Jeffrey Black
CPSC 2150

Requirements Analysis

User Stories

- As a player, I want to be able to place a marker, so I can play Tic-Tac-Toe
- As a player, I want to be able to see the tic-tac-toe grid, so I can view the locations of the markers
- As a player, I want to know when one of the players has won, so I don't have to check after each turn
- As a player, I want to know if there has been a tie, so I don't have to check if there has been a tie
- As a player, I want to know whose turn it is, so I know who has to place their marker
- As a player, I want to know if I placed my marker on a spot that was already claimed, so that multiple players don't place their marker on the same spot.
- As a player, I want to be able to play again after the game ends, so I can play more games without starting the program again
- As a player, I want to be able to change the size of the board, so I can play with different board sizes
- As a player, I want to be able to change the number of markers in a row required to win, so I can change the win condition
- As a player, I want to be able to specify the amount of players, so I can play with however many people I want
- As a player, I want every player to have a different marker, so I don't get confused when playing.
- As a player, I want to be able to choose the amount of players after starting a new game, so I can change the amount of players without restarting the program.
- As a player, I want the program to have a GUI, so that I can easily play the game.

Non-Functional Requirements

- Must have a grid
- System must have a GUI
- System must be coded in Java
- System must follow Model View Controller architectural pattern.
- 2 <= number of players <= 10
- Player marker representations must be capitalized alphabetical characters
- Must have a fast implementation and a memory efficient implementation
- Player 1 must always be the first to play

Jeffrey Black
CPSC 2150

Constructor Tests

GameBoard(int num_rows, int num_cols, int win_req)

Input:	Output: N/A	Reason:
State: unitialized num_rows = MIN_ROWS num_cols = MIN_COLS win_req = MIN_WIN_REQ	State: 0 1 2 0 1 2	This test case is unique and distinct because it's a boundary test for the smallest possible game board you can construct. Function name: test_GameBoard_min_size

GameBoard(int num_rows, int num_cols, int win_req)

Input:	Output: N/A	Reason:
State: uninitialized num_rows = MAX_ROWS num_cols = MAX_COLS win_req = MAX_WIN_REQ	State: 0 ... 99 0 ... 99	This test case is unique and distinct because it's a boundary test for the largest possible game board you can construct. Function name: test_GameBoard_max_size

GameBoard(int num_rows, int num_cols, int win_req)

Input:	Output: N/A	Reason:
State: uninitialized num_rows = 5 num_cols = 7 win_req = MIN_WIN_REQ	State: 0 1 2 3 4 5 6 0 1 2 3 4	This test case is unique and distinct because it tests the ability for an NxM gameboard with N != M. Function name: test_GameBoard_different_rows_cols

Jeffrey Black
CPSC 2150

checkSpace Test

boolean checkSpace(BoardPosition pos)

Input:	Output:	Reason:
State: pos.getRow = 1 pos.getCol = 1	Output: false State: unchanged	This test case is unique and distinct because it tests checkSpace on a space which is already occupied. Function name: test_checkSpace_occupied

	0	1	2	3	4
0					
1		x			
2					
3					
4					

boolean checkSpace(BoardPosition pos)

Input:	Output:	Reason:
State: pos.getRow = 1 pos.getCol = 1	Output: true State: unchanged	This test case is unique and distinct because it tests checkSpace on a space which is not already occupied. Function name: test_checkSpace_unoccupied

	0	1	2	3	4
0					
1					
2					
3					
4					

boolean checkSpace(BoardPosition pos)

Input:	Output:	Reason:
State: pos.getRow = 5 pos.getCol = 1	Output: false State: unchanged	This test case is unique and distinct because it tests checkSpace on a space which is out of the bounds of the game board. Function name: test_checkSpace_out_of_bounds

	0	1	2	3	4
0					
1					
2					
3					
4					

Jeffrey Black
CPSC 2150

checkHorizontalWin Tests

boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 2 lastPos.getCol = 1 Player = x	Output: true State: unchanged	This test case is unique and distinct because lastPos is the leftmost marker in the winning row; therefore, it will have to check markers on the right of lastPos. Function name: test_checkHorizontalWin_win_start_left

	0	1	2	3	4
0					
1					
2		x	x	x	
3					
4					

boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 2 lastPos.getCol = 3 Player = x	Output: true State: unchanged	This test case is unique and distinct because lastPos is the rightmost marker in the winning row; therefore, it will have to check markers on the left of lastPos. Function name: test_checkHorizontalWin_win_start_right

	0	1	2	3	4
0					
1					
2		x	x	x	
3					
4					

Jeffrey Black
CPSC 2150

boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 2 lastPos.getCol = 2 Player = x	Output: true State: unchanged	This test case is unique and distinct because lastPos is the middlemost marker in the winning row; therefore, it will have to check markers on the left and right of lastPos. Function name: test_checkHorizontalWin_win_start_middle

	0	1	2	3	4
0					
1					
2		x	x	x	
3					
4					

boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 2 lastPos.getCol = 1 Player = x	Output: false State: unchanged	This test case is unique and distinct because there is a marker of another player interrupting the row of markers being checked; therefore, there is not a win. Function name: test_checkHorizontalWin_no_win_interrupt

	0	1	2	3	4
0					
1					
2		x	y	x	
3					
4					

Jeffrey Black
CPSC 2150

checkVerticalWin Tests

boolean checkVerticalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 1 lastPos.getCol = 2 Player = x	Output: true State: unchanged	This test case is unique and distinct because lastPos is the uppermost marker in the winning column; therefore, it will have to check markers below lastPos. Function name: test_checkVerticalWin_win_start_top

	0	1	2	3	4
0					
1			x		
2			x		
3			x		
4					

boolean checkVerticalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 3 lastPos.getCol = 2 Player = x	Output: true State: unchanged	This test case is unique and distinct because lastPos is the lowermost marker in the winning column; therefore, it will have to check markers above lastPos. Function name: test_checkVerticalWin_win_start_bottom

	0	1	2	3	4
0					
1			x		
2			x		
3			x		
4					

Jeffrey Black
CPSC 2150

boolean checkVerticalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 2 lastPos.getCol = 2 Player = x	Output: true State: unchanged	This test case is unique and distinct because lastPos is the middlemost marker in the winning column; therefore, it will have to check markers below and above lastPos. Function name: test_checkVerticalWin_win_start_middle

	0	1	2	3	4
0					
1			x		
2			x		
3			x		
4					

boolean checkVerticalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 1 lastPos.getCol = 2 Player = x	Output: false State: unchanged	This test case is unique and distinct because there is a marker of another player interrupting the column of markers being checked; therefore, there is not a win. Function name: test_checkVerticalWin_no_win_interrupt

	0	1	2	3	4
0					
1			x		
2			y		
3			x		
4					

Jeffrey Black
CPSC 2150

checkDiagonalWin Tests

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 3 lastPos.getCol = 1 Player = x	Output: true State: unchanged	Reason: This test case is unique and distinct because lastPos is the lower-leftmost marker in the winning diagonal; therefore, it will have to check markers to the upper-right of lastPos. Function name: test_checkDiagonalWin_win_start_lower_left

	0	1	2	3	4
0					
1				x	
2			x		
3		x			
4					

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 1 lastPos.getCol = 3 Player = x	Output: true State: unchanged	Reason: This test case is unique and distinct because lastPos is the upper-rightmost marker in the winning diagonal; therefore, it will have to check markers to the lower-left of lastPos. Function name: test_checkDiagonalWin_win_start_upper_right

	0	1	2	3	4
0					
1				x	
2			x		
3		x			
4					

Jeffrey Black
CPSC 2150

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 2 lastPos.getCol = 2 Player = x	Output: true State: unchanged	This test case is unique and distinct because lastPos is the middlemost marker in the winning diagonal; therefore, it will have to check markers to the lower-left and upper-right of lastPos. Function name: test_checkDiagonalWin_win_diag1_start_middle

	0	1	2	3	4
0					
1				x	
2			x		
3		x			
4					

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 1 lastPos.getCol = 1 Player = x	Output: true State: unchanged	This test case is unique and distinct because lastPos is the upper-leftmost marker in the winning diagonal; therefore, it will have to check markers to the lower-right of lastPos. Function name: test_checkDiagonalWin_win_start_upper_left

	0	1	2	3	4
0					
1		x			
2			x		
3				x	
4					

Jeffrey Black
CPSC 2150

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 3 lastPos.getCol = 3 Player = x	Output: true State: unchanged	This test case is unique and distinct because lastPos is the lower-rightmost marker in the winning diagonal; therefore, it will have to check markers to the upper-left of lastPos. Function name: test_checkDiagonalWin_win_start_lower_right

	0	1	2	3	4
0					
1		x			
2			x		
3				x	
4					

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 2 lastPos.getCol = 2 Player = x	Output: true State: unchanged	This test case is unique and distinct because lastPos is the lower-rightmost marker in the winning diagonal; therefore, it will have to check markers to the upper-left and lower-right of lastPos. Function name: test_checkDiagonalWin_win_diag2_start_middle

	0	1	2	3	4
0					
1		x			
2			x		
3				x	
4					

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:	Output:	Reason:
State: (win_req = 3) lastPos.getRow = 1 lastPos.getCol = 1 Player = x	Output: false State: unchanged	This test case is unique and distinct because there is a marker of another player interrupting the diagonal of markers being checked; therefore, there is not a win. Function name: test_checkDiagonalWin_no_win_interrupt

	0	1	2	3	4
0					
1		x			
2			y		
3				x	
4					

checkForDraw Tests

boolean checkForDraw()

Input:	Output:	Reason:
State:	Output: false State: unchanged	This test case is unique because it tests checkForDraw on an NxN gameboard for which a draw has not occurred. Function name: test_checkForDraw_no_draw_NxN

	0	1	2	3	4
0	x	x	x	x	x
1	x	x	x	x	x
2	x	x	x	x	x
3	x	x	x	x	x
4	x	x	x	x	

boolean checkForDraw()

Input:	Output:	Reason:
State:	Output: true State: unchanged	This test case is unique because it tests checkForDraw on an NxN gameboard for which a draw has occurred. Function name: test_checkForDraw_draw_NxN

	0	1	2	3	4
0	x	x	x	x	x
1	x	x	x	x	x
2	x	x	x	x	x
3	x	x	x	x	x
4	x	x	x	x	x

boolean checkForDraw()

Input: State:	Output: false State: unchanged	Reason: This test case is unique because it tests checkForDraw on an NxM (N != M) gameboard for which a draw has not occurred. Function name: test_checkForDraw_no_draw_NxM

	0	1	2	3	4	5
0	x	x	x	x	x	x
1	x	x	x	x	x	x
2	x	x	x	x	x	x
3	x	x	x	x	x	x
4	x	x	x	x	x	

boolean checkForDraw()

Input: State:	Output: true State: unchanged	Reason: This test case is unique because it tests checkForDraw on an NxM (N != M) gameboard for which a draw has occurred. Function name: test_checkForDraw_draw_NxM

	0	1	2	3	4	5
0	x	x	x	x	x	x
1	x	x	x	x	x	x
2	x	x	x	x	x	x
3	x	x	x	x	x	x
4	x	x	x	x	x	x

whatsAtPos Tests

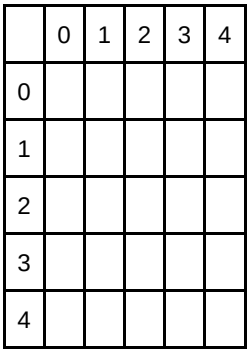
char whatsAtPos(BoardPosition pos)

Input: State:	Output: x State: unchanged	Reason: This test case is unique because it tests whatsAtPos on a valid position on the board; it checks to make sure that it returns the value at pos. Function name: test_whatsAtPos_x_on_board

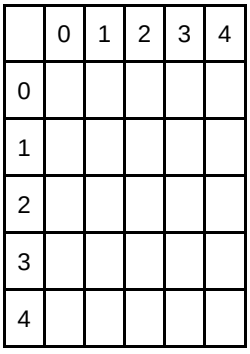
	0	1	2	3	4
0					
1					
2			x		
3					
4					

pos.getRow = 2
pos.getCol = 2

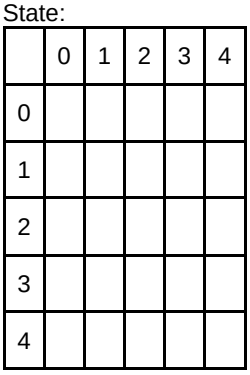
char whatsAtPos(BoardPosition pos)

Input: State:  pos.getRow = 0 pos.getCol = 5	Output: ~ State: unchanged	Reason: This test case is unique because it tests whatsAtPos on a invalid position that is out of bounds. The spot being checked is beyond the right boundary of the board. Function name: test_whatsAtPos_tilde_off_board_right

char whatsAtPos(BoardPosition pos)

Input: State:  pos.getRow = 0 pos.getCol = -1	Output: ~ State: unchanged	Reason: This test case is unique because it tests whatsAtPos on a invalid position that is out of bounds. The spot being checked is beyond the left boundary of the board. Function name: test_whatsAtPos_tilde_off_board_left

char whatsAtPos(BoardPosition pos)

Input: State:  pos.getRow = 5 pos.getCol = 0	Output: ~ State: unchanged	Reason: This test case is unique because it tests whatsAtPos on a invalid position that is out of bounds. The spot being checked is beyond the bottom boundary of the board. Function name: test_whatsAtPos_tilde_off_board_bottom

char whatsAtPos(BoardPosition pos)

Input: State: (0–4 × 0–4, all empty) pos.getRow = -1 pos.getCol = 0	Output: ~ State: unchanged	Reason: This test case is unique because it tests whatsAtPos on a invalid position that is out of bounds. The spot being checked is beyond the upper boundary of the board. Function name: test_whatsAtPos_tilde_off_board_top

isPlayerAtPos Tests

char isPlayerAtPos(BoardPosition pos, char player)

Input: State: (0–4 × 0–4, x at row 2, col 2) pos.getRow = 2 pos.getCol = 2 player = x	Output: true State: unchanged	Reason: This test case is unique because it tests isPlayerAtPos on a valid position on the board with player at the position. Function name: test_isPlayerAtPos_yes

Jeffrey Black
CPSC 2150

char isPlayerAtPos(BoardPosition pos, char player)

Input: State: pos.getRow = 2 pos.getCol = 2 player = x	Output: false State: unchanged	Reason: This test case is unique because it tests isPlayerAtPos on a valid position on the board with a different player at the position. Function name: test_isPlayerAtPos_no_different_player

	0	1	2	3	4
0					
1					
2			y		
3					
4					

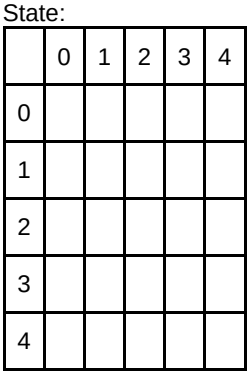
char isPlayerAtPos(BoardPosition pos, char player)

Input: State: pos.getRow = 2 pos.getCol = 2 player = x	Output: false State: unchanged	Reason: This test case is unique because it tests isPlayerAtPos on a valid position on the board with no player at the position. Function name: test_isPlayerAtPos_no_empty

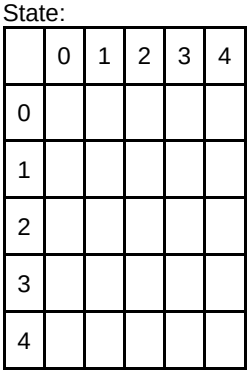
	0	1	2	3	4
0					
1					
2					
3					
4					

Jeffrey Black
CPSC 2150

char isPlayerAtPos(BoardPosition pos, char player)

Input: State:  pos.getRow = 2 pos.getCol = 5 player = x	Output: false State: unchanged	Reason: This test case is unique because it tests isPlayerAtPos on an invalid position (invalid column) Function name: test_isPlayerAtPos_no_invalid_column

char isPlayerAtPos(BoardPosition pos, char player)

Input: State:  pos.getRow = 5 pos.getCol = 2 player = x	Output: false State: unchanged	Reason: This test case is unique because it tests isPlayerAtPos on an invalid position (invalid row) Function name: test_isPlayerAtPos_no_invalid_row

Jeffrey Black
CPSC 2150

placeMarker Tests

void placeMarker(BoardPosition pos, char player)

Input:
State:

	0	1	2	3	4
0					
1					
2					
3					
4					

pos.getRow = 2
pos.getCol = 2
player = x

Output:
State:

	0	1	2	3	4
0					
1					
2			x		
3					
4					

Reason:
This test case is unique because it tests placeMarker on an empty position on the board.

Function name:
test_placeMarker_x_valid_position

void placeMarker(BoardPosition pos, char player)

Input:
State:

	0	1	2	3	4
0					
1					
2			y		
3					
4					

pos.getRow = 2
pos.getCol = 2
player = x

Output:
State:

	0	1	2	3	4
0					
1					
2			y		
3					
4					

Reason:
This test case is unique because it tests placeMarker on a space occupied by another player the board.

Function name:
test_placeMarker_x_occupied_different

Jeffrey Black
CPSC 2150

void placeMarker(BoardPosition pos, char player)

Input:
State:

	0	1	2	3	4
0					
1					
2			x		
3					
4					

pos.getRow = 2
pos.getCol = 2
player = x

Output:
State:

	0	1	2	3	4
0					
1					
2			x		
3					
4					

Reason:
This test case is unique because it tests placeMarker on a space occupied by the same player the board.

Function name:
test_placeMarker_x_occupied_same

void placeMarker(BoardPosition pos, char player)

Input:
State:

	0	1	2	3	4
0					
1					
2					
3					
4					

pos.getRow = 0
pos.getCol = 5
player = x

Output:
State:

	0	1	2	3	4
0					
1					
2					
3					
4					

Reason:
This test case is unique because it tests placeMarker on a position with a column that is out of bounds.

Function name:
test_placeMarker_x_out_of_bounds_col

void placeMarker(BoardPosition pos, char player)

Input:
State:

	0	1	2	3	4
0					
1					
2					
3					
4					

pos.getRow = 5
pos.getCol = 0
player = x

Output:
State:

	0	1	2	3	4
0					
1					
2					
3					
4					

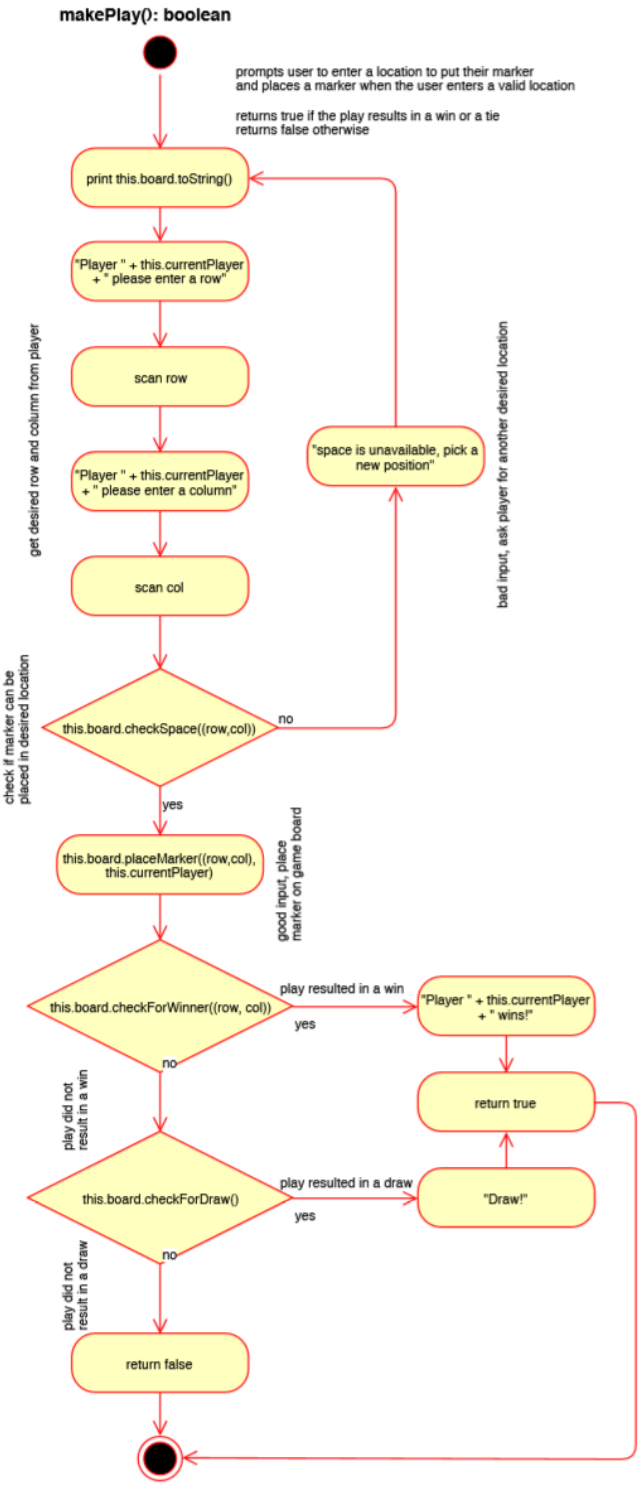
Reason:
This test case is unique because it tests placeMarker on a position with a row that is out of bounds.

Function name:
test_placeMarker_x_out_of_bounds_row

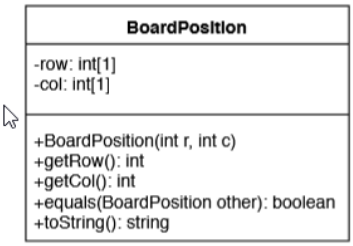
Jeffrey Black
CPSC 2150

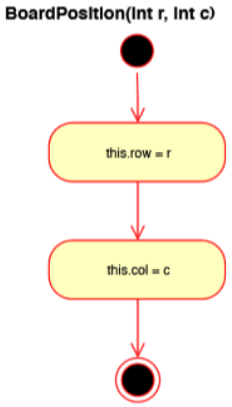
GameScreen
-players:list<Character>[MIN_PLAYERS..MAX_PLAYERS] -currentPlayer: Character[1] {must be a member of players} -board: GameBoard[1] +MAX_PLAYERS:int[1] = 10 +MIN_PLAYERS:int[1] = 2
+main(): void -nextPlayer(): void -makePlay(): boolean

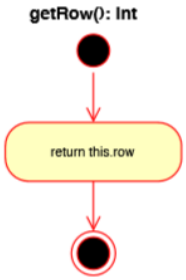
Jeffrey Black
CPSC 2150

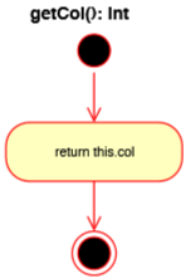


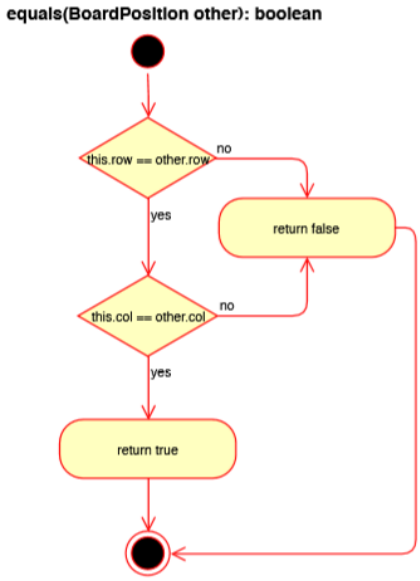
Jeffrey Black
CPSC 2150

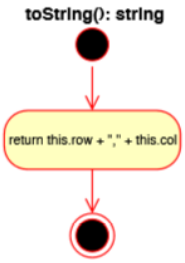




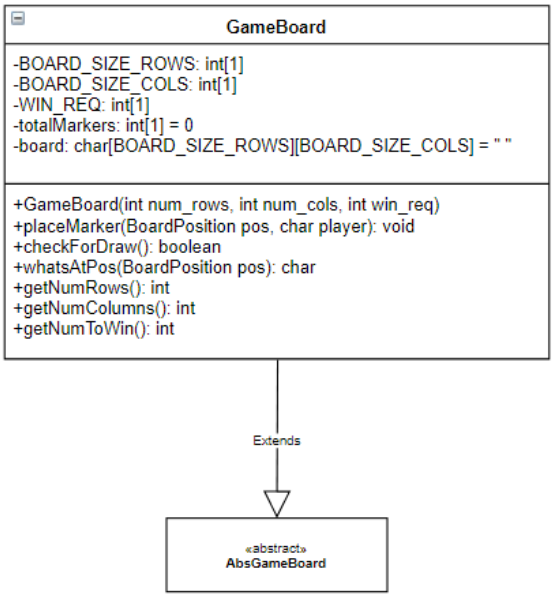


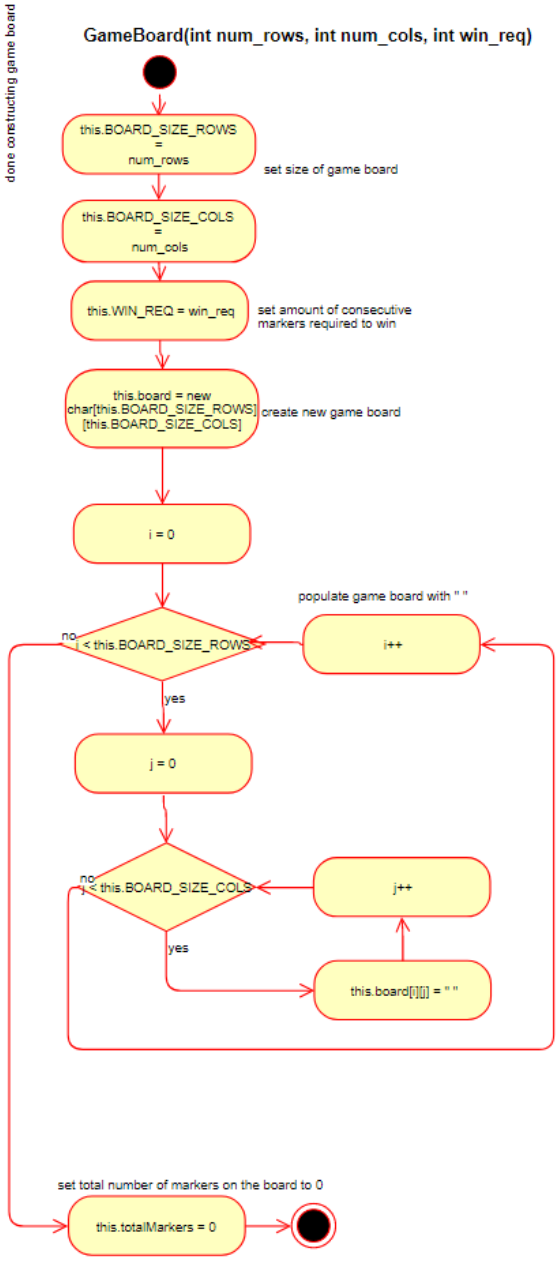


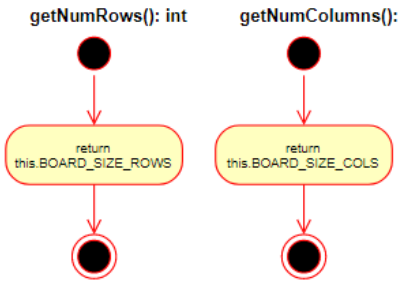


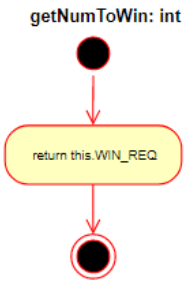


Jeffrey Black
CPSC 2150



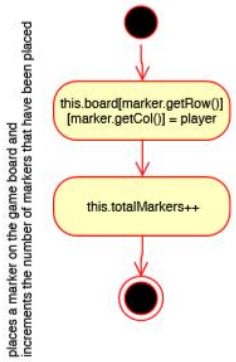




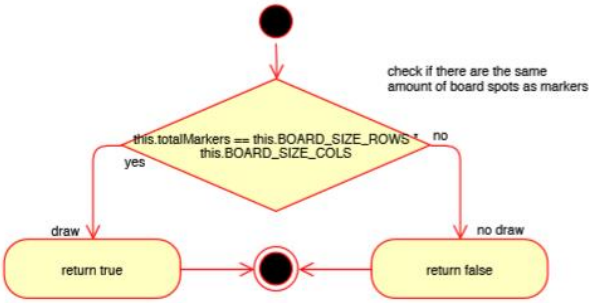


Jeffrey Black
CPSC 2150

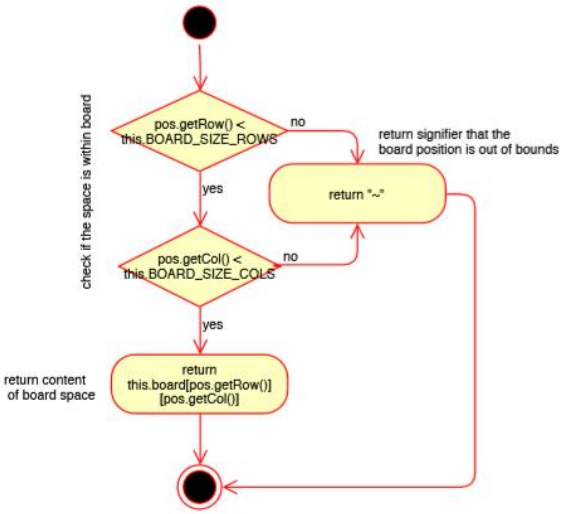
placeMarker(BoardPosition marker, char player): void



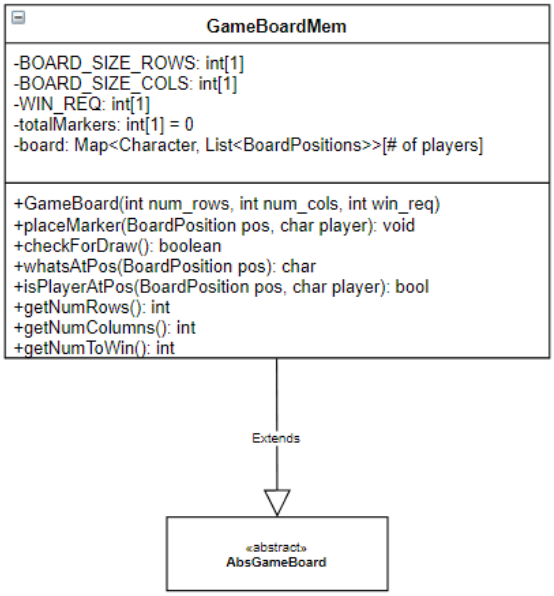
checkForDraw(): boolean

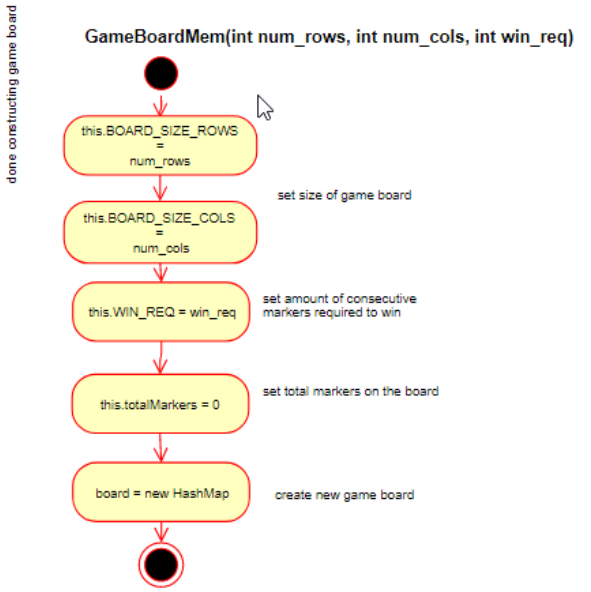


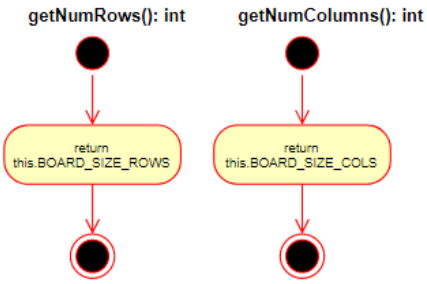
whatsAtPos(BoardPosition pos): char

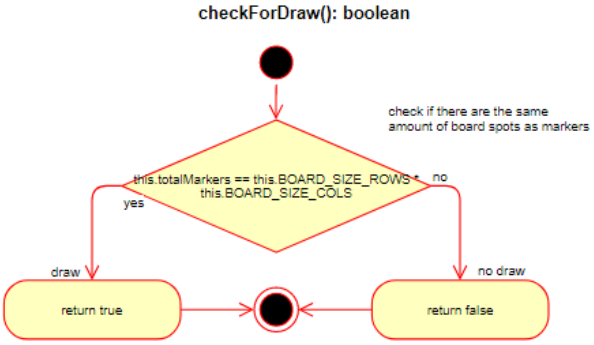


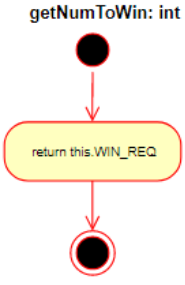
Jeffrey Black
CPSC 2150





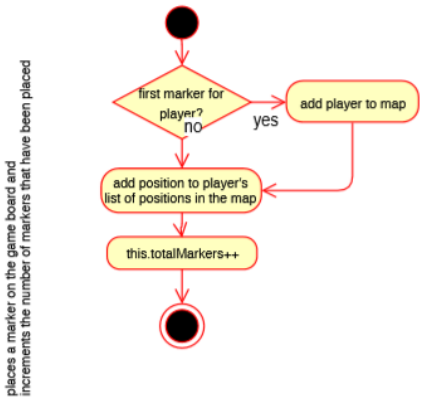




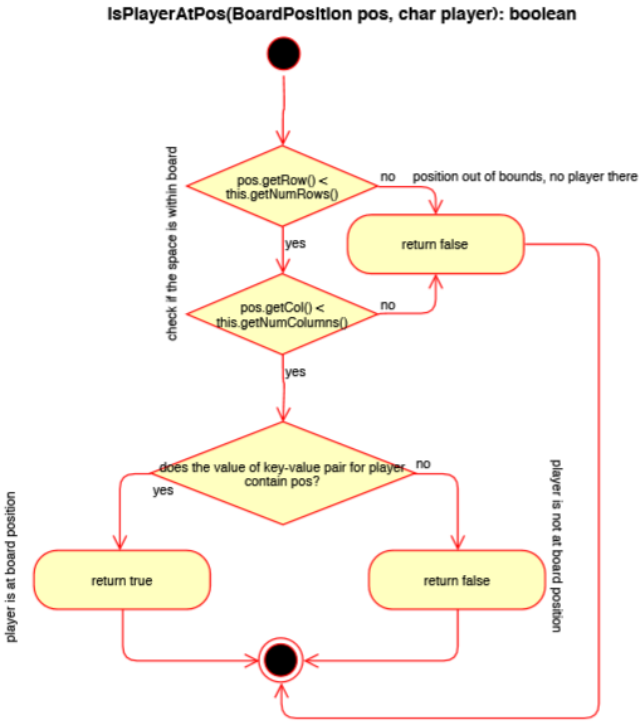


Jeffrey Black
CPSC 2150

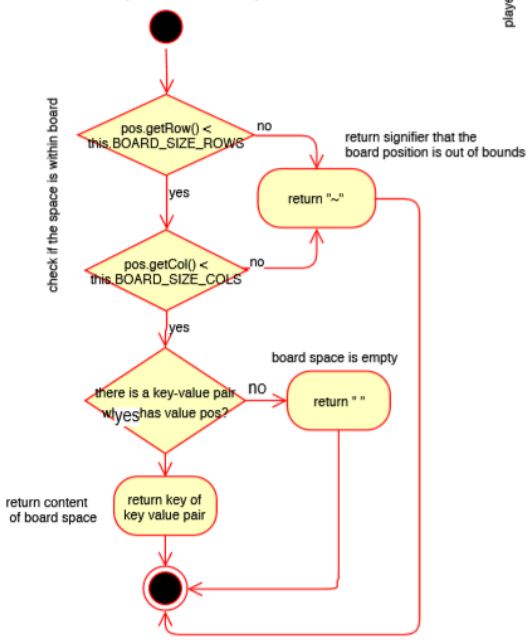
placeMarker(BoardPosition marker, char player): void



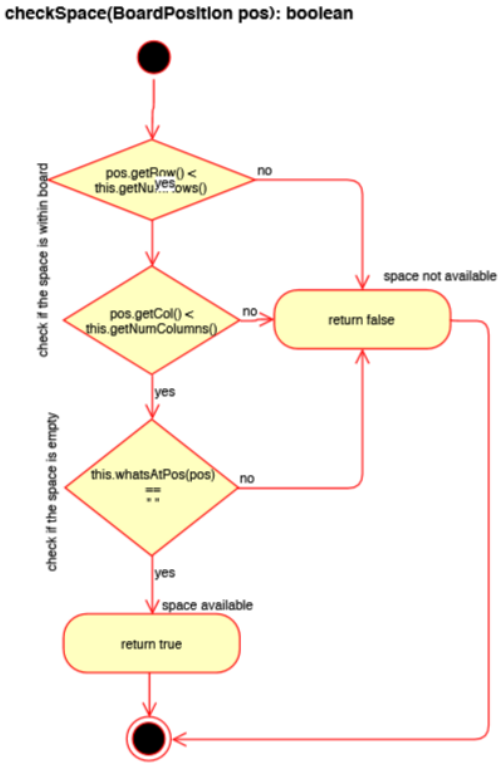
isPlayerAtPos(BoardPosition pos, char player): boolean

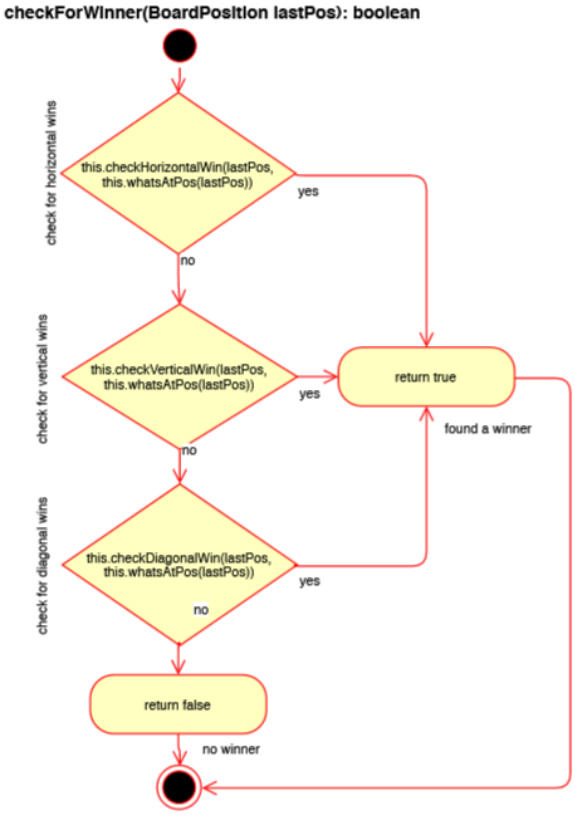


whatsAtPos(BoardPosition pos): char

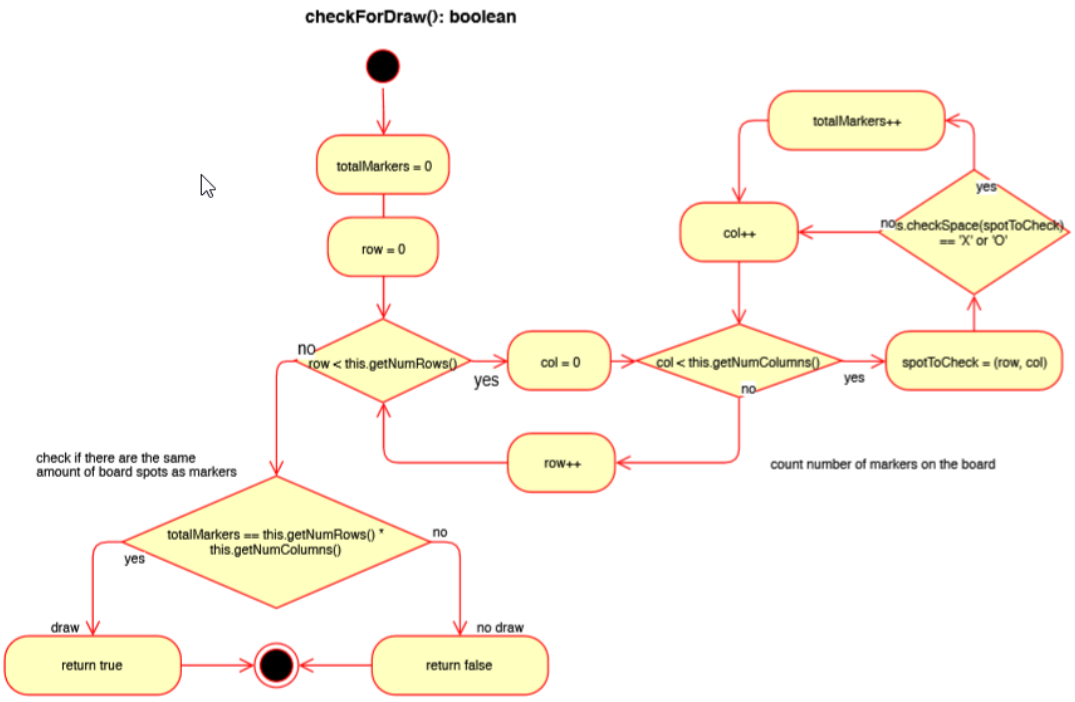


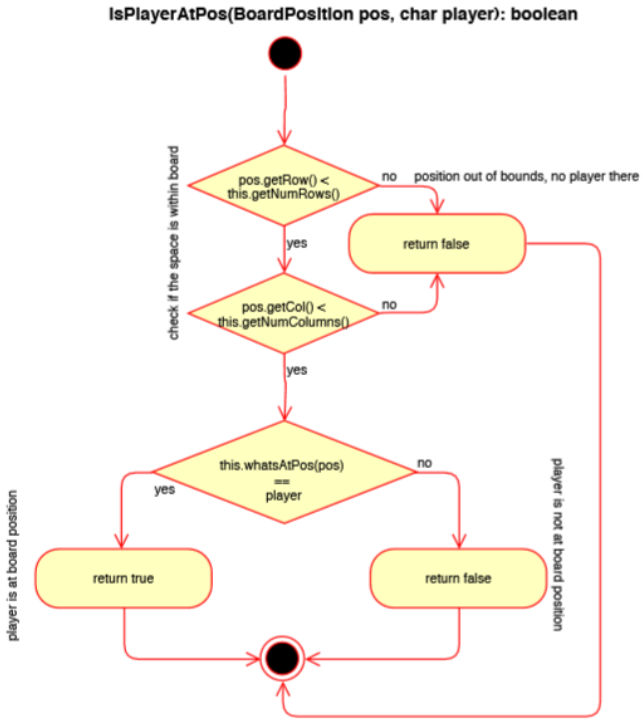
Jeffrey Black
CPSC 2150



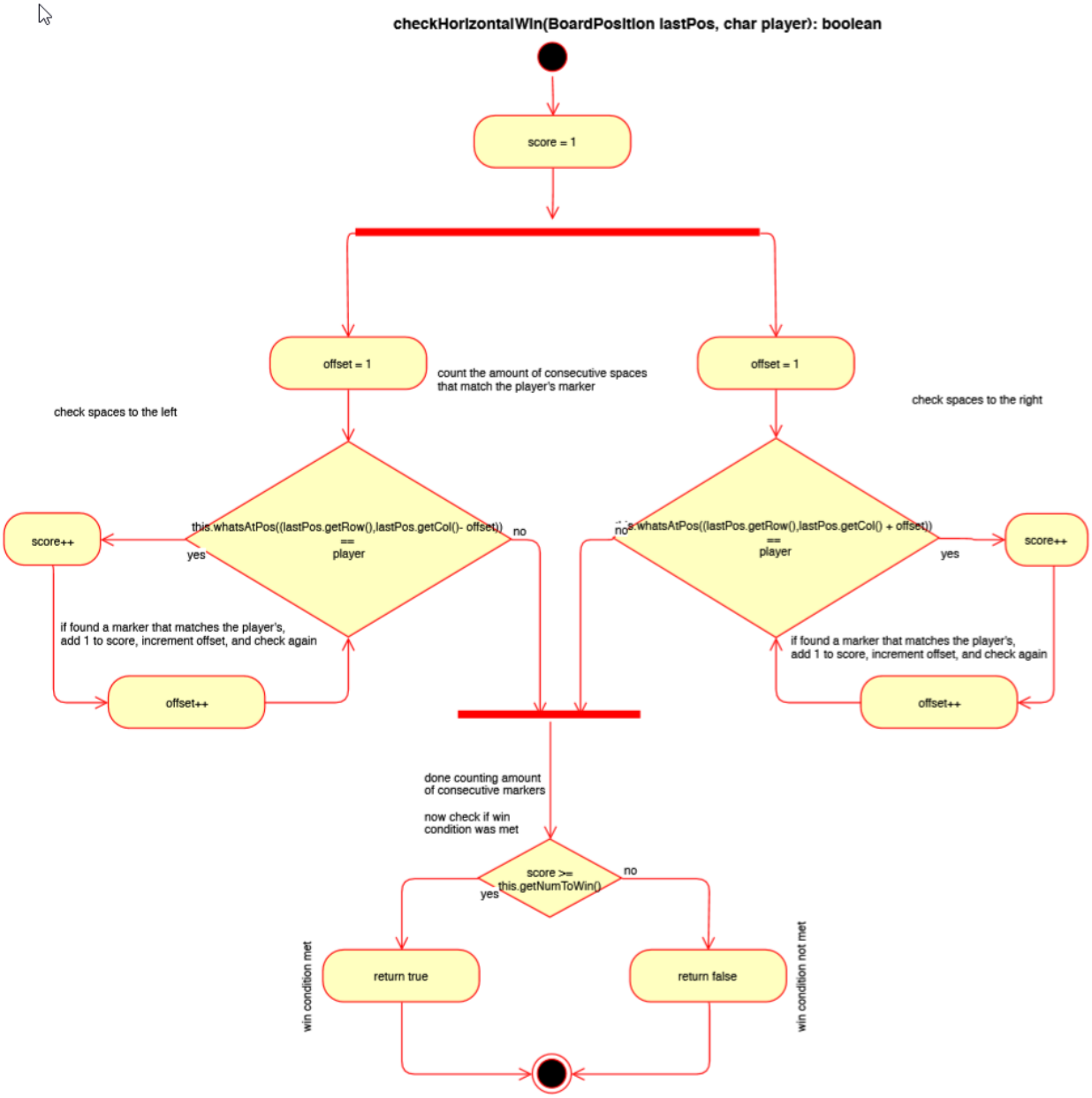


Jeffrey Black
CPSC 2150

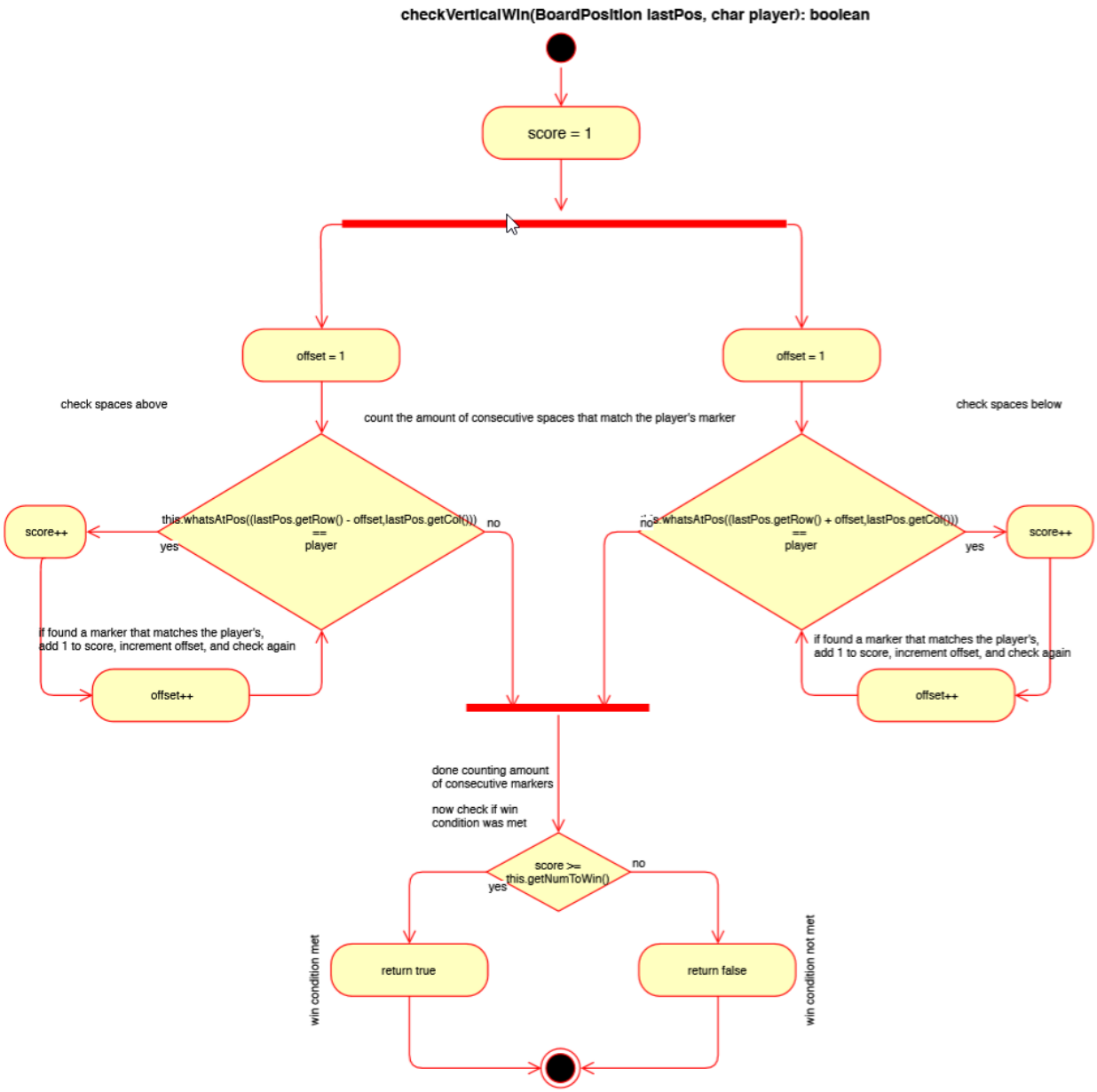




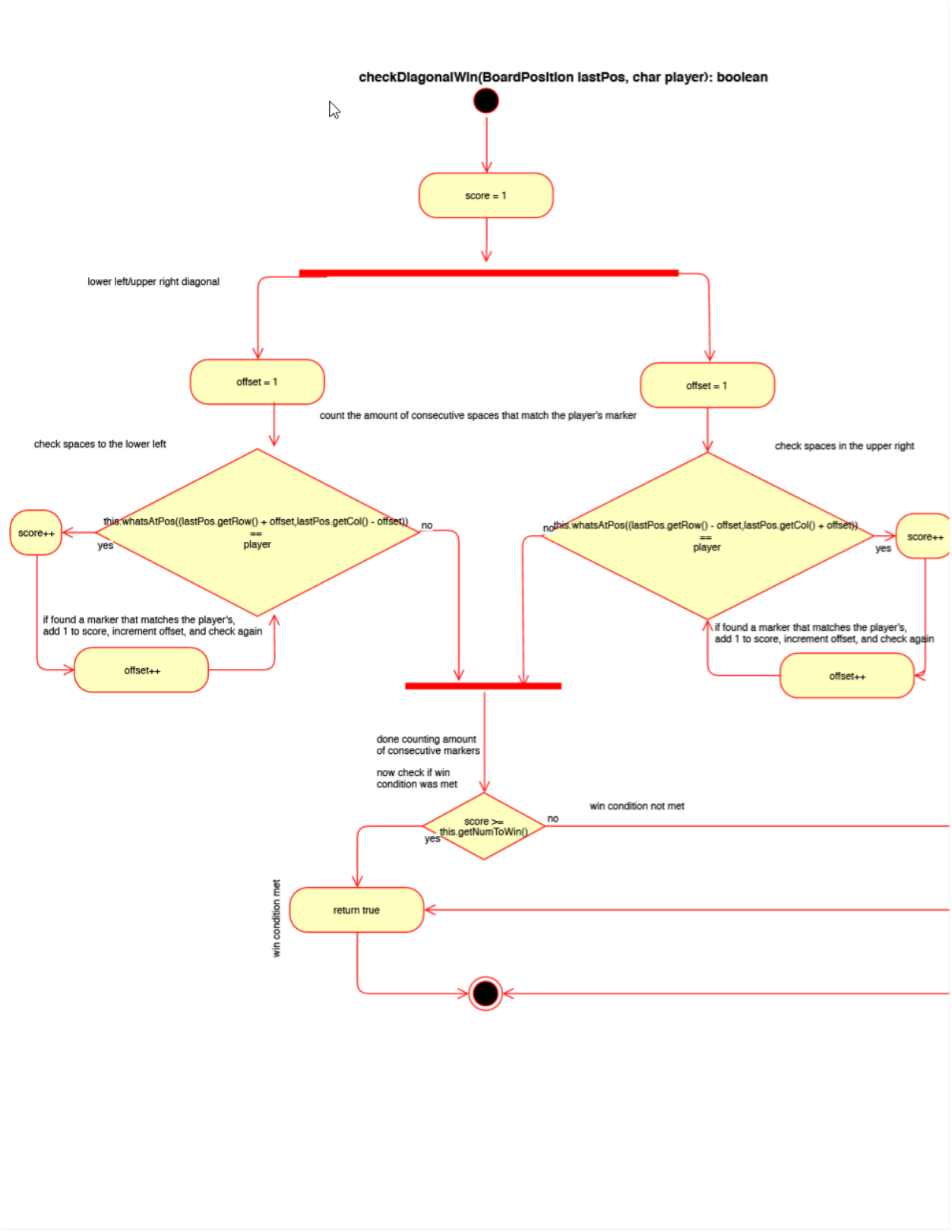
Jeffrey Black
CPSC 2150



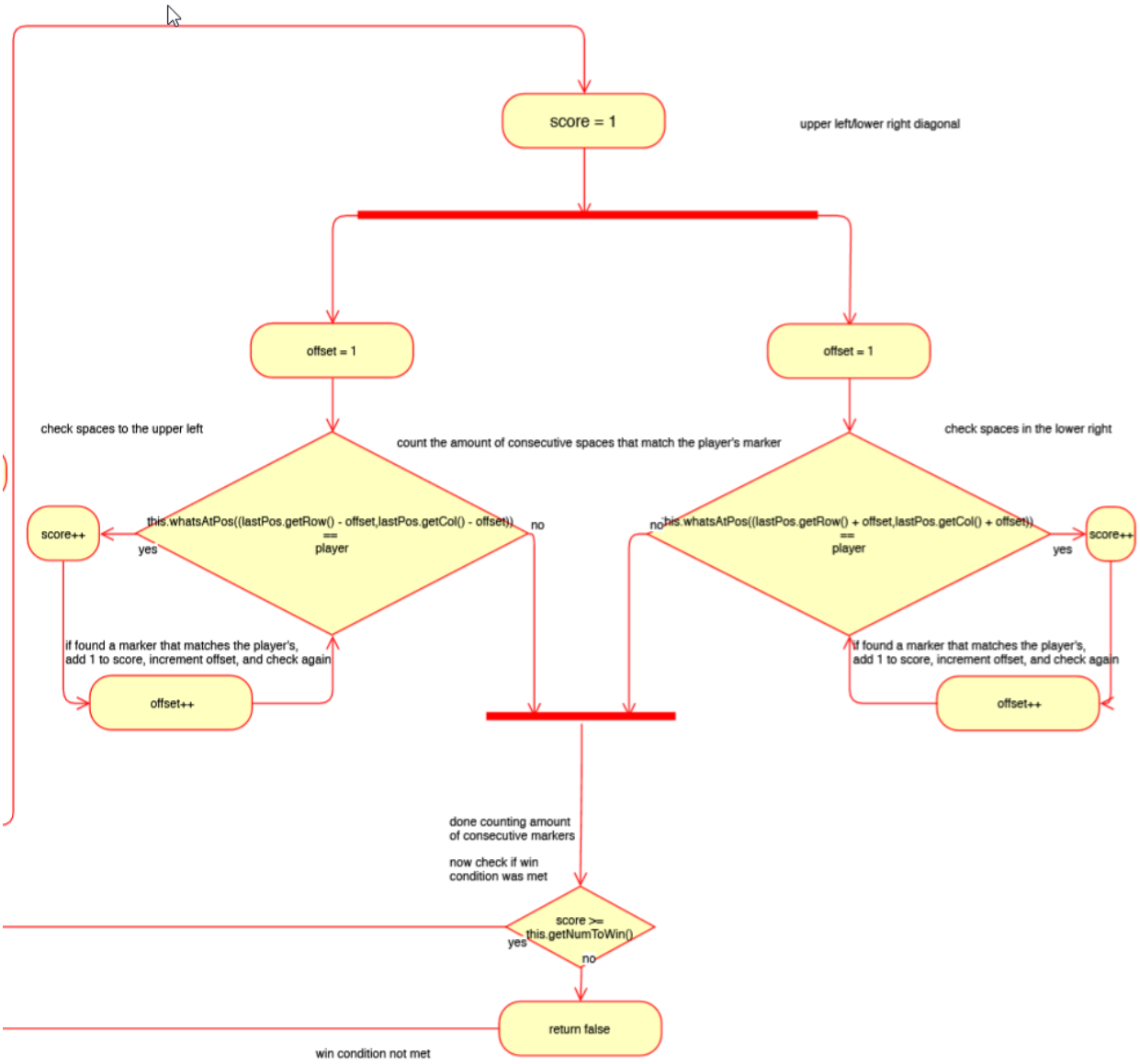
Jeffrey Black
CPSC 2150



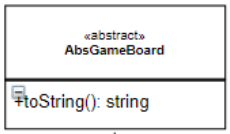
Jeffrey Black
CPSC 2150

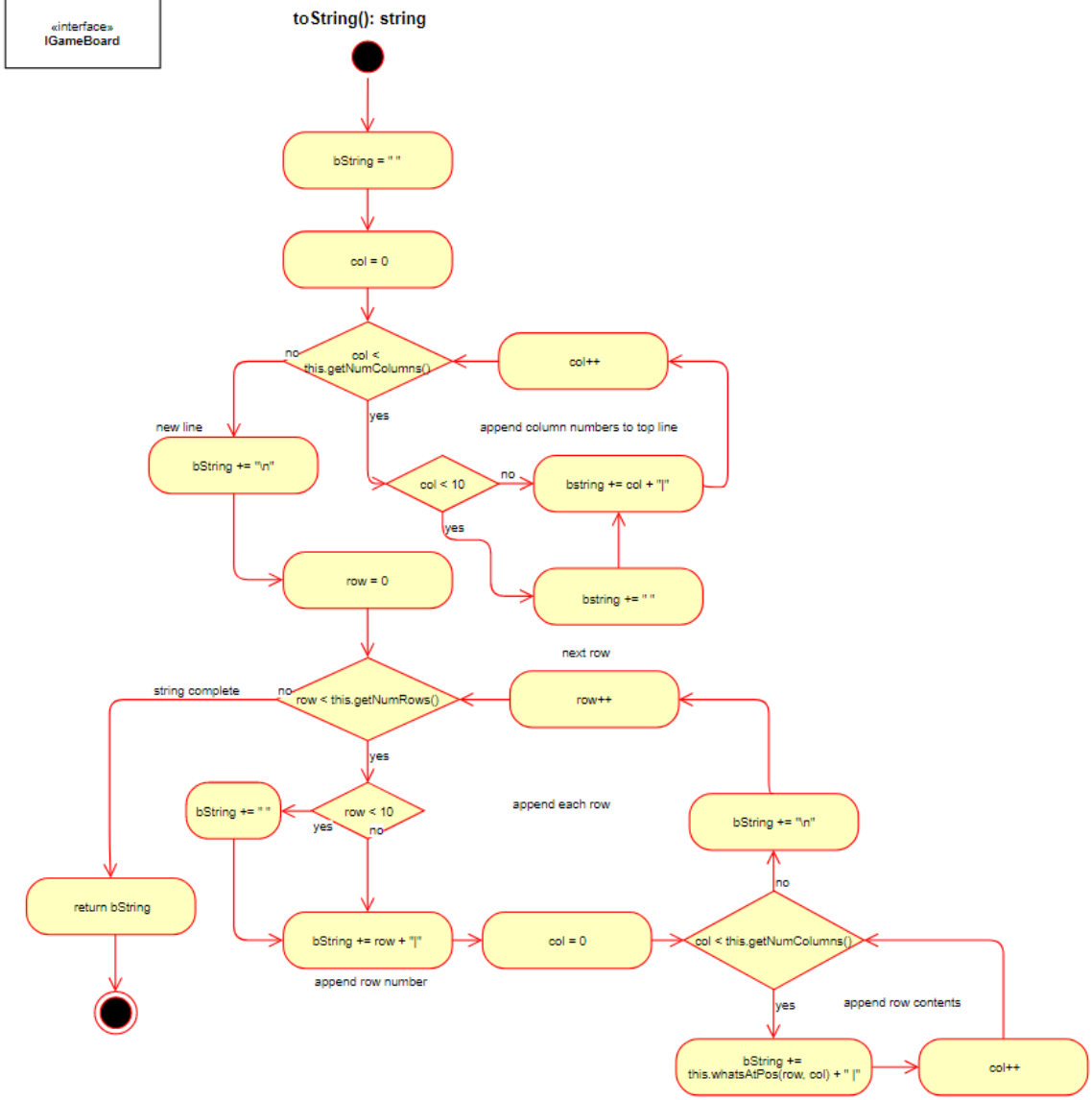


Jeffrey Black
CPSC 2150



Jeffrey Black
CPSC 2150





Jeffrey Black
CPSC 2150

TicTacToeController
-MAX_PLAYERS: int[1] = 10 -MIN_PLAYERS: int[1] = 2 -screen: TicTacToeView[1] -players: List<Character>[MIN_PLAYERS...MAX_PLAYERS] -currentPlayer: Character[1] -curGame: IGameBoard[1] -gameOver: boolean[1] = false
+TicTacToeController() +processButtonClick(int row, int col): void -newGame():void

processButtonClick(int row, int col): void