CPSC 2150 Project 1 Report

Gabriel Hillesheim, Trent Brown, Breanna Mackey

Requirements Analysis

Functional Requirements:

1. As a player I can choose what column I want so that I can make my move.
2. As a player I can select Y at the end of the game so that I can play it over again.
3. As a player I can select N at the end of the game so that I won't have to play the game over again.
4. As a player I should be able to see the game board so that I can strategize my next move.
5. As a player I can see who's turn it is so that I know when to play.
6. As a player I can connect a number of my characters in a row so that I can win.
7. As a player I can see who won at the end of the game so that we can play again.
8. As a player I can see whether the game ends in a tie so that we can play again.
9. As a player I can see what each column is labeled so that I can place my marker in the spot I want.
10. As a player I can see when a column is full so that I won't try to place my marker there.
11. As a player I can be redirected back when I make an invalid input so that I don't lose my turn.
12. As a player I can see my opponents moves so that I can make a counter move.
13. As a player I can see the column lines, so that I can see an organized game board.
14. As a player I can align my characters horizontally so that I can win.
15. As a player I can align my characters vertically so that I can win.
16. As a player I can select to run the GameBoard in a fast implementation or an efficient implementation
17. As a user I can select how many players participate in the game so that multiplayers can play together
18. As a player I can choose how many rows and columns the game board consists of so that I can change the game
19. As a player I can select how many tokens in a row it takes to win so that I can choose the difficulty
20. As a player I can choose what character I wish to be so that I can personalize the experience

Non-Functional Requirements

1. The program must be written in java

2. All input and output to the screen must happen in GameScreen.java

3. Do not write code for each method.

4. Create a project report

5. Runs on Unix/Linux must be a command line application

6. Board is of size of up to 100x100

7. Player one always comes first

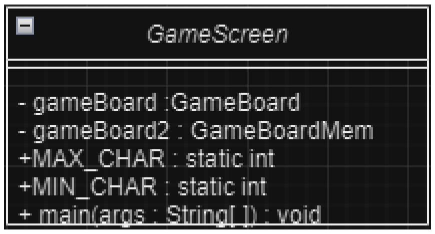
8. (0, 0) is at the bottom left of the board.

9.Board can be run in a fast implementation or an efficient implementation

System Design

GameScreen

Class diagram

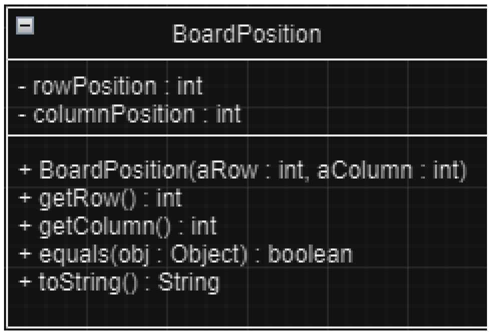


GameScreen

- gameBoard :GameBoard
- gameBoard2 : GameBoardMem
+MAX_CHAR : static int
+MIN_CHAR : static int
+ main(args : String[ ]) : void

BoardPosition

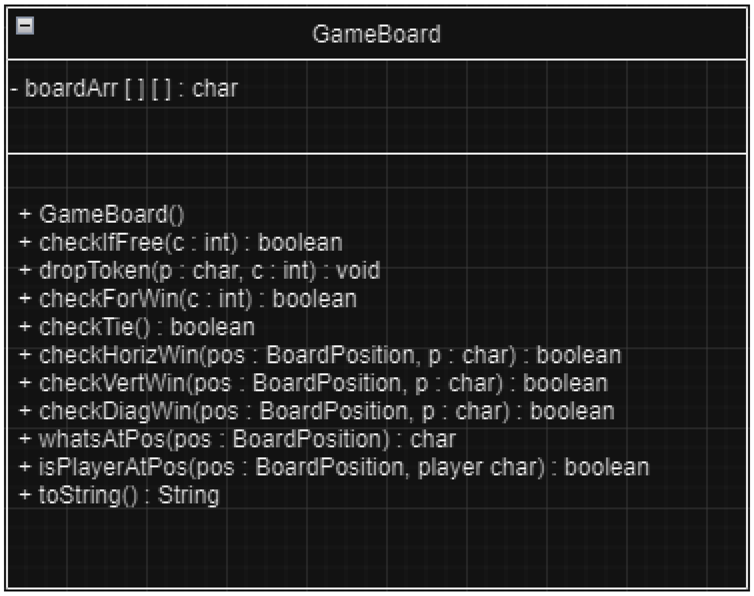
Class diagram



```
BoardPosition
- rowPosition : int
- columnPosition : int
+ BoardPosition(aRow : int, aColumn : int)
+ getRow() : int
+ getColumn() : int
+ equals(obj : Object) : boolean
+ toString() : String
```

GameBoard

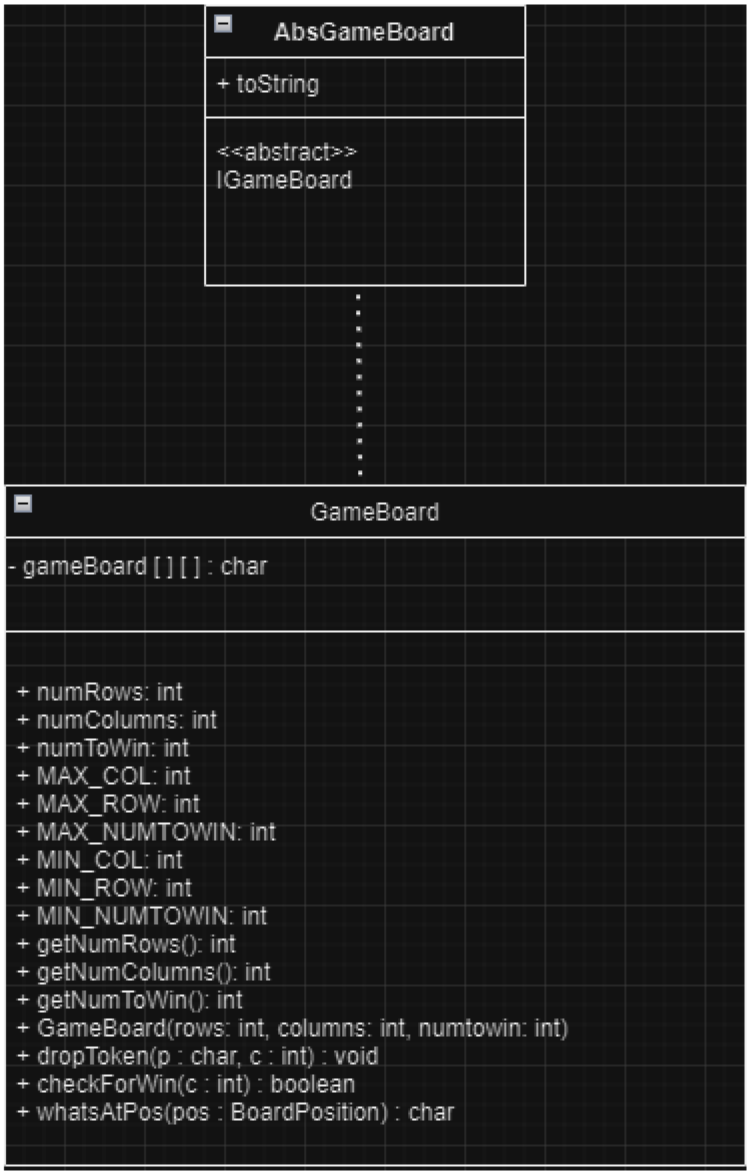
Class diagram



```
GameBoard
- boardArr [ ] [ ] : char
+ GameBoard()
+ checkIfFree(c : int) : boolean
+ dropToken(p : char, c : int) : void
+ checkForWin(c : int) : boolean
+ checkTie() : boolean
+ checkHorizWin(pos : BoardPosition, p : char) : boolean
+ checkVertWin(pos : BoardPosition, p : char) : boolean
+ checkDiagWin(pos : BoardPosition, p : char) : boolean
+ whatsAtPos(pos : BoardPosition) : char
+ isPlayerAtPos(pos : BoardPosition, player char) : boolean
+ toString() : String
```

AbsGameBoard

Class diagram


AbsGameBoard

+ toString

<<abstract>>
IGameBoard

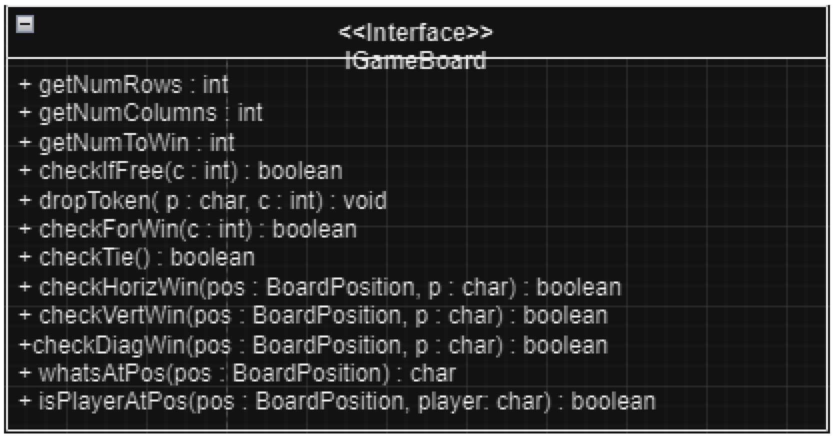
GameBoard

- gameBoard [ ] [ ] : char

+ numRows: int
+ numColumns: int
+ numToWin: int
+ MAX_COL: int
+ MAX_ROW: int
+ MAX_NUMTOWIN: int
+ MIN_COL: int
+ MIN_ROW: int
+ MIN_NUMTOWIN: int
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ GameBoard(rows: int, columns: int, numtowin: int)
+ dropToken(p : char, c : int) : void
+ checkForWin(c : int) : boolean
+ whatsAtPos(pos : BoardPosition) : char

IGameBoard

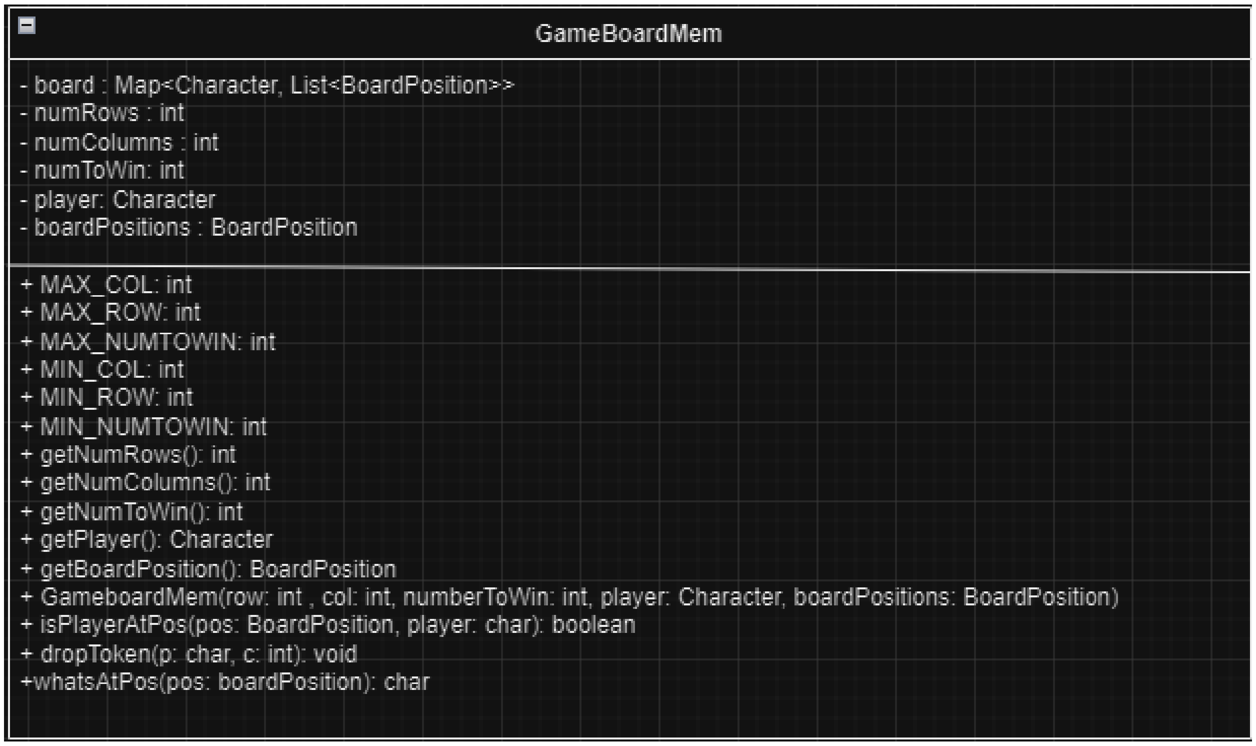
Class diagram


<<Interface>>
IGameBoard

+ getNumRows : int
+ getNumColumns : int
+ getNumToWin : int
+ checkIfFree(c : int) : boolean
+ dropToken(p : char, c : int) : void
+ checkForWin(c : int) : boolean
+ checkTie() : boolean
+ checkHorizWin(pos : BoardPosition, p : char) : boolean
+ checkVertWin(pos : BoardPosition, p : char) : boolean
+checkDiagWin(pos : BoardPosition, p : char) : boolean
+ whatsAtPos(pos : BoardPosition) : char
+ isPlayerAtPos(pos : BoardPosition, player: char) : boolean

GameBoardMem

Class diagram



GameBoardMem

- board : Map<Character, List<BoardPosition>>
- numRows : int
- numColumns : int
- numToWin: int
- player: Character
- boardPositions : BoardPosition

+ MAX_COL: int
+ MAX_ROW: int
+ MAX_NUMTOWIN: int
+ MIN_COL: int
+ MIN_ROW: int
+ MIN_NUMTOWIN: int
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ getPlayer(): Character
+ getBoardPosition(): BoardPosition
+ GameboardMem(row: int , col: int, numberToWin: int, player: Character, boardPositions: BoardPosition)
+ isPlayerAtPos(pos: BoardPosition, player: char): boolean
+ dropToken(p: char, c: int): void
+whatsAtPos(pos: boardPosition): char