CA 5 Design Brief
Turtle Chess

Neo Sahadeo

Table of Contents

Sketch....3
Problem Solving....3
Internal State Position....3
Drawing the Chess board....4
Square function....4
Implementing the Chess board draw function....4
Drawing the Chess pieces....4
Drawing a single piece....4
Drawing the Rook (Polygon)....4
Draw Poly....5
Drawing the rest of the pieces....5
User Input....5
Piece logic....6
Game logic....6
Game Design....6
Finding Checkmate....6

Table of Figures

Figure 1: Chess Pieces Sketch....3
Figure 2: Rook Drawing Example....5
Figure 3: Co-ordinate Conversion....5
Figure 4: Seen and Not-Seen Example....6

Sketch

The following is a sketch of what the pieces will look like. These will either be reduced or increased in detail depending on the performance of turtle.

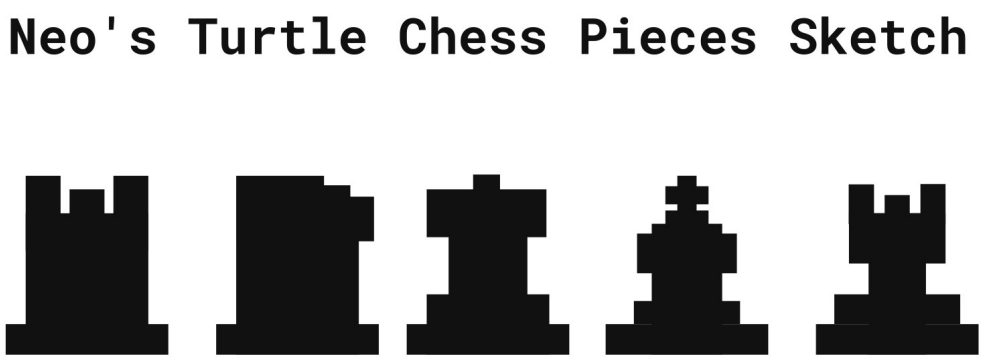

Figure 1: Chess Pieces Sketch

Problem Solving

Creating the chess board and pieces in turtle will be the main concern of this project. The second main priority will be user-input and the functionality of the chess game.

All functions written as part of this problem will be written in a loosely to decoupled state in order to maximise re-usability.

All co-ordinates discussed will be normalised to the width of the screen by dividing the width of the screen by eight that multiplying the unit co-ordinated by the amount to move the turtle to the correct position.

Internal State Position

An list will be used to store the internal state of the board. The list will be a 8x8 matrix. At each position there will be a co-ordinate and piece at that position, as well as the internal state of that index.

Board_[0][0] = ((0,0), rook, seen)

Since generating the matrix will only be done once, it will be hard-coded with initial values of a typical chess board.

Drawing the Chess board

To draw the Chess board will involve create a function called “draw_square” that should take in a co-ordinate, size and colour as arguments.

Square function

The square function should have a co-ordinate, size and colour as parameters. In order to draw a square using turtle the function should have a for-loop that iterates between 0 and 4 such that [0,4).

Pen-up, pen-down, and pen-colour will be used as a wrapper around the loop to make the code decoupled from external pen state.

During the for-loop the pen will move forward by the size amount then rotate 90 degrees to the left using the turtle function “left”.

Implementing the Chess board draw function

There are a total of 64 squares on a Chess board. This may be arranged in a 8 by 8 matrix which can be used to handle co-ordinates.

In order to draw each square in the correct position the turtle will be moved to the bottom-left of the screen and begin to draw the first square. This gives a square with the vertices [(0, 0),(1,0),(1,1), (0,1)]. After this the turtle should move to the column one row two which will be at (0,1) and begin the draw again.

To move to the position the chess board draw function will use the methods “setx” and “sety” to position the turtle in the correct position.

All the above steps will be fit into a for loop that iterates through the Chess board matrix.

The colour will be used to fill in the squares using the “fillcolor” method.

Drawing the Chess pieces

Each piece will be localised and normalised that being when the turtle beings to draw the piece it will end at the centre of the piece. This allows the piece to be centred on each tile of the Chess board without much effort.

Drawing a single piece

Again, each piece will have its’ own function. This will contain all the turtle commands needed to draw the selected piece; eg, ‘pen up’, ‘pen down’, ‘begin fill’, ‘end fill’, ‘fill color’.

Drawing the Rook (Polygon)

The rook requires roughly 5 polygons. Thus a polygon function is needed to draw a polygon with four sides with each side being length n. The function will be called “draw_poly” that has the parameters of width, height, colour, and co-ordinates.

Once this function is implemented the rook function can use it to create the rook piece. This will be involving use turtles “setx” and “sety” method to put the turtle at certain position to draw the desired shape.

The co-ordinates of the rook will known in advance (stored in a list), then the turtle we be programmed to move through those points with a for loop. After completing a polygon the turtle fill will be applied and restarted. This method if filling-in the polygon may change based on whether or not turtle will fill in the points well enough.

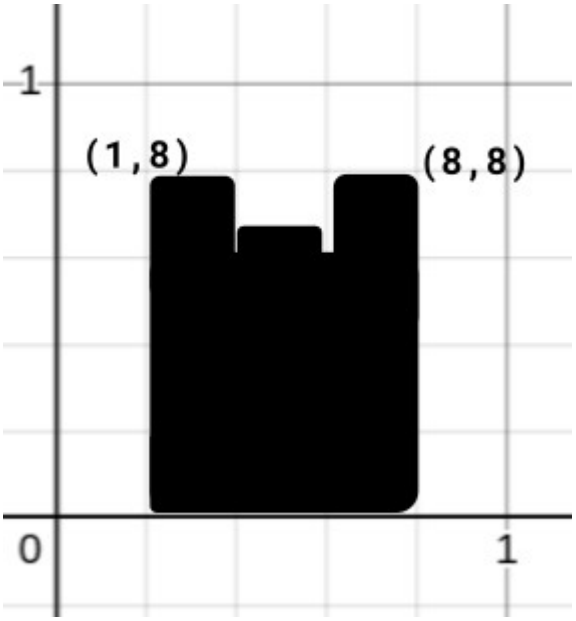

Figure 2: Rook Drawing Example

Draw Poly

The draw poly will be similar to square except the “draw_square” function mentioned earlier. The key difference will be that the forward method called will swap between the width and the height such that it will be in the order -> width, height, width, height.

Drawing the rest of the pieces

All of the piece drawing function will implement the “draw_poly” function to draw its’ respective shapes.

‘draw_queen’

‘draw_king’

‘draw_bishop’

‘draw_knight’

User Input

Getting user-input will involve using the “onscreenclick” method from turtle. This method functions a wrapper around a function that will be implemented to handle the position values returned.

With this input the appropriate piece can be found by taking the co-ordinate and dividing it by the total width and height then multiplying the value by eight and rounding it off; for example, (400, 200) in a screen space of (1280x720) will be translated to (2, 2) in the game space.

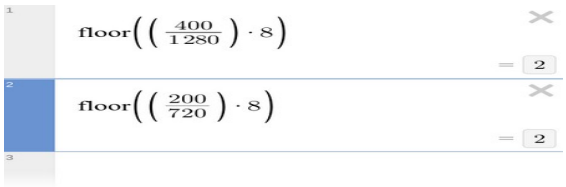

Figure 3: Co-ordinate Conversion

There will be a function called ‘conv_gpos’ that will generate a game position; and it’s opposite, ‘conv_spos’.

The function should than display a message in the console which piece is currently picked up. This then require a variable to store the current piece in the user's hand. If there is enough time left in the assignment the function might be able to control the current tile colour to indicated if a piece is selected or not.

If the user has a piece in their hand and clicks on another position, the piece should be transferred to that positions' tile.

If a piece is in the user's hand and they click elsewhere on the board, the piece will then move to that location and remove from the previous location. The function that controls this will be 'move_piece' that will implement the 'draw_~' function with a respective co-ordinates.

Piece logic

If there is an opponent piece where the user clicks, then the piece should be deleted and the piece in the hand should be drawn at that position.

If the piece is of the same colour, then the currently held piece should be changed to that newly clicked piece.

Game logic

This will be an optional feature where it will dependent on the amount of time left over after completing the previous steps.

All states of the pieces (co-ordinates and etc.) will be stored in a list-tuple.

Game Design

Again, if there is enough time left the program will implement a game-loop using a Python While True. During the process it will call the draw functions and calculate whether the game as ended or not based on site lines (co-ordinates of pieces on the board).

Finding Checkmate

The rough idea of how this works is that each piece will be able to modify that state of each square on the board to either 'seen' or 'not-seen'. This will allow the King to check if a position is able to be traversed. Then when a check if delivered, if surrounding squares are seen then that will be a mate.

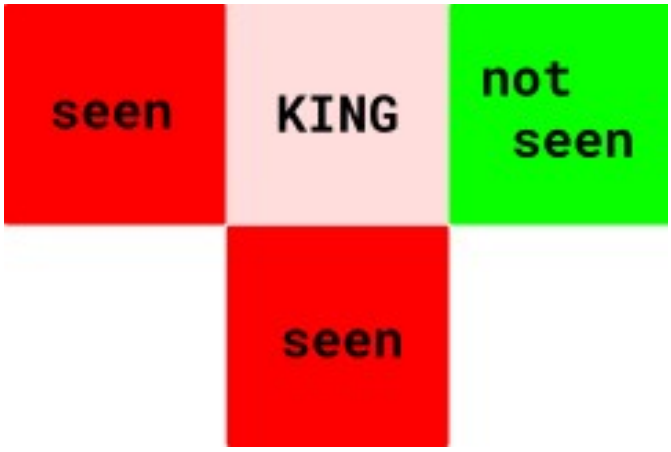

Figure 4: Seen and Not-Seen Example