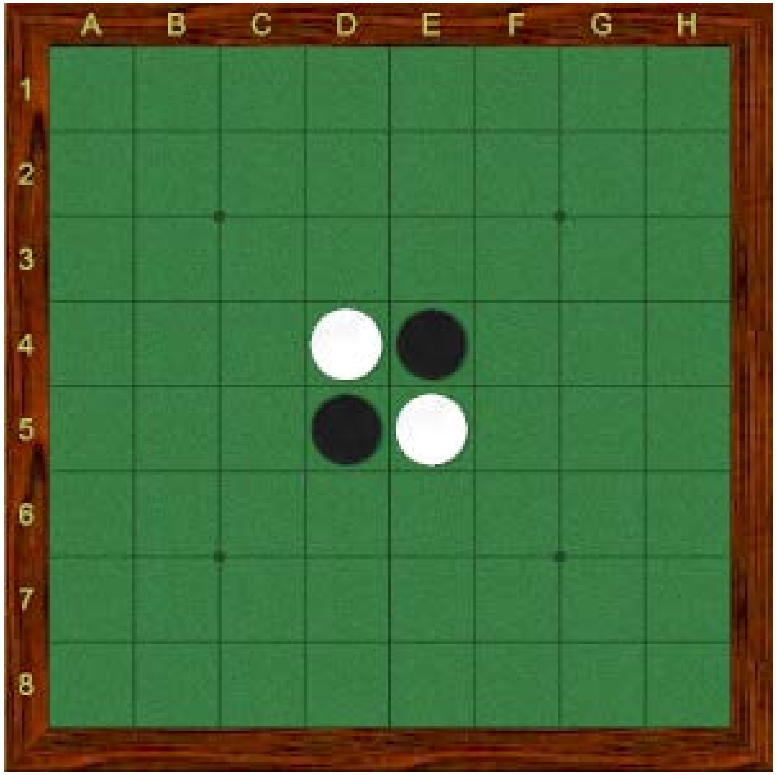
Tutorial 10 – Design Specifications

Week of 20-24, March, 2017

Prepared by: Gurankash Singh

Othello Game

- 8 x 8 board
- 2 Players – Black & White

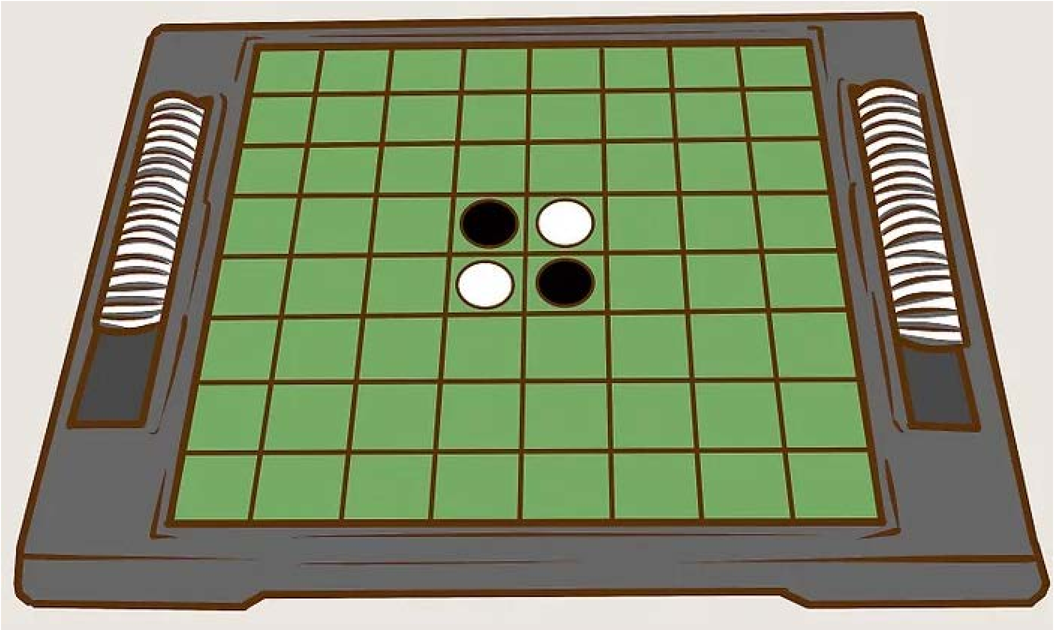


Objective

- To have the majority of your colour discs on the board at the end of the game
- Each player takes 32 discs and chooses one colour
- Pieces are double sided

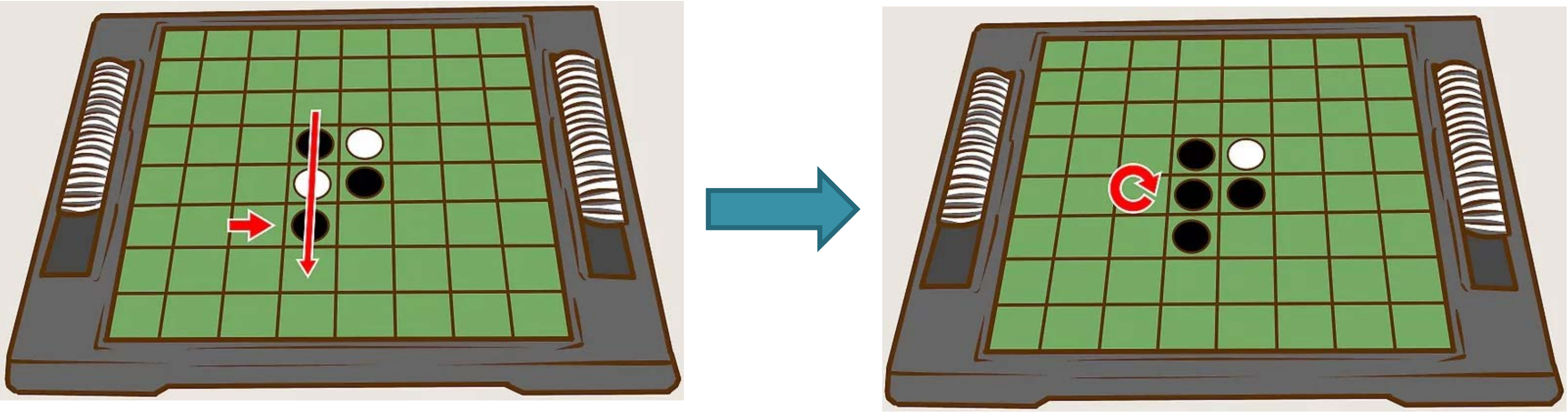
Starting board

- Place 4 discs in the center of the board; 2 black 2 white so that the discs with matching colors touch diagonally
- Black goes first



Scoring

- Outflank - to surround a row of your opponent's discs with two of your own discs
- Row - one or more discs that form a line horizontally, vertically or diagonally

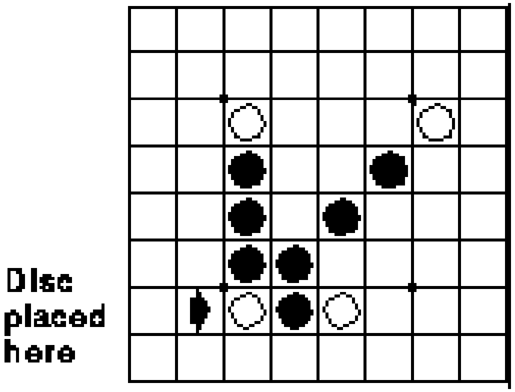


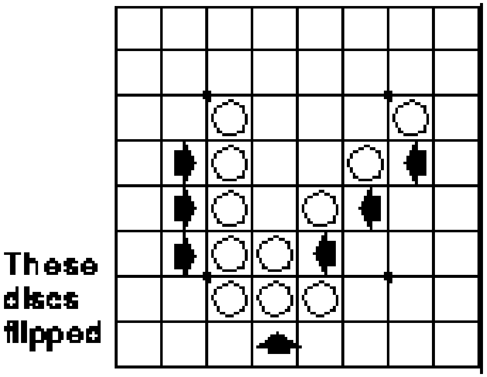
Rules

1. Black always moves first.
2. If on your turn you cannot outflank and flip at least one opposing disc, your turn is forfeited and your opponent moves again. However, if a move is available to you, you may not forfeit your turn.

Rules Continued

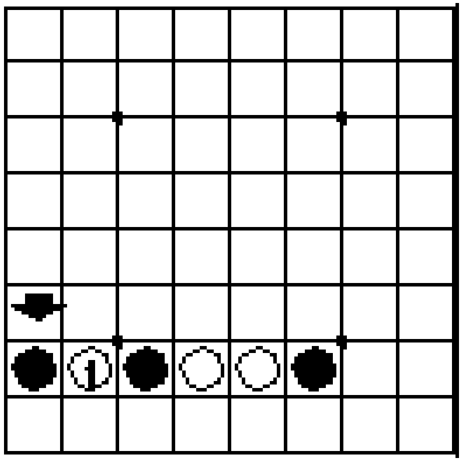
3. A disc may outflank any number of discs in one or more rows in any number of directions at the same time - horizontally, vertically or diagonally (A row is defined as one or more discs in a continuous straight line).





Rules Continued

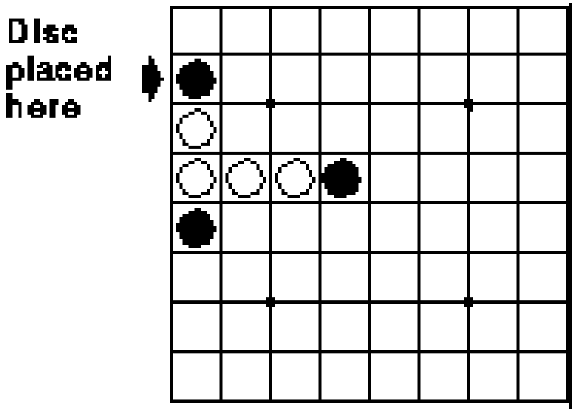
4. You may not skip over your own colour disc to outflank an opposing disc.

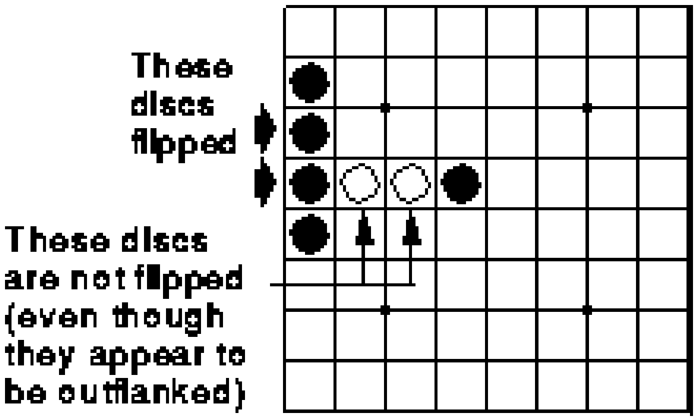


This disc only outflanks and flips White disc 1.

Rules Continued

5. Discs may only be outflanked as a direct result of a move and must fall in the direct line of the disc placed down.




These discs flipped
These discs are not flipped (even though they appear to be outflanked)

Rules Continued

6. All discs outflanked in any one move must be flipped.
7. If a player runs out of discs, but still has an opportunity to outflank an opposing disc on his or her turn, the opponent must give the player a disc to use.

Rules Continued

8. When it is no longer possible for either player to move, the game is over. Discs are counted and the player with the majority of his or her colour discs on the board is the winner. NOTE: It is possible for a game to end before all 64 squares are filled.

Othello Module

- Let's write a module that stores the state of the game board and the status of the game.
- You do not need to worry about modules that display graphics, or control the game play, or determine the strategies of a computer opponent, etc.

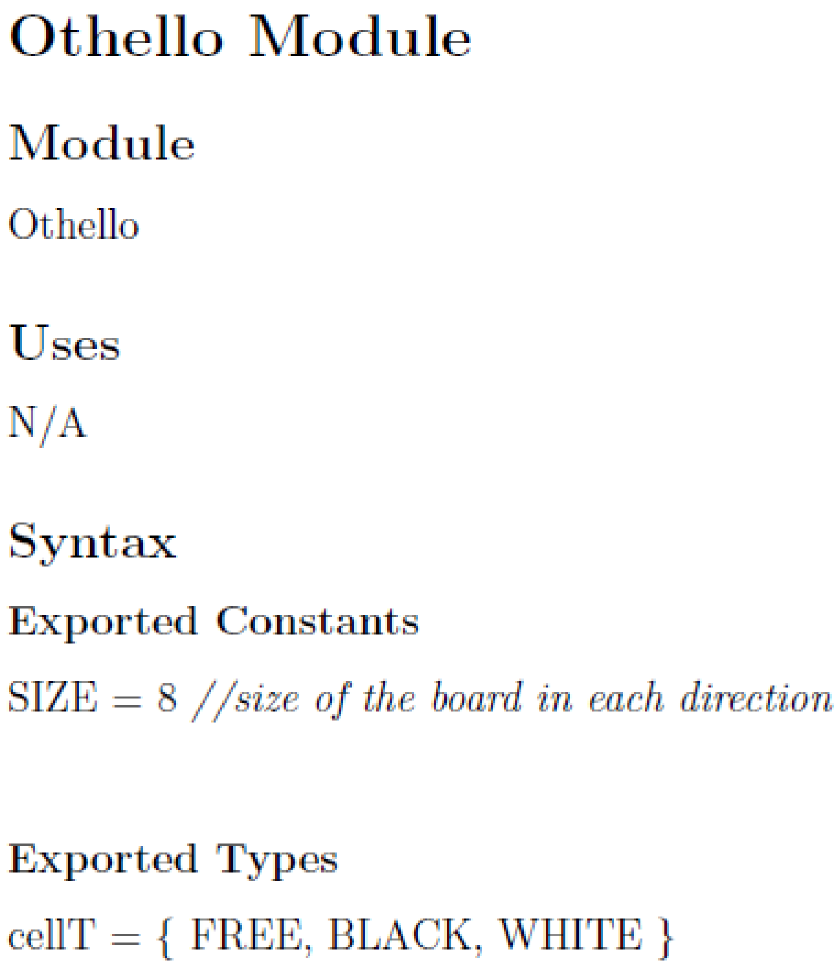
Things to consider

- The state of the game board could be modelled as a two dimensional sequence of {free, black, white}
- You will need a state variable that represents whose turn it is
- The game state module likely makes more sense as an abstract object than it does as an abstract data type
- You will need a routine to initialize the board

Things to consider continued

- You will need to be able to determine whether a move is valid or not
- You will need to be able to inspect the state of any cell of the game board
- You should be able to determine who is winning for any state of the game board
- You need to be able to tell when the game is over
- You will need to be able to determine whether there are any valid moves for a given player

Model, Uses, Syntax



Othello Module

Module

Othello

Uses

N/A

Syntax

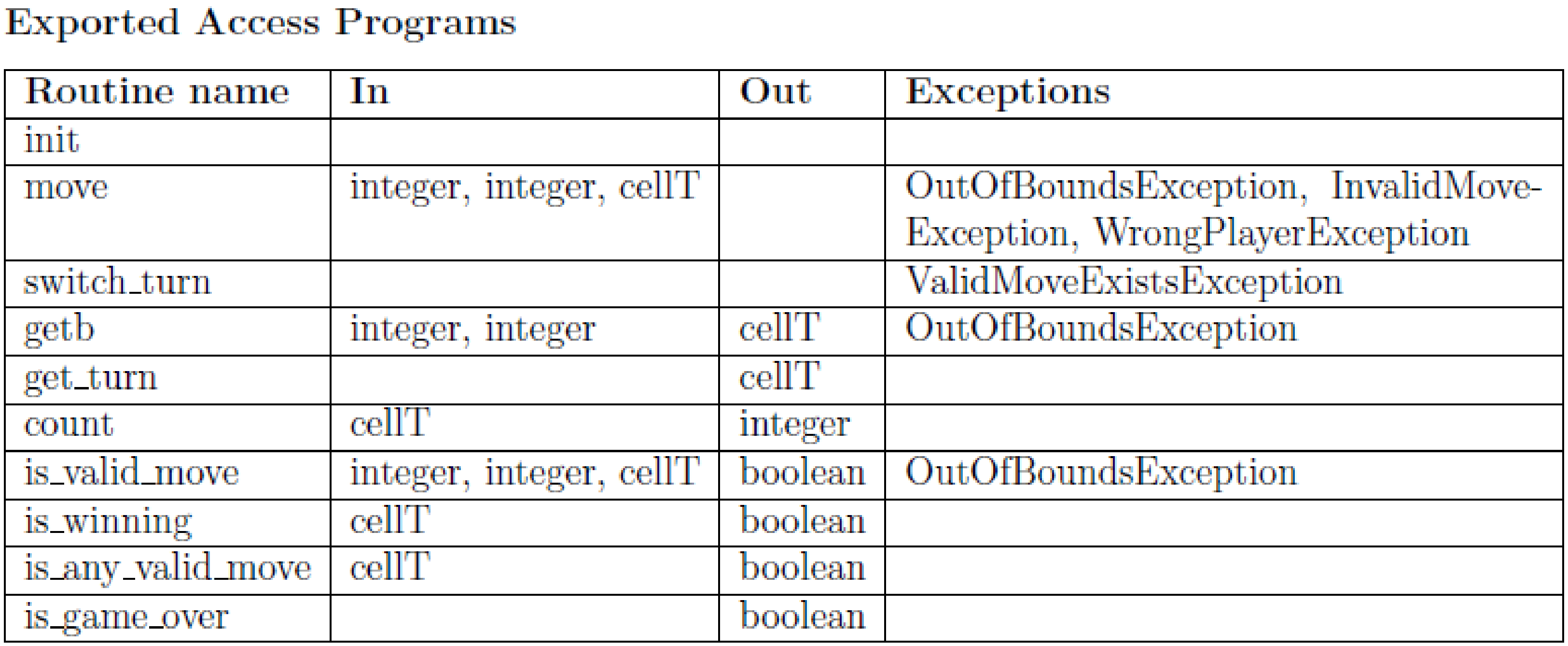
Exported Constants

SIZE = 8 *//size of the board in each direction*

Exported Types

cellT = { FREE, BLACK, WHITE }

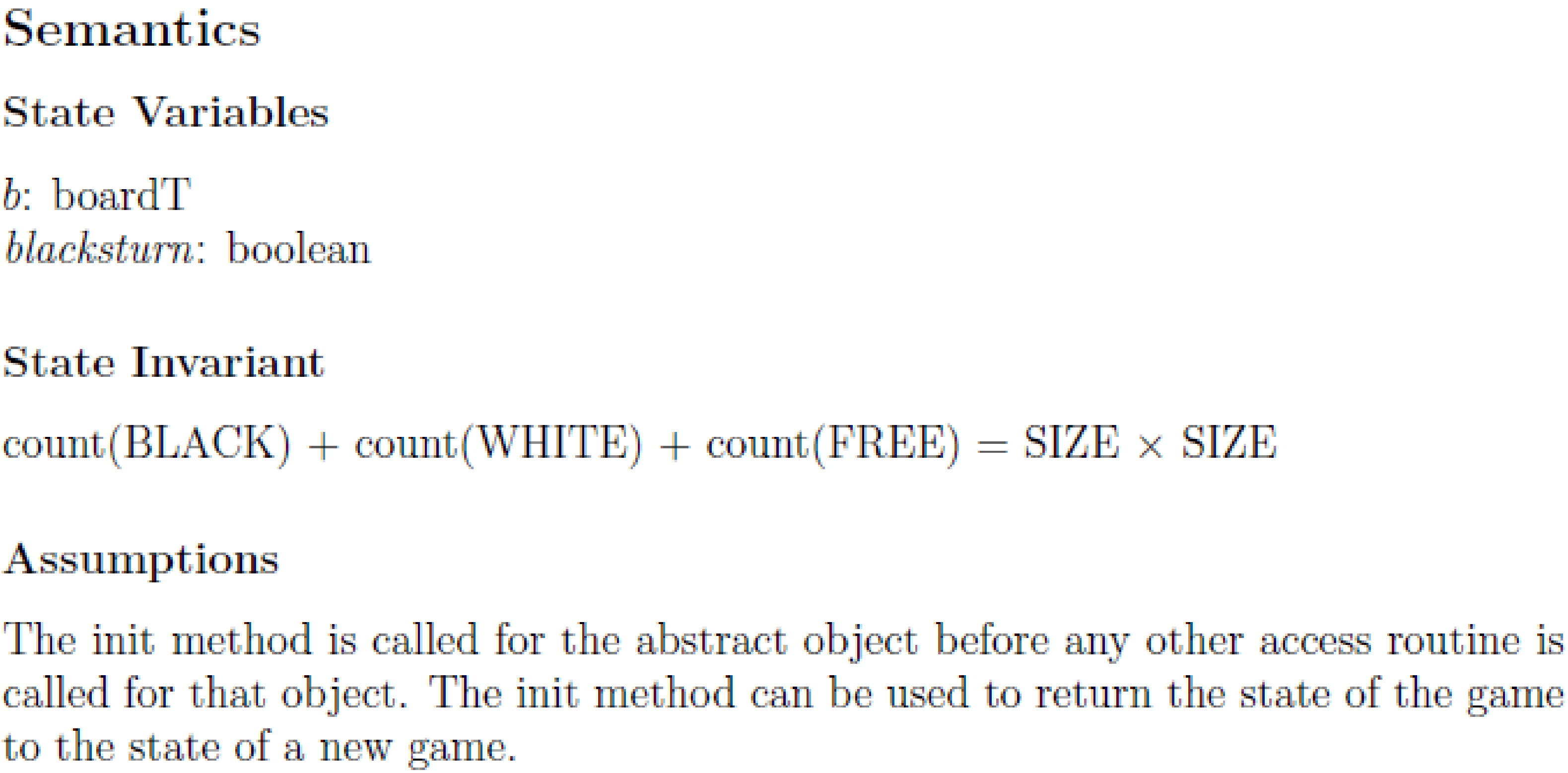
Access Programs



Exported Access Programs

Routine name	In	Out	Exceptions
init			
move	integer, integer, cellT		OutOfBoundsException, InvalidMoveException, WrongPlayerException
switch_turn			ValidMoveExistsException
getb	integer, integer	cellT	OutOfBoundsException
get_turn		cellT	
count	cellT	integer	
is_valid_move	integer, integer, cellT	boolean	OutOfBoundsException
is_winning	cellT	boolean	
is_any_valid_move	cellT	boolean	
is_game_over		boolean	

Semantics



Semantics

State Variables

b: boardT
blacksturn: boolean

State Invariant

$$\text{count}(\text{BLACK}) + \text{count}(\text{WHITE}) + \text{count}(\text{FREE}) = \text{SIZE} \times \text{SIZE}$$

Assumptions

The init method is called for the abstract object before any other access routine is called for that object. The init method can be used to return the state of the game to the state of a new game.

Access Routine Semantics

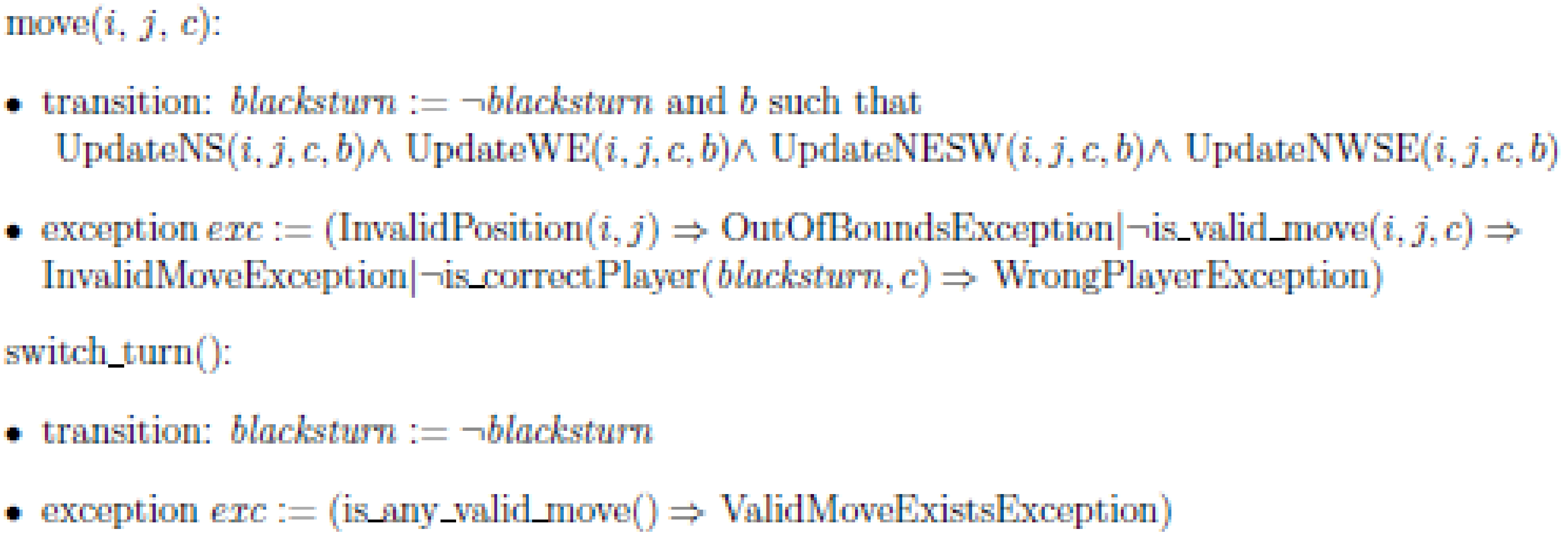
init():

- transition:

$$blacksturn, b := true, <\begin{array}{c} < \mathrm{FREE, FREE, FREE, FREE, FREE, FREE, FREE, FREE} > \\ < \mathrm{FREE, FREE, FREE, FREE, FREE, FREE, FREE, FREE} > \\ < \mathrm{FREE, FREE, FREE, FREE, FREE, FREE, FREE, FREE} > \\ < \mathrm{FREE, FREE, FREE, WHITE, BLACK, FREE, FREE, FREE} > \\ < \mathrm{FREE, FREE, FREE, BLACK, WHITE, FREE, FREE, FREE} > \\ < \mathrm{FREE, FREE, FREE, FREE, FREE, FREE, FREE, FREE} > \\ < \mathrm{FREE, FREE, FREE, FREE, FREE, FREE, FREE, FREE} > \\ < \mathrm{FREE, FREE, FREE, FREE, FREE, FREE, FREE, FREE} > \end{array} >$$

- exception none

Access Routine Semantics



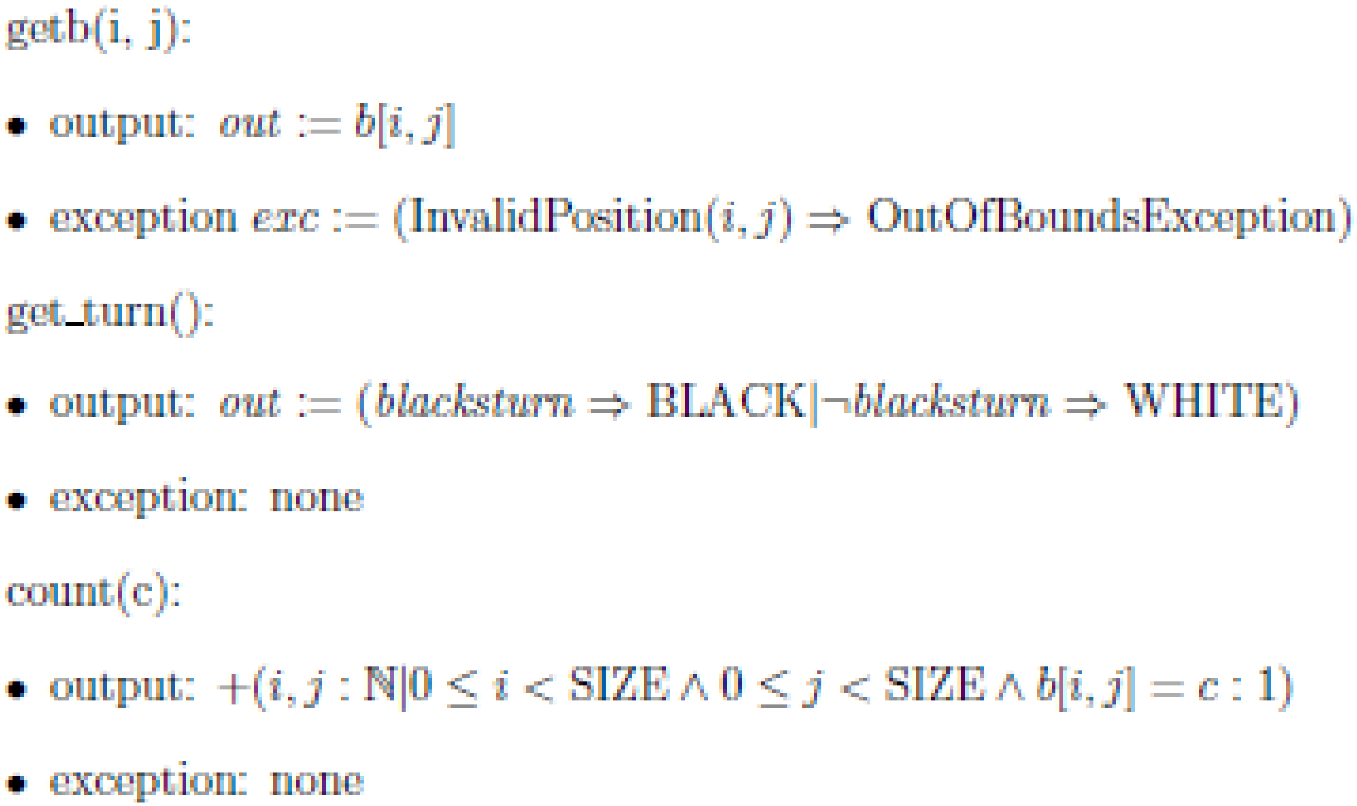
move(i, j, c):

- transition: $blacksturn := \neg blacksturn$ and b such that UpdateNS(i, j, c, b)$\wedge$ UpdateWE(i, j, c, b)$\wedge$ UpdateNESW(i, j, c, b)$\wedge$ UpdateNWSE(i, j, c, b)
- exception $exc :=$ (InvalidPosition(i, j) $\Rightarrow$ OutOfBoundsException$|\neg$is_valid_move(i, j, c) $\Rightarrow$ InvalidMoveException$|\neg$is_correctPlayer($blacksturn, c$) $\Rightarrow$ WrongPlayerException)

switch_turn():

- transition: $blacksturn := \neg blacksturn$
- exception $exc :=$ (is_any_valid_move() $\Rightarrow$ ValidMoveExistsException)

Access Routine Semantics



getb(i, j):

- output: $out := b[i, j]$
- exception $exc := (\text{InvalidPosition}(i, j) \Rightarrow \text{OutOfBoundsException})$

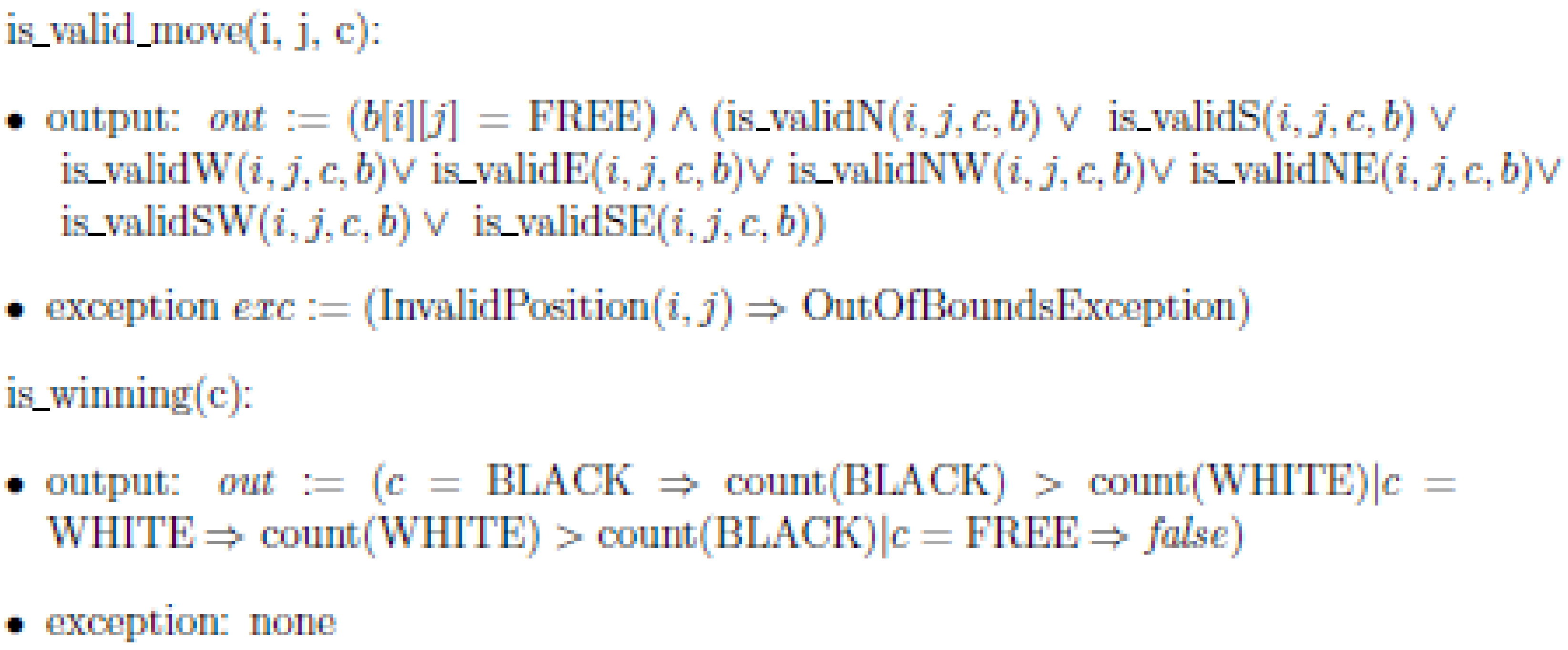
get_turn():

- output: $out := (blacksturn \Rightarrow \text{BLACK} | \neg blacksturn \Rightarrow \text{WHITE})$
- exception: none

count(c):

- output: $+(i, j : \mathbb{N} | 0 \leq i < \text{SIZE} \wedge 0 \leq j < \text{SIZE} \wedge b[i, j] = c : 1)$
- exception: none

Access Routine Semantics



is_valid_move(i, j, c):

- output: $out := (b[i][j] = \text{FREE}) \wedge (\text{is\_validN}(i,j,c,b) \vee \text{is\_validS}(i,j,c,b) \vee \text{is\_validW}(i,j,c,b) \vee \text{is\_validE}(i,j,c,b) \vee \text{is\_validNW}(i,j,c,b) \vee \text{is\_validNE}(i,j,c,b) \vee \text{is\_validSW}(i,j,c,b) \vee \text{is\_validSE}(i,j,c,b))$

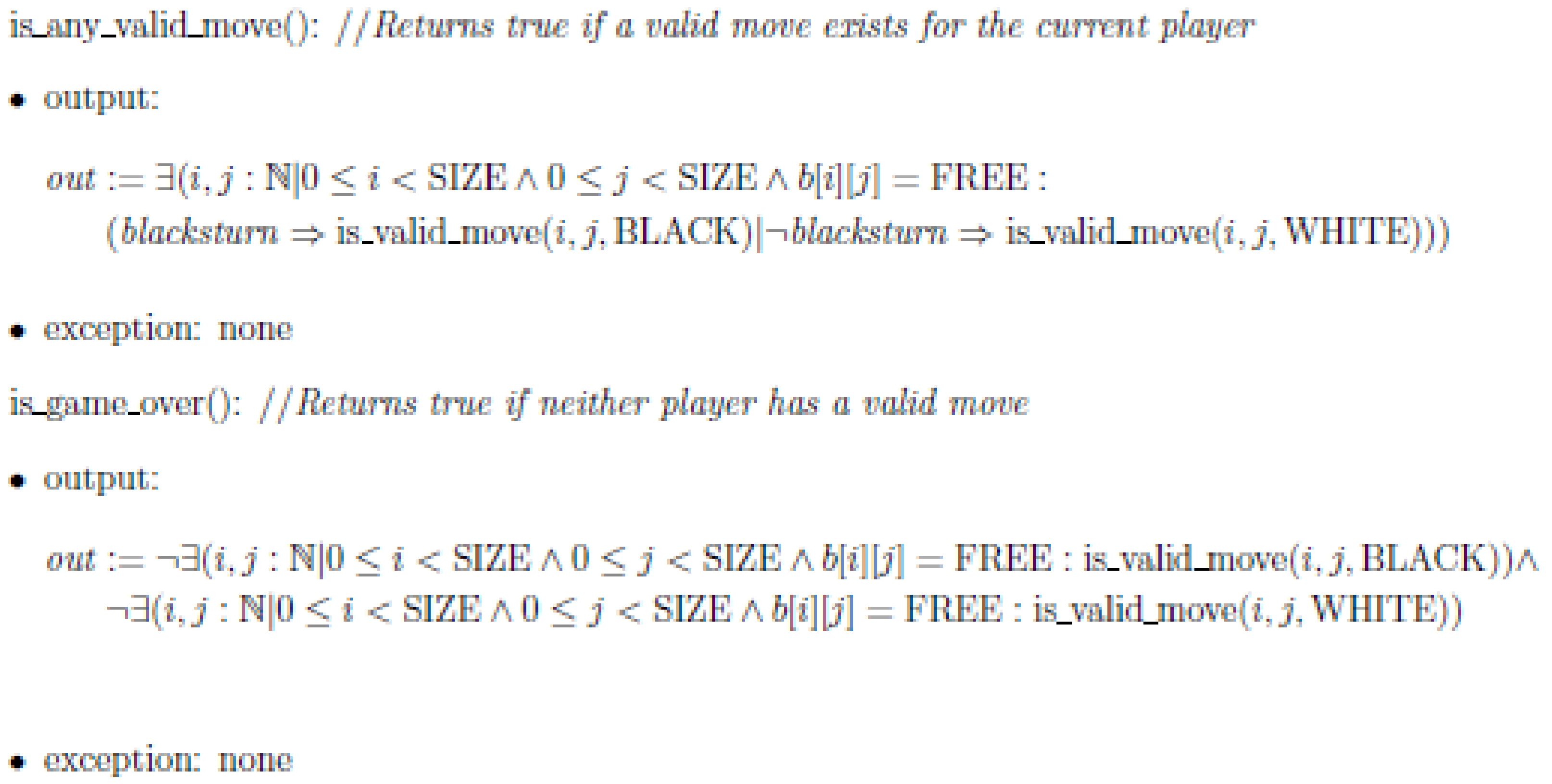
- exception $exc := (\text{InvalidPosition}(i,j) \Rightarrow \text{OutOfBoundsException})$

is_winning(c):

- output: $out := (c = \text{BLACK} \Rightarrow \text{count(BLACK)} > \text{count(WHITE)} | c = \text{WHITE} \Rightarrow \text{count(WHITE)} > \text{count(BLACK)} | c = \text{FREE} \Rightarrow \mathit{false})$

- exception: none

Access Routine Semantics



is_any_valid_move(): *//Returns true if a valid move exists for the current player*

- output:

$$\mathit{out} := \exists(i, j : \mathbb{N} | 0 \leq i < \text{SIZE} \wedge 0 \leq j < \text{SIZE} \wedge b[i][j] = \text{FREE} :\\ (\mathit{blacksturn} \Rightarrow \text{is\_valid\_move}(i, j, \text{BLACK}) | \neg \mathit{blacksturn} \Rightarrow \text{is\_valid\_move}(i, j, \text{WHITE})))$$

- exception: none

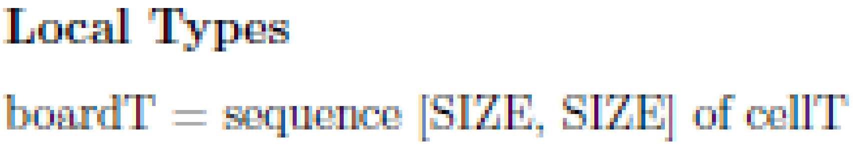
is_game_over(): *//Returns true if neither player has a valid move*

- output:

$$\mathit{out} := \neg\exists(i, j : \mathbb{N} | 0 \leq i < \text{SIZE} \wedge 0 \leq j < \text{SIZE} \wedge b[i][j] = \text{FREE} : \text{is\_valid\_move}(i, j, \text{BLACK})) \wedge\\ \neg\exists(i, j : \mathbb{N} | 0 \leq i < \text{SIZE} \wedge 0 \leq j < \text{SIZE} \wedge b[i][j] = \text{FREE} : \text{is\_valid\_move}(i, j, \text{WHITE}))$$

- exception: none

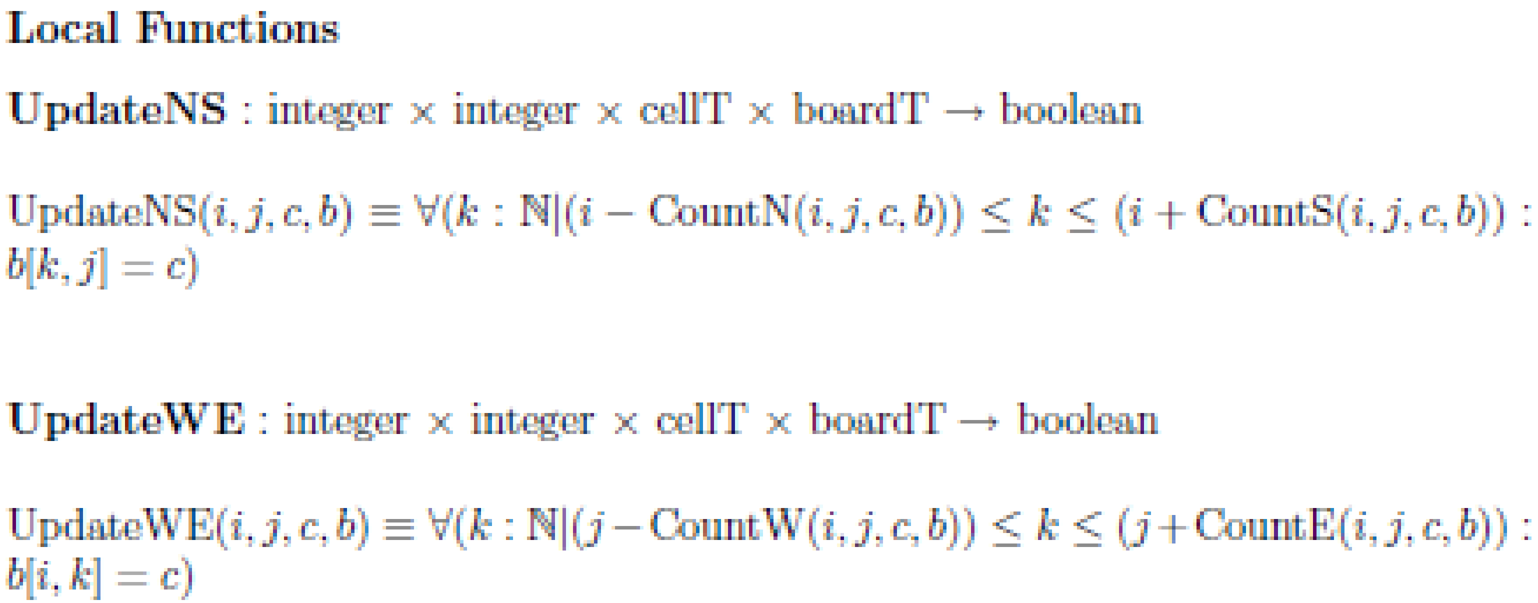
Local Types



Local Types

boardT = sequence [SIZE, SIZE] of cellT

Local Functions



Local Functions

UpdateNS : integer × integer × cellT × boardT → boolean

UpdateNS$(i, j, c, b) \equiv \forall(k : \mathbb{N} | (i - \mathrm{CountN}(i, j, c, b)) \leq k \leq (i + \mathrm{CountS}(i, j, c, b)) : b[k, j] = c)$

UpdateWE : integer × integer × cellT × boardT → boolean

UpdateWE$(i, j, c, b) \equiv \forall(k : \mathbb{N} | (j - \mathrm{CountW}(i, j, c, b)) \leq k \leq (j + \mathrm{CountE}(i, j, c, b)) : b[i, k] = c)$

Local Functions

UpdateNESW : integer × integer × cellT × boardT → boolean

$\text{UpdateNESW}(i, j, c, b) \equiv \forall(k, l : \mathbb{N} | (i - \text{CountNE}(i, j, c, b)) \leq k \leq (i + \text{CountSW}(i, j, c, b)) \wedge ((j - \text{CountSW}(i, j, c, b)) \leq l \leq (j + \text{CountNE}(i, j, c, b)) : b[k, l] = c)$

UpdateNWSE : integer × integer × cellT × boardT → boolean

$\text{UpdateNWSE}(i, j, c, b) \equiv \forall(k, l : \mathbb{N} | (i - \text{CountNW}(i, j, c, b)) \leq k \leq (i + \text{CountSE}(i, j, c, b)) \wedge (j - \text{CountNW}(i, j, c, b)) \leq l \leq (j + \text{CountSE}(i, j, c, b)) : b[k, l] = c)$

Local Functions

CountN : integer × integer × cellT × boardT → integer

$$\begin{aligned}&\text{CountN}(i, j, c, b) \equiv\\&\quad +(k : \mathbb{N} \mid \text{is\_validN}(i, j, c, b) \wedge\\&\quad 0 < k < i \wedge \forall(l : \mathbb{N} \mid k \leq l < i : \text{hostile}(l, j, c, b)) : 1)\end{aligned}$$

CountS : integer × integer × cellT × boardT → integer

$$\begin{aligned}&\text{CountS}(i, j, c, b) \equiv\\&\quad +(k : \mathbb{N} \mid \text{is\_validS}(i, j, c, b) \wedge\\&\quad i < k < (\text{SIZE} - 1) \wedge \forall(l : \mathbb{N} \mid i < l \leq k : \text{hostile}(l, j, c, b)) : 1)\end{aligned}$$

etc.

Local Functions

is_validN : integer × integer × cellT × boardT → boolean

$$\text{is\_validN}(i, j, c, b) \equiv$$
$$\exists(k : \mathbb{N} | 0 \leq k < i : \text{friendly}(k, j, c, b) \wedge \forall(l : \mathbb{N} | k < l < i : \text{hostile}(l, j, c, b)))$$

etc.

friendly: integer × integer × cellT × boardT → boolean

$\text{friendly}(i, j, c, b) \equiv b[i, j] = c$

Local Functions

hostile: integer × integer × cellT × boardT → boolean

$\mathrm{hostile}(i, j, c, b) \equiv (b[i, j] = \mathrm{BLACK} \Rightarrow c = \mathrm{WHITE}\ |b[i, j] = \mathrm{WHITE} \Rightarrow c = \mathrm{BLACK}\ |c = \mathrm{FREE} \Rightarrow \mathit{false})$

InvalidPosition: integer × integer → boolean

$\mathrm{InvalidPosition}(i, j) \equiv \neg((0 \leq i < \mathrm{SIZE}) \wedge (0 \leq j < \mathrm{SIZE}))$

is_correctPlayer: boolean × cellT → boolean

$\mathrm{is\_correctPlayer}(bt, c) \equiv (bt \Rightarrow c = \mathrm{BLACK}|\neg bt \Rightarrow c = \mathrm{WHITE})$

References

- http://www.wikihow.com/Play-Othello
- http://www.hannu.se/games/othello/rules.htm

Play Othello

- http://www.hannu.se/games/othello/othello.asp