Group 1: Reversi Best Moves

Anes Numanovic 21an88

Jacob Cruz 21JTC9

Adam Neto 21amn13

Rena Hajjar 21rsh8

Course Modelling Project

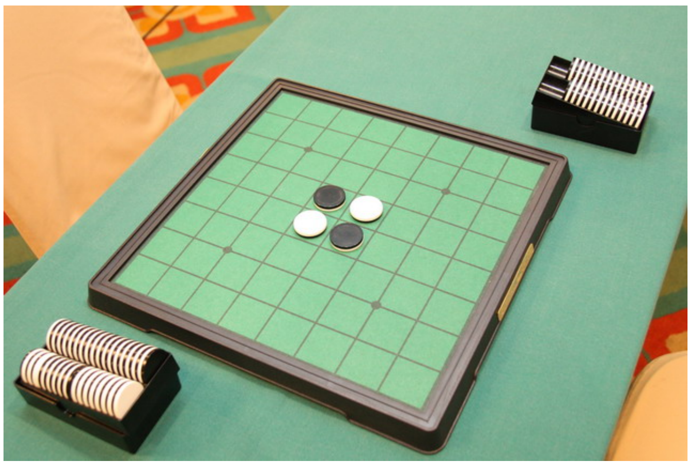
CISC/CMPE 204

Logic for Computing Science

December 8, 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

- $\forall y.((mine\_here(1,y) \lor opp\_here(1,y)) \land (mine\_here(2,y) \lor opp\_here(2,y))... \land (mine\_here(8,y) \lor opp\_here(8,y))) \Rightarrow F$
  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

- $\forall x.\forall y.(\neg opp\_here(x,y) \land \neg mine\_here(x,y)) \Leftrightarrow \forall x.\forall y.empty(x,y)$

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 and vise versa

- $\forall y.(\exists x.opp\_here(x,y) \wedge \exists x.mine\_here(x,y) \wedge \forall x.(\neg(empty(x,y) \vee mine\_here(x,y)))) \Rightarrow \exists x.\forall y.row\_sandwich(x,y)$
  This constraint shows that if there is a row on our board such that, there is a row of opponent's piece with an empty spot on one end and the player's piece on the other end, that row is able to be 'sandwiched'.

- $\forall x.(\exists y.opp\_here(x,y) \wedge \exists y.mine\_here(x,y) \wedge \forall y.(\neg(empty(x,y) \vee mine\_here(x,y)))) \Rightarrow \forall x.\exists y.column\_sandwich(x,y)$
  This constraint shows that if there is a column on our board such that, there is a column of opponent's piece with an empty spot on one end and the player's piece on the other end, that column is able to be 'sandwiched'.

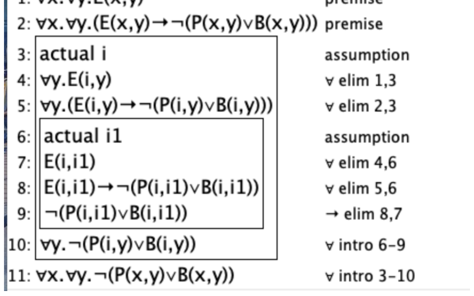
- $\exists x.\exists y.(opp\_here(x,y) \wedge mine\_here(x,y)) \wedge \forall x.\forall y.(\neg(empty(x,y) \vee mine\_here(x,y))) \Rightarrow \exists x.\exists y.diagonal\_sandwich(x,y)$
  This constraint shows that if there is a diagonal on our board such that, there is a diagonal row of opponent's piece with an empty spot on one end and the player's piece on the other end, that diagonal is able to be 'sandwiched'.

- $\forall x.\forall y.(empty(x,y) \wedge (row\_sand(x,y) \vee column\_sand(x,y) \vee diagonal\_sand(x,y))) \Rightarrow \forall x.\forall y.sandwich(x,y)$
  If a spot at (x,y) is empty, and there is either a diagonal sandwich, column sandwich, and/or a row sandwich, that implies that the spot at (x,y) can be 'sandwiched' and is a playable spot.

Jape Proofs

For the Jape Proofs, some name changes for some propositions as we weren't able to use their starting letter within jape. Those changes are that P will represent mine_here, B will represent opp_here, and (in this document) $\mapsto$ will represent semantic implication. Our jape sequences are,

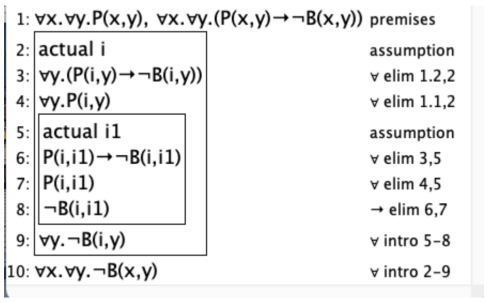
1. Showing that if there is a white piece at some x,y coordinate, that implies that there isn't a black at that coordinate.
$\forall x.\forall y.P(x,y), \forall x.\forall y.(P(x,y) \rightarrow (x,y)) \mapsto \forall x.\forall y.\neg B(x,y)$



```
1: ∀x.∀y.E(x,y)                           premise
2: ∀x.∀y.(E(x,y)→¬(P(x,y)∨B(x,y)))        premise
3: actual i                               assumption
4: ∀y.E(i,y)                              ∀ elim 1,3
5: ∀y.(E(i,y)→¬(P(i,y)∨B(i,y)))           ∀ elim 2,3
6: actual i1                              assumption
7: E(i,i1)                                ∀ elim 4,6
8: E(i,i1)→¬(P(i,i1)∨B(i,i1))             ∀ elim 5,6
9: ¬(P(i,i1)∨B(i,i1))                     → elim 8,7
10: ∀y.¬(P(i,y)∨B(i,y))                   ∀ intro 6-9
11: ∀x.∀y.¬(P(x,y)∨B(x,y))                ∀ intro 3-10
```

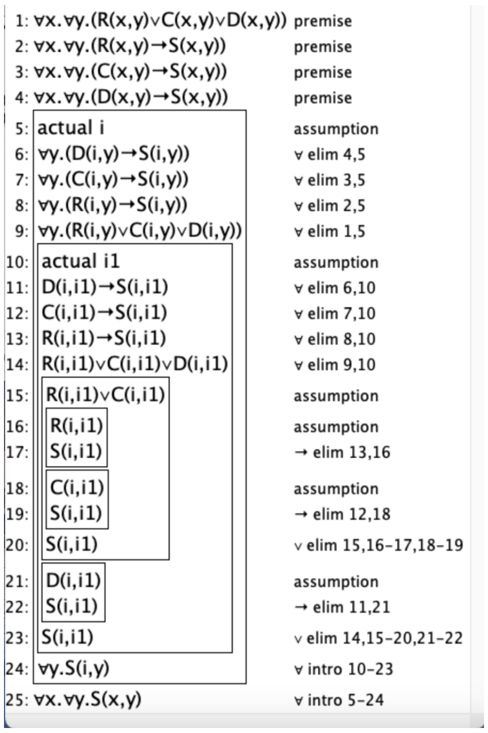
2. Showing that if there is an empty space at some x,y coordinate, that implies that there isn't a black or white piece at that coordinate.

$\forall x.\forall y.E(x,y), \forall x.\forall y.(E(x,y) \rightarrow \neg(P(x,y) \vee B(x,y))) \mapsto \forall x.\forall y.\neg(P(x,y) \vee B(x,y))$



```
 1: ∀x.∀y.P(x,y), ∀x.∀y.(P(x,y)→¬B(x,y))  premises
 2: | actual i                             assumption
 3: | ∀y.(P(i,y)→¬B(i,y))                 ∀ elim 1.2,2
 4: | ∀y.P(i,y)                            ∀ elim 1.1,2
 5: | | actual i1                          assumption
 6: | | P(i,i1)→¬B(i,i1)                   ∀ elim 3,5
 7: | | P(i,i1)                            ∀ elim 4,5
 8: | | ¬B(i,i1)                           → elim 6,7
 9: | ∀y.¬B(i,y)                           ∀ intro 5-8
10: ∀x.∀y.¬B(x,y)                          ∀ intro 2-9
```

3. Showing that if any of the sandwich propositions is true (row, column or diagonal) at a spot (x,y), then that spot will lead to a sandwich of the opponent's pieces.
$\forall x.\forall y.(R(x,y) \rightarrow S(x,y)), \forall x.\forall y.(C(x,y) \rightarrow S(x,y)), \forall x.\forall y.(D(x,y) \rightarrow S(x,y)),$
$\forall x.\forall y.(R(x,y) \vee C(x,y) \vee D(x,y)) \mapsto \forall x.\forall y.S(x,y)$



```
 1: ∀x.∀y.(R(x,y)∨C(x,y)∨D(x,y))   premise
 2: ∀x.∀y.(R(x,y)→S(x,y))          premise
 3: ∀x.∀y.(C(x,y)→S(x,y))          premise
 4: ∀x.∀y.(D(x,y)→S(x,y))          premise
 5: | actual i                      assumption
 6: | ∀y.(D(i,y)→S(i,y))            ∀ elim 4,5
 7: | ∀y.(C(i,y)→S(i,y))            ∀ elim 3,5
 8: | ∀y.(R(i,y)→S(i,y))            ∀ elim 2,5
 9: | ∀y.(R(i,y)∨C(i,y)∨D(i,y))     ∀ elim 1,5
10: | | actual i1                   assumption
11: | | D(i,i1)→S(i,i1)             ∀ elim 6,10
12: | | C(i,i1)→S(i,i1)             ∀ elim 7,10
13: | | R(i,i1)→S(i,i1)             ∀ elim 8,10
14: | | R(i,i1)∨C(i,i1)∨D(i,i1)     ∀ elim 9,10
15: | | | R(i,i1)∨C(i,i1)           assumption
16: | | | | R(i,i1)                 assumption
17: | | | | S(i,i1)                 → elim 13,16
18: | | | | C(i,i1)                 assumption
19: | | | | S(i,i1)                 → elim 12,18
20: | | | S(i,i1)                   ∨ elim 15,16-17,18-19
21: | | | D(i,i1)                   assumption
22: | | | S(i,i1)                   → elim 11,21
23: | | S(i,i1)                     ∨ elim 14,15-20,21-22
24: | ∀y.S(i,y)                     ∀ intro 10-23
25: ∀x.∀y.S(x,y)                    ∀ intro 5-24
```

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 $(5, y)$

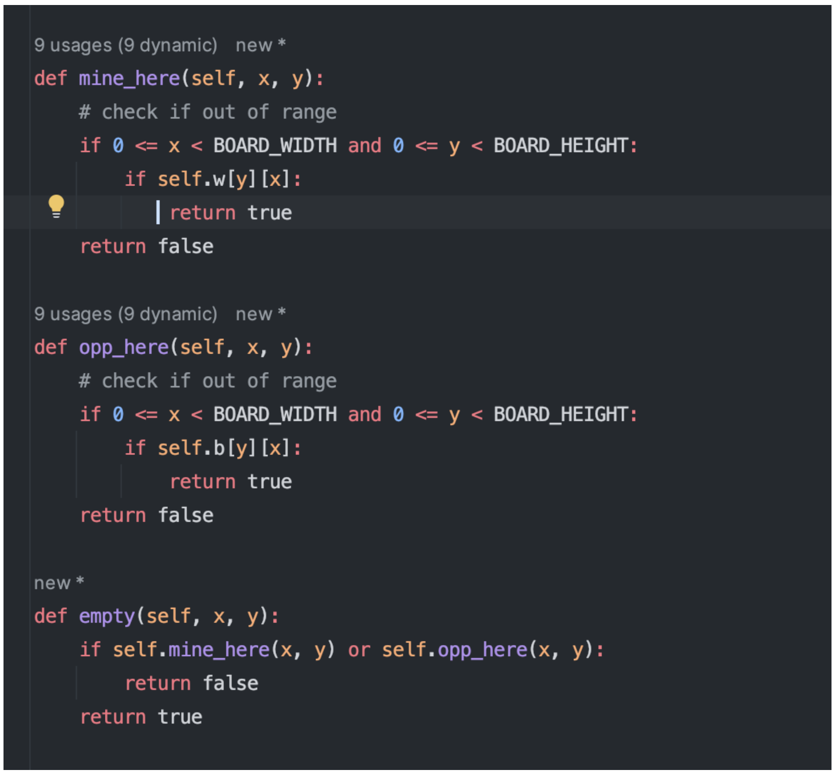
The fourth constraint checks if the spot at (x, y) will sandwich the column, our example from above is for if the player places their piece at position $(x, 1)$ will sandwich the row extending to position $(y, 5)$

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

In the initial stage of our exploration, we had a piece propositional class that had attributes of r (row playable), c (column playable), e (empty), w (white), b (black). Our original idea was to set each proposition for each piece as true or false. We quickly realized that we could implement the tools we've already learned in our programming journey instead of this, to incorporate the logic into our system. By not explicitly declaring these properties and having the propositions speak for themselves, we simplified our process as well as our Piece objects.

We also had each individual proposition as its own function, before seeing we could expand them out greater, leading to less function definitions and more understandable encoding. Shown below is our initial attempt to encode our first 3 constraints:



```
9 usages (9 dynamic)   new *
def mine_here(self, x, y):
    # check if out of range
    if 0 <= x < BOARD_WIDTH and 0 <= y < BOARD_HEIGHT:
        if self.w[y][x]:
            return true
    return false

9 usages (9 dynamic)   new *
def opp_here(self, x, y):
    # check if out of range
    if 0 <= x < BOARD_WIDTH and 0 <= y < BOARD_HEIGHT:
        if self.b[y][x]:
            return true
    return false

new *
def empty(self, x, y):
    if self.mine_here(x, y) or self.opp_here(x, y):
        return false
    return true
```

First-Order 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.

Domain of Discourse:
Objects for pieces (white, black, or neither)
Natural numbers from 0-7 (to model the board)

Predicates:
•$Playable(x, y) : ifaspotissandwichablebyrow, column, ordiagonal$
• $RowPlayable(x, y) : ifaspotissandwichablebyrow$
• $ColPlayable(x, y) : ifaspotisplayablebycolumn$
• $DiagPlayable(x, y) : ifaspotisplayablebydiagonal$