Problem Solving

Neo Sahadeo

Table of Contents

Drawing

This section will explore my problem solving approach to drawing all the pieces, the chess board, and text.

Primitives

The first thing to do it create a basic building block for which I can reuse in various way. This way if I needed to change something I would only have to change it in a single place.

What primitive functions do I need?

There will be three. The first will draw a polygon, the second will draw a square and the last one will draw text.

What will the polygon function take in?

Since I really only want squares and rectangles, the function will take in a position, width, height, and colour.

What will the square function take in?

A square is a polygon with 4 sides will equal sides, the function will take in a position, size, and colour.

What will the draw text function take in?

Draw text will need a position, size, type, and text.

How will the polygon function draw the square?

This function will use the turtle functions ‘forward’ and ‘left’ to draw the polygon. I will use the setx and sety position. This will be done in a for loop

How will the forloop work?

The for-loop will alternate between the width and the height to draw the polygon.

How will it does this?

I will create a swap variable that will swap after moving turtle forward a set amount.

After this the turtle will turn left using the turtle ‘left’ function.

How will the draw square function work?

The square function will just call the polygon function passing in its parameters but instead of a two different values for width and height it will be just one value called width.

How will the text function work?

It will take in some text, a type, a size, and a location. Then it will use the turtle ‘write’ method to draw the text to the screen.

Board

This section discusses the problem solving to draw the board.

How will the board be drawn?

I will create that is O(n^2) where it will iterate of each row and column of an 8x8 matrix. This is a chess board.

How will it iterate?

This will be done with a for loop and a nested for loop.

How will the location of each tile be determined?

I will write a utility function that takes in a screen space coordinate eg: (-250,-300) and then convert that to a game-space coordinate eg: (0,0). [How this function works is talked about in utilties]

What will be draw at each location?

At each location I will call the draw square function passing in the game space coordinates, size, and colour. [The size of each tile is calculated in the utility function ‘get_tile_size’ in the utilities section]

Pieces

How will the piece be drawn part 1?

I will have each piece be drawn in its own function that takes in a piece colour and location (game-space).

How will the piece be drawn part 2?

Each piece will then have a set of calls to draw polygon where it will have unique values that I will calculate by trial and error how to draw the pieces based on the sketch.

Each piece will also have a scale factor and centre alignment that way I can change the size of every piece at random. This is helpful for scaling issues.

How will scaling be done?

Each draw poly width and height will be multiplied by a scaling factor.

How will the pieces stay centred?

I will minus a fixed width which fill calculated based on the tile height and add that as an offset on the x axis. This will be multiplied by the scale factor so it stays scaled. This will follow

$x = x + (1 - WIDTH_M \cdot SCALE) \cdot 0.5$

$y = y + CUSTOM_{position\,scale\,factor} * SCALE$

$width = WIDTH_M \cdot WIDTH \cdot SCALE$

$height = HEIGHT \cdot TILE_{heigth} \cdot SCALE$

such that

$(x + (1 - W_M \cdot S \cdot W_o) \cdot 0.5, y + Offset \cdot S, W \cdot S \cdot W_m, (T_h \cdot H_M) \cdot S)$

Positions

Calculating and converting positions aren't too complicated.

A first few preliminaries

There are a few functions that should be setup before anything else, that being 'get_screen' and 'get_tile_size' which will get the screen size and calculate the tile size respectively.

What does get_screen do?

Get screen will return the width and height of the users turtle screen instance.

How does it do this?

I can use the turtle screen object which will have the window height and window width.

Extras

I will also add padding to the left, top, right, bottom to create a black border around the board.

How will I do this?

Just subtract a small number from the total width and height then offset the axis by the number.

What does get_tile_size do?

This function will return the size of a single tile on the chess board.

How will this value be calculated?

I will call get_screen from above and then use the height of the screen divided by 8 to get my answer. Taking the floor will give me whole numbers.

Utilities

This section will cover the utilities functions that will be created called 'conv_gpos' and 'con_spos' which stands for convert to game space / screen space respectively.

What does conv_gpos do?

This function should take in an x and y screen space coordinate and return a relative coordinate based on an arbitrary axis; aka “game-space”.

How will the function do this?

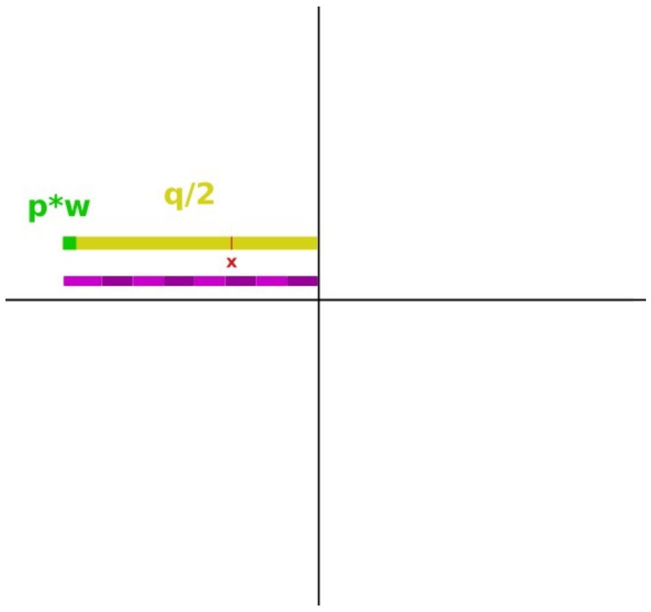
The implementation should be straightforward. I need to first get the width and height of the screen using ‘get_screen’.

The general conversion follows as:

$$\lfloor \frac{p * q + x + (q/2)}{h} \rfloor \cdot 8$$

where q can represent both the width and height. Height should be used for both because the tile size is based purely on the screen height.

This is diagram will hopefully explain what is happening. q/2 means that the offset is starting on the left. p * q + x + (q / 2) means move to the left, then go right by some amount. Dividing by h means that we break it into segments based on height, then when taking the floor and multiplying by 8 we get a nice round number. We don’t want negative numbers. This will then generate a Cartesian plane which maps screen space to game space.



What does conv_spos do?

This is the opposite of conv_gpos so the function will take in game coordinates and return the correct screen space mapping. This is needed because even though the logic my be stored and well defined in its own coordinate system, we still need to display everything.

How does this function do this?

When converting back we care about negative numbers as everything will be an offset from point (0, 0).

This gives us the following formula:

$$-(\frac{w}{2})+\frac{h \cdot x}{8}-p \cdot w$$

$$-(\frac{h}{2})+\frac{h \cdot x}{8}-p \cdot h$$

where p is the padding. We still need the screen width and height and this will come from the get_screen function.

Events

The events section describes how the event handling is expected to work.

Handling user clicks

I can use the built-in Turtle method onscreenclick that will take in a callback function that passing in parameters of the coordinates of the click.

I will write a simple wrapper function around this that will allow myself to create buttons and handle other mouse click events.

How will I do this?

There will be a handle_click function that takes in x and y in parameters. The function will run through a loop of button events and search whether the event is triggered or not.

How do I register button events?

At its simplest a button is a just an area on the screen that something when a mouse click is detected. This way, if I store the button event as its top-right coordinate along with a callback function then I just search bounding boxes.

Calculating the top-right position is pretty straightforward. I would first need to convert the game space coordinate to a screen space coordinate to avoid rounding errors; use conv_spos. I then add the width and height of the box to x, y respectively.

How do I search bounding boxes?

During the handle_click function call, the for loop runs through all button events. Then it should check each button event coordinate with the callback parameters.

How does it do this?

The equation (screen space coordinates):

$$z=(x_{max} \geqslant x \geqslant x_{min}) \wedge (y_{max} \geqslant y \geqslant y_{min})$$

I'll also need a check to find if I clicked a piece or not so this give us (game space coordinates):

$$z=(x \geqslant 0) \wedge (x \leqslant 7) \wedge (7 \leqslant y)(y \geqslant 0)$$

Buttons!

I will create a small button function to generate button that I can use. It should take in button positions, text positions, the text string, button colour, callback function, and size.

How will it work?

It will use draw_poly to render a backplate for the button, then draw_text to render the text on top of that.

I'll then convert this coordinates to screen space using the conv_spos and finally append the coordinate offsets and callback into a global variable called button events.

Internal state

This section covers how the internal state of the game should work. There should be a few constant variables being things like, colours, player scores, piece points, the board state, and data of the game.

This way I can use the values as pass-by-reference so that state won't change per function call.

Update and Get state

There should be two functions called update_state and the other get_state.

update state

Should take in different states and then update them. I'll use a dictionary because I don't know better ways that I similar to what we will learn in class.

In this function there will just be mappings to a data state variable dictionary that updates its' values.

get state

Should fetch data. It will just return a tuple of all the state that I would need like current turn, hand, location, and board. I want a tuple for tuple unpacking which is preferable to indexing. It also allows me to swap out the dictionary system for something later without major codebase changes.

Piece points

"The piece points are some peace points" – Nelson Mandela, probably. Anyway, I will take piece points from online:

Rook = 5

Pawn = 1

Bishop, Knight = 3

Queen = 9

and I'll set King to 39, wait!; Let me cook.

I’ll make every piece value negative so it will look something like this,

Rook = -5

Pawn = -1

Bishop, Knight = -3

Queen = -9

King = 39

If we add up all the piece values we get 0; If we then add these values up when calculating positional status the number should always be positive unless the king dies eg the game is over. This will keep score and there is no need for much else.

Game loop

The game loop is the last section of this problem solving document besides pseudo.

There needs to be four main functions; render board, remove piece, place piece, and move piece.

Render board

This is just a function that called the draw_square function passing in the correct colour and position. Luckly we will setup a game space so the coordinates will follow something similar to (0,0) -> (0, 1) -> (0,2) ... (1, 0) -> (1, 1) ...
I’ll run the turtle tracer method which will turn of screen updates, this will significantly (x1000+) increase the speed that turtle can operate.
There will be a nested loop which should be an 8x8 matrix. The following formula is how the colour will be calculated based on a *i-th* and *j-th* rows and columns similar to the rosette patter Amanda gave us.

$(j+i)\%2$

This will just index at either 0 or 1 for a check board colours.

In the loop I will call the place piece function.

What does place piece do?

This function will take in a piece name, positions, and colour. It will then use a switch statement to select an appropriate draw function for the piece.

How will you get the piece names and colours?

I get these values from the get_state function in the state section. This will return the board with the associated colours and piece names.

I’ll then want a small black-border around the board. All I need for this is to call the get_screen function and compute the square coordinates then call the draw square function.

$(x, y)=(\frac{-b_s}{2\cdot 100}, \frac{-b_s}{2\cdot 100})$

The function above is just a way to centre the border at the centre of the set of square generate during this time.

I’ll then draw some text to the screen an include the hand preview.

What does preview do?

Preview is a special function that is purely visual and display the current piece that user has picked up. It will draw each piece in a designated region of the side of the board. When a piece is changed the tile needs to be re-draw to avoid overlaps.

What does remove piece do?

Remove piece will remove a piece from a board visually. It does not directly effect the state of the board, its purely just for visuals.

It will just take in a position on the board and redraw a square at this position with the correct colour calculated like above using the *i-th* and *j-th* position.

What does move piece do?

I’ve left the best for last. This function should just take in the position of the click; recall the event handler function.

It should call the get_state function to get all the most recent board updates. Then it needs to figure out if a user is swapping a piece, putting a piece back down or capturing the piece.

How does it figure all this out?

Quite simply :)’

Hand is empty

Getting the piece is just an index into the board at the game space coordinate

If the hand is empty then it should just put the piece in the hand variable, update the location variable then call the update_state function.

Hand is not empty

If the hand has something in it then we need to figure out where the piece is going. We can check if its the same colour since this information is stored on the board we just read the second index position on the board.

If the colour is not the same – GOODIE! – we just replace the value of the current piece in the previous position with the new one then swap turns, update the board and zero out the old location.

If it is the same colour we should clear the hand, get the previous location and put the piece back, then update the hand with the new piece.

If it is the same position we need to should just repeat the above senario.

How do we know if its the same position?

The get_state function should returns ‘loc’ which is the previous location. We can then check if that tuple is equal to the click position.

The main function.

The main function will run the game loop using turtles ‘ontimer’ event that triggers a falling-edge timer that is needs to be called recursively.

During the main game loop function we need to call many different sub functions. Namely, “restart” and “render_board”.

The main loop will also draw a small watermark which is just my name and that it was created in turtle and an exit button to exit the game. The exit button will be created using the button function from utilities.

What does restart do?

Restart will allow the user to restart the game. It should get the default value state of the board and then perform a deepcopy to copy the list and sub-structures over to a new memory location. Then it needs to call update_state with this new state.

It then needs to rerun main, and rebind the turtle onscreenclick event. Finally it should restart the game loop.

What does the game loop do?

The game loop should continuously update the score and check if the game should continue.

How does it do this?

To check if the game is over it will iterate through each index in the board and tally pieces based on piece points. If the number is negative then the game is over and someone won. To check who won, I need to have a variable that biases two values based on colour which can be done by importing colours from the state.

If the game continues I just recursively call the game loop with a 500ms delay

If the game is won I will draw a small box that prints the winner and a button asking the user if the wish to restart the game. I will use the utility and draw functions created to do this.

Thanks for reading