Question 4
Does it look like there is any advantage to playing first if both players choose moves randomly? Answer in the space below.

No. The difference in the number of wins is not statistically significant.

Part 2 – Implementing Strategies

In this part, we will be implementing search strategies in order to improve a war of life playing agent's performance. They already have one strategy: `random`, which chooses any piece randomly and moves it randomly. The question is: can we implement any strategy which out-performs this one?

We will look at four strategies:

Bloodlust:
This strategy chooses the next move for a player to be the one which (after Conway's crank) produces the board state with the fewest number of opponent's pieces on the board (ignoring the player's own pieces).

Self Preservation:
This strategy chooses the next move for a player to be the one which (after Conway's crank) produces the board state with the largest number of that player's pieces on the board (ignoring the opponent's pieces).

Land Grab:
This strategy chooses the next move for a player to be the one which (after Conway's crank) produces the board state which maximises this function: Number of Player's pieces – Number of Opponent's pieces.

Minimax:
This strategy looks two-ply ahead using the heuristic measure described in the Land Grab strategy. It should follow the minimax principle and take into account the opponent's move after the one being chosen for the current player.

In your file `my_wol.pl`, write five predicates:

```
bloodlust(+PlayerColour, +CurrentBoardState, -NewBoardState, -Move).
self_preservation(+PlayerColour, +CurrentBoardState, -NewBoardState, -Move).
land_grab(+PlayerColour, +CurrentBoardState, -NewBoardState, -Move).
minimax(+PlayerColour, +CurrentBoardState, -NewBoardState, -Move).
```

These predicates will implement the four strategies described above by choosing the next move for a player. They will all do the same thing: choose the next move for the player. `PlayerColour` will either be the constant `b` for blue or `r` for red. The `CurrentBoardState` will be the state of the board upon which the move choice is going to be made. The predicate will produce a `NewBoardState`, in the usual representation (pair of lists) which will represent the board state after the move, but before Conway's crank is turned. The predicate will also return the `Move` that changed the current to the new board state. This will be represented as a list [r1,c1,r2,c2] where the move chosen is to move the piece at co-ordinate (r1, c1) to the empty cell at co-ordinate (r2, c2).