CPSC 2150 Project 4 Report
Titus Ahlborn, Jake Lunski, Tyler Kriney, and Eric Vien

Requirements Analysis

Functional Requirements

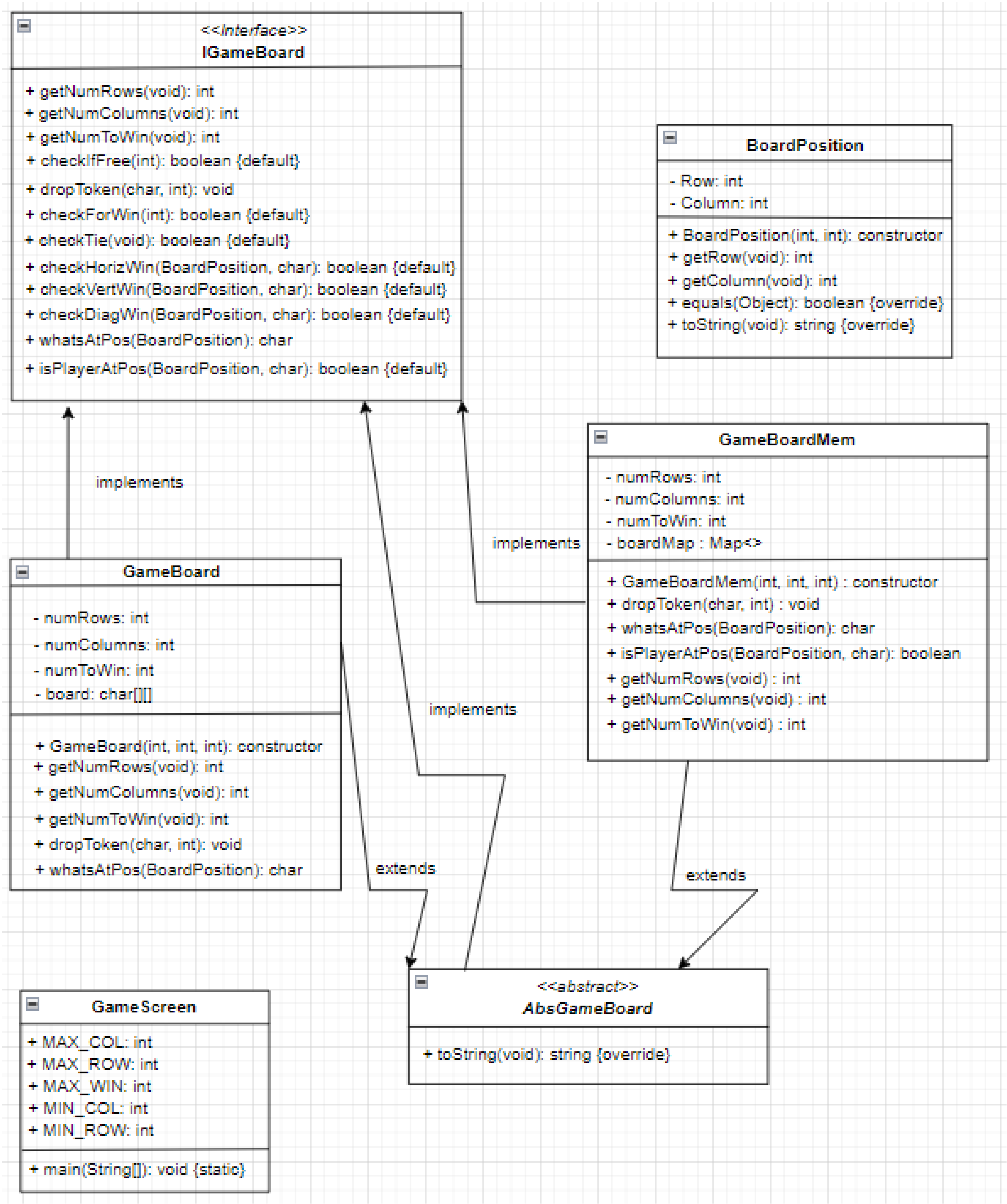
- As a player, I need to be able to choose the size of the board so that I can have more options when playing the game.
- As a player, I need to be able to set the number of in a row tiles needed to win so that I can customize how the game is won.
- As a player, I need to be able to input which character I would like to play as so that I can customize the game to my liking.
- As a player I need a way to start the game so that I can play.
- As a player, I need to choose what row to place my tile in so I can strategise.
- As a player, I need to see whose turn it is so I know when to make a move.
- As a player, I need to know who won and lost so I know the outcome of the game.
- As a player, I need to know where the opponent/s placed their tile so I can play my next move accordingly.
- As a player, I need to be able to see the board clearly so I can make informed decisions about my moves.
- As a player, I need to be able to rematch a player if I lose so I can try to win next time.
- As a player I need to know if the column is full and receive feedback so I can choose a different column.
- As a player I need to know if the game ends in a tie so I know the outcome of the game.
- As a player I need the game to automatically check for a win after every move so that the game ends if there's a winner.
- As a player I need my token to automatically drop to the lowest unoccupied row In the column I choose so it follows the rules of the game.
- As a player I need the game to make sure my input is valid so that the game runs smoothly without errors.
- As a player, I need to see an updated board after every move so that I'm aware of the current state of the game.
- As a player, I need to see a clear board when I start a new game so that I can start fresh.
- As a player I need the game to make sure I enter a column inside board boundaries so that I don't make any illegal moves.
- As a player I need my tiles to be different from my opponents so I can easily tell the difference between them.
- As a player, I need to win when I have my selected amount in a row horizontally so the game ends correctly.
- As a player, I need to win when I have my selected amount in a row vertically so the game ends correctly.
- As a player, I need to win when I have my selected amount in a row diagonally so the game ends correctly.

- As a player, I need clear and concise instructions when playing so that I can play the game without being confused.

Non-Functional Requirements

- Must be written in Java
- Command-line player interface to see the gameboard and make game decisions
- Should run quickly and efficiently through all game state checks at the end of each turn
- Should catch all errors and edge cases.
- Feedback messages are clear and concise
- The code should be well organized and commented
- The board should be resizable between sizes of 3 x 3 to 100 x 100.
- Should have a memory efficient option and regular option of data storage.
- Player one will always go first
- First letter of string inputted for player character will be taken
- Any player character inputted will be capitalized.
- The top left slot will be referred to as (0,0) and it will proceed right for a given number of columns and down a given number of rows.

UMLS


<<Interface>>
IGameBoard
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int
+ checkIfFree(int): boolean {default}
+ dropToken(char, int): void
+ checkForWin(int): boolean {default}
+ checkTie(void): boolean {default}
+ checkHorizWin(BoardPosition, char): boolean {default}
+ checkVertWin(BoardPosition, char): boolean {default}
+ checkDiagWin(BoardPosition, char): boolean {default}
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean {default}
BoardPosition
- Row: int
- Column: int
+ BoardPosition(int, int): constructor
+ getRow(void): int
+ getColumn(void): int
+ equals(Object): boolean {override}
+ toString(void): string {override}
implements
GameBoardMem
- numRows: int
- numColumns: int
- numToWin: int
- boardMap : Map<>
+ GameBoardMem(int, int, int) : constructor
+ dropToken(char, int) : void
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean
+ getNumRows(void) : int
+ getNumColumns(void) : int
+ getNumToWin(void) : int
implements
GameBoard
- numRows: int
- numColumns: int
- numToWin: int
- board: char[][]
+ GameBoard(int, int, int): constructor
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int
+ dropToken(char, int): void
+ whatsAtPos(BoardPosition): char
implements
extends
extends
GameScreen
+ MAX_COL: int
+ MAX_ROW: int
+ MAX_WIN: int
+ MIN_COL: int
+ MIN_ROW: int
+ main(String[]): void {static}
<<abstract>>
AbsGameBoard
+ toString(void): string {override}

Testing

GameBoard(int aRow, int aColumn, int aNumToWin)

Test 1: testConstructor_rows3_cols3_numToWin3

Input:	Output:	Reason:
- Rows: 3 - Columns: 3 - NumToWin: 3	- New 3x3 gameboard is created and toString works correctly - getNumColumns returns 3 - getNumRows returns 3 - getNumToWin returns 3	- This test checks an edge case to see if the constructor works at the smallest constraints we allow for game purposes

Test 2: testConstructor_rows100_cols100_numToWin25

Input:	Output:	Reason:
- Rows: 100 - Columns: 100 - NumToWin: 25	- New 100x100 gameboard is created and toString works correctly - getNumColumns returns 100 - getNumRows returns 100 - getNumToWin returns 25	- This test checks an edge case to see if the constructor works at the largest constraints we allow for game purposes

Test 3: testConstructor_rows35_cols20_numToWin20

Input:	Output:	Reason:
- Rows: 35 - Columns: 20 - NumToWin: 20	- New 35x20 gameboard is created and toString works correctly - getNumColumns returns 20 - getNumRows returns 35 - getNumToWin returns 20	- This test checks an edge case to see if the constructor works when the numToWin is the same as the smallest dimension but is not larger than 25 as specified by the design constraints.

boolean checkDiagWin(BoardPosition pos, char p)

Test 1:

testDiagWin_Rows3_Cols3_NumToWin3_TokenAt_Row_0_1_Col_0_1_PlaceTokenAt_Row_2_Col_2_True

<table>
<tr>
<td>Input:
<table><tr><td></td><td></td><td><b>t</b></td></tr><tr><td></td><td>t</td><td>n</td></tr><tr><td>t</td><td>n</td><td>n</td></tr></table></td>
<td>Output:
- True
- State of board is unchanged</td>
<td>Reason:
- Tests that the function correctly decides player ‘t’ has won in the smallest possible gameboard with correct diagonal win when the last token is top right.</td>
</tr>
</table>

Test 2:

testDiagWin_Rows3_Cols3_NumToWin3_TokenAt_Row_0_2_Col_0_2_PlaceTokenAt_Row_1_Col_1_True

<table>
<tr>
<td>Input:
<table><tr><td></td><td></td><td>t</td></tr><tr><td></td><td><b>t</b></td><td>n</td></tr><tr><td>t</td><td>n</td><td>n</td></tr></table></td>
<td>Output:
- True
- State of board is unchanged</td>
<td>Reason:
- Tests that the function correctly decides player ‘t’ has won in the smallest possible gameboard with correct diagonal win when the last token is in the middle.</td>
</tr>
</table>

Test 3:

testDiagWin_Rows3_Cols3_NumToWin3_TokenAt_Row_1_2_Col_1_2_PlaceTokenAt_Row_0_Col_0_True

<table>
<tr>
<td>Input:
<table><tr><td></td><td></td><td>t</td></tr><tr><td></td><td>t</td><td>n</td></tr><tr><td><b>t</b></td><td>n</td><td>n</td></tr></table></td>
<td>Output:
- True
- State of board is unchanged</td>
<td>Reason:
- Tests that the function correctly decides player ‘t’ has won in the smallest possible gameboard with correct diagonal win when the last token is bottom left.</td>
</tr>
</table>

Test 4:
testDiagWin_Rows5_Cols5_NumToWin5_TokenAt_Row_4_3_2_1_Col_0_1_2_3_PlaceTokenAt_Row_0_Col_4_True

<table>
<tr>
<td>Input:
<table><tr><td>t</td><td></td><td></td><td></td><td></td></tr><tr><td>n</td><td>t</td><td></td><td></td><td></td></tr><tr><td>n</td><td>n</td><td>t</td><td></td><td></td></tr><tr><td>n</td><td>n</td><td>n</td><td>t</td><td></td></tr><tr><td>n</td><td>n</td><td>n</td><td>n</td><td>**t**</td></tr></table></td>
<td>Output:
- True
- State of board is unchanged</td>
<td>Reason:
- Tests that the function correctly decides player ‘t’ has won in a 5x5 game board with correct diagonal win when the last token is bottom right.</td>
</tr>
</table>

Test 5:
testDiagWin_Rows5_Cols5_NumToWin5_TokenAt_Row_4_3_1_0_Col_0_1_3_4_PlaceTokenAt_Row_2_Col_2_True

<table>
<tr>
<td>Input:
<table><tr><td>t</td><td></td><td></td><td></td><td></td></tr><tr><td>n</td><td>t</td><td></td><td></td><td></td></tr><tr><td>n</td><td>n</td><td>**t**</td><td></td><td></td></tr><tr><td>n</td><td>n</td><td>n</td><td>t</td><td></td></tr><tr><td>n</td><td>n</td><td>n</td><td>n</td><td>t</td></tr></table></td>
<td>Output:
- True
- State of board is unchanged</td>
<td>Reason:
- Tests that the function correctly decides player ‘t’ has won in a 5x5 game board with correct diagonal win when the last token is middle.</td>
</tr>
</table>

Test 6:
testDiagWin_Rows5_Cols5_NumToWin5_TokenAt_Row_3_2_1_0_Col_1_2_3_4_PlaceTokenAt_Row_4_Col_0_True

<table>
<tr>
<td>Input:
<table><tr><td>**t**</td><td></td><td></td><td></td><td></td></tr><tr><td>n</td><td>t</td><td></td><td></td><td></td></tr><tr><td>n</td><td>n</td><td>t</td><td></td><td></td></tr><tr><td>n</td><td>n</td><td>n</td><td>t</td><td></td></tr></table></td>
<td>Output:
- True
- State of board is unchanged</td>
<td>Reason:
- Tests that the function correctly decides player ‘t’ has won in a 5x5 game board with correct diagonal win when the last token is top left.</td>
</tr>
</table>

<table>
<tr><td><table><tr><td>n</td><td>n</td><td>n</td><td>n</td><td>t</td></tr></table></td><td></td><td></td></tr>
</table>

Test 7: testDiagWin_Rows3_Cols3_NumToWin3_PlaceTokenAtRow_0_False

<table>
<tr>
<td>Input:
<table><tr><td>t</td><td></td><td></td></tr><tr><td>t</td><td></td><td></td></tr><tr><td>t</td><td></td><td></td></tr></table></td>
<td>Output:
- False
- State of board is unchanged</td>
<td>Reason:
- Tests that the function correctly decides player ‘t’ has not won diagonally in a 3x3 game board with a correct vertical win condition.</td>
</tr>
</table>

default public boolean checkHorizWin(BoardPosition pos, char p)

Test 1:

<table>
<tr>
<td><b>Input:</b>
State: (number to win = 4)
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>t</td><td>t</td><td>t</td><td>t</td><td></td></tr></table>
pos.getRow = 0
pos.getCol = 0
p = ‘t’</td>
<td><b>Output:</b>
checkHorizWin = true

State of the board unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because the last t was placed on the far left side of board with 3 consecutive t’s to the right, so the function will need to count to the right meaning that this test case is a left boundary check for checkHorizWin.

<b>Function name:</b>
testHorizWin_Rows5_Cols5_NumToWin4_TokenAtCol_1_2_3_PlaceTokenAtCol_0_True</td>
</tr>
</table>

Test 2:

<table>
<tr>
<td><b>Input:</b>
State: (number to win = 4)
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>t</td><td>t</td><td>t</td><td>t</td></tr></table>
pos.getRow = 0
pos.getCol = 4
p = ‘t’</td>
<td><b>Output:</b>
checkHorizWin = true

State of the board unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because the last t was placed on the far right side of board with 3 consecutive t’s to the left, so the function will need to count to the left meaning that this test case is a right boundary check for checkHorizWin.

<b>Function name:</b>
testHorizWin_Rows5_Cols5_NumToWin4_TokenAtCol_1_2_3_PlaceTokenAtCol_4_True</td>
</tr>
</table>

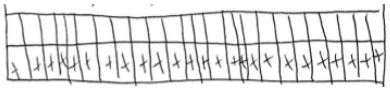
Test 3:

<table>
<tr>
<td><b>Input:</b>
State: (number to win = 3)
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td>t</td><td>t</td><td>t</td></tr></table>
pos.getRow = 0
pos.getCol = 2
p = ‘t’</td>
<td><b>Output:</b>
checkHorizWin = true

State of the board unchanged</td>
<td><b>Reason:</b>
This test case is distinct because this is a routine test case for a game with the lowest amount of rows, columns, and number to win.

<b>Function name:</b>
testHorizWin_Rows3_Cols3_NumToWin3_PlaceTokenAtCol_2_True</td>
</tr>
</table>

Test 4:

<table>
<tr>
<td><b>Input:</b>
State: (number to win = 25)

pos.getRow = 0
pos.getCol = 12
p = ‘t’</td>
<td><b>Output:</b>
checkHorizWin = true

State of the board unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct since there are 12 token to each side of the last placed token at column index 12, so the function will have to check for all tokens to the left side and the right side of the final token. Also, it tests the maximum number to win at 25, which will ensure the max number to win will work with checkHorizWin.

<b>Function name:</b>
testHorizWin_Rows2_Cols25_NumToWin25_TokenAtCol_0_To_11_And_13_To_24_PlaceTokenAtCol_12_True</td>
</tr>
</table>

<u>default public boolean checkVertWin(BoardPosition pos, char p)</u>

Test 1:

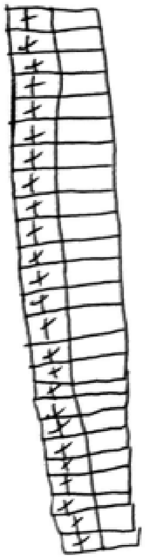
<table>
<tr>
<td><b>Input:</b>
State: (number to win = 3)
<table>
<tr><td>t</td><td></td><td></td></tr>
<tr><td>t</td><td></td><td></td></tr>
<tr><td>t</td><td></td><td></td></tr>
</table>
pos.getRow = 2
pos.getCol = 0
p = ‘t’</td>
<td><b>Output:</b>
checkVertWin = true

State of the board unchanged</td>
<td><b>Reason:</b>
This test case is distinct because this is a routine test case for a game with the lowest number of rows, columns, and number to win as well as testing the upper bound of the checkVertWin function.

<b>Function name:</b>
testVertWin_Rows3_Cols3_NumToWin3_PlaceTokenAtRow_0_True</td>
</tr>
</table>

Test 2:

Input:
State: (number to win = 25)



pos.getRow = 24
pos.getCol = 0
p = ‘t’

Output:
checkVertWin = true

State of the board unchanged

Reason:
This test case is distinct because it tests the maximum number of rows, columns, and number to win for the checkVertWin function.

Function name:
testVertWin_Rows25_Cols2_NumToWin25_TokenAtRow_0_to_23_PlaceTokenAtRow_24_True

Test 3:

Input:
State: (number to win = 3)

t	
t	
t	
e	
e	
t	
t	

Output:
checkVertWin = true

State of the board unchanged

Reason:
This test case is distinct and unique because it tests the checkVertWin function with a number to win of 3 when there are alternating sets of two tokens per player up until the top row where a 3 in a row finally occurs making this a challenging test case.

Function name:
testVertWin_Rows15_Cols2_NumToWin3_Alternating_Different_Tokens_Every_2_To

<table>
<tr>
<td>
<table>
<tr><td>e</td><td></td></tr>
<tr><td>e</td><td></td></tr>
<tr><td>t</td><td></td></tr>
<tr><td>t</td><td></td></tr>
<tr><td>e</td><td></td></tr>
<tr><td>e</td><td></td></tr>
<tr><td>t</td><td></td></tr>
<tr><td>t</td><td></td></tr>
</table>

pos.getRow = 14
pos.getCol = 0
p = ‘t’
</td>
<td></td>
<td>kens_Until_PlaceTokenAtRow_14_True</td>
</tr>
</table>

Test 4:

<table>
<tr>
<td>
<b>Input:</b>
State: (number to win = 3)
<table>
<tr><td></td><td></td><td>e</td><td></td><td></td></tr>
<tr><td></td><td></td><td>t</td><td></td><td></td></tr>
<tr><td></td><td></td><td>t</td><td></td><td></td></tr>
<tr><td></td><td></td><td>t</td><td></td><td></td></tr>
<tr><td></td><td></td><td>e</td><td></td><td></td></tr>
</table>

pos.getRow = 3
pos.getCol = 2
p = ‘t’
</td>
<td>
<b>Output:</b>
checkVertWin = true

State of the board unchanged
</td>
<td>
<b>Reason:</b>
This test case is distinct and unique because it tests if the checkVertWin function will still be able to work if a 3 in a row is sandwiched between two opposing tokens making this a challenging test case.

<b>Function name:</b>
testVertWin_Rows5_Cols5_NumToWin3_FullRow_With_OpposingToken_OnTop_And_OnBottom_3InARow_InMiddle_True
</td>
</tr>
</table>

default public boolean checkIfFree(int c)

Test 1:

<table>
<tr>
<td><b>Input:</b>
State: (number to win = 3)
<table><tr><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td></tr><tr><td> </td><td> </td><td> </td></tr></table>
c = 0</td>
<td><b>Output:</b>
checkIfFree = true

State of the board unchanged</td>
<td><b>Reason:</b>
This test case is distinct because this is a routine test case for the function checkIfFree since it will have to return 'true' if the gameboard is empty.

<b>Function name:</b>
testCheckIfFree_Rows3_Cols3_NumToWin3_NoPositionsFilled_ColToCheck_0_True</td>
</tr>
</table>

Test 2:

<table>
<tr>
<td><b>Input:</b>
State: (number to win = 3)
<table><tr><td>t</td><td> </td><td>t</td></tr><tr><td>t</td><td>t</td><td>t</td></tr><tr><td>t</td><td>t</td><td>t</td></tr></table>
c = 1</td>
<td><b>Output:</b>
checkIfFree = true

State of the board unchanged</td>
<td><b>Reason:</b>
This test case is distinct because it is testing the checkIfFree function to return true when given a full board with the exception of one top row in the column to check, making this an edge case for checkIfFree.

<b>Function name:</b>
testCheckIfFree_AllPositionsFilled_Except_TopRow_At_Col_1_ColToCheck_1_True</td>
</tr>
</table>

Test 3:

<table>
<tr>
<td><b>Input:</b>
State: (number to win = 3)
<table><tr><td>t</td><td>t</td><td>t</td></tr></table></td>
<td><b>Output:</b>
checkIfFree = false

State of the board unchanged</td>
<td><b>Reason:</b>
This test case is distinct because it tests the function checkIfFree to see if it will</td>
</tr>
</table>

<table>
<tr>
<td><table><tr><td>t</td><td>t</td><td>t</td></tr><tr><td>t</td><td>t</td><td>t</td></tr></table>
c = 0</td>
<td></td>
<td>return false when a full board is sent to the function.

<b>Function name:</b>
testCheckIfFree_BoardFull_ColToCheck_0_False</td>
</tr>
</table>

default public boolean checkTie()

Test 1:

<table>
<tr>
<td><b>Input:</b>
State (number to win = 3)
<table><tr><td>x</td><td>o</td><td>x</td></tr><tr><td>x</td><td>o</td><td>x</td></tr><tr><td>o</td><td>x</td><td>o</td></tr></table></td>
<td><b>Output:</b>
checkTie = true

State of the board unchanged</td>
<td><b>Reason</b>:
This test case is designed to verify that the checkTie method correctly identifies a tie when all positions are filled and there is no winner. The alternating pattern of 'X' and 'O' ensures no three alike tokens align, making it a comprehensive test for a full board without a win condition.

<b>Function Name:</b>
testCheckTie_AllPositionsFilled_NoWin()</td>
</tr>
</table>

Test 2:

<table>
<tr>
<td><b>Input:</b>
State (number to win = 3)
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table></td>
<td><b>Output:</b>
checkTie = false

State of the board unchanged</td>
<td><b>Reason</b>:
This case tests if checkTie method properly identifies that the game is not in a tie state when the board is empty. It is good to ensure that the tie logic only triggers under correct conditions.

<b>Function Name:</b>
testCheckTie_BoardEmpty()</td>
</tr>
</table>

Test 3:

<table>
<tr>
<td><b>Input:</b>
State (number to win = 3)
<table>
<tr><td>x</td><td>o</td><td>x</td></tr>
<tr><td>x</td><td>x</td><td>o</td></tr>
<tr><td>o</td><td></td><td>o</td></tr>
</table>
</td>
<td><b>Output:</b>
checkTie = false;

State of the board unchanged</td>
<td><b>Reason:</b>
This test case evaluates the checkTie method's ability to correctly identify a non-tie situation. The test ensures that checkTie only returns true when the board is completely filled without any player having the required tokens in a row for a win.

<b>Function Name:</b>
testCheckTie_OneEmptyPosition_NoWin()</td>
</tr>
</table>

Test 4:

<table>
<tr>
<td><b>Input:</b>
State (number to win = 3)
checkTie = false;
<table>
<tr><td>x</td><td>o</td><td>x</td></tr>
<tr><td>x</td><td>x</td><td>o</td></tr>
<tr><td>o</td><td>x</td><td></td></tr>
</table>
</td>
<td><b>Output:</b>
State:
checkTie = true;
<table>
<tr><td>x</td><td>o</td><td>x</td></tr>
<tr><td>x</td><td>x</td><td>o</td></tr>
<tr><td>o</td><td>x</td><td>o</td></tr>
</table>
</td>
<td><b>Reason:</b>
This test case checks the functionality of checkTie method both before and after a game-ending move. Initially, the game should not be in a tie state with one position open, but placing the final token should result in a tie.

<b>Function Name:</b>
testCheckTie_SecondToLastMoveLeavesOneSpot()</td>
</tr>
</table>

public char whatsAtPos(BoardPosition pos)

Test1:

<table>
<tr>
<td><b>Input:</b>
State: (number to win = 3)
<table>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
</table>
</td>
<td><b>Output:</b>
whatsAtPos = ' ' (empty space)

State of the board unchanged</td>
<td><b>Reason:</b>
This test case validates the whatsAtPos method's ability to return the correct token symbol or empty position at a given position. It ensures that an empty position is accurately reported.

<b>Function Name:</b>
testWhatsAtPos_EmptyPosition()</td>
</tr>
</table>

Test 2:

<table>
<tr>
<td><b>Input:</b>
State:
<table><tr><td>x</td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table></td>
<td><b>Output:</b>
whatsAtPos = ‘x’</td>
<td><b>Reason:</b>
This test case confirms that the whatsAtPos method correctly identifies a 'x' token placed at a specific position, ensuring accurate game state representation.

<b>Function Name:</b>
testWhatsAtPos_PositionFilledByPlayerX()</td>
</tr>
</table>

Test 3:

<table>
<tr>
<td><b>Input:</b>
State:
<table><tr><td>o</td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table></td>
<td><b>Output:</b>
whatsAtPos = ‘o’</td>
<td><b>Reason:</b>
The purpose of this test is to make sure the whatsAtPos method accurately detects a 'o' token at the specified location, reflecting the correct board status. This also proves that it can recognize more than 1 char input which means the user can choose to play with any char they choose.

<b>Function Name:</b>
testWhatsAtPos_PositionFilledByPlayerO()</td>
</tr>
</table>

Test 4:

<table>
<tr>
<td><b>Input:</b>
State:
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td>x</td><td>o</td><td>x</td></tr></table></td>
<td><b>Output:</b>
whatsAtPos for each position should return 'X', 'O', and 'X' respectively</td>
<td><b>Reason:</b>
This test case challenges the whatsAtPos method to accurately return the token at multiple positions within a single column, ensuring the method's reliability across different board heights.

<b>Function Name:</b>
testWhatsAtPos_MultipleTokensInColumn()</td>
</tr>
</table>

Test 5:

Input:	**Output:**	**Reason:**

<table>
<tr>
<td>State:
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td>x</td><td>o</td><td>x</td></tr><tr><td></td><td>x</td><td>o</td></tr><tr><td></td><td></td><td></td></tr></table></td>
<td>whatsAtPos = ‘ ’ (empty space)</td>
<td>This scenario tests the whatsAtPos method's ability to confirm that a position above the highest token in a partially filled column is indeed empty, which is crucial for the integrity of the game logic.

<b>Function Name:</b>
testWhatsAtPos_HigherEmptyPositionInPartiallyFilledColumn()</td>
</tr>
</table>

default public boolean isPlayerAtPos(BoardPosition pos, char player)

Test 1

<table>
<tr>
<td><b>Input:</b>
State:
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td><b>X</b></td><td></td><td></td><td></td><td></td></tr></table></td>
<td><b>Output:</b>
isPlayerAtPos(BoardPosition(0,0), ‘X’) = true</td>
<td><b>Reason:</b>
This scenario tests the isPlayerAtPos method’s ability to find the simple scenario of the first position with one character present on the board.

<b>Function Name:</b>
testisPlayerAtPos_BottomCorner()</td>
</tr>
</table>

Test 2

<table>
<tr>
<td><b>Input:</b>
State:
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td><b>Output:</b>
isPlayerAtPos(BoardPosition(0,0), ‘X’) = false</td>
<td><b>Reason:</b>
This scenario tests the isPlayerAtPos method’s ability to confirm that the token checked is not the same as the token present in the board position.

<b>Function Name:</b>
testisPlayerAtPos_PlayerDifferentAtPos()</td>
</tr>
</table>

<table>
<tr>
<td><table><tr><td><b>O</b></td><td></td><td></td><td></td><td></td></tr></table></td>
<td></td>
<td></td>
</tr>
</table>

Test 3

<table>
<tr>
<td><b>Input:</b>
State: 100 x 100 grid
Showing the 100th column with rows 95-100 showing.
<table><tr><td><b>X</b></td></tr><tr><td>O</td></tr><tr><td>X</td></tr><tr><td>O</td></tr><tr><td>X</td></tr><tr><td>...(Rows 94 to 0 below.)</td></tr></table></td>
<td><b>Output:</b>
isPlayerAtPos(BoardPosition(99,99), ‘X’) = true</td>
<td><b>Reason:</b>
This scenario tests the isPlayerAtPos method’s ability to check the top boundary position of the largest top corner of the largest board.

<b>Function Name:</b>
testisPlayerAtPos_Largest_Top_Right_Corner()</td>
</tr>
</table>

Test 4

<table>
<tr>
<td><b>Input:</b>
State:
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td><b>X</b></td><td></td><td></td><td></td><td></td></tr></table></td>
<td><b>Output:</b>
isPlayerAtPos(BoardPosition(0,0), ‘X’) = true</td>
<td><b>Reason:</b>
This scenario tests the isPlayerAtPos method’s ability to check the correct position when tokens are placed on top of it.

<b>Function Name:</b>
testisPlayerAtPos_Multiple_Tokens_on_Top_of_tested_position()</td>
</tr>
</table>

Test 4

<table>
<tr>
<td><b>Input:</b>
State:
<table><tr><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td><b>Output:</b>
isPlayerAtPos(BoardPosition(0,0), ‘X’) = false</td>
<td><b>Reason:</b>
This scenario tests the isPlayerAtPos method’s ability to return false when checking for a player’s token when the board is empty.</td>
</tr>
</table>

<table>
<tr>
<td><table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td></td>
<td><b>Function Name:</b>
testisPlayerAtPos_when_board_is_empty()</td>
</tr>
</table>

default public void dropToken(char p, int c)

Test 1

<table>
<tr>
<td><b>Input:</b>
Four tokens of the same type around each other.
State:
<table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td><b>A</b></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td><b>A</b></td><td>A</td><td><b>A</b></td><td></td><td></td><td></td></tr></table></td>
<td><b>Output:</b>
A 10 x 10 board array with the token ‘A’ in positions (0,4), (0,5), (0,6), and (1,5)</td>
<td><b>Reason:</b>
This scenario tests the dropToken method’s ability to drop tokens around a token already placed.

<b>Function Name:</b>
testDropToken_Dropped_Same_Token_On_Top_And_To_Right_And_To_Left_Of_Starting_Token_In_Middle()</td>
</tr>
</table>

Test 2

Input: Token, ‘X’, placed in position (0,0) State:	**Output:** A 5 x 5 board array with the token ‘X’ in the first position (0,0).	**Reason:** This scenario tests the dropToken method’s ability to drop a token in the first position.

<table>
<tr>
<td><table>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>X</b></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td></td>
<td><b>Function Name:</b>
testDropToken_One_Token_Dropped_First_Column()</td>
</tr>
</table>

Test 3

<table>
<tr>
<td><b>Input:</b>
Token, ‘X’, placed in position (0,99)
State:
<table>
<tr><td></td><td>.. (Rows: 1-99 after this)</td></tr>
<tr><td>.. (Columns: 0-98 before this)</td><td>X</td></tr>
</table></td>
<td><b>Output:</b>
A 5 x 5 board array with the token ‘X’ in the last column (0,99).</td>
<td><b>Reason:</b>
This scenario tests the dropToken method’s ability to drop a token in the last position.

<b>Function Name:</b>
testDropToken_One_Token_Dropped_in_Largest_Column()</td>
</tr>
</table>

Test 4

<table>
<tr>
<td><b>Input:</b>
Max number of player tokens in the same column.
State:
<table>
<tr><td><b>J</b></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>I</b></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>H</b></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>G</b></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>F</b></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>E</b></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>D</b></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Output:</b>
A 10 x 10 board array with 10 different tokens in the first column.</td>
<td><b>Reason:</b>
This scenario tests the dropToken method’s ability to drop different tokens on top of each other.

<b>Function Name:</b>
testDropToken_Dropped_Max_Number_Player_Ten_In_Same_Column()</td>
</tr>
</table>

<table>
<tr>
<td>
<table>
<tr><td><b>C</b></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>B</b></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>A</b></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td></td>
<td></td>
</tr>
</table>

Test 5

<table>
<tr>
<td>
<b>Input:</b>
Max number of player tokens in the same row.
State:
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>A</b></td><td><b>B</b></td><td><b>C</b></td><td><b>D</b></td><td><b>E</b></td><td><b>F</b></td><td><b>G</b></td><td><b>H</b></td><td><b>I</b></td><td><b>J</b></td></tr>
</table>
</td>
<td>
<b>Output:</b>
A 10 x 10 board array with 10 different tokens in the first row.
</td>
<td>
<b>Reason:</b>
This scenario tests the dropToken method’s ability to drop different tokens next to each other.

<b>Function Name:</b>
testDropToken_Dropped_Max_Number_Player_Ten_In_Same_Row()
</td>
</tr>
</table>