CPSC 2150 Project 4 Report

Jason Russo
Joseph Becker
Evan Schwartz

Requirements Analysis

Functional Requirements:

1. As a player, I need to be able to choose a column to drop a chip so I can take a turn.
2. As a player, I need to know which columns are full, so I do not overload one.
3. As a player, I need to know whose turn it is, so I do not act out of turn.
4. As a player, I need to know the size of the board, so I do not select a column out of the bounds.
5. As a player, I need to know when the game is over, so I know when to stop making selections.
6. As a player, I need to know who won the game, so I can track wins over other players.
7. As a player, I need to have the option to play again so multiple games can be played.
8. As a player, I need to know if the game resulted in a tie so I can quit making moves and track my record.
9. As a player, I need to know what player token I am so that I can keep track of my tokens
10. As a player, I need to be informed if my choice of move is invalid so that I can pick another column
11. As a player, I need to see the game board after each turn so that I can stay informed on the progress of the game.
12. As a player, I should be prompted and able to start a new game after I complete my game so that I can play again without restarting the program
13. As a player, I need to be presented with a fresh board if I choose to play again, so I can start a game from scratch.
14. As a player, I need to know which number corresponds with each column so that I can accurately place my token
15. As a player, I need to be informed of the controls for the game so that I can play without making mistakes
16. As a player, I need to know if my last placed token completed the required same tokens in a row horizontally to win the game so that I know the game is over
17. As a player, I need to know if my last placed token completed the required same tokens in a row vertically to win the game so that I know the game is over
18. As a player, I need to know if my last placed token completed the required same tokens in a row diagonally to win the game so that I know the game is over.
19. As a player, I need to be able to choose from a fast or memory efficient gameboard so that I can choose one that better suits my game
20. As a player, I can select a number of players so that I can play with more than 2 players
21. As a player, I can select my player token so that I can distinguish myself from other players

22. As a player, I can select the number of rows, columns, and tokens in a row to win so that I can create different games

Non-Functional Requirements

1. The game must run on Unix
2. The game must run on the command line
3. The program must be written in Java
4. The program must be able to create a board for fast play
5. The program must be able to create a board for memory efficient play
6. The board size must be adjustable, with a min size 3x3 and max size 100x100
7. The board size must be able to be a square or a rectangle
8. The number of tokens to win must be between 3 and 25
9. The first player to choose their token must go first
10. The players must take their turns in the order their tokens were selected
11. (0,0) must be the bottom left position of the board

System Design

GameBoard:

Class diagram

GameBoard
- board: char[][]
+ GameBoard(): void + dropToken(char p, int c): void + whatsAtPos(BoardPosition pos): char + ToString(): string

GameBoardMem
- Board: Map<Character, List<BoardPosition>>

+ GameBoard(): void + dropToken(char p, int c): void + whatsAtPos(BoardPosition pos): char + ToString(): string

GameScreen:

Class diagram

GameScreen
- MIN_PLAYERS: int - MAX_PLAYERS: int - MIN_ROWS: int - MAX_ROWS: int - MIN_COLS: int - MAX_COLS: int - MIN_NUM_TO_WIN: int - MAX_NUM_TO_WIN: int
+ main() : void

BoardPosition:

Class diagram

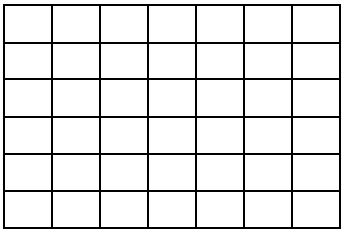
BoardPosition
- Row: int - Column: int
+ BoardPosition(aRow: int, aColumn: int) + getRow(): int + getColumn(): int + equals(obj: Object): bool + toString(): string

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

AbsGameBoard
+ toString(): String

Testing

GameBoard(int rows, int cols, int win)

Input: State:  rows = 6 columns = 7	Output: State of board matches the 6x7 input and is blank	Reason: Verifies a blank gameboard is generated to start the game. Unique because of the standard Connect-4 size. Function Name: testGameBoardConstructorStandardVals

win = 4		

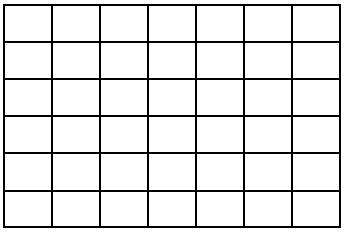
GameBoard(int rows, int cols, int win)

Input: State:  rows = 3 columns = 3 win = 3	Output: State of board matches the 3x3 input and is blank	Reason: Verifies a blank gameboard is generated to start the game. Unique because it is the minimum size. Function Name: testGameBoardConstructorMinVals

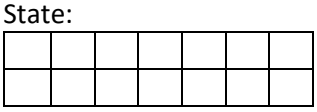
GameBoard(int rows, int cols, int win)

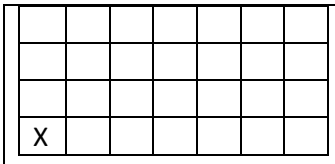
Input: State: 100x100 blank gameboard rows = 100 columns = 100 win = 25	Output: State of board matches the 100x100 input and is blank	Reason: Verifies a blank gameboard is generated to start the game. Unique because it is the maximum size. Function Name: testGameBoardConstructorMaxVals

boolean checkIfFree(int c)

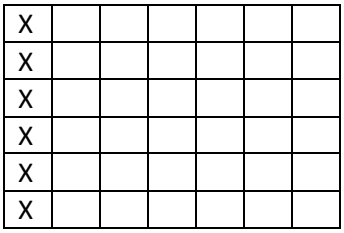
Input: State:  C = 'X'	Output: checkIfFree = True State of board is unchanged	Reason: Unique because it verifies a cell is free in a column with no tokens. Function Name: testGameBoardCheckIfFreeEmptyColumn

boolean checkIfFree(int c)

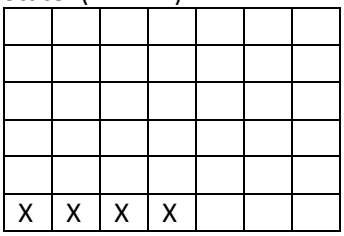
Input: Input: State: 	Output: checkIfFree = True State of board is unchanged	Reason: Unique because it verifies a cell is free in a partially full column. Function Name: testGameBoardCheckIfFreePartiallyFilledColumn

 c = 'X'		

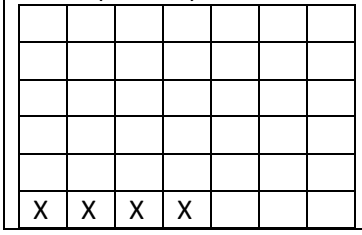
boolean checkIfFree(int c)

Input: State:  c = 'X'	Output: checkIfFree = False State of board is unchanged	Reason: Unique because it verifies a cell is not free in a full column. Function Name: testGameBoardCheckIfFreeFilledColumn

boolean checkHorizWin(BoardPosition pos, char p)

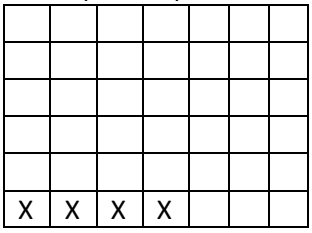
Input: State: (win = 4)  pos.getRow = 0 pos.getCol = 0 p = 'X'	Output: checkHorizWin = True State of board is unchanged	Reason: Unique because it tests a horizontal win starting from the right side and going to the left of the win pattern. Function Name: testGameBoardCheckHorizWinBottomLeft

boolean checkHorizWin(BoardPosition pos, char p)

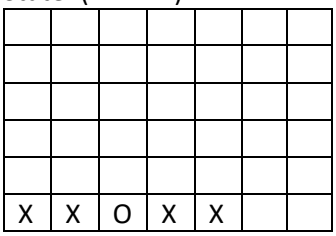
Input: State: (win = 4) 	Output: checkHorizWin = True State of board is unchanged	Reason: Unique because it tests a horizontal win starting from the left side and going to the right of the win pattern. Function Name: testGameBoardCheckHorizWinBottomRight

pos.getRow = 0 pos.getCol = 3 p = 'X'		

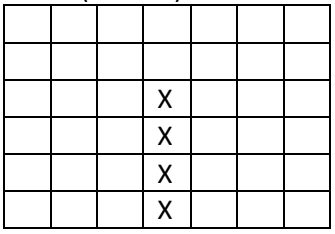
boolean checkHorizWin(BoardPosition pos, char p)

Input: State: (win = 4)  pos.getRow = 0 pos.getCol = 1 p = 'X'	Output: checkHorizWin = True State of board is unchanged	Reason: Unique because it tests a horizontal win starting from the left side of the board and dropping a token in the middle of the win pattern. Function Name: testGameBoardCheckHorizWinDropTokenInMiddle

boolean checkHorizWin(BoardPosition pos, char p)

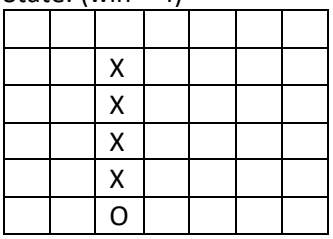
Input: State: (win = 4)  pos.getRow = 0 pos.getCol = 4 p = 'X'	Output: checkHorizWin = False State of board is unchanged	Reason: Unique because it tests a horizontal win is not given when a horizontal pattern is interrupted by an opposing player's piece. Function Name: testGameBoardCheckHorizWinWithBrokenLine

boolean checkVertWin(BoardPosition pos, char p)

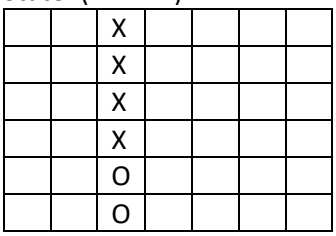
Input: State: (win = 4) 	Output: checkVertWin = True State of board is unchanged	Reason: Unique because it tests a vertical win starting from the bottom of an empty column going up. The win touches the bottom of the range. Function Name: testGameBoardCheckVertWinEmptyColumn

pos.getRow = 3 pos.getCol = 3 p = 'X'		

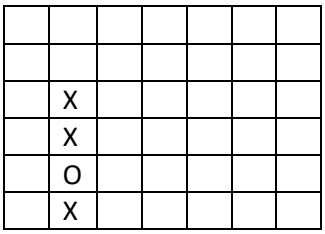
boolean checkVertWin(BoardPosition pos, char p)

Input:	Output:	Reason:
State: (win = 4)  pos.getRow = 4 pos.getCol = 2 p = 'X'	checkVertWin = True State of board is unchanged	Unique because it tests a vertical win starting from the bottom of a column with an opponent's piece inserted, going up. The win does not reach the top of the column, or touch the bottom. Function Name: testGameBoardCheckVertWinMiddleColumn

boolean checkVertWin(BoardPosition pos, char p)

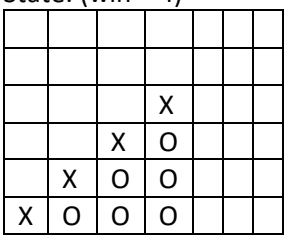
Input:	Output:	Reason:
State: (win = 4)  pos.getRow = 5 pos.getCol = 2 p = 'X'	checkVertWin = True State of board is unchanged	Unique because it tests a vertical win starting from the bottom of a column with two of the opponent's pieces inserted, going up. The win reaches the top of the column, which tests a boundary. Function Name: testGameBoardCheckVertWinTopOfColumn

boolean checkVertWin(BoardPosition pos, char p)

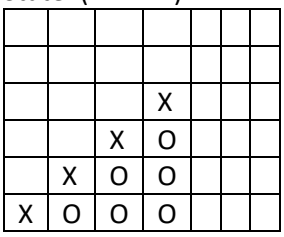
Input:	Output:	Reason:
State: (win = 4) 	checkVertWin = False State of board is unchanged	Checks for a vertical win when opposing pieces form enough pieces to win a game. Since they are all not the same, no win should be given. Function Name: testGameBoardCheckVertWinWithBrokenLine

pos.getRow = 3 pos.getCol = 1 p = 'X'		

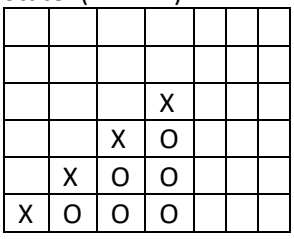
boolean checkDiagWin(BoardPosition pos, char p)

Input:	Output:	Reason:
State: (win = 4)  pos.getRow = 3 pos.getCol = 3 p = 'X'	checkDiagWin = True State of board is unchanged	Tests a diagonal win starting from the bottom of a column, ascending. Unique because the last piece is placed at the top. Function Name: testGameBoardCheckDiagWinAscendingLastTokenTopRight

boolean checkDiagWin(BoardPosition pos, char p)

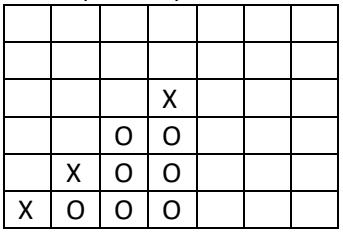
Input:	Output:	Reason:
State: (win = 4)  pos.getRow = 0 pos.getCol = 0 p = 'X'	checkDiagWin = True State of board is unchanged	Tests a diagonal win starting from the bottom of a column, ascending. Unique because the last piece is placed at the bottom. Function Name: testGameBoardCheckDiagWinAscendingLastTokenBottomLeft

boolean checkDiagWin(BoardPosition pos, char p)

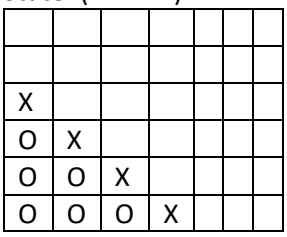
Input:	Output:	Reason:
State: (win = 4)  pos.getRow = 1	checkDiagWin = True State of board is unchanged	Tests a diagonal win starting from the bottom of a column, ascending. Unique because the last piece is placed in the middle. Function Name: testGameBoardCheckDiagWinAscendingLastTokenMiddle

pos.getCol = 1 p = 'X'		

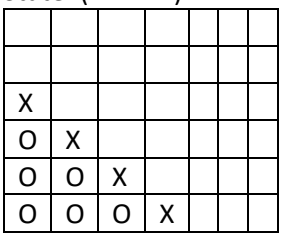
boolean checkDiagWin(BoardPosition pos, char p)

Input: State: (win = 4)  pos.getRow = 3 pos.getCol = 3 p = 'X'	Output: checkDiagWin = False State of board is unchanged	Reason: Tests a diagonal win starting from the bottom of a column, ascending. Unique because the diagonal test is false as an opposing piece breaks up the win condition. Function Name: testGameBoardCheckDiagWinAscendingBrokenLine

boolean checkDiagWin(BoardPosition pos, char p)

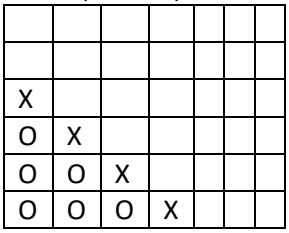
Input: State: (win = 4)  pos.getRow = 3 pos.getCol = 0 p = 'X'	Output: checkDiagWin = True State of board is unchanged	Reason: Tests a diagonal win starting from the bottom of a column, descending. Unique because the last piece is placed in the top left. Function Name: testGameBoardCheckDiagWinDescendingLastTokenTopLeft

boolean checkDiagWin(BoardPosition pos, char p)

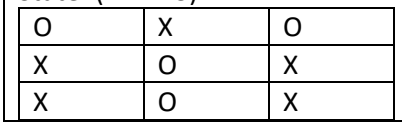
Input: State: (win = 4)  pos.getRow = 0 pos.getCol = 3	Output: checkDiagWin = True State of board is unchanged	Reason: Tests a diagonal win starting from the bottom of a column, descending. Unique because the last piece is placed in the bottom right. Function Name: testGameBoardCheckDiagWinDescendingLastTokenBottomRight

p = 'X'		

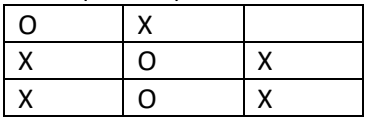
boolean checkDiagWin(BoardPosition pos, char p)

Input:	Output:	Reason:
State: (win = 4)  pos.getRow = 2 pos.getCol = 2 p = 'X'	checkDiagWin = True State of board is unchanged	Tests a diagonal win starting from the bottom of a column, descending. Unique because the last piece is placed in the middle Function Name: testGameBoardCheckDiagWinDescendingLastTokenMiddle

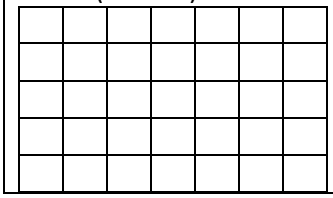
boolean checkTie()

Input:	Output:	Reason:
State: (win = 3) 	checkTie = True State of board is unchanged	Tests to make sure checkTie works correctly on a full board. Function Name: testGameBoardCheckTieFullBoard

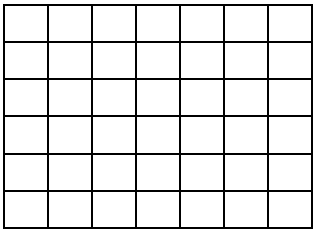
boolean checkTie()

Input:	Output:	Reason:
State: (win = 3) 	checkTie = True State of board is unchanged	Tests to make sure checkTie works correctly on a board with an open space, where no win can be achieved. Function Name: testGameBoardCheckTieEmptyBoard

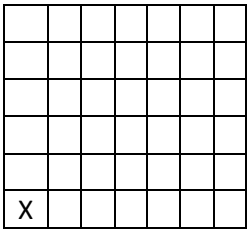
boolean checkTie()

Input:	Output:	Reason:
State: (win = 4) 	checkTie = False State of board is unchanged	Checks to make sure checkTie works as expected on a board where a tie is not a guarantee. Function Name: testGameBoardCheckTieOneFreeColumn

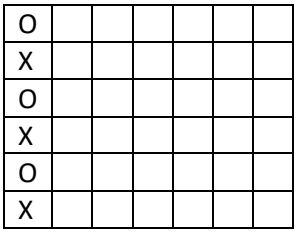
char whatsAtPos(BoardPosition pos)

Input: State:  pos.getRow = 0 pos.getCol = 0	Output: whatsAtPos = ‘ ‘ State of board is unchanged	Reason: Tests whatsAtPos when no pieces are placed. Function Name: testGameBoardWhatsAtPosEmptyBoard

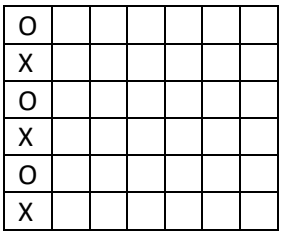
char whatsAtPos(BoardPosition pos)

Input: State:  pos.getRow = 0 pos.getCol = 0	Output: whatsAtPos = ‘X’ State of board is unchanged.	Reason: Tests whatsAtPos when one piece is placed at the bottom of a column. Function Name: testGameBoardWhatsAtPosBottomOfColumnOneTokenPlayed

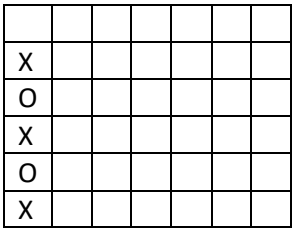
char whatsAtPos(BoardPosition pos)

Input: State:  pos.getRow = 5 pos.getCol = 0	Output: whatsAtPos = ‘O’ State of board is unchanged.	Reason: Tests whatsAtPos when a piece is placed at the top of a column, causing it to be full. This tests another boundary. Function Name: testGameBoardWhatsAtPosTopOfColumnFullColumn

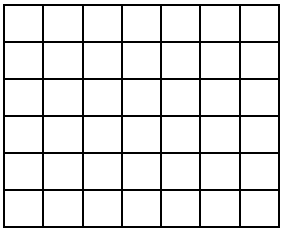
char whatsAtPos(BoardPosition pos)

Input: State:  pos.getRow = 2 pos.getCol = 0	Output: whatsAtPos = 'X' State of board is unchanged.	Reason: Tests whatsAtPos when a piece is placed at the top of top of a column, causing it to be full. This tests for a middle value, which falls within boundaries. Function Name: testGameBoardWhatsAtPosMiddleOfColumnFullColumn

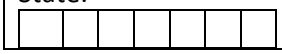
char whatsAtPos(BoardPosition pos)

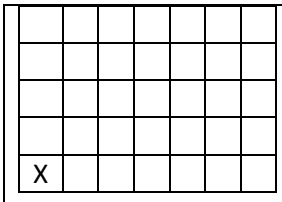
Input: State:  pos.getRow = 4 pos.getCol = 0	Output: whatsAtPos = 'X' State of board is unchanged.	Reason: Tests whatsAtPos when a piece is placed at the top of top of a column. This tests for a middle value in a partially full column. Function Name: testGameBoardWhatsAtPosTopOfPartiallyFullColumn

boolean isPlayerAtPos(BoardPosition pos, char player)

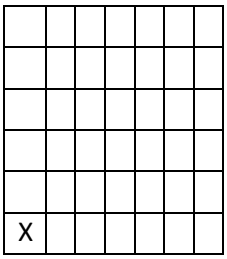
Input: State:  pos.getRow = 0 pos.getCol = 0 player = 'X'	Output: isPlayerAtPos = False State of board is unchanged	Reason: Tests if the player is at the position chosen, is unique due to the space being empty. Function Name: testGameBoardWhatsAtPosTopOfPartiallyFullColumn

boolean isPlayerAtPos(BoardPosition pos, char player)

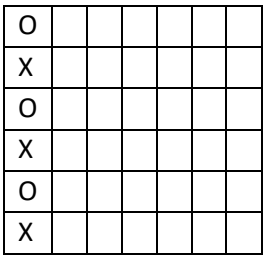
Input: State: 	Output: isPlayerAtPos = True	Reason: Tests if the player is at the position chosen, is unique due to the space being occupied by the expected character.

 pos.getRow = 0 pos.getCol = 0 player = 'X'	State of board is unchanged	Function Name: testGameBoardIsPlayerAtPosOccupiedSpaceCorrectPlayer

boolean isPlayerAtPos(BoardPosition pos, char player)

Input: State:  pos.getRow = 0 pos.getCol = 0 player = 'O'	Output: isPlayerAtPos = False State of board is unchanged	Reason: Tests if the player is at the position chosen, is unique due to the space being occupied by a character, but not the one that is expected. Function Name: testGameBoardIsPlayerAtPosOccupiedSpaceIncorrectPlayerBottomOfColumn

boolean isPlayerAtPos(BoardPosition pos, char player)

Input: State:  pos.getRow = 5 pos.getCol = 0 player = 'O'	Output: isPlayerAtPos = True State of board is unchanged	Reason: Tests if the player is at the position chosen, is unique due to a check on a column that is full, and the occupied space being at the top of the column. Function Name: testGameBoardIsPlayerAtPosOccupiedSpaceTopOfColumn

boolean isPlayerAtPos(BoardPosition pos, char player)

Input:	Output:	Reason: This test ensure that isPlayerAtPos works as intended for positions in the middle of columns Function Name: testGameBoardIsPlayerAtPosOccupiedSpaceMiddleOfColumn
O X O X O X pos.getRow=2 Pos.getCol=0 Player= 'X'	IsPlayerAtPos = true;	

void dropToken(char p, int c)

Input:	Output:	Reason: This test ensures that dropToken begins filling a column from the bottom Function Name: testGameBoardDroptokenEmptyColumn
`board.dropToken('X', 0);`	X	

void dropToken(char p, int c)

Input:	Output:	Reason: This test ensures that dropToken correctly fills a column from bottom to top without overriding what already exists in the column Function Name: testGameBoardDroptokenPartialColumn

`board.dropToken('X', 0);`
`board.dropToken('O', 0);`
`board.dropToken('X', 0);`
	X
	O
	X	

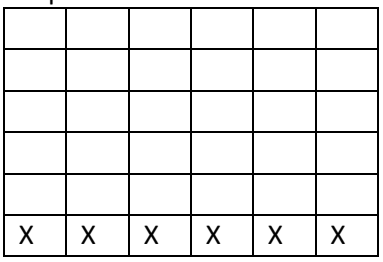
void dropToken(char p, int c)

Input:	Output:	Reason: This test ensures dropToken will work for all rows of the gameboard Function Name: testGameBoardDroptokenFullColumn

`for (int row = 0; row < rows; row++) {`
`board.dropToken('X', 0);`
`}`
	X
	X
	X
	X
	X
	X	

void dropToken(char p, int c)

Input:	Output:	Reason: This test ensures dropToken works for all columns of the gameboard Function Name: testGameBoardDroptokenFullRow

`for (int col = 0; col < cols; col++) {`
`  board.dropToken('X', col);`
`}`
		

void dropToken (char p, int c)

Input:	Output:	Reason: This test ensures dropToken is capable of filling an entire gameBoard, which is necessary if no player wins Function Name: testGameBoardDroptokenFillEntireBoard

`board.dropToken('X', 0);`
`board.dropToken('X', 0);`
`board.dropToken('O', 0);`
`board.dropToken('O', 1);`
`board.dropToken('O', 1);`
`board.dropToken('X', 1);`
`board.dropToken('X', 2);`
`board.dropToken('X', 2);`
`board.dropToken('O', 2);`