Queen's UNIVERSITY

Group 1: Reversi Best Moves

Anes Numanovic 21an88

Jacob Cruz 21JTC9

Adam Neto 21amn13

Rena Hajjar 21rsh8

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

November 3, 2023

Abstract

Our project revolves around finding the "best move(s)" in the board game Reversi. The game Reversi consists of an 8x8 grid with a 2x2 grid in the centre of the board consisting of 2 white pieces diagonal from each other, as well as 2 black pieces diagonal from each other. Each player takes a turn placing one of their pieces on the board in an attempt to "sandwich" the opponent's pieces. The term "sandwich" refers to when a player's pieces on the board have the opponent's pieces on both ends, causing the player's pieces to turn to the opponent's colour.

Propositions

mine_here(x,y) - True if the player's piece is on the position [x][y]
opp_here(x,y) - True if an opposing player's piece is on the position [x][y]
empty(x,y) - True if there is no piece on the position [x][y], the position is empty
sandwich(x,y) - True if placing your own piece on the position [x][y] will create a sandwich
row_sand(x,y) - True if placing your own piece on the position [x][y] will create a horizontal sandwich
column_sand(x,y) - True if placing your own piece on the position [x][y] will create a vertical sandwich
diagonal_sand(x,y) - True if placing your own piece on the position [x][y] will create a diagonal sandwich

Constraints

$\forall y.((mine\_here(1,y) \vee opp\_here(1,y)) \wedge mine\_here(2,y) \vee opp\_here(2,y))... \wedge mine\_here(8,y) \vee opp\_here(8,y)))) \Rightarrow F$
If every space in a row is filled with either a black or white piece, this implies that the board is full and no more pieces can be placed

$\forall x.\forall y.(\neg opp\_here(x,y) \wedge \neg mine\_here(x,y)) \Rightarrow \forall x.\forall y.empty(x,y)$
If at a specific position (x,y) there is neither the player's piece nor an opponent's piece, this implies the square is empty

$\forall y.(opp\_here(2,y) \wedge mine\_here(5,y) \wedge (\neg(empty(3,y) \vee empty(4,y) \vee mine\_here(3,y) \vee mine\_here(4,y)))) \Rightarrow \forall y.row\_sandwich$
If there are two of your own pieces that can surround one or more opponent pieces horizontally, this implies that you can have a sandwich in that row.

$\forall x.(opp\_here(x,2) \wedge mine\_here(x,5) \wedge (\neg(empty(x,3) \vee empty(x,4) \vee mine\_here(x,3) \vee mine\_here(x,4)))) \Rightarrow \forall x.column\_sandwich$
If there are two of your own pieces that can surround one or more opponent pieces vertically, this implies that you can have a sandwich in that column

$(opp\_here(2,2) \wedge mine\_here(5,5) \wedge (\neg(empty(3,3) \vee empty(4,4) \vee mine\_here(3,3) \vee mine\_here(4,4)))) \Rightarrow diagonal\_sandwich$
If there are two of your own pieces that can surround one or more opponent pieces diagonally, this implies that you can have a sandwich on that diagonal.

$\forall x.\forall y.(empty(x,y) \wedge (row\_sand(x,y) \vee column\_sand(x,y) \vee diagonal\_sand(x,y))) \Rightarrow \forall x.\forall y.sandwich(x,y)$

If a square(x,y) is empty, and there is either a diagonal sandwich, column sandwich, or a row sandwich, this implies that the square is a sandwich and playable square.

Model Exploration

Our model uses the following constraints to apply logic to our program:

The first constraint will check if all spots on the grid are taken with either a black or white piece (Our constraint example from above is only for one row in the grid).

The second constraint checks if the spot at position (x, y) has neither a black or white piece, if the spot has neither it implies the spot is empty.

The third constraint checks if the spot at (x, y) will sandwich the row, our example from above is for if the player places their piece at position $(1, y)$ will sandwich the row extending to position $(5, y)$

The fourth constraint checks if the spot at (x, y) will sandwich the column, our example from above is for if the player places their piece at position $(x, 1)$ will sandwich the row extending to position $(y, 5)$

The fifth constraint checks if the spot at (x, y) will sandwich the diagonal, our example from above is for if the player places their piece at position $(1, 1)$ will sandwich the row extending to position $(5, 5)$

Finally, the sixth constraint checks if the spot at position (x, y) will sandwich a line of the opponent's pieces. Our example from above checks if the spot at (x, y) is empty and also checks if one or more of the row, column, or diagonal constraints evaluate to true

First-Order Extension

Describe how you might extend your model to a predicate logic setting, including how both the propositions and constraints would be updated. There is no need to implement this extension!

For the reversi move finder, predicate logic would be implemented by having propositions for each square's possible state (empty, opponent piece, or players piece), as well as propositions for whether or not a piece can be placed in any given position. The propositions and constraints would need to be updated after every turn is simulated, as the board would likely be significantly changed after each move. To find the next move the program would check how many points would be given for each possible move on a given turn and decide based on the highest number what the best move would be. A challenge with finding the truly best move would be having the need to look into the future for what would be the best move in the long run, but we will not be implementing that in this project as that would be much too difficult and unreasonable.

Useful Notation

Feel free to copy/paste the symbols here and remove this section before submitting.

$$\wedge \quad \vee \quad \neg \quad \rightarrow \quad \forall \quad \exists$$