return prematurely. When the method returns, we always reset the point it was testing at the current depth.
- After each move is tested a score is given depending on how many pieces of the ladder it will have completed subsequently with this move. Ex: If a potential move results in a 1-piece ladder we award it a score of 2 or -2 depending on if the player is maximizing or minimizing. A 2-piece ladder is awarded score of 4 or -4. A 3-piece ladder is awarded a score of 16 or -16. A 4-piece ladder is awarded a score of 256 or -256, and finally a 5-piece ladder or goal state ladder is awarded a score of 65536 or -65536. Each subsequent piece of a ladder is worth the previous ladder's score squared. This ensures that a move resulting in a higher score is given exponentially more points than a move resulting in a lesser score.
- Besides returning and checking other positions when max depth is reached we have two immediate stopping conditions at depth 0. 1) If the A.I. player can win this turn (depth 0), stop and return immediately, the position found is given the highest score possible. The A.I. will take this move and win the game. 2) If the opposing player can win immediately (depth 0), stop and return immediately. This position is given the highest score and the A.I. will take this move in order to ensure that the opponent player does not win on his/her next turn. Priority is given to the A.I. player winning over blocking an opponent's ladder. This means that if both the A.I. player and opponent player can win on their next turn, the A.I. will choose to win the game over blocking its opponent. When one of these conditions are met the A.I. selects its move without ever leaving depth 0 resulting in a significantly lower compute time than normal.
- Once minimax has completed we call the getComputerMove method which iterates through the scores of each move computed by the minimax and returns either the highest score if A.I. was max or the lowest score if the A.I. was min and the coordinates of the point associated with it. This move is then input as the A.I. player's move.

Tournament results:

- No time was allotted for the tournament, only for demos.

We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality – See attached documents with signatures