CPSC 2150 Project 3 Report

Jack Huber, Derek Smith, Just Kang

Requirements Analysis

Functional Requirements:

1. As a player, I need to select how many players I want, so I can play with my desired amount of players
2. As a player, I need to be prompted to re-select how many plays I want if I have chosen to few or too many players
3. As a player, I want to select a character to represent each player's token so that I know who each player is
4. As a player, I need to re-select a character to represent my token if I have selected a character that is already in use, so that I don't have more than one player with the same token
5. As a player, I need to select how many rows and columns I want my board to be so that I can choose the size of the game board
6. As a player, I want to re-select how many rows and columns I want my board to be if I have selected a row or column size that is too small or too big, so that my game board isn't too big or too small.
7. As a player, I want to select how many tokens in a row I want in order to win the game so that I can choose how many tokens in a row I want to win the game
8. As a player, I want to re-select how many tokens in a row I need to win the game if I have selected too few or too many, so that I don't play a game with too few or too many tokens in a row in order to win
9. As a player, I want to be able to select either a fast game or a memory efficient game, so that I can choose my game to either run faster or be more memory efficient.
10. As a player, I need to see which player's turn it is so I know

when to place my marker.

11. As a player, I need to input which column I'd like to place my marker so I can specify where I want to place my marker.
12. As a player, I want to know if I've chosen to place my marker at an unavailable location.
13. As a player, I need to be able to pick again if I picked a full column
14. As a player, I want a prompt to display when a player has won the game, so I know when the game has ended.
15. As a player, I want a prompt to display when the game has ended in a tie, so I know when the game has ended.
16. As a player, I want the game to move on to the next player turn when the game has resulted in neither a tie or a win, so the game will proceed.
17. As a player, I want to see the board displayed after each player's turn, so that I can see the board after each players turn
18. As a player, I want the option to play again after the game has ended, so I can choose whether or not I want to continue to play.
19. As a player, I want the option to select the number of players in the game, the character for each player, the rows and columns size of the board, and how many tokens are needed in a row to win after I have chosen to play again so that I can re customize the game

Non-Functional Requirements

1. The game must be able to run smoothly

2. The game must have no errors and work properly

3. The game must be easy to play with no confusing steps

4. The rows and columns of the game board must be no greater than 100 and no less than 3.

5. The coordinates (0,0) must be the bottom left of the board

6. Program must run on Unix

7. Program must be written in Java

8. The number of players must be no less than 2 and no greater than 10

9. The number of tokens in a row to win must not exceed 25 and be no fewer than 3

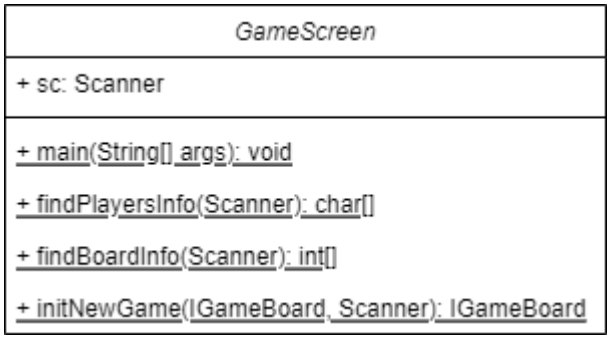
Deployment Instructions

Details in Projects 2-5.

System Design

Class 1: GameScreen

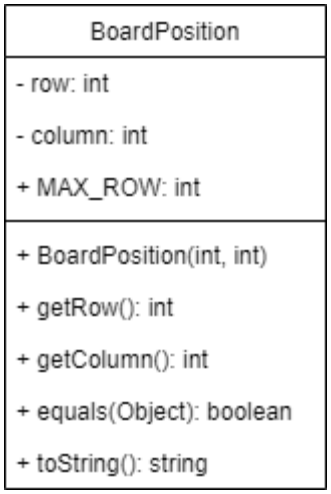
Class diagram



Activity diagrams

Class 2: BoardPosition

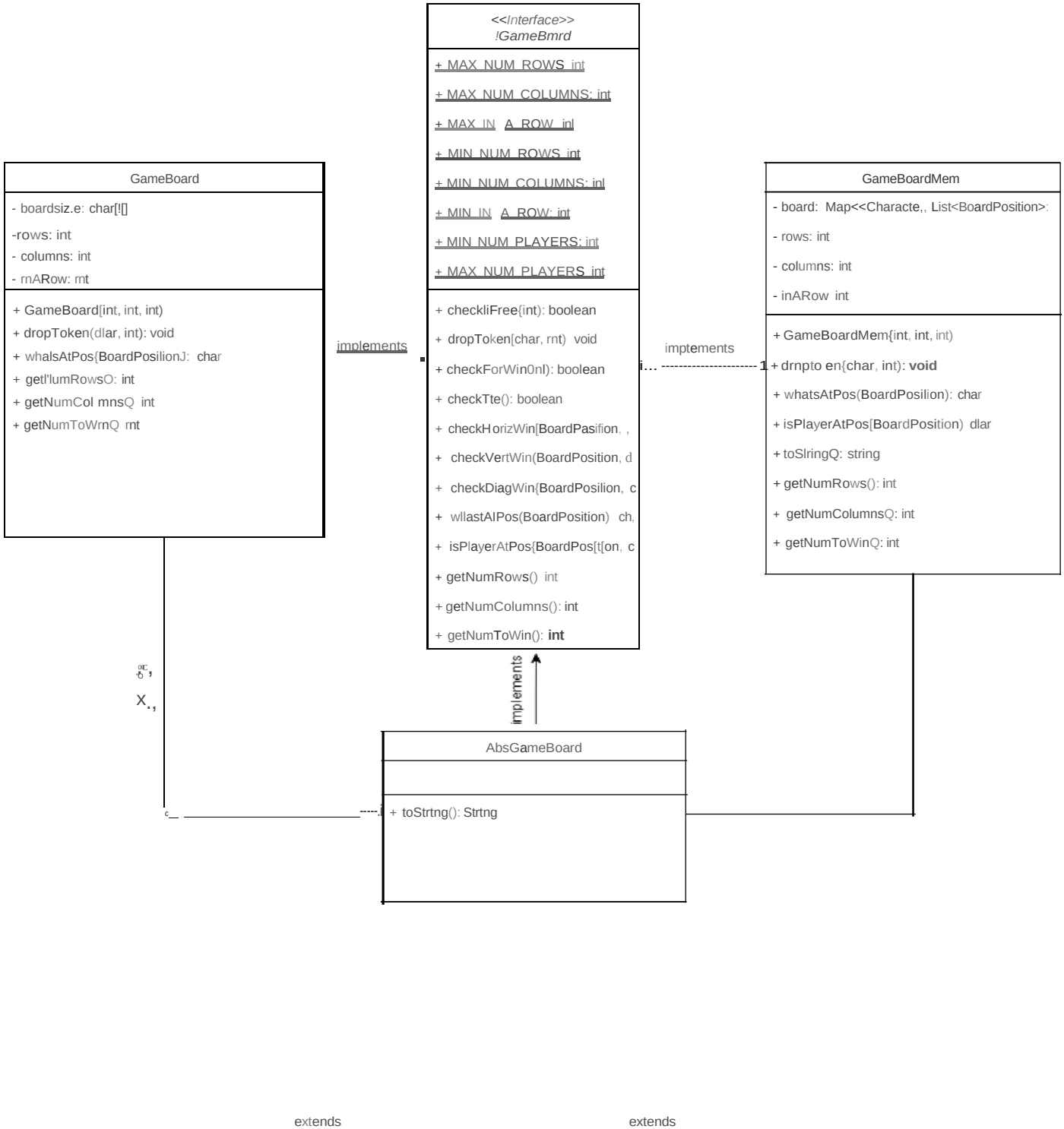
Class diagram



Activity diagrams

Class 3: GameBoard, GameBoardMem, IGameBoard, AbsGameBoard

Class diagram


<<Interface>>
!GameBmrd
+ MAX_NUM_ROWS: int
+ MAX_NUM_COLUMNS: int
+ MAX_IN_A_ROW: inl
+ MIN_NUM_ROWS: int
+ MIN_NUM_COLUMNS: inl
+ MIN_IN_A_ROW: int
+ MIN_NUM_PLAYERS: int
+ MAX_NUM_PLAYERS: int
+ checkliFree(int): boolean
+ dropToken(char, rnt) void
+ checkForWin0nI): boolean
+ checkTte(): boolean
+ checkHorizWin[BoardPasifion, ,
+ checkVertWin(BoardPosition, d
+ checkDiagWin{BoardPosilion, c
+ wllastAlPos(BoardPosition) ch,
+ isPlayerAtPos{BoardPos[t[on, c
+ getNumRows() int
+ getNumColumns(): int
+ getNumToWin(): int
GameBoard
- boardsiz.e: char[![]
-rows: int
- columns: int
- rnARow: rnt
+ GameBoard[int, int, int)
+ dropToken(dlar, int): void
+ whalsAtPos{BoardPosilionJ: char
+ getl'lumRowsO: int
+ getNumCol mnsQ int
+ getNumToWrnQ rnt
implements
imptements
1
GameBoardMem
- board: Map<<Characte,, List<BoardPosition>:
- rows: int
- columns: int
- inARow int
+ GameBoardMem{int, int, int)
+ drnpto en{char, int): void
+ whatsAtPos(BoardPosilion): char
+ isPlayerAtPos[BoardPosition) dlar
+ toSlringQ: string
+ getNumRows(): int
+ getNumColumnsQ: int
+ getNumToWinQ: int
implements
AbsGameBoard
+ toStrtng(): Strtng
extends
extends

Test Cases

Details in Project 4.

GameBoard(int row, int column, int numWin)

Input: State: <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>	**Output:** GameBoard = 3 x3 board with blank characters in each position Rows = row Columns = column inARow = numWin	**Reason:** This test case is unique and distinct because it checks that the gameboard is able to be created at the smallest row and column size possible. **Function Name:** testConstructor_r3_c3_win3_Smallest_Size

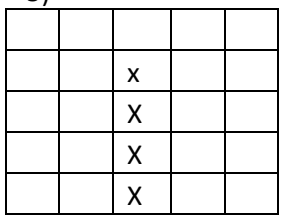
GameBoard(int row, int column, int numWin)

Input: State: 100 x 100 Empty Board with a blank char for each value	**Output:** GameBoard = 100 x 100 board with blank characters in each position Rows = rows Columns = column inARow = numWin	**Reason:** This test case is unique and distinct because it checks that the gameboard is able to be created at the smallest row and column size possible. **Function name:** testConstructor_r100_c100_win25_Largest_Size

GameBoard(int row, int column, int numWin)

Input: 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></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></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>	**Output:** GameBoard = 10 x 5 board with blank characters in each position Rows = rows Columns = column inARow = numWin	**Reason:** This test case is unique and distinct because it checks that the gameboard is able to be created when the amount of rows and greater than the amount of columns. **Function Name:** testConstructor_r10_c5_win5

Boolean checkIfFree(int c)

Input: State: (number to win = 5)  c = 2	**Output:** checkIfFree = True state of the board is unchanged	**Reason**: This test case is unique and distinct because it checks that the middle column's position is free when every position in the column is full except for the last position. **Function Name:** testCheckIfFree_r5_c5_win5_Check_Middle_Column_One_Space_Left

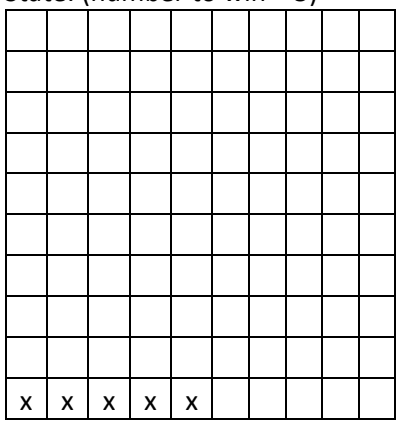
Boolean checkIfFree(int c)

Input: State: (number to win = 25) 100 x 100 Board with each position containing an 'X'.	**Output:** checkIfFree = false state of the board is unchanged	**Reason:** This test case is unique and distinct because it checks that when the largest possible board size contains a character in each position, it will return that there is not an open position on the table. **Function Name:** testCheckIfFree_r100_c100_win25_Largest_Size_Full_Board

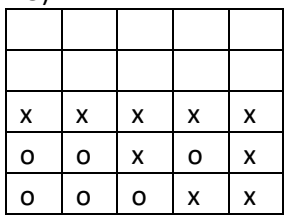
Boolean checkIfFree(int c)

Input: State: (number to win = 4) 	**Output:** checkIfFree = true state of the board is unchanged	**Reason:** This test case is unique and distinct because it checks that checkIfFree returns true when checking with a completely empty board. **Function Name:** testCheckIfFree_r5_c6_win4_Empty_Board

Boolean checkHorizWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 5)  pos.getRow = 0 pos.getCol = 4 p = 'x'	checkHorizWin = true state of the board is unchanged	This test case is unique and distinct because it checks that checkHorizWin returns true when it checks for a win at the bottom row **Function Name:** testHorizontalWin_r10_c10_win5_String_Of_Character_At_Bottom

Boolean checkHorizWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 5)  pos.getRow = 2 pos.getCol = 2 char p = 'x'	checkHorizWin = true state of the board is unchanged	This test case is unique and distinct because it checks that checkHorizWin returns true when checking for a win when the last token placed is a token in the middle of, making it check the tokens from both the left and the right to ensure that the number to win has or has not been reached **Function Name:** testHorizontalWin_r5_c5_win5_Last_Character_Is_Middle_Of_String

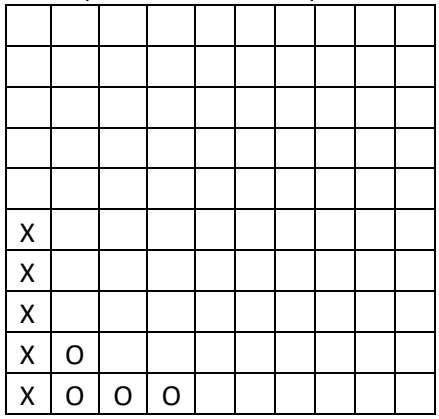
Boolean checkHorizWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 3) o o o o x o x O o x x O x o x x pos.getRow = 3 pos.getCol = 3 char p = 'o'	checkHorizWin = true state of the board is unchanged	This test case is unique and distinct because it checks that checkHorizWin returns true when checking for a win when the number of tokens in a row needed to win are present in the top most row. **Function Name:** testHorizontalWin_r4_c4_win3_Character_String_At_Top_Most_Row

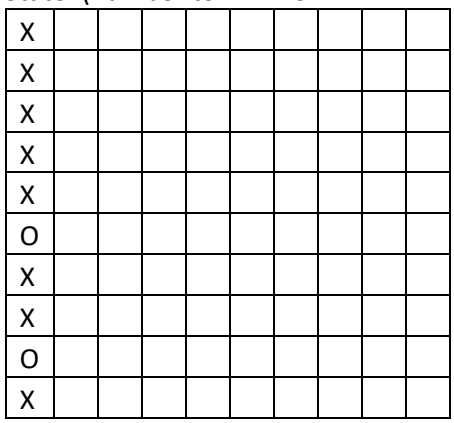
Boolean checkHorizWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 5) o x x x x x x o o O x o pos.getRow = 1 pos.getCol = 5 char p = 'x'	checkHorizWin = true state of the board is unchanged	This test case is unique and distinct because it checks that checkHorizWin returns true when checking for a win when the number of tokens in a row needed to win are present in the top most row. **Function Name:** testHorizontalWin_r10_c10_win5_Last_Character_Is_In_End_Of_String

Boolean checkVertWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 5)  pos.getRow = 4 pos.getCol = 0 char p = 'x'	checkVertWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkVertWin is true when the win condition is in the bottom left of the board **Function Name:** testVerticalWin_r10_c10_win5_Bottom_Left_Corner

Boolean checkVertWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 5  pos.getRow = 10 pos.getCol = 0 char p = 'X'	checkVertWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkVertWin is true when the win condition is in the top left of the board **Function Name:** testVerticalWin_r10_c10_win5_Top_Left

Boolean checkVertWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 5) I I I I I X I I I X pos.getRow = 9 pos.getCol = 9 char p = 'I'	checkVertWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkVertWin is true when the win condition is in the top left of the board **Function Name:** testVerticalWin_r10_c10_win5_Top_Right

Boolean checkVertWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 3) X X o o x Pos.getRow = 2 Pos.getCol = 5 Char p = '5'	checkVertWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkVertWin is true when the win condition is in the bottom right of the board **Function Name:** testVerticalWin_r5_c5_win3_Bottom_Right

Boolean checkDiagWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 5) X X O X O O O X O X X X O O X O Pos.getRow = 4 Pos.getCol = 4 Char p = 'X'	checkDiagWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkDiagWin is true when the amount in a row is a right diagonal **Function Name:** checkDiagWin_r5_c5_win5_Right_Diagonal

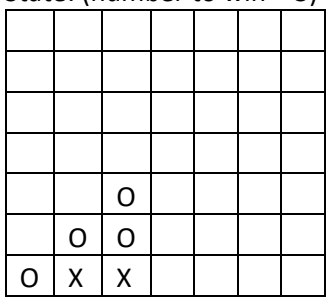
Boolean checkDiagWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 5) X X X O O X O X O X O X X O x Pos.getRow = 4 Pos.getCol = 0 Char p = 'X'	checkDiagWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkDiagWin is true when the amount in a row is a left diagonal **Function Name:** checkDiagWin_r5_c5_win5_Left_Diagonal

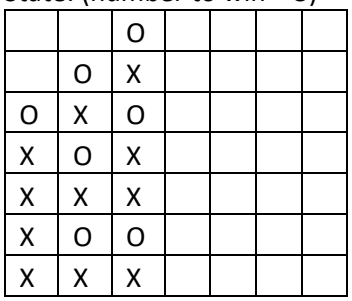
Boolean checkDiagWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 5) X X X O O X O O O X O O X O X Pos.getRow = 2 Pos.getCol = 2 Char p = 'X'	checkDiagWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkDiagWin is true when there is a left diagonal and the last character is placed in the middle of the string of characters **Function Name:** checkDiagWin_r5_c5_win5_Left_Diagonal_Last_Character_Middle_Of_String

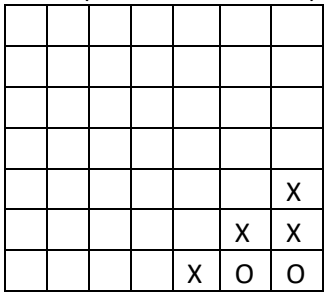
Boolean checkDiagWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 3)  Pos.getRow = 0 Pos.getCol = 0 Char p = 'O'	checkDiagWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkDiagWin is true when there is a right diagonal that is in the bottom left of the board **Function Name:** checkDiagWin_r7_c7_win3_Right_Diagonal_Bottom_Left_Of_Board

Boolean checkDiagWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 3)  Pos.getRow = 6 Pos.getCol = 2 Char p = 'O'	checkDiagWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkDiagWin is true when there is a right diagonal in the top left of the board **Function Name:** checkDiagWin_r7_c7_win3_Right_Diagonal_Top_Left_Of_Board

Boolean checkDiagWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: (number to win = 3) 	checkDiagWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkDiagWin is true there is a right diagonal in the bottom right of the board **Function Name:** checkDiagWin_r7_c7_win3_Right_Diagonal_Bottom_Right_Of_Board

Pos.getRow = 2 Pos.getCol = 6 Char p = 'X'		

Boolean checkDiagWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State (number to win = 3) O O X O O X O X O O X O X O X X O X Pos.getRow = 4 Pos.getCol = 4 Char p = 'O'	checkDiagWin = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkDiagWin is true when there is a right diagonal in the top right of the board **Function Name:** checkDiagWin_r7_c7_win3_Right_Diagonal_Top_Right_Of_Board

Boolean checkTie()

Input:	**Output:**	**Reason:**
State x x x x x o o o o o x x x x x	checkTie = true the state of the board is unchanged	This test case is unique and distinct because it checks that checkTie is true when the board is completely full **Function Name:** checkTie_r3_c5_win3_Full_Board

Boolean checkTie()

Input:	**Output:**	**Reason:**
State x x x x o o o o o x x x x x	checkTie = false the state of the board is unchanged	This test case is unique and distinct because it checks that checkTie is false when there is one empty spot on the board **Function Name:** checkTie_r3_c5_win3_One_Space_Open

Boolean checkTie()

Input:	**Output:**	**Reason:**
State X X X X O O O O X X X X O O O O X X X X	checkTie = False the state of the board is unchanged	This test case is unique and distinct because it checks that checkTie is false when there is one empty column on the right side of the board **Function Name:** checkTie_r5_c5_win5_Left_Most_Column_Empty

Boolean checkTie()

Input:	**Output:**	**Reason:**
State X I I X X I I X X I I X	checkTie = false the state of the board is unchanged	This test case is unique and distinct because it checks that checkTie is false when there is one empty row at the top of the board **Function Name:** checkTie_r4_c4_win3_Top_Most_Row_Empty

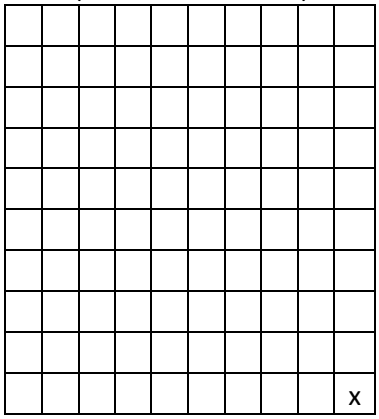
Char whatsAtPos(BoardPosition pos)

Input:	**Output:**	**Reason:**
State (number to win = 5) X Pos.getRow = 0 Pos.getCol = 0	whatsAtPos = 'X' the state of the board is unchanged	This test case is unique and distinct because it checks that whatsAtPos returns the position of a character at the bottom left of the board **Function Name:** checkWhatsAtPos_r5_c5_numWin_5_What_Character_At_Bottom_Left

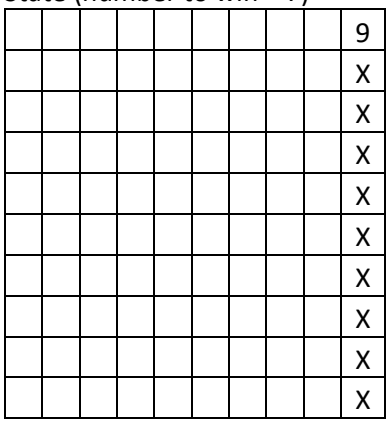
Char whatsAtPos(BoardPosition pos)

Input:	**Output:**	**Reason:**
State (number to win = 5) O X Pos.getRow = 2 Pos.getCol = 3	whatsAtPos = ' ' the state of the board is unchanged	This test case is unique and distinct because it checks that whatsAtPos returns the position of a character even when that character is a blank character **Function Name:** checkWhatsAtPos_r5_c5_numWin_5_Character_Is_Space

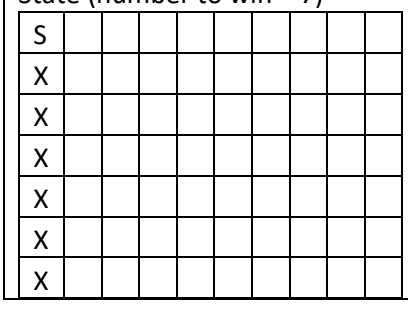
Char whatsAtPos(BoardPosition pos)

Input:	**Output:**	**Reason:**
State (number to win = 7)  Pos.getRow = 0 Pos.getCol = 9	whatsAtPos = 'X' the state of the board is unchanged	This test case is unique and distinct because it checks that whatsAtPos returns the position of a character when it is in the bottom right of the board **Function Name:** checkWhatsAtPos_r10_c10_numWin_5_What_Character_At_Bottom_Right

Char whatsAtPos(BoardPosition pos)

Input:	**Output:**	**Reason:**
State (number to win = 7)  Pos.getRow = 9 Pos.getCol = 9	whatsAtPos = '9' the state of the board is unchanged	This test case is unique and distinct because it checks that whatsAtPos returns the position of a character when it is at the top right of the board **Function Name:** checkWhatsAtPos_r5_c5_numWin_5_What_Character_At_Top_Right

Char whatsAtPos(BoardPosition pos)

Input:	**Output:**	**Reason:**
State (number to win = 7) 	whatsAtPos = 'S' the state of the board is unchanged	This test case is unique and distinct because it checks that whatsAtPos returns the position of a character when it is at the top left of the board **Function Name:** checkWhatsAtPos_r5_c5_numWin_5_What_Character_At_Top_Left

X X X Pos.getRow = 9 Pos.getCol = 0		

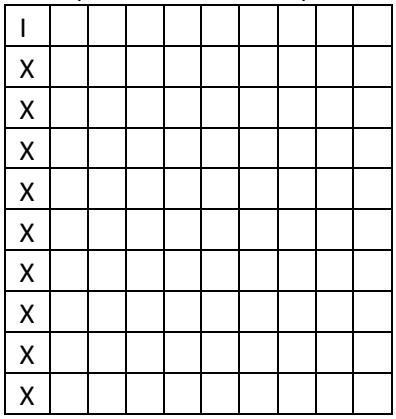
Boolean isPlayerAtPos(BoardPosition pos, char player)

Input:	**Output:**	**Reason:**
State (number to win = 7) X Pos.getRow = 0 Pos.getcol = 0 Player = 'X'	isPlayerAtPos = true the state of the board is unchanged	This test case is unique and distinct because it checks that isPlayerAtPos returns true when the character is at the bottom left position of the board **Function Name:** checkIsPlayerAtPos_r10_c10_numWin_7_Player_Character_At_Bottom_Left

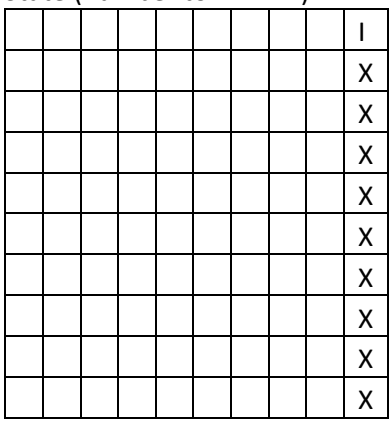
Boolean isPlayerAtPos(BoardPosition pos, char player)

Input:	**Output:**	**Reason:**
State (number to win = 7) X Pos.getRow = 0 Pos.getCol = 9 Player = 'X'	isPlayerAtPos = true the state of the board is unchanged	This test case is unique and distinct because it checks that isPlayerAtPos returns true when the character is at the bottom right position of the board **Function Name:** checkIsPlayerAtPos_r10_c10_numWin_7_Character_At_Bottom_Right

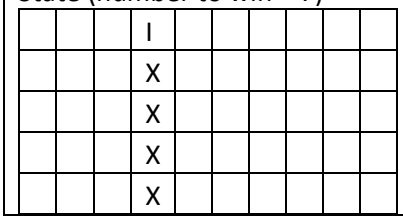
Boolean isPlayerAtPos(BoardPosition pos, char player)

Input:	**Output:**	**Reason:**
State (number to win = 7)  Pos.getRow = 9 Pos.getCol = 0 Player = 'I'	isPlayerAtPos = true the state of the board is unchanged	This test case is unique and distinct because it checks that isPlayerAtPos returns true when the character is at the top left position of the board **Function Name:** checkIsPlayerAtPos_r10_c10_numWin_7_Character_At_Top_Left

Boolean isPlayerAtPos(BoardPosition pos, char player)

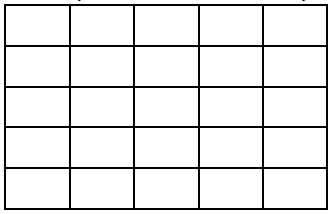
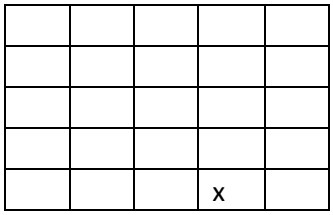
Input:	**Output:**	**Reason:**
State (number to win = 7)  Pos.getRow = 9 Pos.getCol = 9 Player = 'I'	isPlayerAtPos = true the state of the board is unchanged	This test case is unique and distinct because it checks that isPlayerAtPos returns true when the character is at the top right position of the board **Function Name:** checkIsPlayerAtPos_r10_c10_numWin_7_Character_At_Top_Right

Boolean isPlayerAtPos(BoardPosition pos, char player)

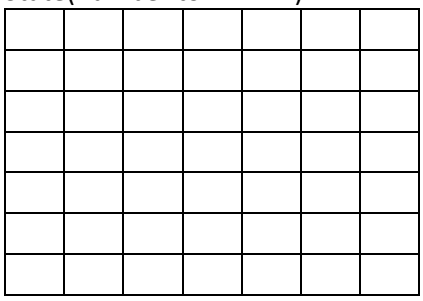
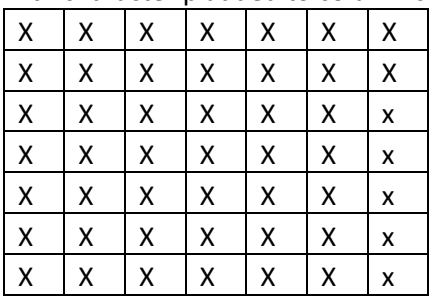
Input:	**Output:**	**Reason:**
State (number to win = 7) 	isPlayerAtPos = true the state of the board is unchanged	This test case is unique and distinct because it checks that isPlayerAtPos returns true when the character is in the top row **Function Name:** checkIsPlayerAtPos_r10_c10_numWin_7_Character_Is_At_Top_Row

			X					
			X					
			X					
			X					
			X					

Void dropToken(char p, int c)

Input:	**Output:**	**Reason:**
State (number to win = 5)  P = 'X' c = 3	dropToken = the previous board with character p added to column c 	This test case is unique and distinct because it checks that dropToken will place a character is the correct column when the column is completely empty **Function Name:** dropToken_r5_c5_numWin_5_Drop_Token_Empty_Column

Void dropToken(char p, int c)

Input:	**Output:**	**Reason:**
State(number to win = 7)  P = 'X' c = (each column of the board until the column is full)	dropToken = the previous board with character p added to column c 	This test case is unique and distinct because it checks that dropToken will place a character in each position of the board **Function Name:** dropToken_r7_c7_numWin_7_All_Positions

Void dropToken(char p, int c)

Input:

State (number to win = 5)

P = 'X' && c = 3
P = 'O' && c = 3

Output:

dropToken = the previous board with character p added to column c

			O	
			X	

Reason:
This test case is unique and distinct because it checks that dropToken will place a character on the row above the position of a character that is already in the row below it

Function Name:

dropToken_r5_c5_numWin_5_Column_With_Token_In_It

Void dropToken(char p, int c)

Input:

State (number to win = 5)

P = 'X' && c = 3
P = 'O' && c = 3
P = 'X' && c = 3
P = 'O' && c = 3
P = 'O' && c = 3

Output:

dropToken = the previous board with character p added to column c

			O	
			O	
			X	
			O	
			X	

Reason:
This test case is unique and distinct because it checks that dropToken will place a character in every position in a column

Function Name:

dropToken_r5_c5_numWin_5_Almost_Full_Column

Void dropToken(char p, int c)

Input:

State (number to win = 5)

P = 'X' && c = 4
P = 'o' && c = 4
P = 'X' && c = 4
P = 'o' && c = 4
P = 'o' && c = 4

Output:

dropToken = the previous board with character p added to column c

				O
				O
				X
				O
				X

Reason:
This test case is unique and distinct because it checks that dropToken will place a character in the correct position in the last column of the board

Function Name:

dropToken_r5_c5_numWin_5_Token_Dropped_Last_Column