Evaluation

Satisfying Requirements

1) This requirement was achieved in full effect using a text-based interface.

2) This requirement was achieved in full effect using Pygame.

3) This requirement was achieved in full effect using Pygame.

4) This requirement was achieved in full effect using a text-based interface.

5) This requirement was achieved in full effect using a text-based interface.

6) This requirement was achieved in full effect using a text-based interface.

7) This requirement was achieved in full effect using a text-based interface in conjunction with pygame.

8) This requirement was achieved in full effect using a module called pickle.

9) This requirement was achieved in full effect using a module called pickle.

10) This requirement was achieved in full effect using a text-based interface in conjunction with pickle.

11) This requirement was achieved in full effect using Pygame.

12) This requirement was achieved in full effect using a text-based interface.

13) This requirement was achieved in full effect using Pygame.

14) This requirement was achieved in full effect using a text-based interface.

15) This requirement was achieved in full effect using a text-based interface.

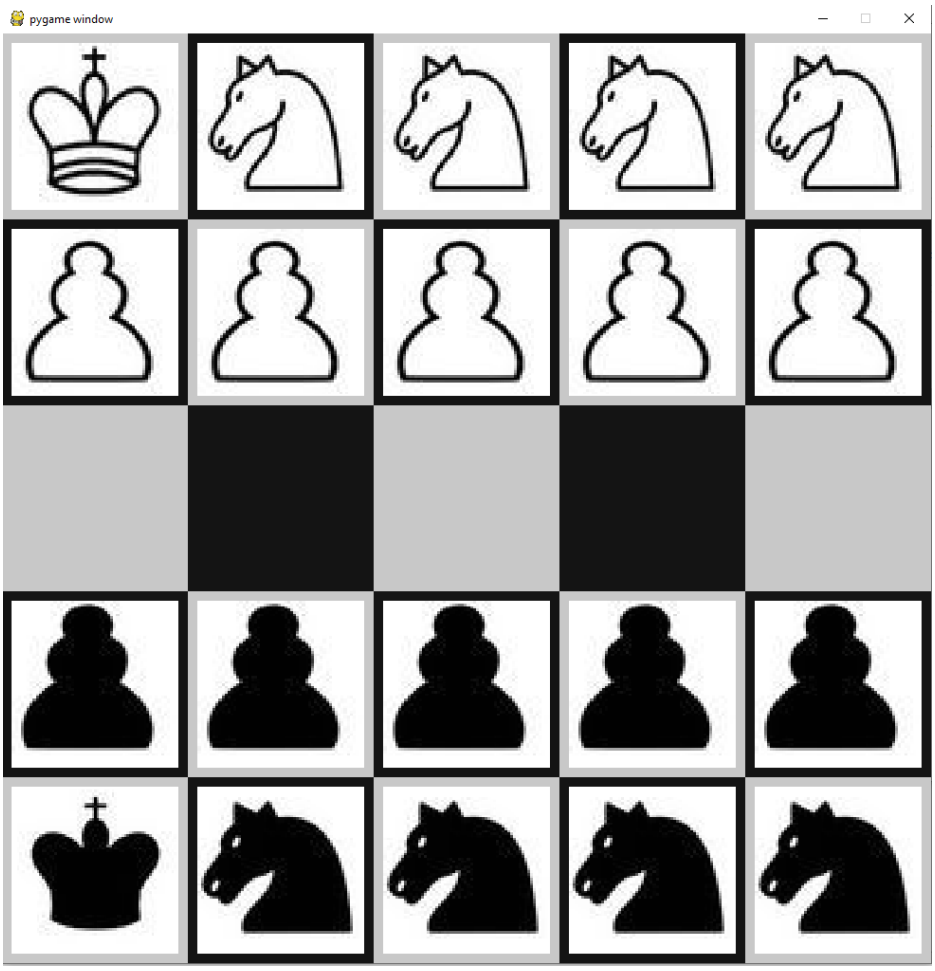
This is all demonstrated in the testing section. Requirement 13 is shown by the tests for maximum and minimum board size.

Meeting Objective

The program runs efficiently and is simple and easy to use. Chess variants can be played, created and deleted very quickly and with minimal knowledge from the user. This was largely thanks to the simple interface offered by Pygame. The more complex aspects are taken care of by the program so pieces and variants can be designed easily. The rules of the game are perfectly executed so no erroneous moves can be made.

User Feedback

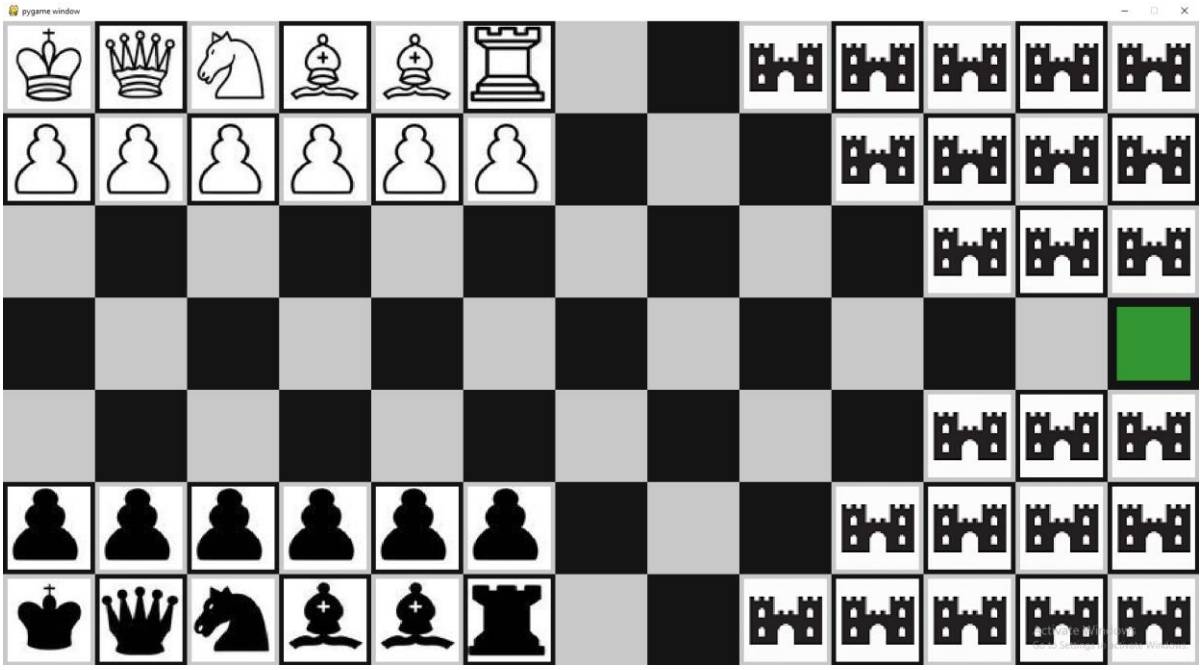
Users found that the program achieved exactly what it set out to do. One significant example is a game of Fischer Random+ that was played between users. The game board looked like this.



Experience in conventional chess says that knights are three times as valuable as pawns. However, on this board knights have significantly fewer movement options due to its small size meaning they are less valuable. Pawns, on the other hand, are significantly more powerful because the diagonal structures they form threaten a much more significant proportion of the board. On top of this their ability to promote is more powerful because the small board means that reaching the other side is no longer as difficult.

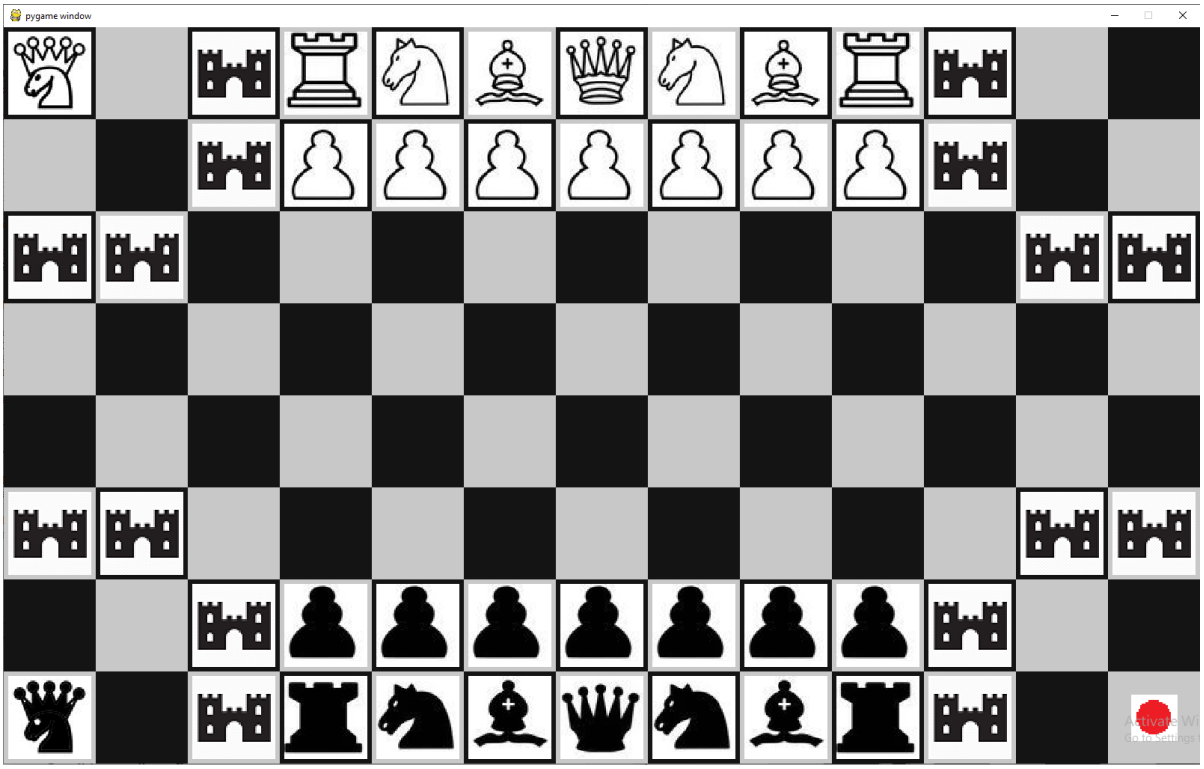
One user relied on conventional chess experience and favoured knights over pawns while the other user realised that pawns were more valuable leading to their victory. This is evidence of experience being ineffectual and creativity and intelligence leading to success.

Another example is a user defined variant called ‘The Race’. The board looked like this:



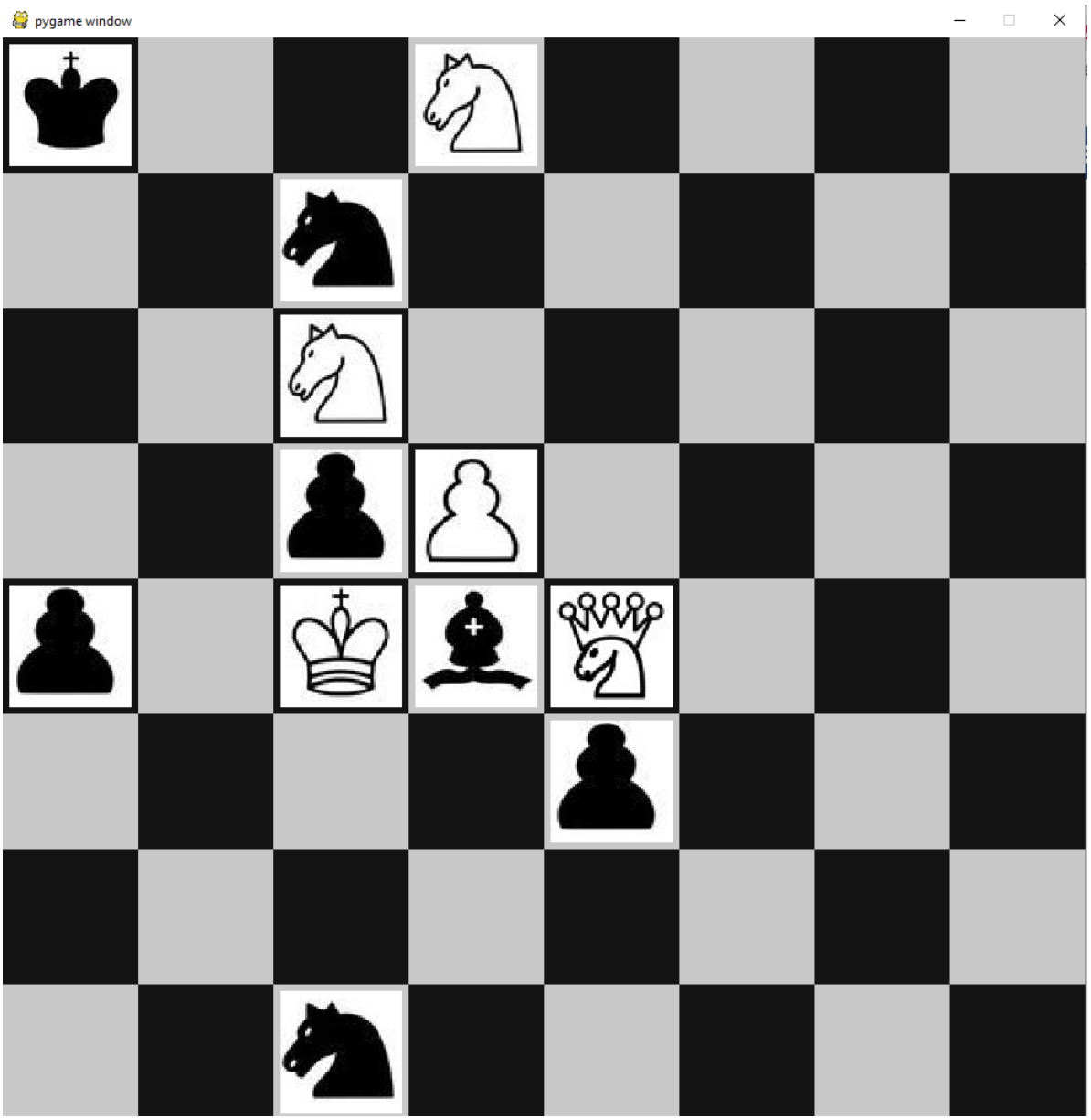
This variant adds another layer of complexity to the game because alongside checkmating your opponent, you can try to get your king to the hill square on the right of the board to win. This makes positional value of pieces a new interesting puzzle for players to tackle.

Another example is a user defined variant called ‘Backdoor Chess’. The board is as follows:

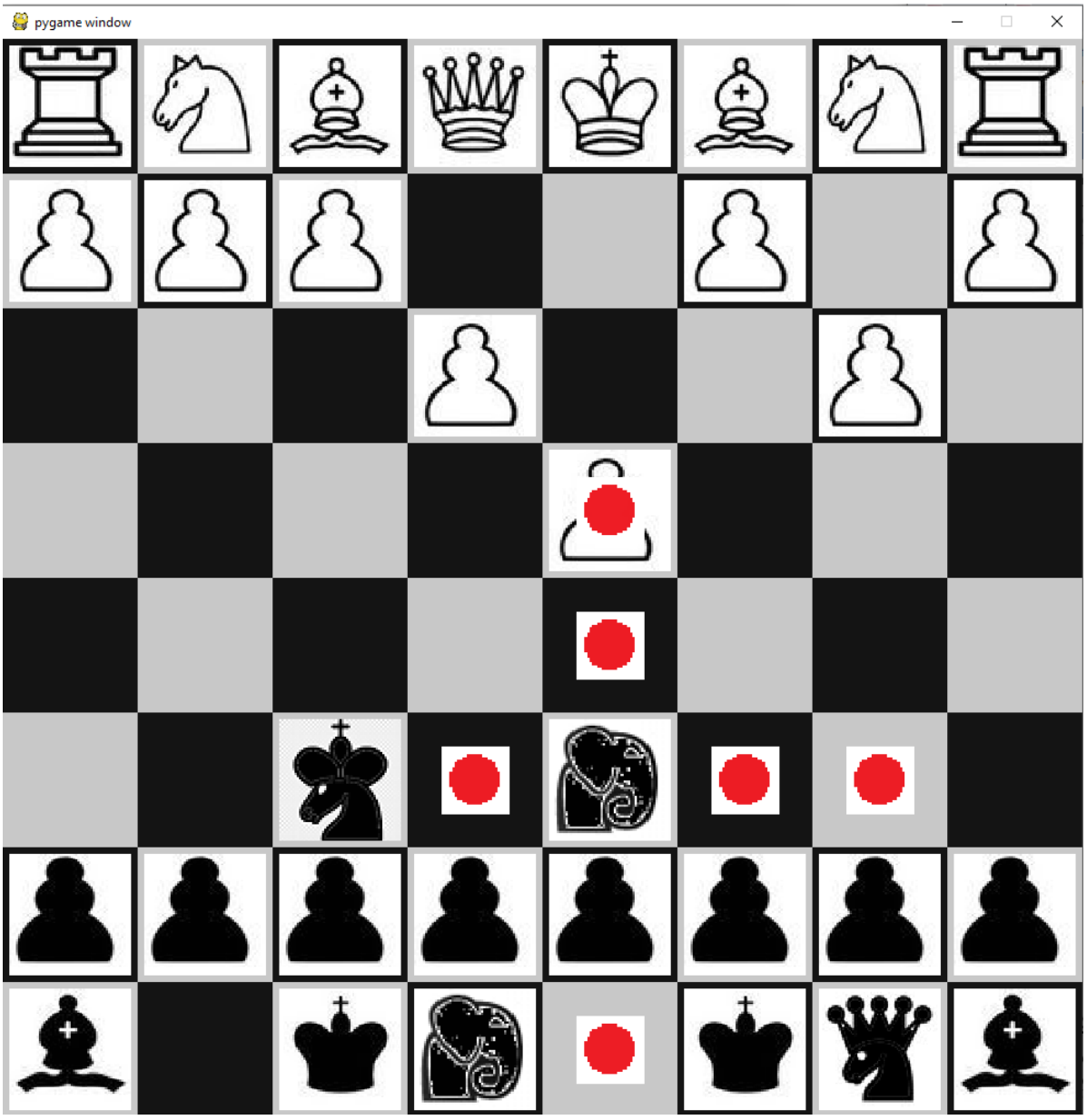


In this game mode the king has been replaced by a royal leaper that moves by a [12, 7] vector. This means it can only jump from the corner of the board to the opposite corner, in the section enclosed by walls. The enemy royal is the same but in the other corners. The goal of this game is to checkmate but the unique movement property of the royal piece means the user needs to threaten both squares on either side of the board to win. This creates unique gameplay because having a widely spread formation is now as good as if it not better than having a tightly packed formation, and the value of knights and rooks is significantly reduced because they cannot threaten the enemy royal.

Similarly, some of the sections in the possible uses and benefits in the analysis were achieved. Below is a fully functional recreation of the nightrider puzzle detailed in section 2 of the possible uses and benefits.



Next there is a recreation of Spartan Chess detailed in section 5 of the possible uses and benefits. It does not function exactly how the game was intended to, but it is most of the same and similar enough to convey the same historical information.



These examples show that not only were the individual requirements met but that when combined they managed to capture the overall goal of the program as well.

There was also negative user feedback.

One issue that was raised was that if a user defines a piece and then wishes to use that piece in another game mode, they must re-define it.

Another was a lack of features included in most common chess apps. Some of these were omitted for a reason, (see omissions in Analysis) but others such as an undo button, timers or AI opponents were valid suggestions.

Additionally, although functional a text-based interface can become tedious to navigate.

Improvements/changes

Mutability

The current parameters which the user can adjust allow for a very wide variety of different rule sets with unique properties to be designed. However, implementing spartan chess made me realise they are still somewhat limited. Ideally an indeterminate number of pieces could be designed. This would be hard to achieve because there would need to be some way to generate images that look like chess pieces, to preserve the aesthetic of the software. Alternatively, there could be a universal custom piece image with a number on so that they can be differentiated. Another solution would be to allow users to design their own pieces visually as well as mechanically, but player's may not wish to go through this process every time as the game play is more the focus than the aesthetic.

Following on from this, the current system for designing piece movement captures all existing chess pieces and has scope for many more but is still somewhat limited when compared with piece movement across chess variants. It is unfeasible to create a software that captures al possible types of movement, but the scope could be widened. For example, the user could apply directional constraints on certain vectors or have piece's that take differently from how they move.

Similarly, there could be more scope in designing win conditions. Currently there is a set list of functions to choose from, but creativity could be enhanced if the user can design them. For example, specifying how different royal pieces react to different win conditions (like a piece that can't be checkmated but can get to a hill square to win) or having different win conditions for black and white.

This increase in creative freedom could be applied to many differen5t aspects of the software and would probably be the first improvement to consider as the focus of the software is the unique and interesting game play that can arise from it.

User interface

Ideally the main menu would make use of a GUI because this would be more user friendly making the design of chess variants even easier. However, a text-based interface works fine so I deemed a GUI to be non-essential and didn't end up implementing it as a result. However, if

improving the software or revisiting the development, I would consider this as a high priority improvement because ease of access is an important feature.

Storage

A module called pickle was used to store objects in a file. The code was written in a way such that all information about each ruleset was stored in one object, the game board, and then saved to a file. This modular approach made it very easy to handle creating and deleting rulesets and reduced the scope for error. As rapid generation was a key part of the objective this was a very significant success. However, this means that every time a new rule set is created, its board, pieces and win conditions need to be redefined by the user. A potential improvement would be storing all user defined entities in a database. This would mean that instead of having to recreate every previously made feature, a user could simply call on a board or piece that has already been designed to use for their ruleset. They would then be stored in a relational database, so that each piece only needs to be stored once. If this software were to be used by many users or consistently for a long period of time this could be a very worthwhile improvement and end up saving much time and space, but for the current usage, it would make very little difference.

Leader board

A leader board feature that records players, games and outcomes of games could be a useful feature in organised institutions, such as a chess club, in which players wish to improve through competition.

AI

Finally, an AI that attempts to play user defined chess variants would be a very valuable improvement. Currently, two players are needed to thoroughly explore the nuance of a chess variant. An AI opponent would be invaluable for user enjoyment. However, this is a very complex task and no such AI currently exists.