## 1.1 Problem Description– 605 million adults in the world play chess regularly. People are attracted to chess as it is a highly complex game with more possible board states than atoms in the universe. Chess is entertaining to play as it requires you to think and look at what could happen after you make a given move in order to evaluate that move over others. This is an entertaining passtome and there is clearly a demand for it as one of the largest websites chess.com has over 77 million users who play chess online.

There is clearly a large demand for online chess programs that allow users to play chess against a computer. This is because these programs solve the problem of providing entertainment.

My chess program will have a web interface in which users can play chess against the computer. The chess engine will be written in python and will run on a web server that hosts the website.

The goal is not to create the best chess engine possible but instead to maximize the user's experience and available features for a casual game to provide more entertainment. For example the user will be able to **set the difficulty** as they choose to allow them to start playing even when they are not well practiced at chess or to seek a challenge if that is what they want.

I have also asked my Dad about the features he likes in a chess program as he plays chess daily on his phone for entertainment. He said he would appreciate it if you could see the last move the computer made. He also said being able to set up your own custom starting position would be fun. For example, starting with 8 pawns and 7 queens and a king against the hardest difficulty setting. This feature could also be easily adapted so that a variety of game states could be stored in a library of puzzles and the game states are all 1 or 2 moves from checkmate. In this way I aim to make a user friendly chess program aimed at casual players for entertainment that includes features they may want to make it more entertaining than other online chess games.

## 1.2 Stakeholders

In this project the only stakeholder is the User as how the project turns out will affect how entertaining a game it is for them to play. My user demographic is 8+ as chess is a highly complex game that mostly only older children and adults play. However there are no restrictive elements to it, like gore in a game for example, so a child who wants to play chess is able to; they are just unlikely to be interested and as such are not the target audience.

Therefore the stakeholders are people of age 8+ who are looking to play chess for entertainment.

Note my program could in theory be used to practice chess and improve in skill level but this is not the main purpose.

## 1.3 Why can the problem be solved by computational methods

**Thinking Ahead:**

**Abstraction:**

This is the process of identifying and removing unnecessary details and information from a problem. This allows me to get a better understanding of the core or crux of the problem and reduces complexity. This makes it more likely that my solution will successfully solve the problem and reduces unnecessary complexity in my code.

**How can I use abstraction:**

· **Previous moves abstracted:** The chess engine only needs to be given the current board state in order to determine the next best possible move. For example the history of the last few moves the user and then computer made are irrelevant. This is because all the information needed to make the next move is on the chess board and not hidden. The program will not analyze the users moves as it will assume they will play optimally to reduce complexity

· **Pieces are abstracted in the backend:** A chess piece can be abstracted as how it looks, its design and exact color are irrelevant. The only relevant attributes of a non-king piece is:

o **Who owns the piece:** Which player it belongs to

o **What possible moves can it make:** The set of vector

o **What is the relative value of each piece**

· **The board is abstracted:** The board can be abstracted to a coordinate grid in terms of x and y or vector space. The colors of the squares can also be ignored.

**Decomposition:**

The project will have 2 core parts:

* A chess engine that determines the best move that the computer can make in a chess game. This will be interacted with using shell commands or an API like a restful API or Websockets
* A user interface which provides the user with all necessary menus to select difficulty etcetera as well as a simple to use and intuitive interface for playing chess against the computer

Create and explain a diagram that breaks each of these (core features ownly) down into subcomponents see here for how to apply decomposition to the user interface:

<https://www.youtube.com/watch?v=RUusIWfNCak>

## 1.4 Research

Research into how chess engines are created:

Firstly to define terms I like wikipedia’s definition of a chess engine which is:

In [computer chess](https://en.wikipedia.org/wiki/Computer_chess), a **chess engine** is a [computer program](https://en.wikipedia.org/wiki/Computer_program) that analyzes [chess](https://en.wikipedia.org/wiki/Chess) or [chess variant](https://en.wikipedia.org/wiki/List_of_chess_variants) positions, and generates a move or list of moves that it regards as strongest.[[1]](https://en.wikipedia.org/wiki/Chess_engine#cite_note-1)

A chess engine is usually a [back end](https://en.wikipedia.org/wiki/Front_and_back_ends) with a [command-line interface](https://en.wikipedia.org/wiki/Command-line_interface) with no graphics or [windowing](https://en.wikipedia.org/wiki/Windowing_system).

(<https://en.wikipedia.org/wiki/Chess_engine> link also in sources)

Interestingly this wiki entry also mentions how GUI’s are often standardized in order to allow users to quickly understand how to play chess against a computer if they have already used similar websites. This is done using a touch screen or a keyboard input. As a result, free versions of a chess GUI are already available. This means that early iterations could focus on the chess engine by using one of these tools for the user interface. These include XBoard and ChessBase.

Chess is a game with more possible board states than atoms in the universe. This means that it is not a solved game.

An example of a solved game is tic tac toe as there are a small enough set of all possible combinations of future moves that you can represent them all in a tree where the arcs are moves and the nodes are game states. All the leaf nodes will be game states that lead to either a win, loss or draw. As a result it is possible to make optimal moves to guarantee a win or a draw.

Whereas with chess if you continued to expand your tree of all possible moves, resulting game states and all possible moves the tree would never finish expanding. As such with finite computing power a chess program cannot see the move that is guaranteed to lead to victory. In order to get around this chess programs either learn patterns of moves (strategies of a sort) by training in the case of machine learning.

It seems that there are 2 main strategies for creating a chess engine (for projects of varying scope)

Talk about game theory, assumptions of rationality, differences in scope e.g. neural networks require more expertise and resources. t

**Machine learning:**

**Decision trees:**

## 1.5 Features of the proposed solution - limitations

**Due to the potential size of this project it is important that prototypes with fewer features that solve a simplified version of the problem are developed first. Features should be added in order of importance.**

In my first prototype the chess engine will play a simplified game of chess (e.g. all pawns) and it will have a command line interface. I will then create a separate graphical user interface using C#. This will be my simplest version of the program.

Later prototypes will have a more advanced chess engine that will use the minimax algorithm and will play actual chess (minus castling and en passant which are special moves). It will have a web server interface that uses flask and a RESTFUL api. The user interface will be web based and a web server will be written in flask or node will host html and javascript that will form the user interface. It will include all necessary menus and the chessboard. It will include all necessary validation.

The final version will include a chess engine and a websocket server that forms a full duplex connection with the client to allow for moves to be made easily and quickly. Other information like leaderboard information and user information will be served over a restful API or a graphql API. The web interface should allow for a greater range of inputs while still being simple to use. For example users can play one move to checkmate puzzles or load a game they left off. Validation may be done on the backend via websockets. A javascript framework may be used to reduce the code complexity as the website grows in size. For example I may use VUE.js.

## 1.6 Software and hardware requirements

## 1.7 Success criteria