Exploring the Python Chess Module

Liam Vallance

April 19, 2018

Abstract

In this project it is shown that python modules and pexpect can be used to great effect to create a Graphical User Interface to work with any terminal based script or application. In order to test and demonstrate this a simple and clean Graphical User Interface written in python is produced for the m2chess engine in order to explore the uses of the python language for User Interface production along with the use of modules of pexpect to link the created Graphical User Interface with the m2chess engine. The overall outcome of the Graphical User Interface is visually pleasant and functional providing the user all the functionality of the terminal base m2chess engine in a clean and easy to use Graphical User Interface.

Contents

1 Introduction

Python is a widely used and very popular programming language particularly in the Linux environment. Python makes great use of modules that are separate files that have a specific function and can be brought into existing python projects with very minimal effort. This has allowed for many well written and flexible modules to be plugged into many existing projects of varying kinds and still work very well and perform their functions. This project will explore the creation and use of python modules. The python chess engine 'm2chess' (a simple terminal-based chess game) will be used as a starting base ground for this project. The project will focus on using the understanding of modules gained from research throughout the project to create a clean user interface module for the 'm2chess' engine, this module can then be plugged into the chess engine to overwrite the terminal-based interface it currently uses. If this module is written flexibly and effectively then the module should be self-contained without any dependencies on the existing engine meaning that it can then be used on other python chess engines with very little effort.

1.1 Statement

This project is aimed to create a greater understanding of python and python modules such as the python chess module which will be used throughout this project. The end of the project should produce an entirely new User Interface module for any python chess engine using the python chess module. This new module should be designed flexibly as to allow the "plug and play" of the module into many python chess engines with very little work needed. This report should grant any reader a great understanding of how the python modules work as well as have an understanding of how to both use and create new python modules.

1.2 Objectives

- Gain a deep understanding of the use of modules in the python language
- Produce a GUI for the chess game engine I am working with that runs efficiently and elegantly working around any bugs within the unfinished engine
- Encapsulate the GUI into a new python module that can be included and used with any python chess game engine
- Explore Regular Expression

2 Background Research

2.1 Resources Used

Python:
The Python language designed in 1990 by Guido Rossum is an interactive object-oriented programming language with a very simplistic and elegant syntax which provides high-level data structures such as list and associative arrays (called dictionaries), dynamic typing and dynamic binding, modules, classes, exceptions, automatic memory management, etc... Just like many scripting languages Python is a free to use language for both personal and commercial use. The Python language can be run on virtually any modern computer system as it is compiled automatically by the interpreter to be platform independent byte code which can then be interpreted by the host system. (Sanner et al. 1999)

One of the major draw backs of using python is it's limitations for graphical user interfaces, building a GUI with just python is very difficult but luckily there are many tools that can be used to produce very simplistic interfaces such as Tkinter and wxWidgets, in this case the framework chosen was Tkinter. The problem with requiring a separate framework package in order to have the ability to produce a decent graphical user interface is that there are now many different syntaxes for solving problems based on the framework you have selected making trouble shooting a little harder as your trouble shooting now needs to be framework specific.

Modules:
With Python if you quit out of the interpreter and then re-enter then all the definitions made such as functions and variables are lost. This is where Python modules come into play, a longer program written in python is likely to need a file produced in a text editor to be passed in as the inputs for the interpreter when the Python program is executed, this is known as a script. You may also wish to use certain functions many times throughout one or many separate programs. By splitting these into many separate files the file structure becomes much easier to understand as each file becomes much more specific to its own functions and purpose while simply including and calling definitions from other modules (a file containing Python definitions and statements) imported at the top of the file. (Anon 2017)

Tkinter:
The Tkinter module ("Tk interface") is a standard python interface GUI toolkit that will be used to generate the GUI for this project, Tkinter was chosen as the interface module to use to its popularity among python developers, simplicity of understanding while still providing versatile use and its native OS design making it look great on whichever system it is run on by staying consistent with OS theme (Lundh 1999). Tkinter was written by Fredrik Lundh in 1999 who wrote two version of his An Introduction to Tkinter: the first in 1999 which was his complete documentation of the module he had created and a revised version in 2005 which presented a few newer features (Shipman 2013). The 1999 version can be accessed at: `http://www.tcltk.co.kr/files/TclTk_Introduction_To_Tkiner.pdf` and a copy of the 2005 version at `http://effbot.org/tkinterbook/`.

Tkinter although works well for this use case is in no means perfect and has its own flaws and imperfections. Tkinter objects are not strictly python objects and much of Tkinter works by containing many widgets in wrappers that can make debugging a bit harder and can sometimes give some strange and confusing errors however these issues can be less of an issue once more familiar with the Tkinter framework.

m2chess:
m2chess is the base chess engine provided by Gaius Mulley that this projects GUI is to be generated for, the m2chess engine plays a simple game of chess in the Linux terminal, this chess engine is capable of taking moves from users and has a built in simple AI competitor to play against. This chess engine is not written in python whereas this GUI project will be and so pexpect will need to be used in order to create a connection between the GUI and the console commands that m2chess prints and expects.

2.2 Similar Projects

Iwerdna Chess:
One of the main research influences for this project was a chess game written in python and gui designed with Tkinter by (Lamoureux 2012), the similarity in this project started by Lamoureux in 2012 and last updated in April 2018 meant that this was a great project to research in order to gain a greater understanding of some of the methods others have come up with to produce a similar outcome to this project. Lamoureux project was used along with others in order to gain a greater knowledge base of techniques and approaches in order to come up with solutions to the problems throughout this project.

Simple Python Chess:
Simple Python Chess is a chess engine with a console based unicode GUI and a TkInter generated GUI made by Liudmil Mitev as seen in Figure 1. Simple Python Chess as its name indicates runs a very simple game of chess in python, this does not contain a playable AI and can only be played by two humans. Regardless of its simplicity the GUI is nice and clean with some nice features such as colouring the tiles indicating legal moves. This project has many similarities to the project i am currently undergoing and will prove very beneficial and educational to investigate, it has many features that are desired and thus could help provide deeper understanding on how to best set up and use these features while there are still many features lacking to be included in this project. (Mitev 2011)

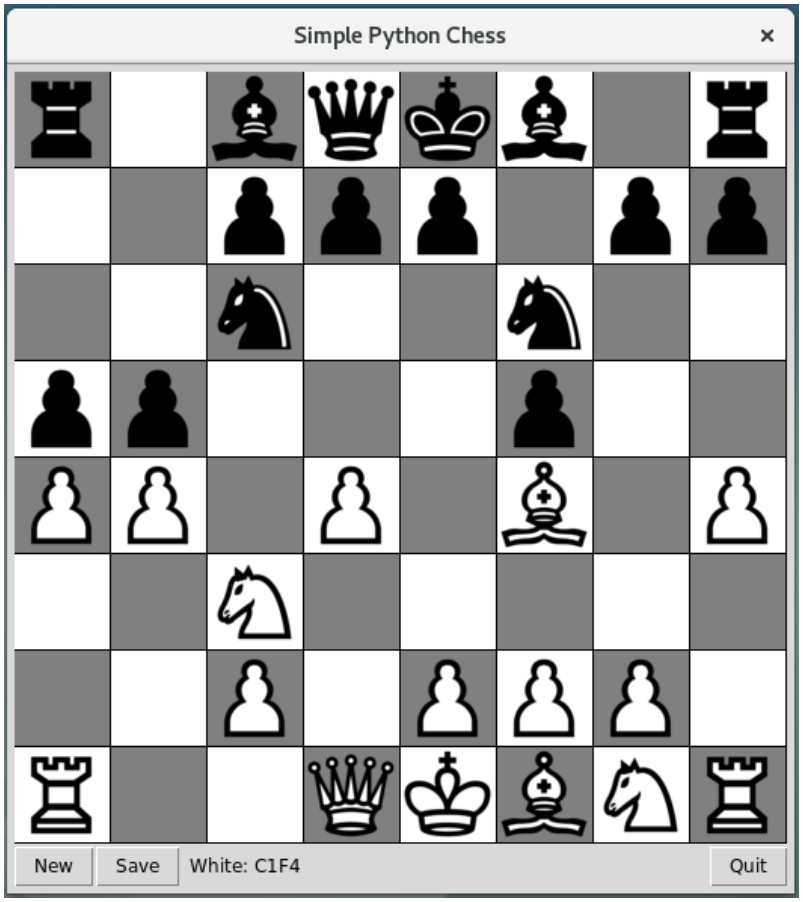

Figure 1: Simple Python Chess GUI

3 Progress

The first step of this project was to setup a Linux system as this is the operating system that the project is being carried out on. The chosen system was the latest version of Debian (Debian 9 Stretch) Once this was installed the python chess module m2chess could be acquired, after the acquisition the first problem was faced as the m2chess module was not unpacking correctly on the system, after going to the source this issue was later resolved with some trial and error debugging in the command line, with this module now successfully installed on the Debian system the project could continue.

With Python being an unfamiliar language, a decent understanding of the workings and syntax of Python needed to be acquired and so some playing around with the language and some background research helped to build a better understanding on how the m2chess modules works and how to use the Python language for the remainder of the project.

A key obstacle found at the start of this project is with an incompatibility with the system of work so far, the m2chess module is currently unable to correctly build on the system, the initial assumption as to the cause of this problem is a lack of certain libraries install on the system, this issue needs deeper research in order to find the exact problem and the issue must be resolved before any progress on the work can be made.

The source of the m2chess engine not building was the makefile containing incorrect build commands for the engine and with the makefile amended the m2chess engine now correctly builds on all work systems tested, with this complication resolved the m2chess engine has been explored and tested to see how it works in its current terminal-based state.

Using the python package Tkinter an initial basic GUI layout containing a chessboard grid with coordinate markings a-h across the bottom and 1-8 up the left side with a text field off to the right to display the previous moves made and forward and backward buttons for the user to undo and redo moves has been generated as seen in Figure 2. The GUI will remain in this basic state for the time being while the GUI gets hooked into the m2chess engine, once the GUI works well with the engine further advancements may be made with the GUI structure and design.

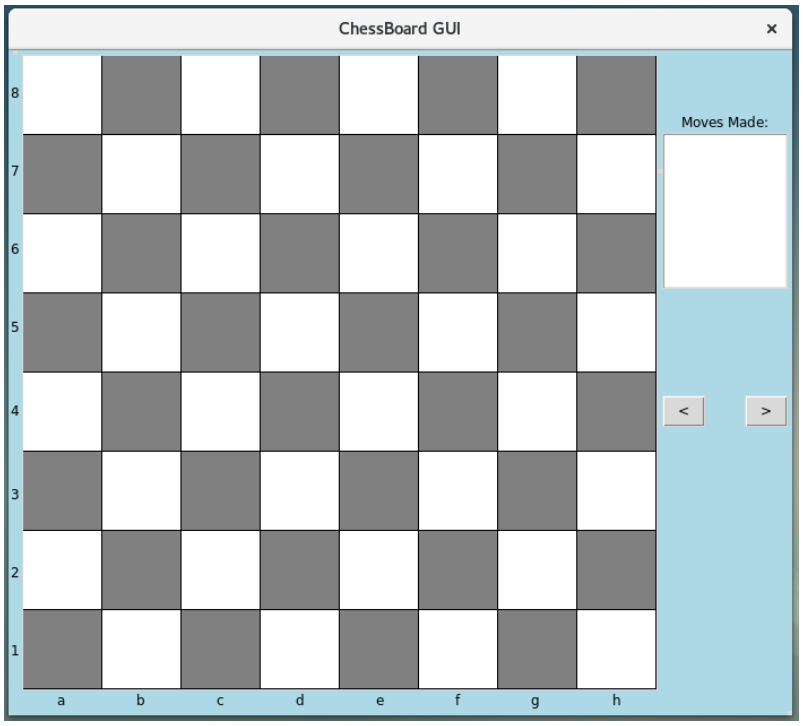

Figure 2: First GUI Prototype designed in Tkinter

The next step made was to bring in the chess piece images and draw them on screen, this proved a tricky task as there are many approaches to drawing images to the screen in python and Tkinter and the first few methods caused many problems such as having images that could not utilize the transparent background that the image png had, in the end with some deeper research into other similar projects (in this case the Iwerdna Chess game) the chess pieces were finally drawn to screen and placed on tiles with transparency however the new issue is the placing code, at this stage every piece is drawn on every tile as the looping code that selects a tile and selects a piece to pass into the place piece code is cycling through every tile and every piece, this will need to be adjusted before continuing.

With some minor adjustments to the code the pieces are now drawn only in the desired starting positions however this is only a temporary fix as the pieces are placed manually and the desired finished project will need to be able to understand game states and place pieces based on the current game state, this way there can be a simple starting state that is used at the start of the game but as moves are made the game state updates meaning that when the board s refreshed for any reason the pieces are placed back in the correct places, this will also allow the use of saving and loading games along with providing the desired undo and redo moves functionality by keeping track of the last few states an any given time.

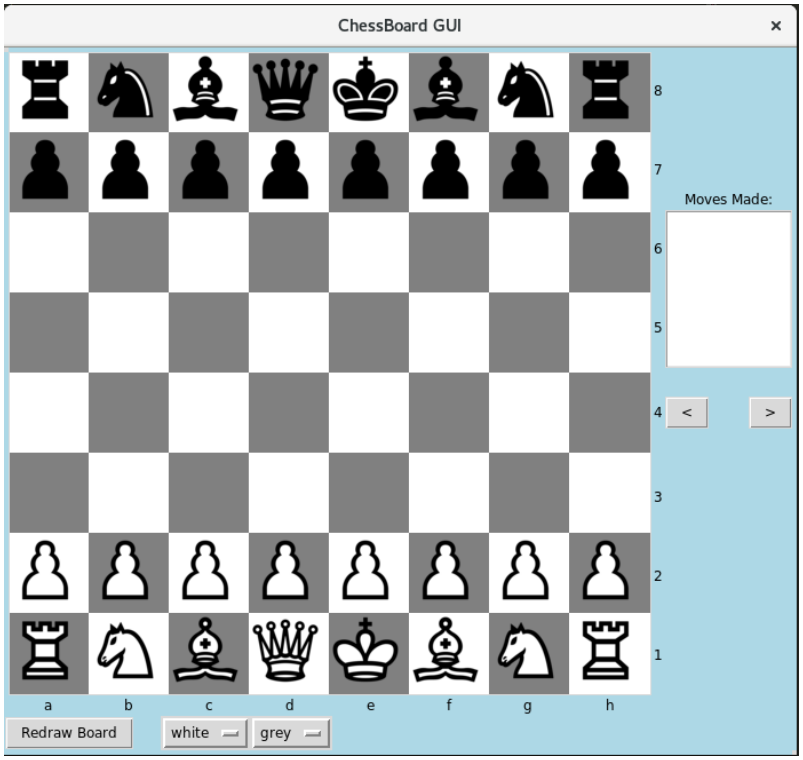

Figure 3: Manual Piece Placement

Figure 3 above also shows the refresh board and colour selection features added to the GUI these are currently a work in progress and although the board tile colour change works entirely as intended, once the board is refreshed via the 'Redraw Board' button the piece placement code currently goes out of range and all drawn pieces are lost, this issue should solve itself once the pieces can be placed via a game state rather than simply manually looped through.

With the game state proving to be far more complicated than first thought the project has moved forward without this for the time being and is sticking with the initial manual placement of the game pieces. Now the project is moving forward with linking into the m2chess engine and making moves within that engine through the GUI.

Using pexpect to run the m2chess engine and provide the base setup lines that the engine requires such as AI thinking time and whether a colour is being controlled by a human or by the AI, the chess engine in now run in conjunction with the GUI, a ”Make Move” button was additionally added just below the redo and undo moves buttons which using pexpect calls the makeMove function and sends the move gathered from the GUI to the m2chess engine to make the move itself.

A highlight feature is currently in the workings too and the user can now select a tile to highlight it in yellow, only two tiles can be selected and highlighted at a time to represent the piece to move and where to move it to, see Figure 4. At the moment this highlight does not disappear when the move has been made and this will need to be resolved, once this issue has been resolved a new ”Cancel Move” button will be added in order to clear the move and highlight without sending any information to the m2chess engine.
the piece images also do not currently move in the GUI to represent the sate of the game in the m2chess engine.

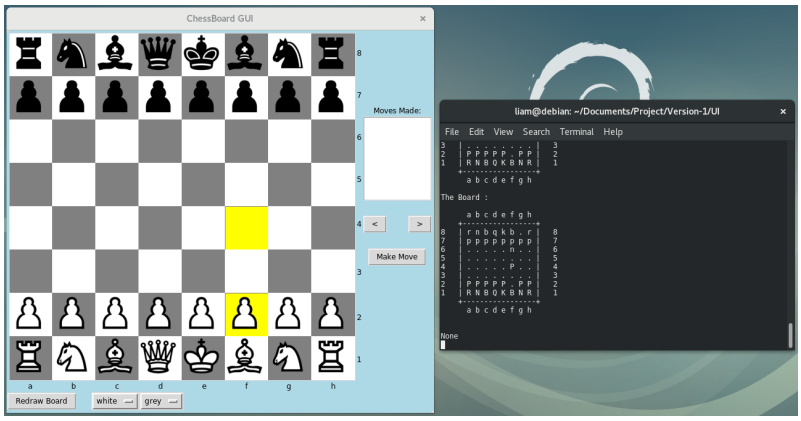

Figure 4: Move making and highlight prototype

Now that the GUI links into the m2chess engine and is capable of making moves however not able to show these moves in the GUI work as returned to the game states and getting the pieces placed based on a fen string of the current board state. To start with the fen is simply declared in it’s starting state by passing in the FEN STARTING that reads as a simple string as such: - ’rnbqkbnr/pppppppp/......../......../......../......../PPPPPPPP/RNBQKBNR’ this fen shows the state of the game board and in this case all the pieces are in their starting location. The pieces are now placed on the board based on this fen rather than placed on specific tiles manually meaning that now the GUI-Engine link can move forward with getting the engines current board state in order to use that to update the GUI board with each pieces new locations meaning that the pieces will then move on the GUI to represent their position according to the engine.

The chess GUI is now able to read in the m2chess engines current board state with the readState function that once a move has been made reads in the child.before into a variable and then looking through that variable the lines are split when the ’—’ delimiter is hit segmenting this buffer into a list that every other element shows a line of the game board, with this the every second element (the information needed) is joined onto the back of a string separated by a ’/’, this now gives a string of the engines game board in equal syntax to the fen string that is being used for the GUI, this string is then passed back to the GUI and the fen is set to equal that string and the GUIs game board is updated with the new piece locations according to the new fen. This now successfully shows the pieces where they appear in the engine. With this a very simple game of chess is now playable through the GUI in the engine with some minor missing functionality and bugs.

With this new method of placing the pieces on the board the redrawBoard function to colour the board tiles to the users preference now works correctly and the user is now able to change the colour of the tiles at any point in the game without affecting the layout of the pieces on the board at all. This also means that the highlight feature now works fully and the user is able to highlight two tiles to show the move they wish to make and when the user commits to the move

by selecting the Make Move button the board will update and the highlights will disappear setting the highlighted tiles back to the colour they should be according to the users chosen board colours. There is now an additional Cancel Move button added to allow the user to remove the highlights and select a new desired move before the move is made if they do not wish to make the first move they selected.

A new bug has been discovered while testing the current state and the GUI currently does not react well to a game check mate, when check mate is reached in the engine it prints the winner and a well done message followed by closing the engine as the game is now over. Due to the fact that the GUI is not set to react to this unexpected engine closure the final move is not made in the GUI and the console windows shows many errors suitable to the issue at hand.The GUI will now need to be set to be able to understand this endgame scenario in the engine.

The GUI is now able to represent game overs by expecting one of three statements when getting move, it now expects the engine to ask for a move, state that black wins or state that white wins, depending on the statement that it reads it will now react accordingly, if the engine asks for a move then the GUI continues as should by taking another move and passing that into the engine. If one of the two other alternative expected statements is read then the board will update the final move on the GUI board and send the user to endgame in which a message box will pop up on the users screen stating that check mate has been reached and what colour won as seen in Figure 5. Once the user dismisses this box then the user is prompted with the option to either start a new game or close the application.

Figure 5: Check Mate Message Box

The ability for the user to save and load a game state has now been implemented, the GUI now has 3 additional buttons added to the bottom right of the GUI, New Game, Load Game and Save Game. The new game button works very simply, once the new game button is clicked the engine is restarted passing in the startup variables to the engine such as the AI thinking time, as this is a new game being started the engine is also passed this ans so is set to start the engine in the opening stage and that the board is in it's initial state. From this the board GUI is then reset to represent a fresh game and is linked into the engine.

Saving the game simply takes the current GUI board state which is saved as a string called fen and passes that fen string into a saveGame text file, if the file already exists then the file is overwritten with the new game save string however if the file does not already exist then a new file is created and the fen is written to that file. The load game button looks for the saveGame file and if it is unable to find one then an error message box pops up informing the user that no save game was found and so a new game is being started instead. If there is a saveGame text file present then the string is read from the file into the fen string and the engine started this time being set to load a game state and is passed this fen string for the engines board state, the GUI is then also updated to show the loaded game sate and the user is able to continue playing their game from that point.

A final Possible Moves button has been added to the GUI which bring up a message box showing the user all the possible moves that can be made at that point in the game. The undo and redo buttons have been removed from the GUI and the feature has been discarded in favor of a more traditional feel to the game in which once the move has been made the user is committed to that move as the user has ample opportunity to decide on their committed move with the highlight, cancel and make features of the game.

Finally some start up question boxes have been added when a user loads the game asking if the user would like to load their last save game and asking the user if they would like to play against a computer AI opponent. The engine is then setup based on the answers that the user selects to these questions. The user is asked whether they would like to play against a computer AI opponent when the new game button is selected also. Figure 6 shows the final result of the GUI designed for this project.

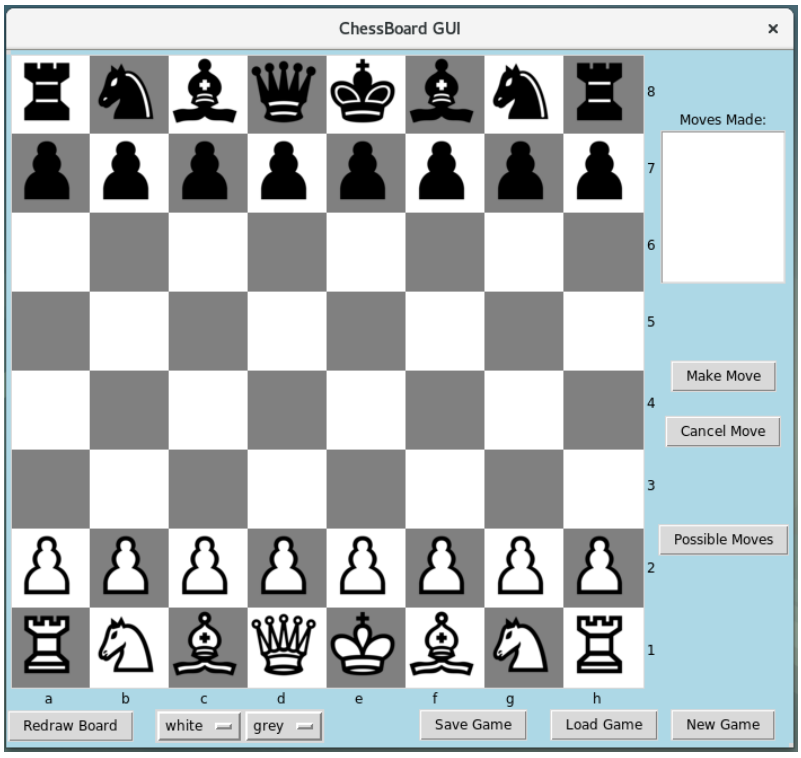

Figure 6: Final GUI State

4 Results

The final product of this project is a simple and clean Graphical User Interface that provides a user with the ability to play a simple game of chess using the m2chess engine, this chess engine allows for a player verses player game to be played but also has a simple built in AI that can play a game of chess with a thinking time that can be set, a higher thinking time provides the user with a greater challenge. This Graphical User Interface come with a simple but clean chess board with varying tile colour options for the user to choose from giving them a personalised chess experience. The board can be interacted with through simple mouse click events allowing the user to select tiles on the board with their mouse in order to highlight the move they wish to make, upon making a move the piece will update its position on the board and the move is added to the Moves Made text field in order to show the user the history of the moves they have made. This simple chess Graphical User Interface allows the user to save a load games on their system as well as start fresh with the click of a button. For beginner players who wish to learn and improve their chess game the Graphical User Interface has a show possible moves feature in which the user can display all the possible moves that can be made in the current state.

This Chess Graphical User Interface provides all the basic functionality of a game of chess with some nice extra features in order to make the user experience better and more enjoyable. The feedback from testers has been good and users seem to really like the ability to change the colour of the tiles to what they wish and the highlight features in particular.

5 Critical Evaluation

The results of this project have shown that although requiring external tool framework packages and not being a go to language for Graphical User Interface production python can be used to great effect for clean and simple User Interfaces with admirable effort put into the layout of the interface. The ease of use of modules in python help with having file communicating between each other as was needed between the GUI module and the link to the engine module.

This project has shown the effectiveness of python for uses where a link to applications and scripts of alternative languages is required. Through the use of pexpect this python chess Graphical User Interface is able to communicate with the m2chess engine with ease regardless of the language clash of the two elements.

The use of python for this project does have its limitations however, as stated above python is not a common language for a Graphical User Interface, this id due to the limitations it has as a language, although a simple UI is producible it is much harder to produce more advanced designs of User Interface and so is commonly reserved for interfaces that mostly require a few buttons, labels and text fields such as a simple login window and would have a much harder job of being able to design a full music player application such as the Spotify desktop application.

The use of pexpect to communicate between scripts and applications can easily be limited as the writer of the program would have to know exactly what the scrip or application they are linking to is going to try and do in order for them to expect that output, pexpect will not be helpful without strict understanding of the expected output of the script or application.

6 Conclusion

The desired outcome of this project was to create a simple python based Graphical User Interface for a simple chess game as the product for research into the effectiveness of using python and its optional frameworks for creating fully functional User Interfaces. Throughout the production of the Graphical User Interface there were many occasions of which greater understanding of python as a programming language was achieved, many of the skills learned for creating the Graphical User Interface are transferable and can easily be of use in many other python based projects in the future.

The use of pexpect in order to link the created Graphical User Interface into the provided m2chess engine was an entirely new concept to be explored at the start of the project and has proven its extreme effectiveness and uses throughout the Computer Science industry. The ability to expect output statements and react according to that means it has a great number of uses with any application or script which concise outputs to a terminal or uses a terminal to ask for some kind of input user based or otherwise.

The use of a Graphical User Interface and a link to the m2chess provided the perfect opportunity to split these two elements up and have the two Modules communicate between each other too as a testbed for using Modules in python for cross file and cross folder file communication. The ability to communicate between file in a programming language is not a new concept however python handles this with remarkable ease and in a very easy to understand method of connecting due to the fact that the directions needed to be given to python in order to locate a specific file matches that of how the user would use the terminal to navigate themselves to that file.

In conclusion this project was a great method of exploring python as a language as well as the uses of python modules and pexpect. The Chess game Graphical User Interface contains sufficient features to be played and enjoyed by many users however with more time and some minor improvements to time management in the future there are a few things that could possibly be implemented on top of the current feature base.

6.1 Future Improvements

Alternate chess pieces:
In the future there could be the option for the users to be able to select varying types of chess pieces much like with the board tiles colour choice already present, this feature could allow the user to pick from different art styles for the chess pieces.

More settings choices:
In the future a second child canvas window could be designed to allow the users to pick more settings about the game such as the ability to choose a difficulty of the computer AI allowing them to have more thinking time to improve its choices on the moves to make, choosing which side the user would like to be so that the user could possibly play as black and allow the compute AI to go first as white.

Timers:
In the future the ability to turn on a timed game could be added so that each player has a limited amount of time in the game as is present in speed chess rules so that once a user makes a move their timer stops decrementing and the opponents timer begins to continue decrementing.

Window resizability:
In the future the ability to resize the game window and have all of the elements on that canvas resize with it could be implemented allowing for a wider user base as the chess game will be more pleasant to view on a much wider range of resolutions.

References

Anon (2017). *The Python Tutorial*. URL: https://docs.python.org/3/tutorial/modules.html.

Lamoureux, Andrew (2012). *Iwerdna Chess*. URL: https://github.com/lwerdna/chess.

Lundh, Fredrik (1999). "An introduction to tkinter". In: *URL: www. pythonware. com/library/tkinter/introduction/inde htm*.

Mitev, Liudmil (2011). *Simple Python Chess*. URL: https://github.com/liudmil-mitev/Simple-Python-Chess.

Sanner, Michel F et al. (1999). "Python: a programming language for software integration and development". In: *J Mol Graph Model* 17.1, pp. 57–61.

Shipman, John W (2013). "Tkinter 8.4 reference: a GUI for Python". In: *dostopno na http://infohost. nmt. edu/tcc/help/pubs/tkinter. pdf*.

A Appendix

A.1 Statement of Originality

CS3D660 Individual Project

This is to certify that, except where specific reference is made, the work described within this project is the result of the investigation carried out by myself, and that neither this project, nor any part of it, has been submitted in candidature for any other award other than this being presently studied.

Any material taken from published texts or computerized sources have been fully referenced, and I fully realize the consequences of plagiarizing any of these sources.

Student Name (Printed): LIAM VALLANCE

Student Signature: Liam Vallance

Registered Course of Study: MComp Computer Games Development Year 3

Date of Signing: 19/04/2018

A.2 Ethics Checklist

This form is only applicable for assessed exercises that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, or getting information about how a system could be used, or evaluating a working system. If your proposed activity does not comply with any one or more of the points below then please contact your project supervisor and/or project coordinator for advice. If your evaluation does comply with all the points below, please sign this form and submit it with your assessed work.

1. Participants were not exposed to any risks greater than those encountered in their normal working life. Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback.
2. The experimental materials were paper-based, or comprised software running on standard hardware. Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones and PDAs.
3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project. If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant. Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.
4. No incentives were offered to the participants. The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.
5. No information about the evaluation or materials was intentionally withheld from the participants. Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.
6. No participant was under the age of 16. Parental consent is required for participants under the age of 16.
7. No participant has an impairment that may limit their understanding or communication. Additional consent is required for participants with impairments.
8. Neither I nor my supervisor is in a position of authority or influence over any of the participants. A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.
9. All participants were informed that they could withdraw at any time. All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.
10. All participants have been informed of my contact details. All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.
11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions. The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation.
12. All the data collected from the participants is stored in an anonymous form. All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Student Name: LIAM VALLANCE
Student ID: 14014610
Student's Signature: Liam Vallance
Date: 19/04/2018

A.3 Code Base:

A.3.1 gui.py:

```
#!/usr/bin/env python
import pexpect, sys, string, os, tkMessageBox
from pexpect import *
from Tkinter import *
from PIL import Image

import pexpectLink

FEN_STARTING = 'rnbqkbnr/pppppppp/......../......../......../......../PPPPPPPP/RNBQKBNR'
#global fen
fen = FEN_STARTING
foo = None

# main window setup
canvas = Tk()
canvas.title("ChessBoard GUI")
canvas.configure(background = "lightblue")
# main window setup

# base variables
grid = []
size = 70
gridLetters = ['a','b','c','d','e','f','g','h','j']
gridNumbers = [8,7,6,5,4,3,2,1,0]
coords = []
squares = []
for row in '87654321':
for col in 'abcdefgh':
colrow = col + row
squares.append(colrow)

posMoves = []
whiteTurn = True
player_move = []
# base variables

# canvas padding
padtl_board = Canvas(canvas, width=2, height=2)#padding value
padtl_board.grid(row=0, column=0)#top left of board
padbr_board = Canvas(canvas, width=2, height=2)#padding value
padbr_board.grid(row=2, column=9)#bottom right of board
padright = Canvas(canvas, width=2, height=2)#padding value
padright.grid(row=30, column=30)#padding right of everything
# canvas padding

# setup new game
def newGame():
play_ai = tkMessageBox.askyesno('Human vs ?', 'Play vs computer?')
global fen, foo
fen = FEN_STARTING

movesField.config(state=NORMAL)
movesField.delete(1.0, END)
movesField.config(state=DISABLED)
```

```
scriptDir = os.getcwd()
foo = pexpectLink.test(True)
os.chdir(scriptDir)

foo.freshGame(play_ai)

refresh(fen)
# setup new game

# setup loaded game
def loadGame():
global fen, foo

try:
fen = open('saveGame.txt','r').read()
except:
tkMessageBox.showinfo('No Save', 'No save game found\n Starting new game')
newGame()
return

play_ai = tkMessageBox.askyesno('Human vs ?', 'Play vs computer?')

scriptDir = os.getcwd()
foo = pexpectLink.test(True)
os.chdir(scriptDir)

foo.loadGame(fen, play_ai)

refresh(fen)
# setup loaded game

# save game fen to file
def saveGame():
global fen

saveFile = open('saveGame.txt', 'w')
saveFile.write(fen)
saveFile.close()
tkMessageBox.showinfo('Saved Game', 'Game has been saved')
# save game fen to file


pieceToPhoto = {}
for piece in 'pnbrqkPNBRQK':
pieceToPhoto[piece] = PhotoImage(file='./img/%s.png' % piece)


# chessboard setup
board = Canvas(canvas, width=8*size, height=8*size)
def drawBoard(colour1, colour2):
board.delete('tags')

for ridx, row in enumerate(list('87654321')):
for fidx, col in enumerate(list('abcdefgh')):
tag = col + row
color = [colour1, colour2][(ridx-fidx)%2]
shade = ['light', 'dark'][(ridx-fidx)%2]
```

```
tags = [col+row, shade, 'square']

board.create_rectangle(
fidx*size, ridx*size,
fidx*size+size, ridx*size+size,
outline=color, fill=color, tag=tags)
board.grid(row=1, column=1, rowspan=8, columnspan=8)
# chessboard setup


# place chess images on tile
def placePiece(square, piece):
#if not piece in 'pnbrqkPNBRQK':
#    raise "invalid piece: '%s'" % piece

if piece == '':
return

item = board.find_withtag(square)

coords = board.coords(item)

photo = pieceToPhoto[piece]
image = board.create_image(coords[0], coords[1], image=photo,
state=NORMAL, anchor=NW, tag='piece')
# place chess images on tile

# refresh game board to math fen
def refresh(fen):
redrawBoard()

board.delete('piece')
i=0
for x in squares:
if fen[i] == '/':
i+=1
if fen[i] == '.': pass
else: placePiece(x, fen[i])
i+=1
# refresh game board to math fen

# set colour of tiles
def colourBoard(colour1, colour2):
for rank, row in enumerate('87654321'):
for file , col in enumerate('abcdefgh'):
tile = col + row
color = [colour1, colour2][(rank-file)%2]

item = board.find_withtag(tile)
board.itemconfigure(item, fill=color)
# set colour of tiles

# board colour select
OPTIONS = ["white","grey","blue","lightblue","green","lightgreen"
,"orange","pink","purple"]

var1 = StringVar(canvas)
var1.set(OPTIONS[0])
```

```
colourPick1 = OptionMenu(canvas, var1, *OPTIONS)
var2 = StringVar(canvas)
var2.set(OPTIONS[1])
colourPick2 = OptionMenu(canvas, var2, *OPTIONS)

colourPick1.grid(row=10, column=2,columnspan=2, sticky=E)
colourPick2.grid(row=10, column=4,columnspan=2, sticky=W)

# give colour board users choice
def redrawBoard():
colourBoard(var1.get(), var2.get())
# give colour board users choice

btn_drawBoard = Button(canvas, text="Redraw Board", command=redrawBoard)
btn_drawBoard.grid(row=10, column=0, columnspan=3,padx=1, sticky=W)
# board colour select

# send player move to engine
def makeMove():
global fen, whiteTurn
move = player_move[0]+player_move[1]
foo.makeMove(move)
movesField.config(state='normal')
movesField.insert(END, 'Player: '+move+'\n')
movesField.config(state='disabled')

if (foo.getMove()):
fen = foo.endGame(fen).replace(' ','').replace('s','.').replace('S','.')
refresh(fen)
gameOver(whiteTurn)
else:
fen = foo.readState(fen).replace(' ','').replace('s','.').replace('S','.')
refresh(fen)

whiteTurn = not whiteTurn
del player_move[:]
# send player move to engine

# cancel selected move
def cancelMove():
del player_move[:]
redrawBoard()
# cancel selected move

# list current possible moves
def possibleMoves():
posMoves = foo.posMoves()
tkMessageBox.showinfo('Possible Moves', posMoves)
# list current possible moves

# mouse grid coords, print tile clicked
def mousePressed(event):
if len(player_move) < 2:
x = (canvas.winfo_pointerx()-canvas.winfo_rootx())/size
y = (canvas.winfo_pointery()-canvas.winfo_rooty())/size
x = gridLetters[x]
y = gridNumbers[y]
#bounding fix
```

```
if x == gridLetters[8]:
x = gridLetters[7]
if y == gridNumbers[8]:
y = gridNumbers[7]
tile = str(x)+str(y)
#print tile

item = board.find_withtag(tile)
board.itemconfigure(item, fill='yellow')

player_move.append(tile)

board.bind("<Button-1>", mousePressed)
# mouse grid coords, print tile clicked

# bottom grid coords
for i in range (0, 8):
lbl_i = Label(canvas, text=gridLetters[i], bg="lightblue")
lbl_i.grid(row=9, column=i+1)
# left grid coords
for i in range (0, 8):
lbl_i = Label(canvas, text=gridNumbers[i], bg="lightblue")
lbl_i.grid(row=i+1, column=9)

# moves made title
lbl_moves = Label(canvas, text="Moves Made:", bg="lightblue")
lbl_moves.grid(row=1, column=10, sticky=S)
# moves made text field
scroll=Scrollbar(canvas)
movesField = Text(canvas, width=15, height=10, state='disabled')
movesField.grid(row=2, column=10, rowspan=2)
scroll.configure(command=movesField.yview)
movesField.configure(yscrollcommand=scroll.set)
# moves made text field


# make\cancel\possible moves buttons
btn_mkmove = Button(canvas, text="Make Move", command=makeMove)
btn_mkmove.grid(row=5, column=10, sticky=N)
btn_cnclmove = Button(canvas, text="Cancel Move", command=cancelMove)
btn_cnclmove.grid(row=5, column=10, sticky=S)

btn_posMoves = Button(canvas, text="Possible Moves", command=possibleMoves)
btn_posMoves.grid(row=7, column=10, sticky=N)
# make\cancel\possible moves buttons

# new\load\save Game buttons
btn_newGame = Button(canvas, text="New Game", command=newGame)
btn_newGame.grid(row=10, column=10, sticky=N)
btn_loadGame = Button(canvas, text="Load Game", command=loadGame)
btn_loadGame.grid(row=10, column=8, columnspan=2,sticky=N)
btn_saveGame = Button(canvas, text="Save Game", command=saveGame)
btn_saveGame.grid(row=10, column=6, columnspan=2,sticky=N)
# new\load\save Game buttons

# show end game check mate
def gameOver(whiteWin):
if whiteWin:
```

```
tkMessageBox.showinfo('Chekc Mate!', 'White Wins!')
else:
tkMessageBox.showinfo('Chekc Mate!', 'Black Wins!')

answer = tkMessageBox.askyesno('New Game', 'Would you like to play a new game?')

if answer:
newGame()
else:
canvas.destroy()
# show end game check mate

# draw initial board with default colours
drawBoard('white', 'grey')
# draw initial board with default colours

# ask user to load save game
load_game = tkMessageBox.askyesno('Load Game', 'Would you like to load last save?')

# load game or start new game
if load_game:
loadGame()
else:
newGame()
# load game or start new game



canvas.mainloop()
```

A.3.2 pexpectLink.py:

```
#!/usr/bin/env python
import pexpect, sys, string, signal, os
from pexpect import *

board = []
state = []
moves = []


class test:
def __init__ (self, debugging = False, level = 1, filename = "./m2chess", directory = "../m2chess-
if os.path.isdir(directory):
os.chdir(directory)
print "cd", directory, " and running ", filename
else:
print "error as, directory: ", directory, " does not exist"
sys.exit(0)

self.child = pexpect.spawn(filename)
self.child.delaybeforesend = 0.1
self.level = level
self.finished = False
self.debugging = False


def freshGame(self, play_ai):
self.child.expect('Enter Stage Of Game')
self.child.sendline('opening\n')

self.child.expect('Is the present Board in initial position?')
self.child.sendline('yes')

self.child.expect('Enter maximum thinking time')
self.child.sendline('2')

self.child.expect('Human')
self.child.sendline('h')

self.child.expect('Human')
if play_ai:
self.child.sendline('c')
else:
self.child.sendline('h')

self.child.expect('Please enter move')
print self.child.before
print self.child.after


def loadGame(self, fen, play_ai):
self.child.expect('Enter Stage Of Game')
self.child.sendline('opening\n')

if self.debugging:
print self.child.before
self.child.expect('Is the present Board in initial position?')
self.child.sendline('no')
```

```
self.child.expect('Enter Board :')
del state[:]
for x in fen.split('/'):
state.append(x)

for y in range(len(state)):
self.child.sendline(state[y])


self.child.expect('Is the castling status in the initial state?')
self.child.sendline('yes')

self.child.expect('Enter maximum thinking time')
self.child.sendline('2')

self.child.expect('Human')
self.child.sendline('h')

if play_ai:
self.child.sendline('c')
else:
self.child.sendline('h')

self.child.expect('Enter Colour to make first move :')
self.child.sendline('white')


self.child.expect('Please enter move')
print self.child.before
print self.child.after



def makeMove(self, move):
self.child.sendline(move)
print move

def getMove(self):
print "getting move"


i = self.child.expect(['Please enter move', 'The game has ended with a Win for Black', 'The game h
if i==0:
print self.child.before
print self.child.after
return False
elif i==1:
print self.child.before
print self.child.after
return True
elif i==2:
print self.child.before
print self.child.after
return True

def posMoves(self):
self.child.sendline('help')
```

```
self.child.expect('Please enter move')
moves =  ''.join(self.child.before.split(' '))
moves = moves[36:]
print self.child.after
return moves



def readState(self, fen):
del state[:]
board = self.child.before[-277:]
for x in (board.split('|')):
state.append(x)
board = '/'.join(state[::2])
#print board
if board[0] == ':':
return fen
else:
return board

def endGame(self, fen):
del state[:]
board = self.child.before[-275:]
for x in (board.split('|')):
state.append(x)
board = '/'.join(state[::2])
#print board
return board
```