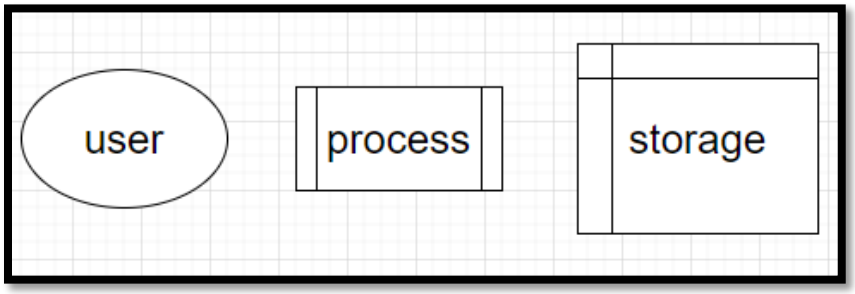
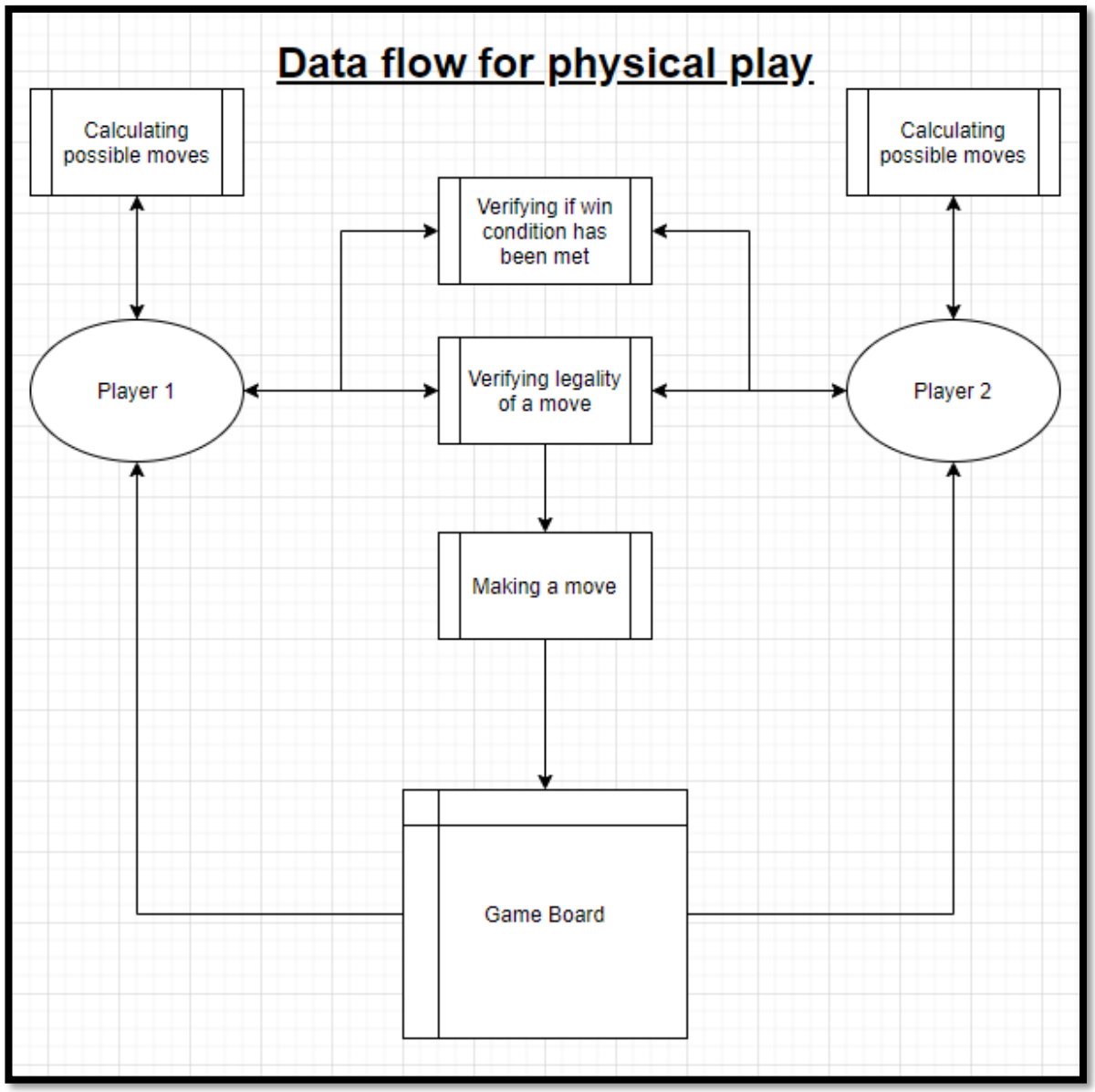
Design

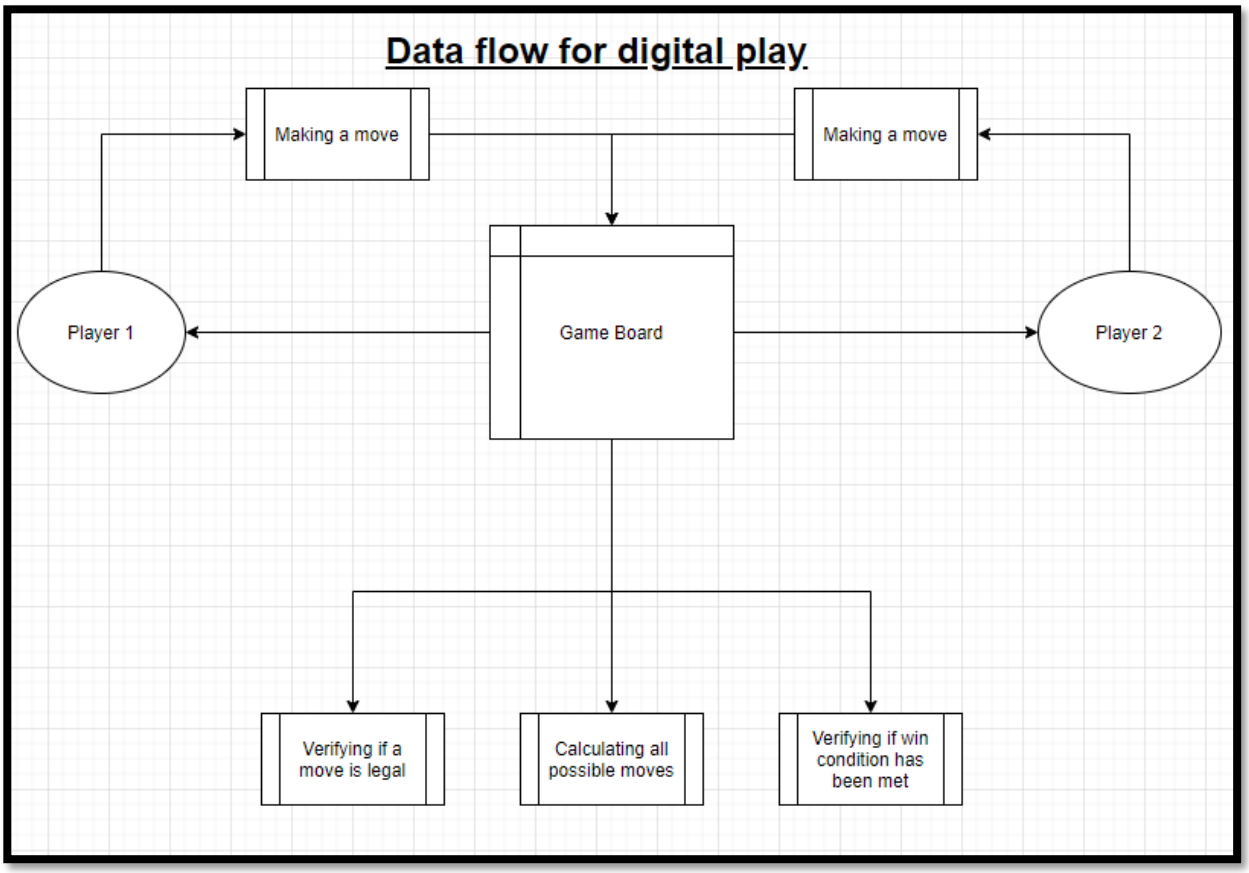
Data Flow

Chess is an information perfect game, so an automated change does not significantly change where data flows however it does handle a significant amount of verification which is normally done by the players.

Data Flow key



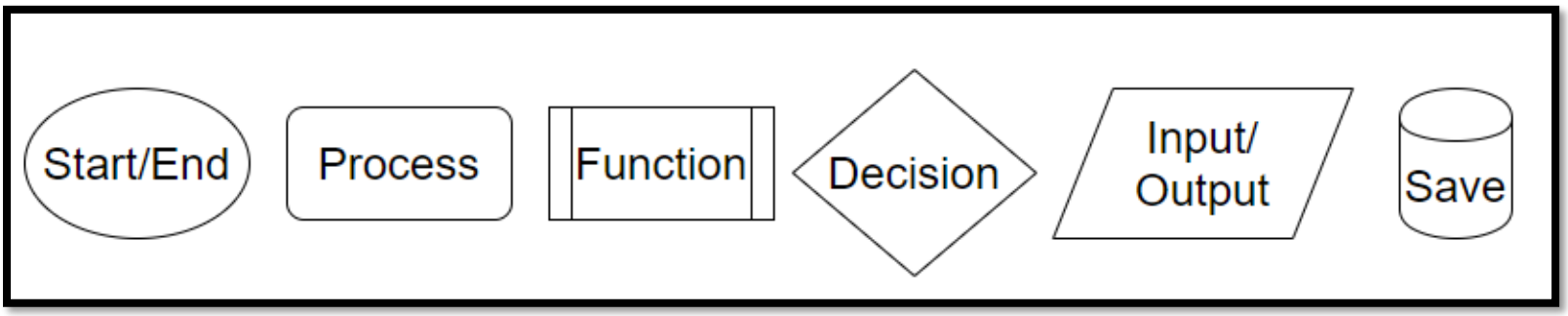




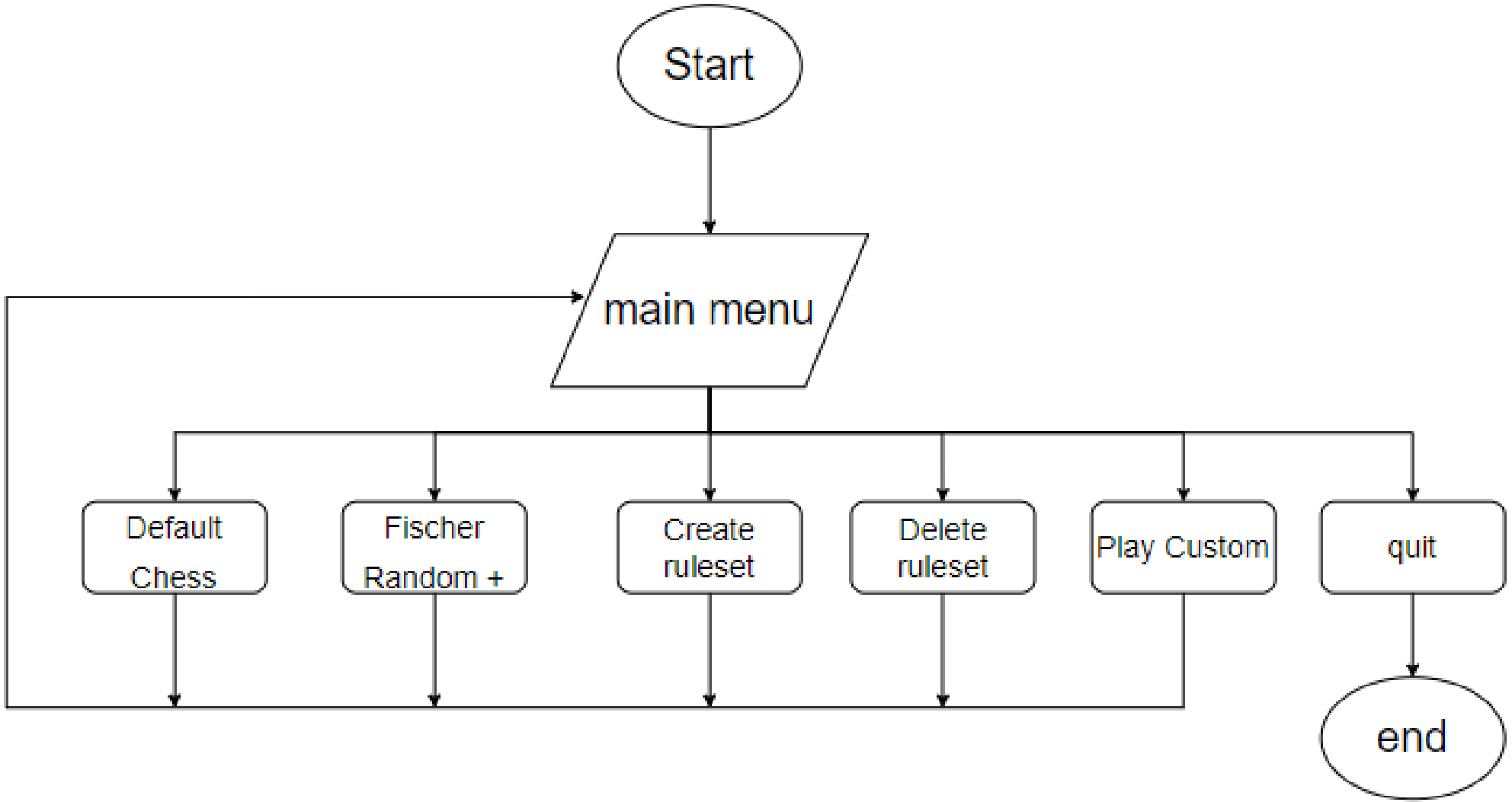
In the first diagram, the players must communally enforce the rules of the games, a process which, especially when playing variants of chess, is susceptible to human error. In the second diagram, the system handles all the verification processes and presents it to the player through the game board allowing them to focus purely on making intelligent moves and to not worry about the legality of their opponents moves. Essentially, what was formerly a three-way system between both players and the game board has been replaced with a one to one relationship between each user and the game board.

Flowcharts

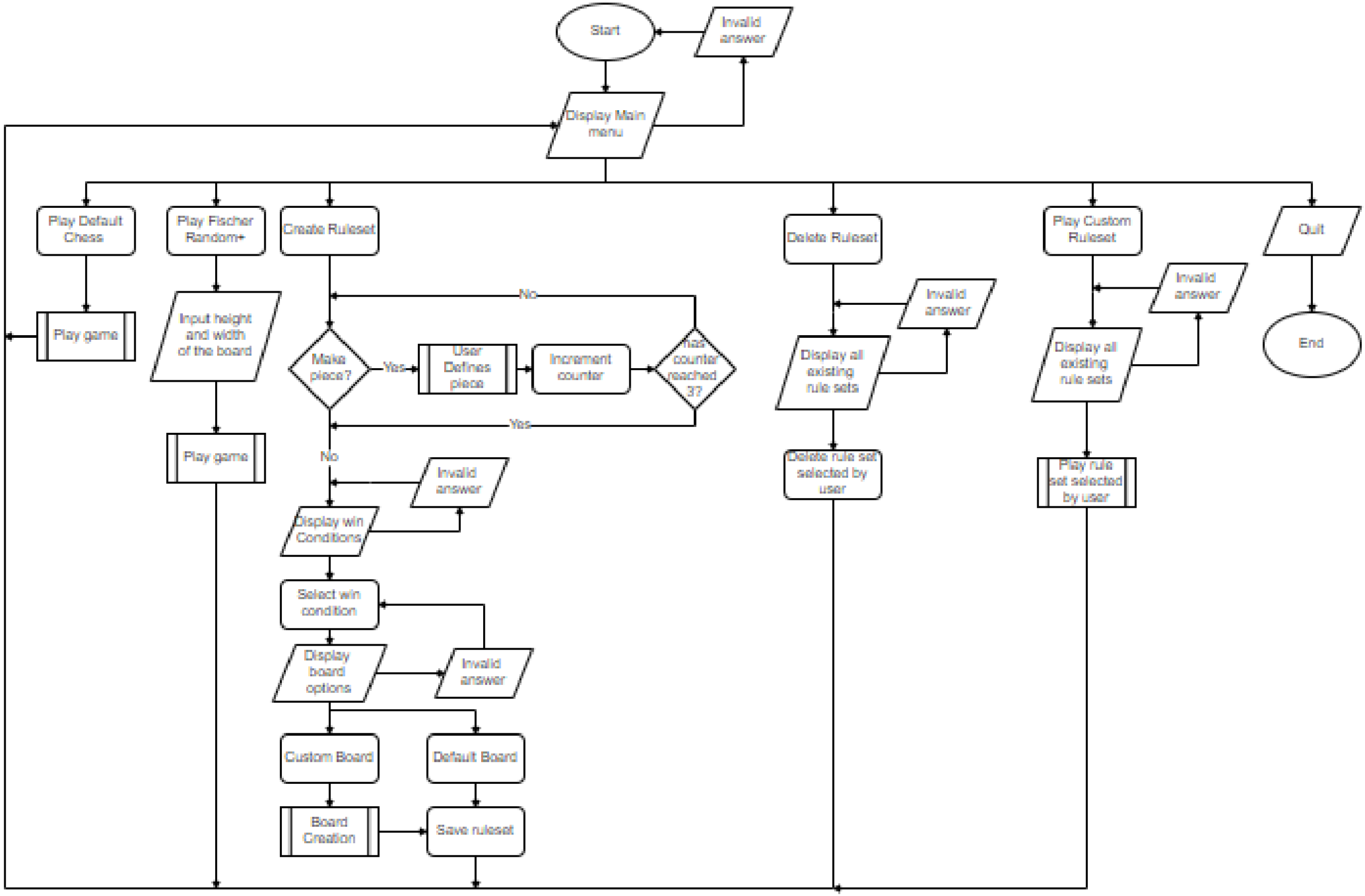
Flowchart Key



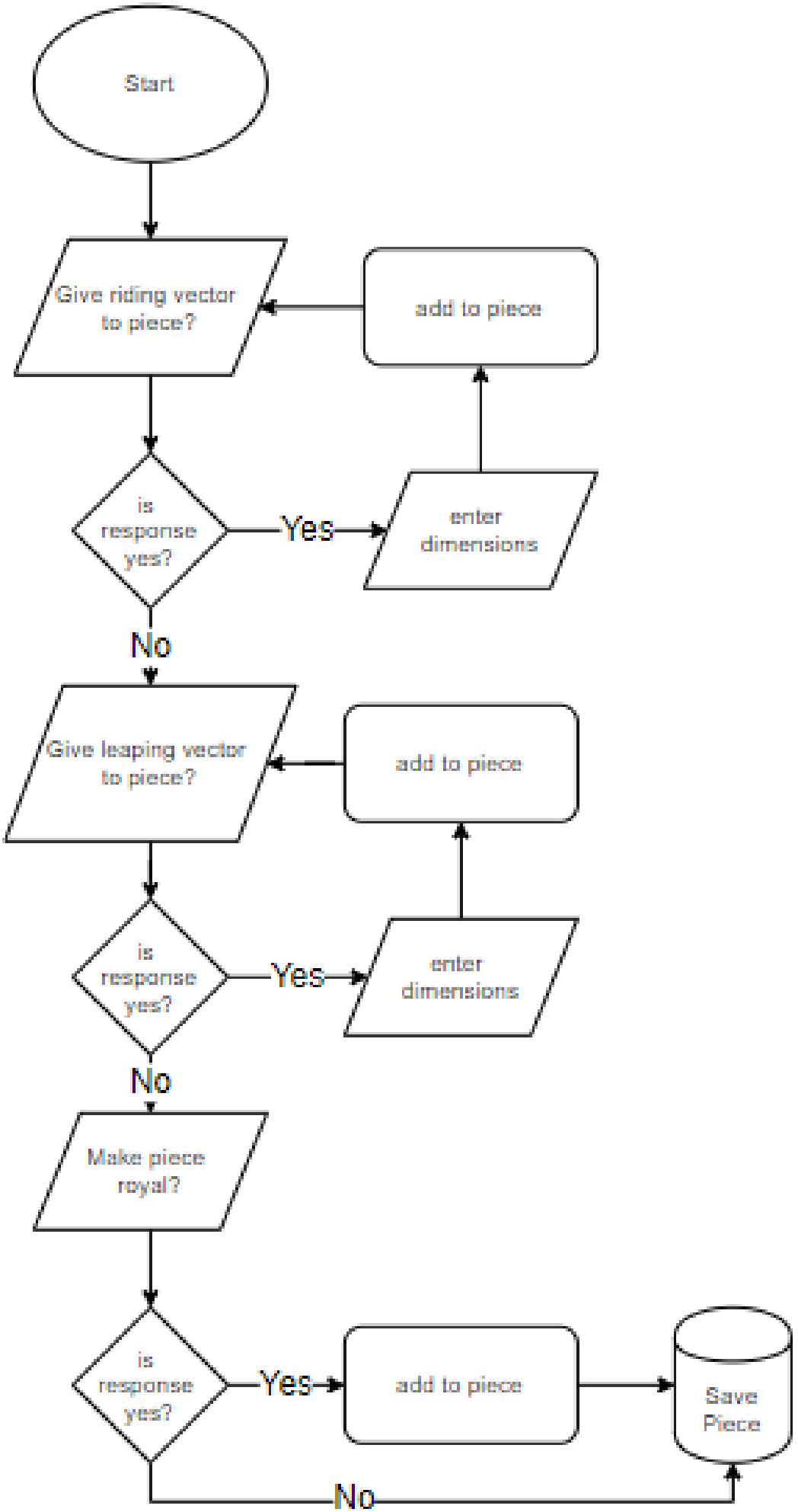
Main Flowchart: Level 0



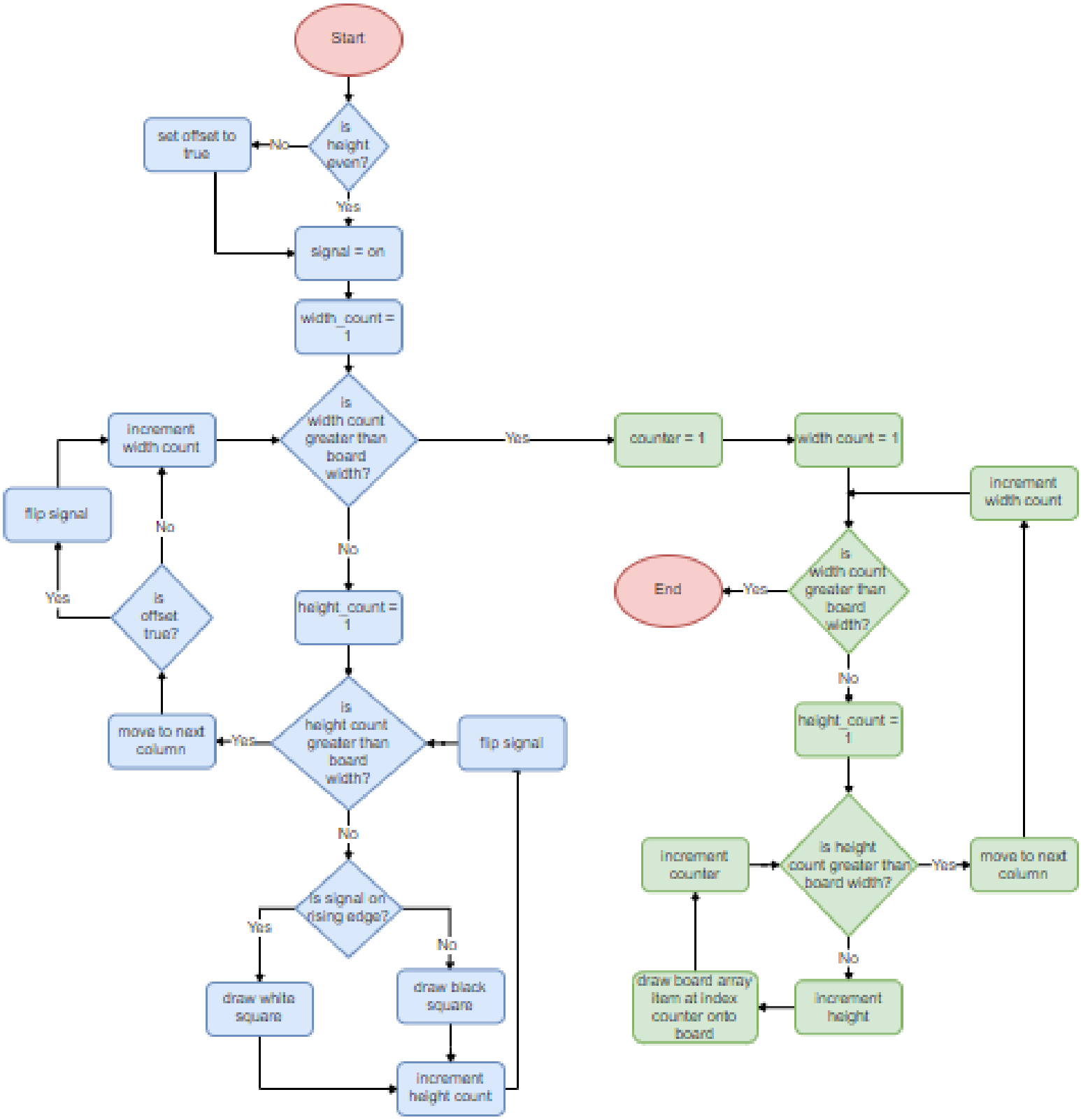
Main Flowchart: Level 1



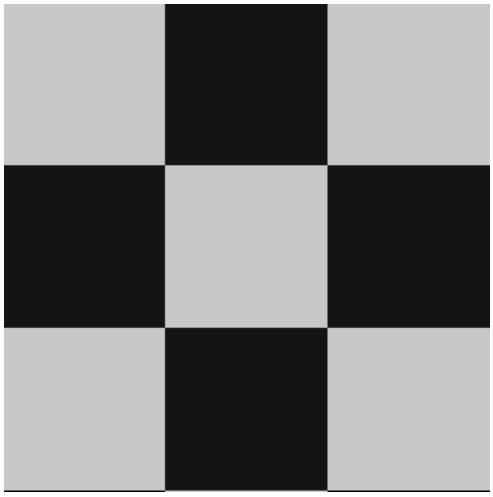
Piece Design



Draw Board


Start
is height even?
No
set offset to true
Yes
signal = on
width_count = 1
is width count greater than board width?
Yes
counter = 1
width count = 1
increment width count
No
height_count = 1
is height count greater than board width?
Yes
flip signal
is offset true?
Yes
No
move to next column
Yes
flip signal
No
Is signal on rising edge?
Yes
No
draw white square
draw black square
increment height count
End
Yes
is width count greater than board width?
No
height_count = 1
increment counter
is height count greater than board width?
Yes
move to next column
No
draw board array item at index counter onto board
increment height

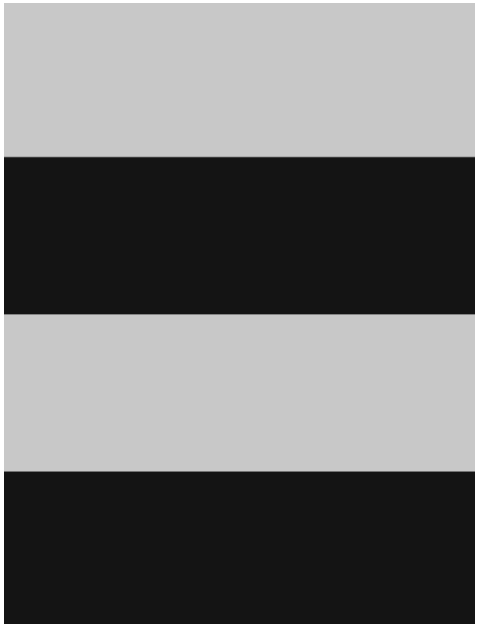
The Blue section represents the part of the algorithm which draws the empty board. The goal of this is to produce a board with alternating black and white squares. For example:



The function draws the board column by column, alternating the square colour each time.

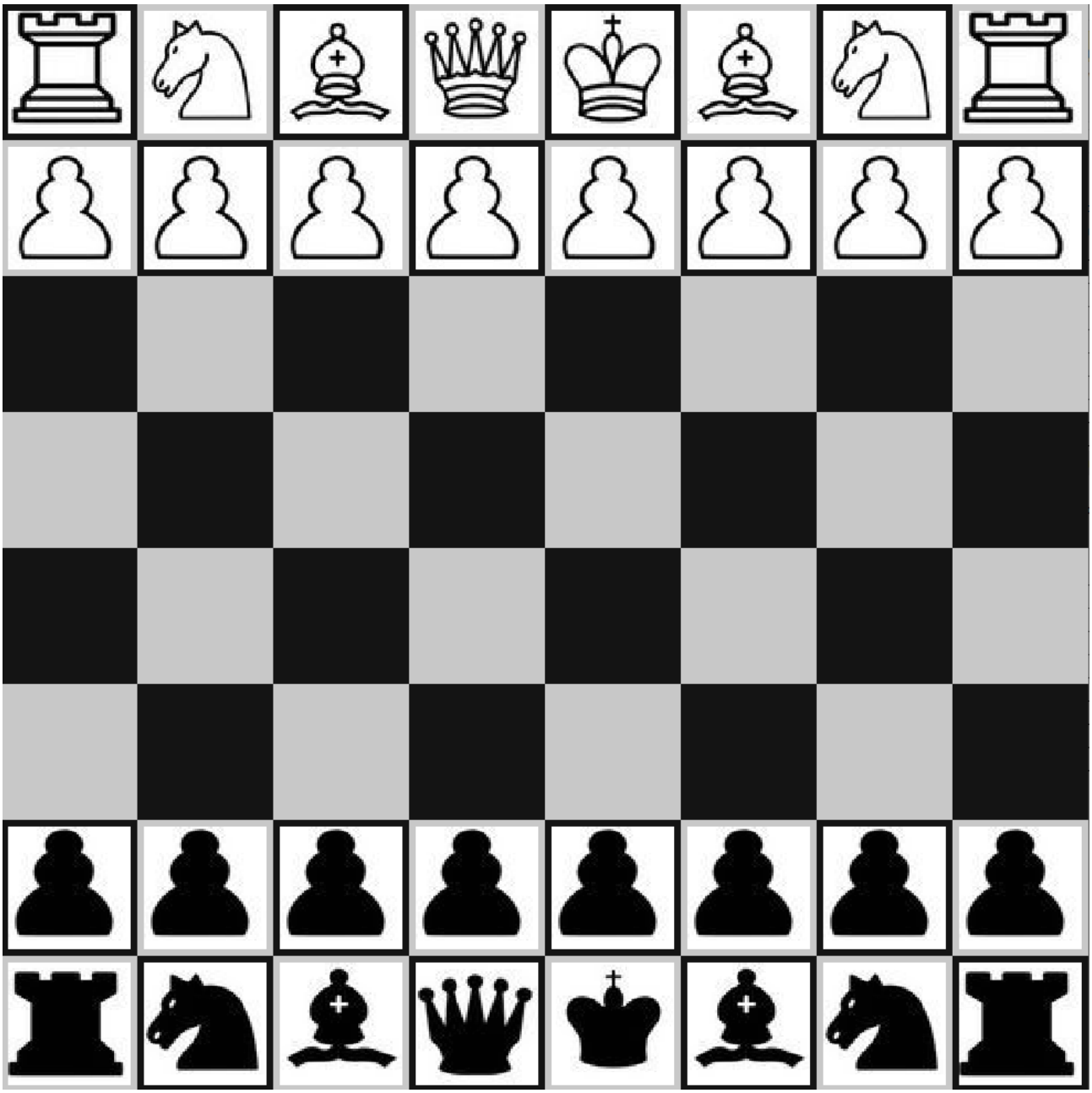
For boards with an odd height value, it is enough to have the colour change every time a square is drawn. This is because, if we were to order all squares sequentially, the even ones would all be one colour and the odd ones the other. When represented as a 2D grid, this also works because the final square of each column will be the same colour as the first square. This means that the next column starts with a different colour from the previous column. Therefore, every square is effectively offset by 1 colour, resulting in the alternating effect.

However, for boards with an even height value, the colour of the square about to be drawn at the start of each column will be the same, resulting in this.

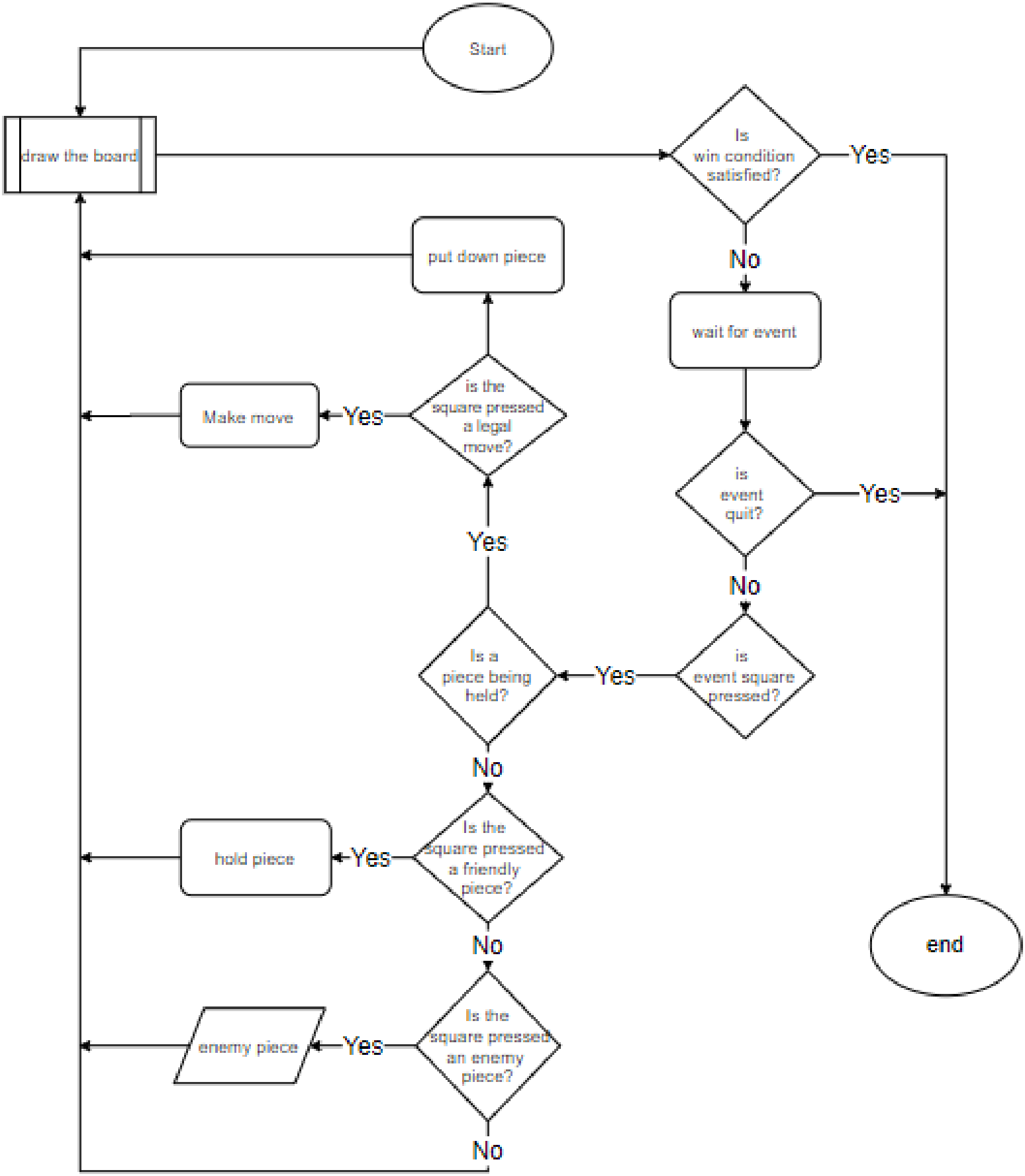


Evidently the alternating pattern is lost hence an artificial offset needs to be introduced to achieve the desired result.

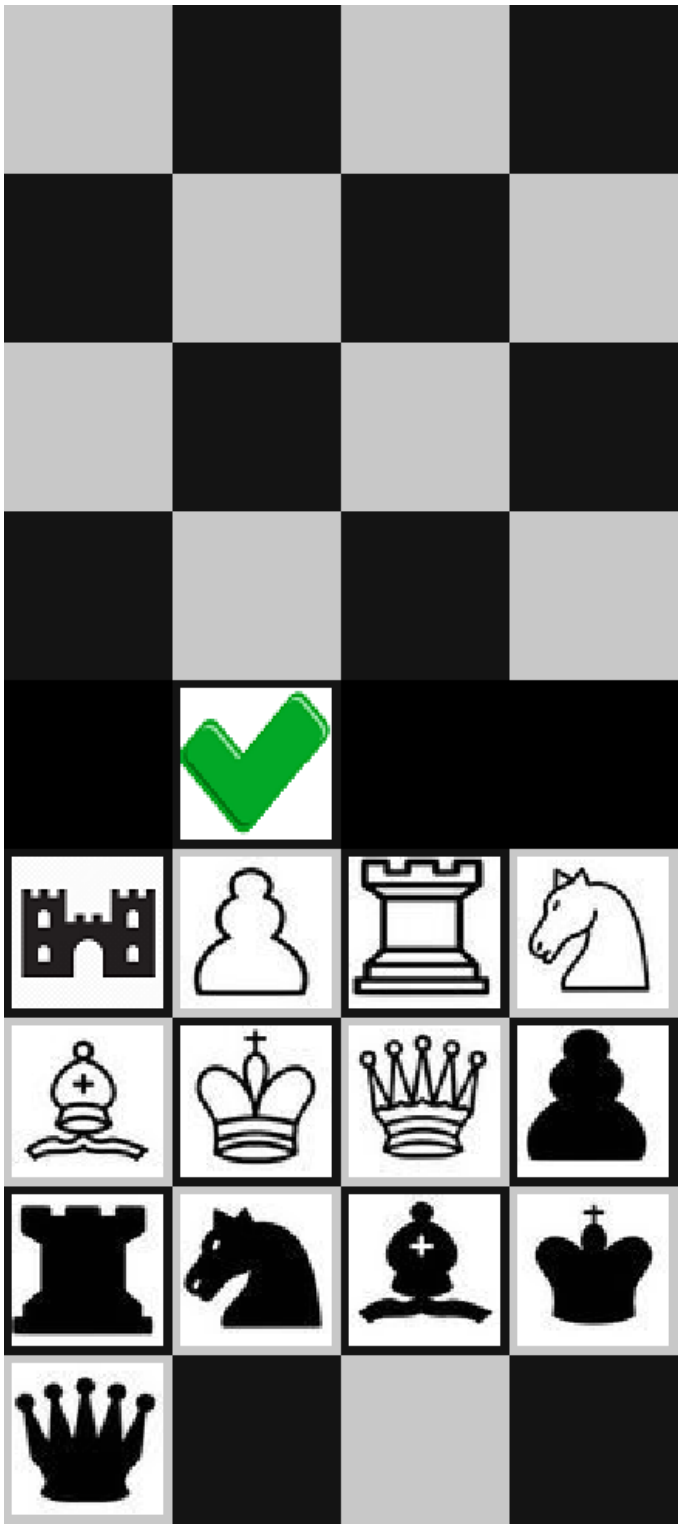
The green section represents the pieces being drawn onto the board. I intend to store the pieces in a 2-dimensional array that mirrors the board. So, the algorithm will use the piece's location in the array to determine where on the board to place it. The result will look like this.



Game Loop


Start
draw the board
Is win condition satisfied?
Yes
No
end
put down piece
wait for event
Make move
Yes
is the square pressed a legal move?
is event quit?
Yes
Yes
No
Is a piece being held?
Yes
is event square pressed?
No
hold piece
Yes
Is the square pressed a friendly piece?
No
enemy piece
Yes
Is the square pressed an enemy piece?
No

The next Flowchart shows the creation loop. It is event driven. The interface will contain the board, the pieces the user is able to select from and a tick button.



The board refers to the section above the tick button and everything else is the creative tools. So, in this case a 4 by 4 board has been specified.

Creation loop