MiniMax Algorithm and AlphaBeta Pruning Post Mortem

Successful

- AI code
    - Minimax was implemented and can handle long games with the help of the depth variable
    - Alpha beta pruning was successful
    - Implemented a randomize function IF the computer plays second AND the player places a piece in the middle when they are playing first. It chooses one of the four corners of the board so the AI doesn't select the same corner every single time. The minimax function will be called on the computers next turn.
- Other code
    - Customization allows for custom player pieces

Unsuccessful

- Coding without planning
    - It took a while for me to understand how to code the MiniMax recursion as I didn't write out any pseudo code to make sure I was including everything I needed to code.
- Forgetfulness when coding
    - I spent 7hrs wondering why my AlphaBeta Pruning wouldn't work due to a hard-coded value I placed in a parameter for the MiniMax code causing it to fail every time. I probably implemented AlphaBeta pruning successfully in 5 different ways without realising because of it.

Lessons Learned

- Minimax is a great way to code AI
- Pruning can cut out a lot of computation time (tested by using counters where there were over 60000 moves less with the pruning)
- Pseudo code helps a lot when coding confusing functions such as MiniMax and AlphaBeta Pruning
- Learnt how to make sure the user doesn't input anything while the program is calling Sleep. (Computer will sleep for a second so it seems like they are making a decision)
- Commenting while you code and including function headers when you create a new function can save a lot of time and confusion rather than including it when you are finished with the code