Project Proposal – Robert Lucas

# Concept

Chess is an ancient game played by millions around the world and has seen many apps made for competitive and casual play however these tend to offer only the default versions of chess or slight spinoffs that still play on the same sized board with the same pieces but with changes to the rules.

I plan to make a chess game with support for adding many game modes as well as multiplayer support, save game support and AI that will work with any new game mode allowing game modes to be released monthly or even weekly to keep players interested.

# Stakeholders

The stakeholders in this project are myself and the players as all decisions should be made to either make development more feasible for me or to improve the player experience as this will bring in more players and improve player retention.

# Computational methods used

Abstraction:

* The game will be 2D and not realistic simplifying it reducing development time and making the board easier to understand for a user
* The game will use Unity to handle most of the rendering, IO and packaging the game into an executable reducing development time
* After the multiplayer system is developed, the game logic will only call exposed functions on the networking classes without having to worry about handling networking allowing different sections of the solution to remain more independent allowing for easier iteration on different parts of the code
* IP addresses will be encoded to alphabetic strings to make them easier to remember and pass on to friends

Thinking ahead:

* As the game is symmetrical, code can be reused for both players decreasing development time
* Instead of writing a client and a client-server hybrid to allow one of the players to host, the client can be reused and a separate server can be made. The local client can then connect to the local server. This will reduce code duplication between a client and a client-server hybrid reducing development time and decreasing the chance of errors occurring due to mismatches between how a client and how a client-server works.
* The board can be serialised to save its state and this needs to be done in a smart way that can work for any game mode and any number of custom pieces with custom data as well as not breaking with future updates

Decomposition:

* The networking library, game logic, AI and visuals can all be developed separately as they only interact with each other in limited ways allowing the project to be easier to manage
* These subsystems are further divided into smaller chunks such as the networking being divided into client and server and the chess manager being broken down into the game, save and input system

Procedures:

* The game will use procedures such as:
  + The network library going from receiving data to validating it to processing it to passing it to the game logic for it to be applied to the board
  + The client clicking on a square to move a piece to the input manager receiving that to the game manager validating the move to the game manager applying the move to the game manager checking for a check or checkmate and finally the game manager updating the visuals manager

Logic:

* Logic will be used very frequently for example evaluating which moves a player can and can’t make or for not letting more players join a lobby once it is full both of which are essential for the game to work

Concurrency:

* The networking library will need to run concurrently to be able to receive and process data from other players independently of the game’s framerate
* The AI will need to run on a separate concurrently to the main thread as calculating the best move can take up to 30 seconds and the game shouldn’t freeze during this time as this can ruin the user experience

Divide and Conquer:

* AI will run on multiple threads at the same time with each thread searching different possible moves

Encapsulation:

* Input manager will encapsulate Unity’s input system to make it easier to use and adding features such as support for multiple key presses

Backtracking:

* The Mini-Max algorithm uses backtracking

Heuristics:

* The AI will use heuristics such as not going further down a path that has a very low score

# Features based on existing solutions

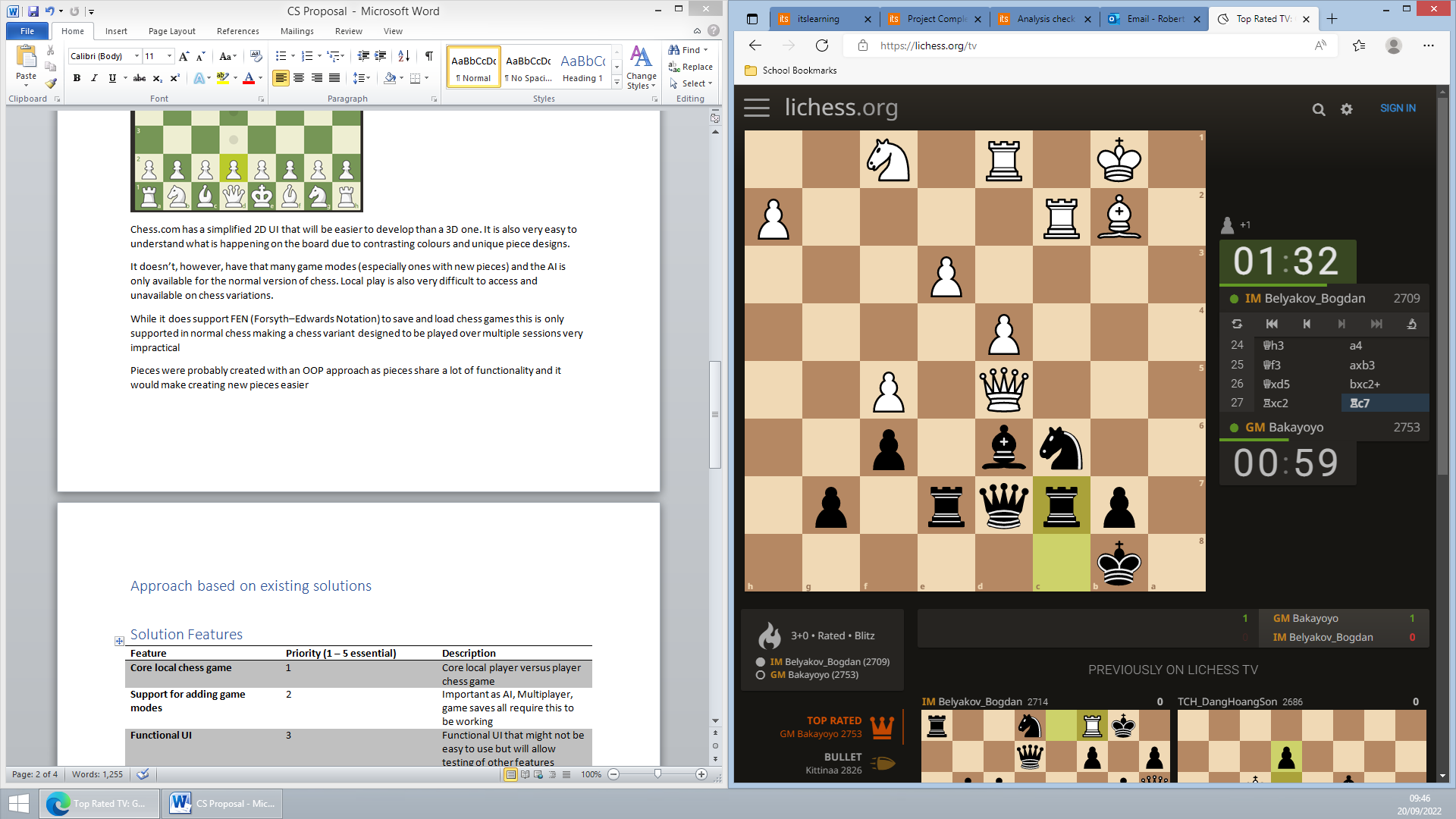


Chess.com has a simplified 2D UI that will be easier to develop than a 3D one. It is also very easy to understand what is happening on the board due to contrasting colours and unique piece designs.

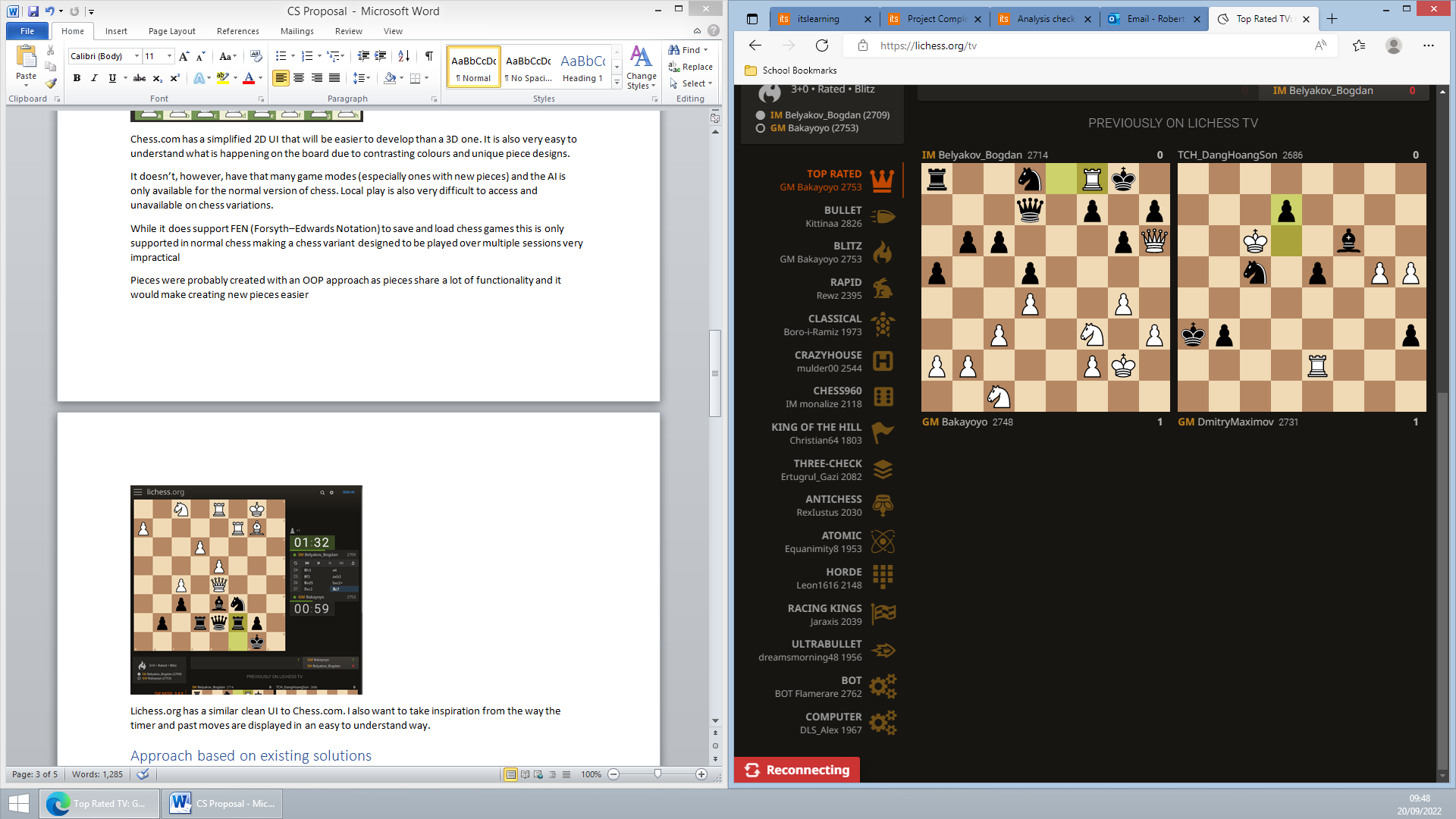
It doesn’t, however, have that many game modes (especially ones with new pieces) and the AI is only available for the normal version of chess. Local play is also very difficult to access and unavailable on chess variations.

While it does support FEN (Forsyth–Edwards Notation) to save and load chess games this is only supported in normal chess making a chess variant designed to be played over multiple sessions very impractical

Pieces were probably created with an OOP approach as pieces share a lot of functionality and it would make creating new pieces easier



Lichess.org has a similar clean UI to Chess.com. I also want to take inspiration from the way the timer and past moves are displayed in an easy to understand way.



Lichess does also have quite a few modes however, like Chess.com, Lichess doesn’t stray too far from normal chess and doesn’t support saving games in progress that can’t be represented with FEN



This is a generic app on the Play Store with the board represented in 3D. I find this representation to be more cluttered and more difficult to understand.

A 3D view would also require more developer time as models for custom game modes would have to be created and there are extra complications around camera controls and ensuring that the contrast is good enough on all the pieces to see them clearly

I do, however, like that when you click on a piece it shows you a preview of legal moves and believe that it would greatly improve the user experience

# Approach based on existing solutions

Most chess games will probably use an object-oriented approach to create the pieces as they need to share some functionality which can be provided by a parent class. I will do this and also use a similar approach for implementing multiple game modes as large amount of functionality will be shared between them

A lot of chess AIs use the minimax algorithm with alpha-beta pruning and I will use this as I have some experience with it. The minimax algorithm (excluding some optimisations) will also work with any game mode as it requests all possible moves from a piece which custom pieces can also provide. I might also use Zobrist hashing to create transposition tables for the board but this would be a massive increase in complexity, especially for custom game modes.

Sockets are commonly used for low-level and efficient communications and as I want very fine control over network communications I will be using that.

# Stakeholders (TODO)

# Solution Features

|  |  |  |
| --- | --- | --- |
| Feature | Priority (1 – 5 essential) | Description |
| Core local chess game | 1 | Core local player versus player chess game |
| Support for adding game modes | 2 | Important as AI, Multiplayer, game saves all require this to be working |
| Functional UI | 3 | Functional UI that might not be easy to use but will allow testing of other features |
| Multiplayer | 4 | Allow players to play with each other across the internet  Must support working with any future game modes |
| AI | 4 | An AI that can play any game mode |
| Save games | 5 | Allows player to save the current state of the game and load it later including loading it into a multiplayer game |
| Good UI | 6 | An intuitive UI that is easy for a new player to use, possibly with tutorials |
| Animations | 7 | Animations for pieces moving from square to square |
| Themes | 7 | Allow user to change board themes |

# Limitations

### Art

This game will require art assets for each piece as well as a logo, fonts and UI elements. Because of my limited knowledge in this field and time limitations I will use third-party assets for some components and simplistic elements for most other things

### Money

Third-party asset packs, animations packs, font packs and more cost money which has to be managed carefully as this project is on a small budget so in most cases free alternatives will be used

### Computational power

The AI can’t be too complex as the typical user system won’t have a powerful CPU and this game needs to be accessible to as many people as possible meaning that the AI may be quite limited in its skill

### Time

Development time is limited due to the lack of other developers and the limited time frame therefore each feature must be considered with the time it takes to make it

### Platform

While ideally this game would be made on a website, due to my more limited knowledge of JavaScript and the large extra time requirement creating a website has I’ve decided to use Unity and C# as an alternative. This would reduce player numbers as players as players would need to also have a game distribution platform such as Steam installed

This game also won’t be cross platform as despite Unity supporting IOS and Android, these require extra development time due to UI considerations and would require scaling back the AI even further

# Requirements

* A powerful computer capable of running Unity, Visual Studio 2022 and completing Unity builds in a reasonable amount of time
* Ability to code C# as well as knowing Unity and socket specific parts of C#
* Ability to use Unity
* A second computer capable of running the built game and connecting to a hotspot to test multiplayer support

# Success Criteria

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Category | Measured by | Justification |
| All essential solution features included | Functionality  Usability | Tick list  Testing | The game must be complete |
| Error-free | Robustness | Stress testing (possibly automated) | Clients should never have to restart the game due to errors to ensure a good UX |
| Secure (optional) | Robustness | Ensure data is encrypted  Use Wireshark to inspect packets sent and ensure they are encrypted | Optional as no sensitive data is transferred |
| User friendliness | Usability | Tests with target audience | Users must be able to use the software without any external assistance |
| AI performance reasonable | Functionality | AI makes moves (on average) in under 1 minute on mid-range hardware | Users should be able to use the AI without requiring very powerful hardware |
| First functional build released by end of November | Functionality | - | Ensures project is completed on time |

# Testing

Excel spreadsheets for testing will be created as the project progresses as the UI, AI and game modes are all subject to change and thus might require different tests.

Testing itself will be carried out through a combination of manual tests and automated tests. These automated tests will be in a class in Unity that is only compiled and ran in the editor, not the release build.

# The Game Mode System

This game will support many different game modes. This will be achieved using OOP techniques, namely polymorphism, so that the Game Manager (handling who’s turn it is and general game rules), Board Manager (handling how large the board is and where pieces can go) and Pieces (holding a piece’s value and methods for getting its possible move) are al interchangeable. The Pieces, Board and Game will all have serialisation and deserialization methods to save and load games.

Each Game Manager will have a UID which can be used to synchronise the game mode between players.