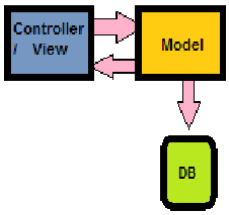
Text written in darker font colour is additions to assignment 2
Text Written in Red is additions to assignment 3

Key note about the pygame library being exported. Unfortunately you will manually install them from the pygame website, as pinpointing the exact modules in the library would be quite tedious in the pygame library as it is quite large. The EXE will do for now

4.6 Internal Review and Evaluation of our Design

Our Group decided to implement a, "Model View Controller I" or MVC I for short.



This model divides things into two sections a Controller and a View, and a model. The controller and the view will handle user inputs, while the Model will handle business logic all while accessing databases or saved material.
In our implementation of checkers:
The **"Draw "** module act similar to the Controller and Views taking in user input and interpretting that information for the model to use, all the while being called back to by the model to draw changes to the graphical user interface.

The Model is then the **"Gameloop module"**, handling most of the checker's games business logic, in the case of assignment one changing the matrix variables that currently exist

While the **Main** and **Classes** module **act as interfaces running everything together as one unit. Note** that for the design of assignment one, no database has been fully implemented as assignment one did not require much data to be saved or modified. However for assignment two and three a database will be implemented.

In terms of assignment 2, the Draw module still acts as the controller/view handling the input and the handling the drawing logic of the system. Once again similar to assignment 1, Main and Classes module, acts as an interface uniting all the code. Difference, is in regards to the SAVE module, it is a new addition to the model component of our MVC model. The SAVE module acts the component that manipulates and accesses our database. In addition, the database, or save file is in the form of a txt file. The save module does so by means of reading the txt file, for loading game, and writing to the txt file, for saving the game.

Summary:
Main,Draw,Classes,Gameloop, still behave the same way they did in assignment one's MVC model.

Save module is a new addition to the model component, and a txtfile will act as a database saving the information about the board.

Assignment 3 Involved Many Changes that affected the design of the overall project.
The addition of the AI module acts another part of the **Controller** of MVC it connects the user interaction and interpretting it and output a visual addition that gives the user an appearance of another player, in which we call a computer player.

In regards to the controller/view part of the design, the Class module originally containing a small move logic has now had its contents moved to the AI class. Originally the classes module contained logic that was believed to contribute to the overarching design of the interface. However through development it came to our attention that more and more of the movement logic/algorithms became incorporated in the interface, which goes against the purpose of an interface. Thus clearly the intelligent choice was to move that piece of code to the AI module to handle movement, already acting as a major VIEW/CONTROLLER component.

Draw module additions of a winning screen and a game mode screen still keep in the view part of the design no changes in regard to design here.

Gameloop Module is still considered part of the model as it still handles the main parts of the Businness logic(placement of pieces and jump) and acts as the major component of the Model, only now is that it incorporates the movement of the computer sent from the conroller.

Save module still remains a major component of the Model as well, saving the information of the board into a txt "database".

Classes and Screen also remain as the interface and View respectively.

Advantages to this design philosphy include:

Seperation of concerns

More specifically that we can divide work easier, one group simply works on graphical interfaces while the other can work on the logic of the system.
Note that we did not decide to use MVC II as splitting controller and view seems unneccesary for such simple requirements, but as the project goes on the stability of MVC I may be uncertain and splitting into the contoller and view may be necessary.

Loosely coupling elements

As mentioned above MVC allows for work to be dvided easily and that each element hardly affects the other one, this allows for **parrallel development** and as such makes it much more productive and easy. In addition, with this, **future changes can easily be made,** allowing removing difficulty in programming and changes in requirements.

4.7 Additional documentation of the code

The choice to use python was simply due to it's easy to use library, more specifically pygame, came with functions that was ideal for making a checkers game. This included drawing circles, squares, button, and rerendering of images. Now, many of these libraries my exist in Java, but other advantages for Python include

- Python has dynamic typing, making it faster to code and less tedious and since the project is small enough and due to short time contraints python was the better choice due to syntax
- Python's dynamic typing also make it easier to trace and read as there is less syntax, more information dense.
- Python is very flexible aand simple to use, recompilation of files can be done on demand and changes can be seen immediately.

We decided to build the program using pygame as pygame comes with many useful premade functions, such as rendering and creating objects, this make making the program much easier.Rather than completely starting from scratch we decided to use premaade libraries that exist today on the internet and are easy to use.
The design for assignment one currently consists of the five modules,

- Main

- Classes
- Gameloop
- Draw
- __init__
- Save
- Save1 txtfile

Refer to document 4.3 for the hierarchy
The **Main** module runs and calls all other modules implementing all other modules together.
Lines such as,

```
win = screen.makewindow(400)
board = gameloop.createBoard(8,8)
board = gameloop.placePieces(win, board) s
gameloop.run(win, board)
```

call other modules and help unify the entire program together, simply working with the model view controller on its own individual part is difficult and unweildy. Instead it is much easier to take thos MVC parts and instead unify them into one larger unit.

The **Classes** module acts as the interface, defining the behaviour of all other global variables and functions that exist in the program.
The interface includes,

```
class Tile
class Piece:
class Board:
```

these classes help define the behaviour of all the objects that will be created under these titles. Such as the Tile, it will consist of the square colours on the board. The pieces class will contain a total of all the pieces, keeping count of the remaining pieces in hand. The Board class keeps track of info on the board itself by means of a matrices and keep track of occupancy of squares and placement.The Classes module describes the overarching possible behaviour by preallotting variables and functions to that class acting as a contract if an object wishes to be of that type.

The **Gameloop** module acts as the model of the system, interpreting and creating the business logic of the system.

```
if click_index[0] > 7:  # clicking near the buttons
    if click_index[1] < 1:
        __placing_player_piece = True
        __placing_computer_piece = False
        __placing_player_king = False
        __placing_computer_king = False
```

More specifcally it handles most of the events, it obtains the events and from then interprets it into logic later allowing allowing us draw onto the board
The **Draw** module acts as the view and controller producing visual information and handles most of the user output.

```
def drawBoard(win, board): # drawing of the board
  for column in board.tile_matrix:
    for tile in column:
      if tile.colour == 'dark':
        colour = black
      else:
        colour = white
      pygame.draw.rect(win, colour, tile.rect)
```

It is called by GameLoop, however output from the function of the model are then placed into the draw module calls, and then the board is updated for the user to interact with.

The __init__ module is used to mark the folder as python directory and is most of the time an empty file. Reference to : http://freepythontips.wordpress.com/2013/07/28/what-is-__init__-py/

The Save module, follows the design of MVC, by being an additional component to the model of MVC I. This modules accesses and manipulates the database, in our case the database is a txtfile. This module does so by reading and writing to a series of matrices that represent the board, tiles and pieces. The module uses reading and writing function provided in python and will transport information such that the controller/view and the business logic can use later.

The txtfile save1 consists of simple row of integers that represent the pieces that was saved onto the board. The location of the integer represents where the piece is and the number itself represents the type of piece.

```
if piece_list[index] == '0':          #if empty, do nothing
   piece_matrix[i][j] = None
elif piece_list[index] == '1':        #add red normal piece
   piece_matrix[i][j] = classes.Piece([i,j], 0, 0)
elif piece_list[index] == '3':        #add red king piece
   piece_matrix[i][j] = classes.Piece([i,j], 0, 1)
elif piece_list[index] == '2':        #add blue normal piece
   piece_matrix[i][j] = classes.Piece([i,j], 1, 0)
elif piece_list[index] == '4':        #add blue king piece
   piece_matrix[i][j] = classes.Piece([i,j], 1, 1)
```

The looping in the code will handle the position in which the pieces occur.
The choice of a txtfile was used to keep thing simplistic and the board and pieces variables are represented just by a matrices of numbers anyways.

Note that blue pieces is always at the bottom and red pieces is always at the top.

The Design/Working of the AI required multiple steps:

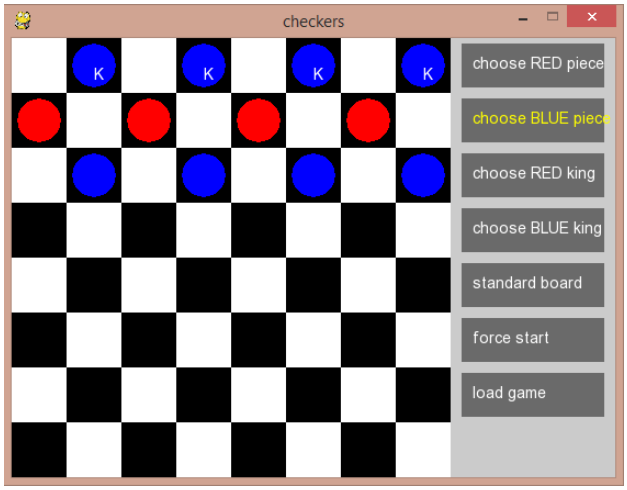
1. Checking of the legal moves-i.e the checking of the four diagonal squares located around each piece that is currently owned by the computer on the board.
2. Finally from those legal moves it takes the first semi-random(based on how it was sorted in the array). Note that if one of the legal moves is a killed move the computer will have to take the move, similar to a human player.

With this in Mind, this means that the behaviour of the AI is not 100% winning or hard, the behaviour of the AI is completely random, however the AI still follows all the rules that exists in checkers, this includes jumps and legal light squares only.

Reference to Rules we Used:
http://boardgames.about.com/cs/checkersdraughts/ht/play_checkers.htm

The Stalemate
There are two types of "Stalemates" in these checker board like games:

1. Where both players cannot make legal moves



- With forced capture , present in the game this is not possible except prematurely set up before the game starts.
- If one move is possible this type of stalemates can no longer be induced and thus **Stalemates were programmed only for the first turn.**

2. **Where both players make the same move over and over and both players wish to declare a draw.**This usually occurs when each player only has one piece left over.
    - This felt too subjective when a computer or another player should declare a draw. The question was how many turns does it take before I give up.
    - In addition, no where in our rules did it mention this "draw".
    - Thus this was not coded into the game.