CPSC 2150 Project Report

MARK SANDOMENO

Requirements Analysis

Functional Requirements:

- As a player, I can drop my token into a specific column on the grid in order to complete my turn.
- As a player, I can only drop a token when it is my turn only so that there is a fair and even order.
- As a player, I can choose another column to drop my token in if I choose a fully occupied column, due to it being an invalid coordinate.
- As a player, I can choose another column to drop my token in if my selection is out-of-bounds, due to it being an invalid coordinate.
- As a player, if I have 4 tokens in a row in the horizontal direction I have won the game of Connect 4.
- As a player, if I have 4 tokens in a row in the vertical direction I have won the game of Connect 4.
- As a player, if I have 4 tokens in a row in the diagonal direction I have won the game of Connect 4.
- As a player, if I am token ‘X’ I will go first in the game every time.
- As a player, if I am token ‘O’ I will go second in the game every time.
- As a player, at the end of the game I will either be a winner or loser because there can only be one winner.
- As a player, if all columns are filled with 6 tokens and neither player achieved 4 in a row, the game results in a tie.
- As a player, at the conclusion of the game I have the choice to clear the board and play again, or exit the program so that multiple games can be played in a row.

Non-Functional Requirements

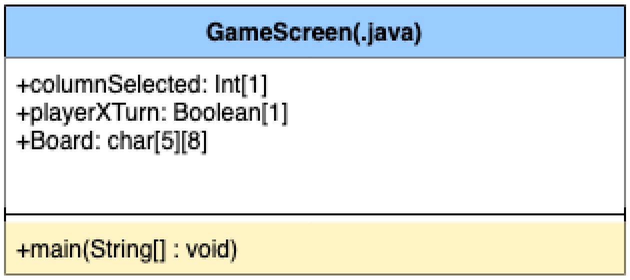
- As an implementer, this program will be written in Java
- As an implementer, this program will be run on a single command line.
- As an implementer, this program will not require any data to be saved outside of the running program.
- As an implementer, this program must display to the console an updated game board after each player turn.
- As an implementer, this program will have 3 separate files for the screen, board, and position logic respectively.
- As an implementer, the bottom left of the board will be representative of the coordinate (0, 0).

Deployment Instructions

Details in Projects 2-5.

System Design

Class 1: GameScreen.java

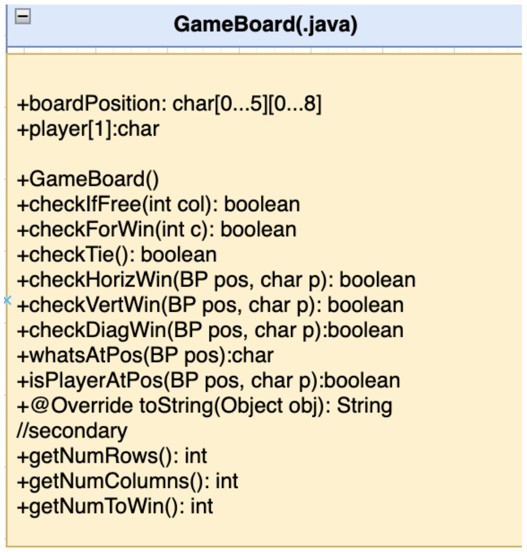


GameScreen(.java)

+columnSelected: Int[1]
+playerXTurn: Boolean[1]
+Board: char[5][8]

+main(String[] : void)

Class 2: GameBoard.java

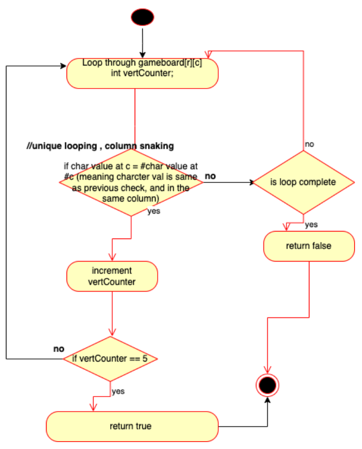


GameBoard(.java)

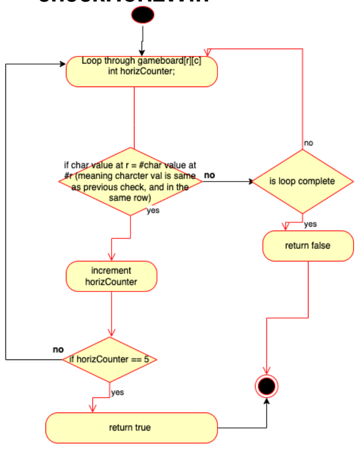
+boardPosition: char[0...5][0...8]
+player[1]:char

+GameBoard()
+checkIfFree(int col): boolean
+checkForWin(int c): boolean
+checkTie(): boolean
+checkHorizWin(BP pos, char p): boolean
+checkVertWin(BP pos, char p): boolean
+checkDiagWin(BP pos, char p):boolean
+whatsAtPos(BP pos):char
+isPlayerAtPos(BP pos, char p):boolean
+@Override toString(Object obj): String
//secondary
+getNumRows(): int
+getNumColumns(): int
+getNumToWin(): int

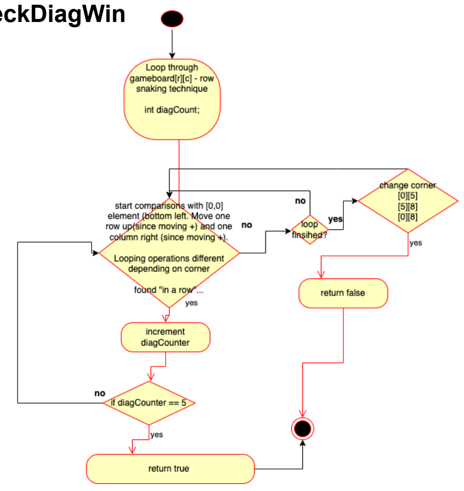
checkVertWin


Loop through gameboard[r][c]
int vertCounter;
//unique looping , column snaking
if char value at c = #char value at #c (meaning charcter val is same as previous check, and in the same column)
no
yes
is loop complete
no
yes
return false
increment vertCounter
if vertCounter == 5
no
yes
return true

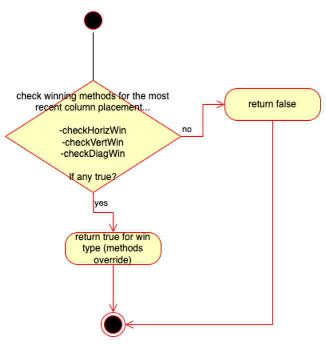
checkHorizWin


Loop through gameboard[r][c]
int horizCounter;
if char value at r = #char value at #r (meaning charcter val is same as previous check, and in the same row)
no
yes
is loop complete
no
yes
return false
increment horizCounter
if horizCounter == 5
no
yes
return true

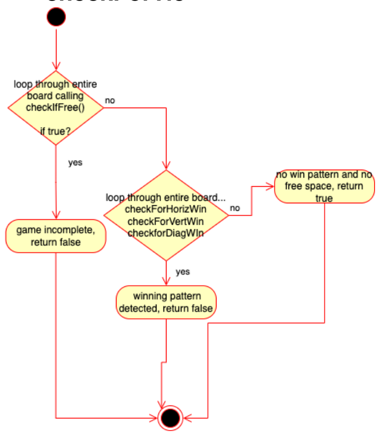
checkDiagWin


Loop through gameboard[r][c] - row snaking technique
int diagCount;
start comparisons with [0,0] element (bottom left. Move one row up(since moving +) and one column right (since moving +).
Looping operations different depending on corner
found "in a row"...
yes
no
loop finsihed?
no
yes
change corner [0][5] [5][8] [0][8]
yes
return false
increment diagCounter
if diagCounter == 5
no
yes
return true

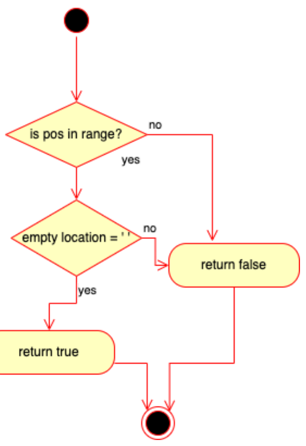
checkForWin


check winning methods for the most recent column placement...
-checkHorizWin
-checkVertWin
-checkDiagWin
If any true?
no
return false
yes
return true for win type (methods override)

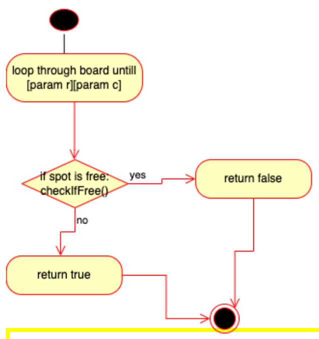
checkForTie


loop through entire board calling checkIfFree()
if true?
no
yes
game incomplete, return false
loop through entire board...
checkForHorizWin
checkForVertWin
checkforDiagWIn
no
no win pattern and no free space, return true
yes
winning pattern detected, return false

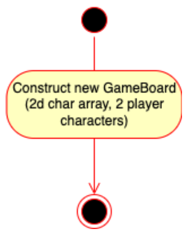
checkIfFree



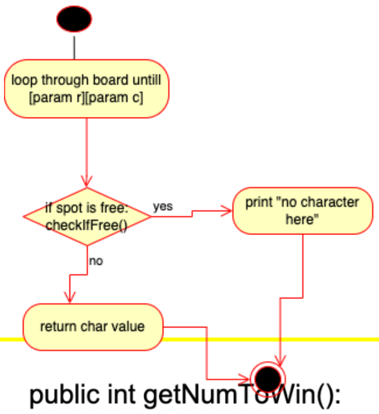
isPlayerAtPos


loop through board untill [param r][param c]
if spot is free: checkIfFree()
yes
return false
no
return true

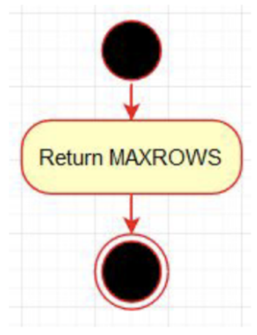
Constructor


Construct new GameBoard (2d char array, 2 player characters)

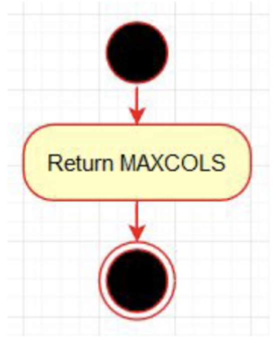
whatsAtPos


loop through board untill [param r][param c]
if spot is free: checkIfFree()
yes
print "no character here"
no
return char value

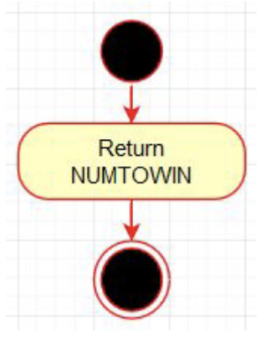
public int getNumRows():



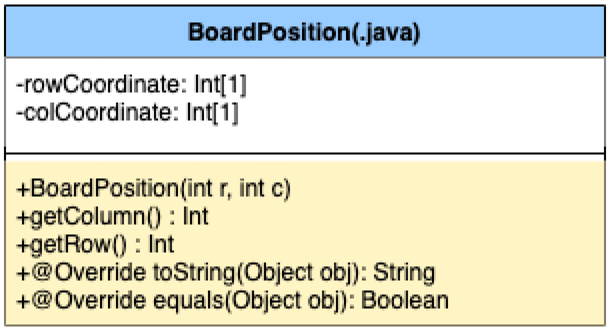
public int getNumColumns():



public int getNumToWin():



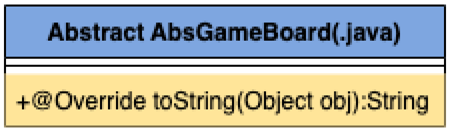
Class 3: BoardPosition.java


BoardPosition(.java)

-rowCoordinate: Int[1]
-colCoordinate: Int[1]

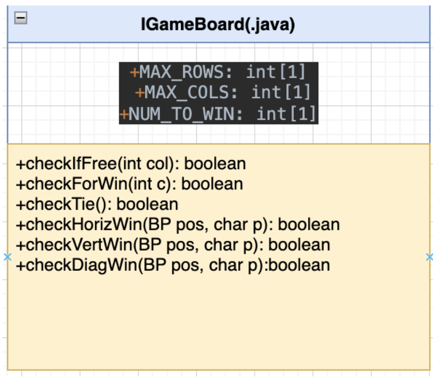
+BoardPosition(int r, int c)
+getColumn() : Int
+getRow() : Int
+@Override toString(Object obj): String
+@Override equals(Object obj): Boolean

Class 4: AbsGameBoard.java


Abstract AbsGameBoard(.java)

+@Override toString(Object obj):String

Class 5: IGameBoard.java


IGameBoard(.java)

```
+MAX_ROWS: int[1]
+MAX_COLS: int[1]
+NUM_TO_WIN: int[1]
```

+checkIfFree(int col): boolean
+checkForWin(int c): boolean
+checkTie(): boolean
+checkHorizWin(BP pos, char p): boolean
+checkVertWin(BP pos, char p): boolean
+checkDiagWin(BP pos, char p):boolean