Project Report

Odd Semester (2021)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:

Surname: Koesmanto **Given Name:** Edward Alvin **Student ID Number:** 2501963141

Course Code	: COMP6047001	**Course Name**	: Algorithm and Programming
Class	: L1AC	**Lecturer**	: Jude Joseph Lamug Martinez, MCS

Type of Assignments: Term Final Project

Submission Pattern

Due Date	: 17 January 2022	**Submission Date**	: 17 January 2022

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Edward Alvin Koesmanto

Project Specification

Overview

This program is a game of Sudoku, the goal of the game is to finish a randomly generated Sudoku board with a decent difficulty. The program uses recursive backtracking to keep a solution in mind of the computer.

1. Program Input:
    a. Mouse Click that signifies the position selected in the grid
    b. Number inputted by the user from both Numpad and Number Column of the keyboard
2. Program Output:
    a. A sketch of the number inputted (grey colour) by the user
    b. A number inputted by the user if it is entered and correct
    c. A strike if number of inputted by the user is wrong

Third-Party libraries/modules used

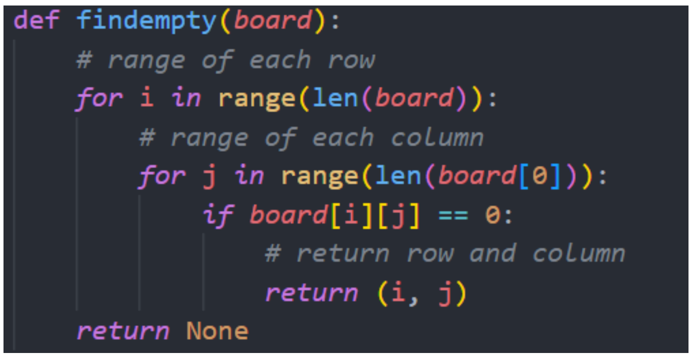
1. Pygame
    - Making games with Python Language
2. Numpy
    - Data Processing
3. Dokusan
    - Generating the Sudoku Board

Solution Design

Files Involved are solver.py and SudokuGrid.py

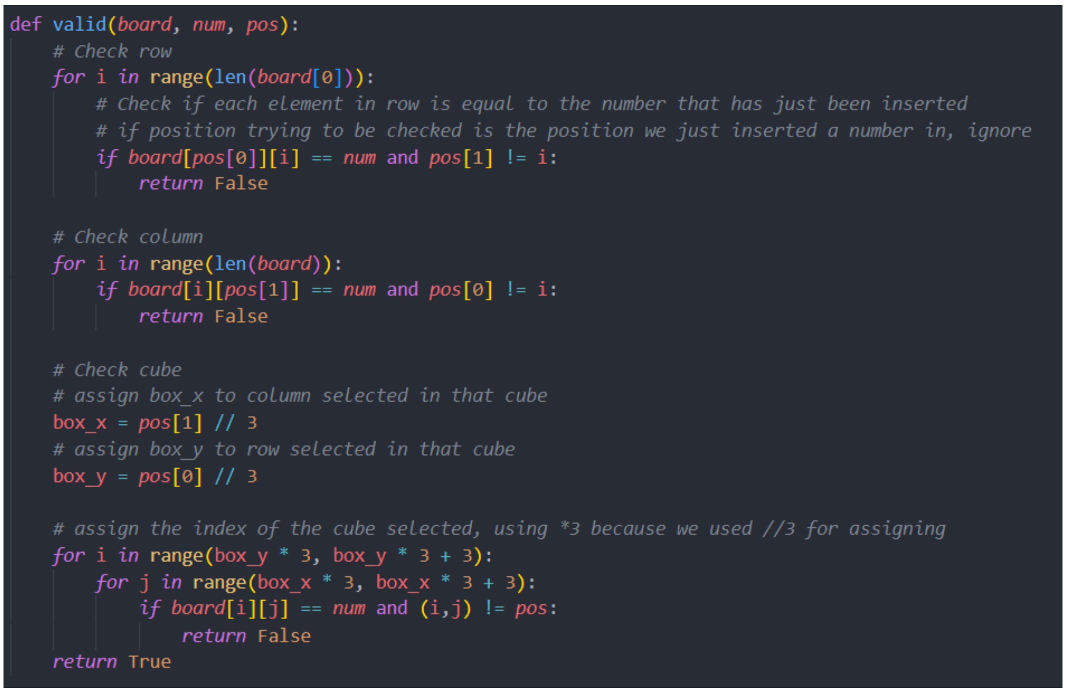
Solver.py

- Purpose: Stores the functions that makes the program able to do recursive backtracking that will be involved in the main file.



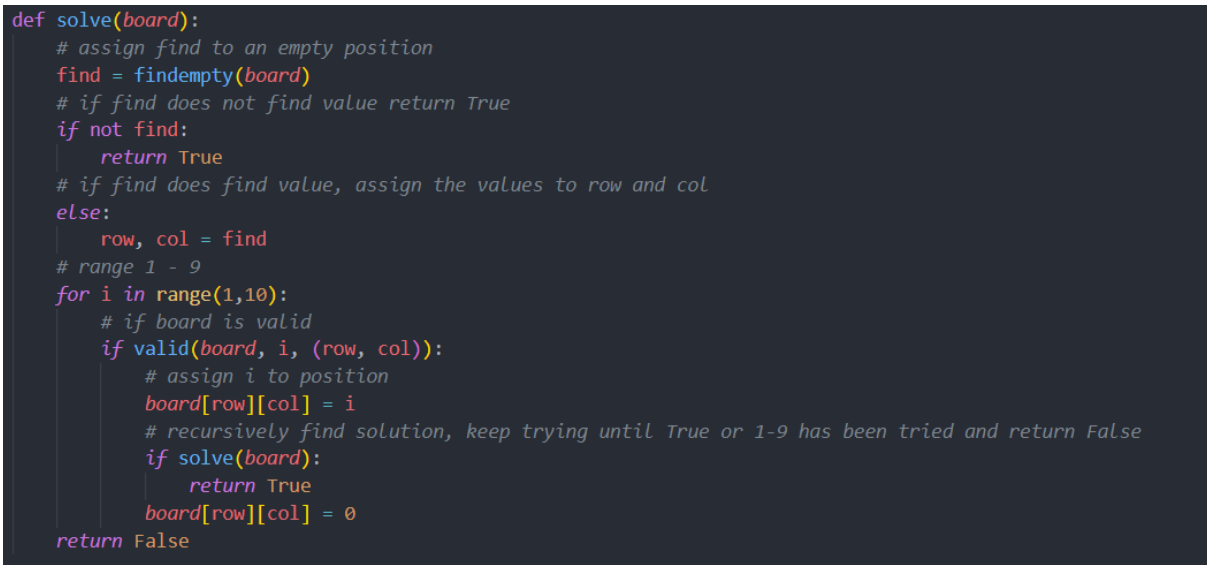
```python
def findempty(board):
    # range of each row
    for i in range(len(board)):
        # range of each column
        for j in range(len(board[0])):
            if board[i][j] == 0:
                # return row and column
                return (i, j)
    return None
```

- The findempty() function takes in 1 parameter, the board.
- It will see if there is a cube empty in the board and returns the index of that cube.



```python
def valid(board, num, pos):
    # Check row
    for i in range(len(board[0])):
        # Check if each element in row is equal to the number that has just been inserted
        # if position trying to be checked is the position we just inserted a number in, ignore
        if board[pos[0]][i] == num and pos[1] != i:
            return False

    # Check column
    for i in range(len(board)):
        if board[i][pos[1]] == num and pos[0] != i:
            return False

    # Check cube
    # assign box_x to column selected in that cube
    box_x = pos[1] // 3
    # assign box_y to row selected in that cube
    box_y = pos[0] // 3

    # assign the index of the cube selected, using *3 because we used //3 for assigning
    for i in range(box_y * 3, box_y * 3 + 3):
        for j in range(box_x * 3, box_x * 3 + 3):
            if board[i][j] == num and (i,j) != pos:
                return False
    return True
```

- The valid() function takes in 3 parameters the board, number entered and position of the board being checked.
- It will check if each element in the row and column is equal to the number just entered or not.
- If the position being checked is the position that the number has just been entered, it will ignore it and move on to another position.
- box_x and box_y will be assigned to their boxes of row and column.
- It will assign the index of cube being checked by multiplying by 3 because in the assignment, the box is being divided by 3 to get the value of the box.
- If value of the position being checked is equal to the number just entered and the index is equal to the position of the board where the number has just been entered, it will ignore it and move on to another position.
- If valid() is false then the board is not suitable.

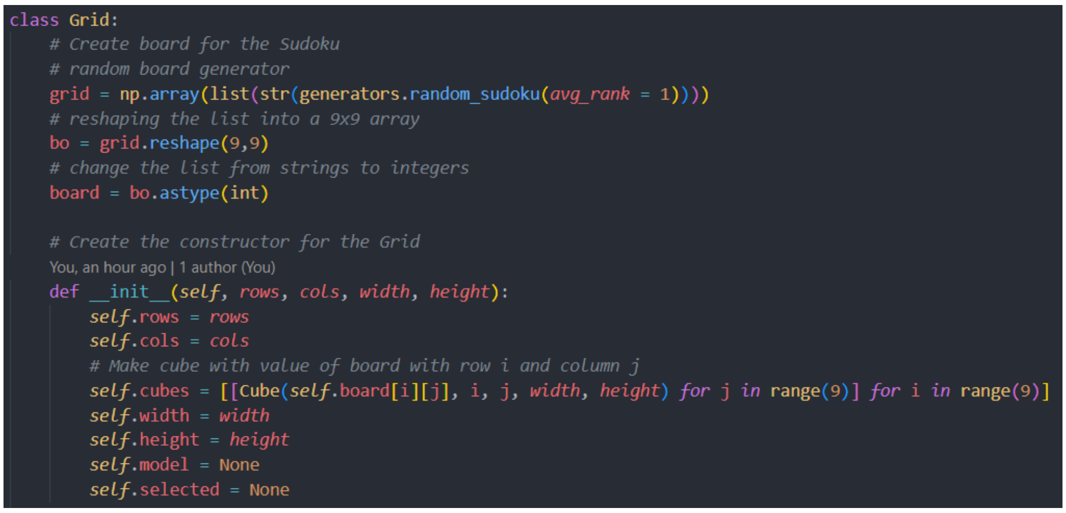


```python
def solve(board):
    # assign find to an empty position
    find = findempty(board)
    # if find does not find value return True
    if not find:
        return True
    # if find does find value, assign the values to row and col
    else:
        row, col = find
    # range 1 - 9
    for i in range(1,10):
        # if board is valid
        if valid(board, i, (row, col)):
            # assign i to position
            board[row][col] = i
            # recursively find solution, keep trying until True or 1-9 has been tried and return False
            if solve(board):
                return True
            board[row][col] = 0
    return False
```

- The solve function takes 1 parameter which is the board.
- It will assign the findempty() function to the find variable.
- If findempty() does not find an empty cube, it will return True.
- If findempty() finds an empty cube, it will assign row and col to find
- It checks if index selected is valid or not using the valid() function.
- If it is valid, it assigns the index of row and col of the board to i.
- It will then check the possible number that may be entered to complete the board from 1-9 again and again until True or 1-9 has all been tried and nothing is correct. It will then return False.

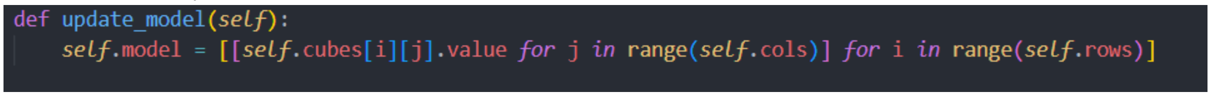
SudokuGrid.py

- Purpose: The making of the GUI is in this file.



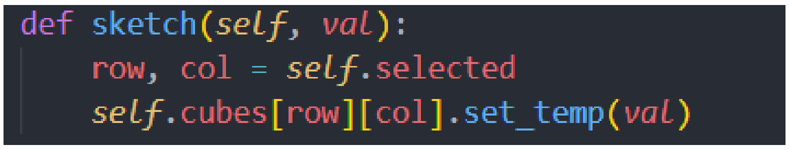
```python
class Grid:
    # Create board for the Sudoku
    # random board generator
    grid = np.array(list(str(generators.random_sudoku(avg_rank = 1))))
    # reshaping the list into a 9x9 array
    bo = grid.reshape(9,9)
    # change the list from strings to integers
    board = bo.astype(int)

    # Create the constructor for the Grid
    You, an hour ago | 1 author (You)
    def __init__(self, rows, cols, width, height):
        self.rows = rows
        self.cols = cols
        # Make cube with value of board with row i and column j
        self.cubes = [[Cube(self.board[i][j], i, j, width, height) for j in range(9)] for i in range(9)]
        self.width = width
        self.height = height
        self.model = None
        self.selected = None
```

- The Grid class will generate a random board using Dokusan and with the help of Numpy, turn it into an array. It will then reshape the array to a 9x9 board and change the type of the elements inside the board as integers.
- The constructor method defines the rows, columns, cubes of board, width, height, model, and the position selected.



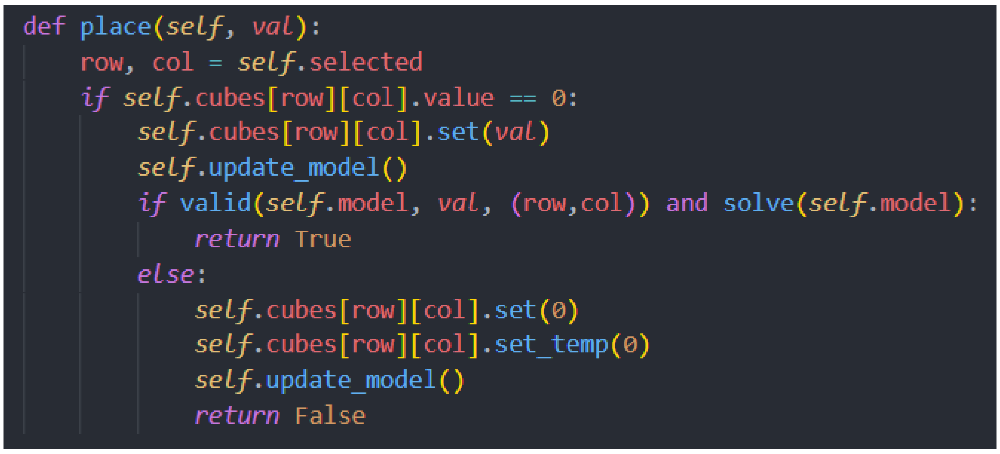
```python
def update_model(self):
    self.model = [[self.cubes[i][j].value for j in range(self.cols)] for i in range(self.rows)]
```

- The update_model() function will update the model according to the 9x9 columns and rows.



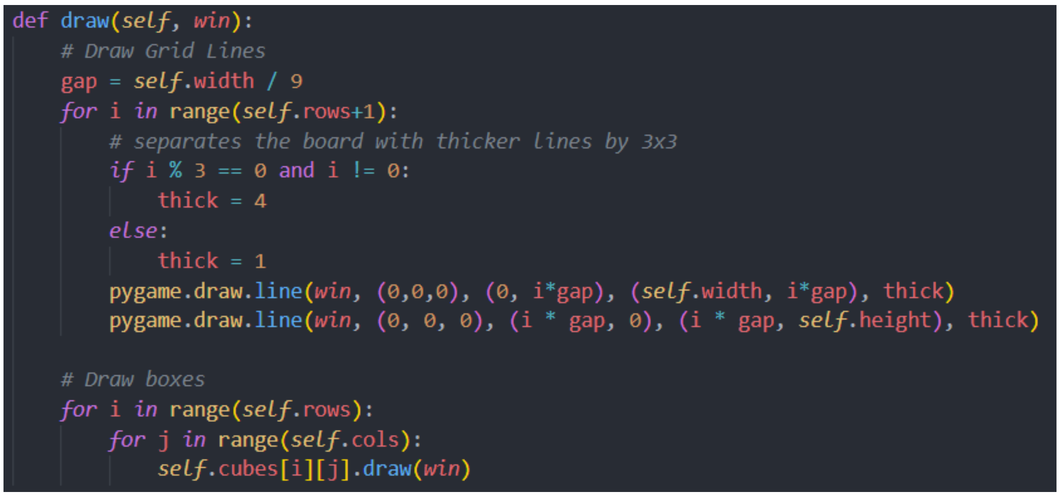
```python
def sketch(self, val):
    row, col = self.selected
    self.cubes[row][col].set_temp(val)
```

- The sketch() function will allow the user to sketch a number inputted by the user but not entered yet. It is a preview of the number.



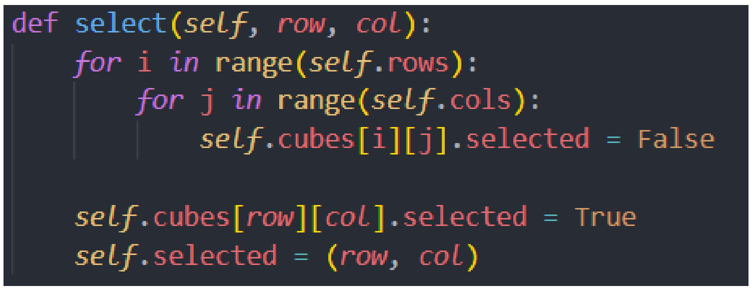
```python
def place(self, val):
    row, col = self.selected
    if self.cubes[row][col].value == 0:
        self.cubes[row][col].set(val)
        self.update_model()
        if valid(self.model, val, (row,col)) and solve(self.model):
            return True
        else:
            self.cubes[row][col].set(0)
            self.cubes[row][col].set_temp(0)
            self.update_model()
            return False
```

- The place() function will validate if the number sketched by the user is suitable or not to be entered.



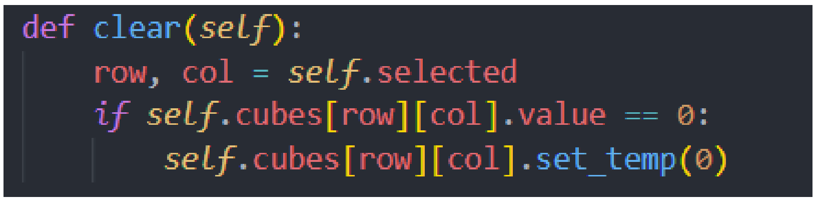
```python
def draw(self, win):
    # Draw Grid Lines
    gap = self.width / 9
    for i in range(self.rows+1):
        # separates the board with thicker lines by 3x3
        if i % 3 == 0 and i != 0:
            thick = 4
        else:
            thick = 1
        pygame.draw.line(win, (0,0,0), (0, i*gap), (self.width, i*gap), thick)
        pygame.draw.line(win, (0, 0, 0), (i * gap, 0), (i * gap, self.height), thick)

    # Draw boxes
    for i in range(self.rows):
        for j in range(self.cols):
            self.cubes[i][j].draw(win)
```

- The draw() function will draw the board.
- It will first draw the grid lines with a thicker line to separate the boxes when it is 3x3.



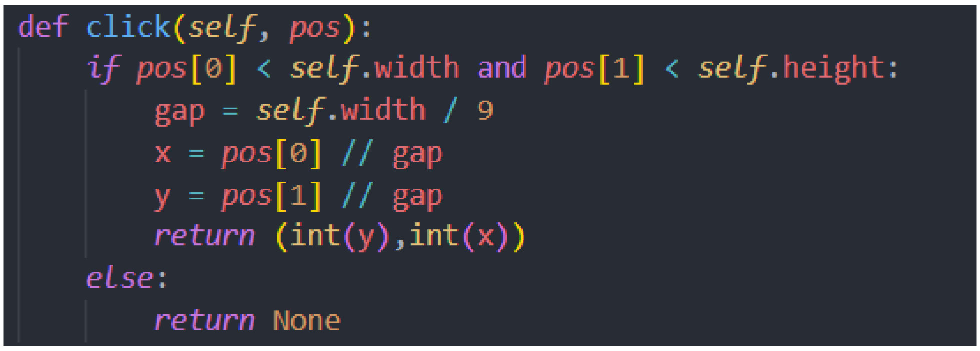
```python
def select(self, row, col):
    for i in range(self.rows):
        for j in range(self.cols):
            self.cubes[i][j].selected = False

    self.cubes[row][col].selected = True
    self.selected = (row, col)
```

- The select() function will select the box clicked by the user



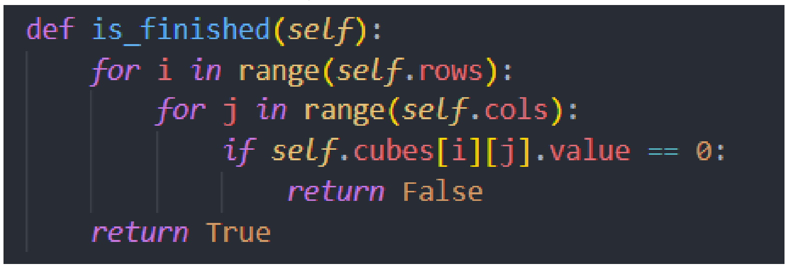
```python
def clear(self):
    row, col = self.selected
    if self.cubes[row][col].value == 0:
        self.cubes[row][col].set_temp(0)
```

- The clear() function will clear the box selected



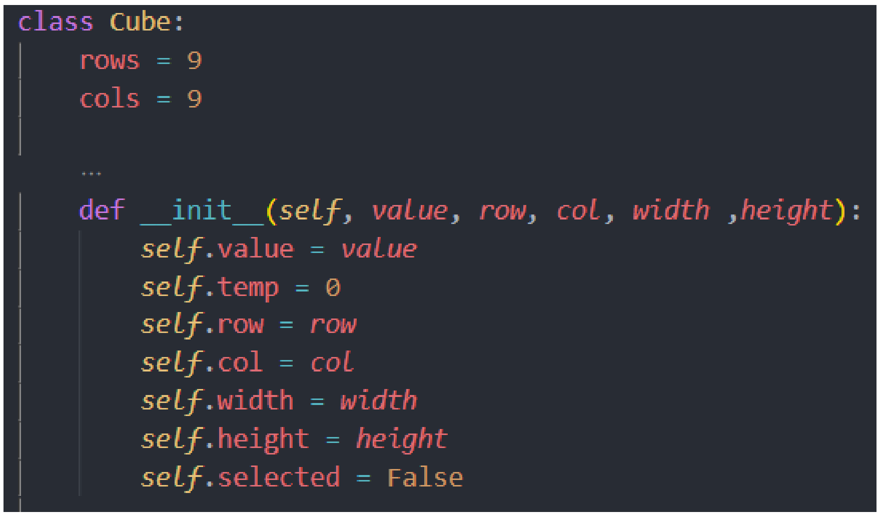
```python
def click(self, pos):
    if pos[0] < self.width and pos[1] < self.height:
        gap = self.width / 9
        x = pos[0] // gap
        y = pos[1] // gap
        return (int(y),int(x))
    else:
        return None
```

- Returns the box clicked by the user



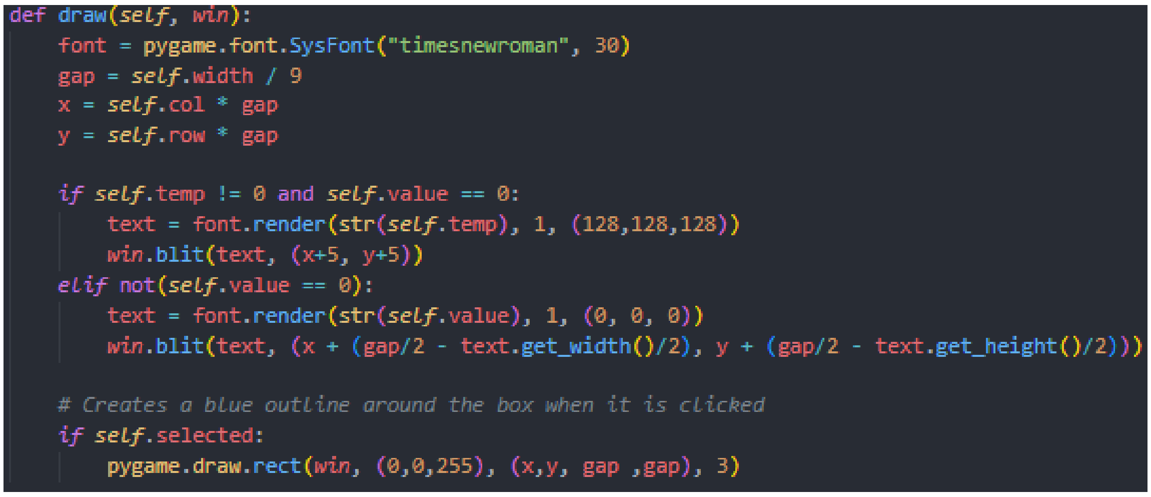
```python
def is_finished(self):
    for i in range(self.rows):
        for j in range(self.cols):
            if self.cubes[i][j].value == 0:
                return False
    return True
```

- The is_finished() function checks if the board has any missing number or not.



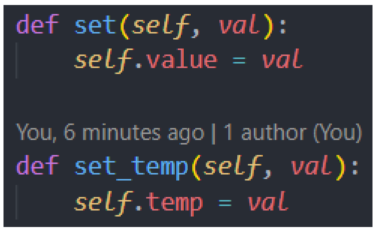
```python
class Cube:
    rows = 9
    cols = 9

    ...

    def __init__(self, value, row, col, width ,height):
        self.value = value
        self.temp = 0
        self.row = row
        self.col = col
        self.width = width
        self.height = height
        self.selected = False
```

- The Cube class defines the values needed to make the GUI such as value which cannot be changed, the temp which is the number sketched by the user, row, column, width and height of the window.



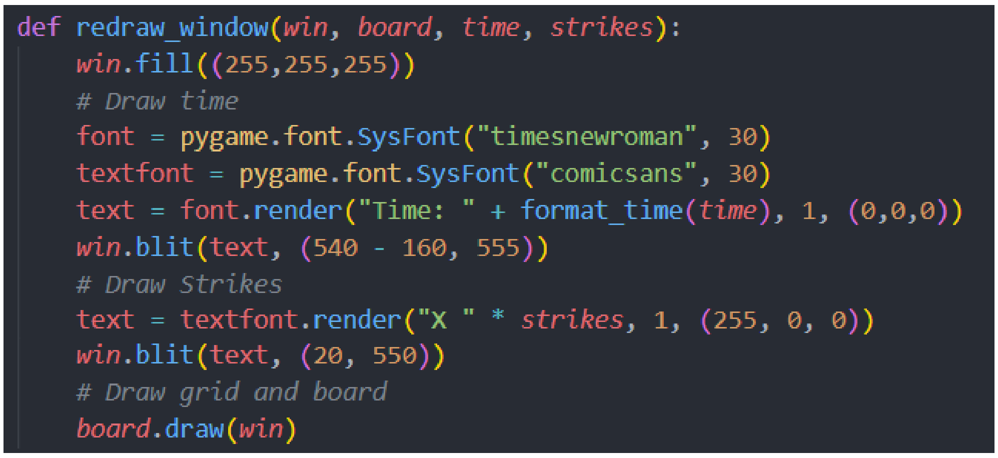
```python
def draw(self, win):
    font = pygame.font.SysFont("timesnewroman", 30)
    gap = self.width / 9
    x = self.col * gap
    y = self.row * gap

    if self.temp != 0 and self.value == 0:
        text = font.render(str(self.temp), 1, (128,128,128))
        win.blit(text, (x+5, y+5))
    elif not(self.value == 0):
        text = font.render(str(self.value), 1, (0, 0, 0))
        win.blit(text, (x + (gap/2 - text.get_width()/2), y + (gap/2 - text.get_height()/2)))

    # Creates a blue outline around the box when it is clicked
    if self.selected:
        pygame.draw.rect(win, (0,0,255), (x,y, gap ,gap), 3)
```

- The draw() function will draw the window and boxes of the grid. It will also show the selected box with a blue outline.



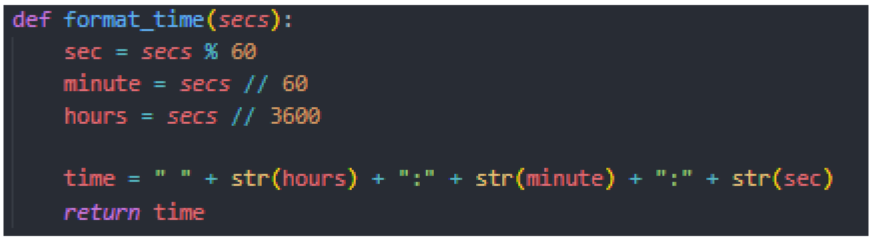
```python
def set(self, val):
    self.value = val

You, 6 minutes ago | 1 author (You)
def set_temp(self, val):
    self.temp = val
```

- The set() function will set the value of the number inputted.
- The set_temp() function will set the value sketched by the user.



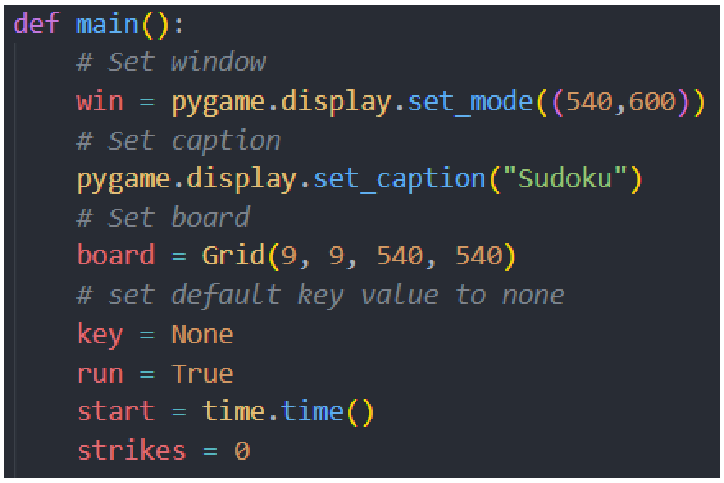
```python
def redraw_window(win, board, time, strikes):
    win.fill((255,255,255))
    # Draw time
    font = pygame.font.SysFont("timesnewroman", 30)
    textfont = pygame.font.SysFont("comicsans", 30)
    text = font.render("Time: " + format_time(time), 1, (0,0,0))
    win.blit(text, (540 - 160, 555))
    # Draw Strikes
    text = textfont.render("X " * strikes, 1, (255, 0, 0))
    win.blit(text, (20, 550))
    # Draw grid and board
    board.draw(win)
```

- The redraw_window() function will redraw the window everytime there is a change to the window, such as number sketched, inputted, time allocated and strikes.



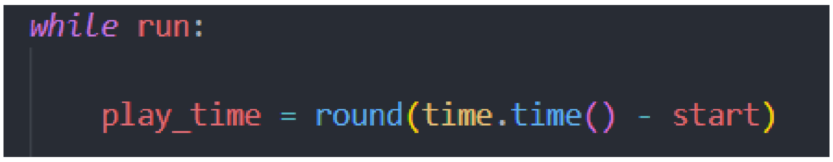
```python
def format_time(secs):
    sec = secs % 60
    minute = secs // 60
    hours = secs // 3600

    time = " " + str(hours) + ":" + str(minute) + ":" + str(sec)
    return time
```

- The format_time() function will format the time to hours:minutes:seconds and show the time running the program.



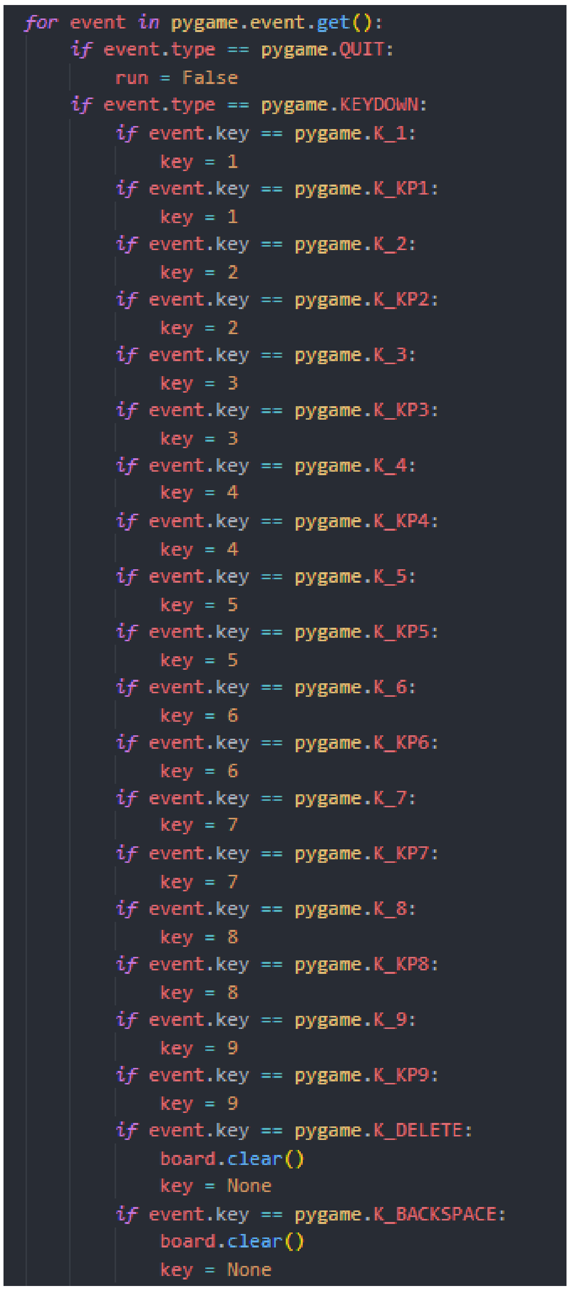
```python
def main():
    # Set window
    win = pygame.display.set_mode((540,600))
    # Set caption
    pygame.display.set_caption("Sudoku")
    # Set board
    board = Grid(9, 9, 540, 540)
    # set default key value to none
    key = None
    run = True
    start = time.time()
    strikes = 0
```

- The main() function will set everything for the run of the program.
- It will draw the window, set caption, set board and set values to none.



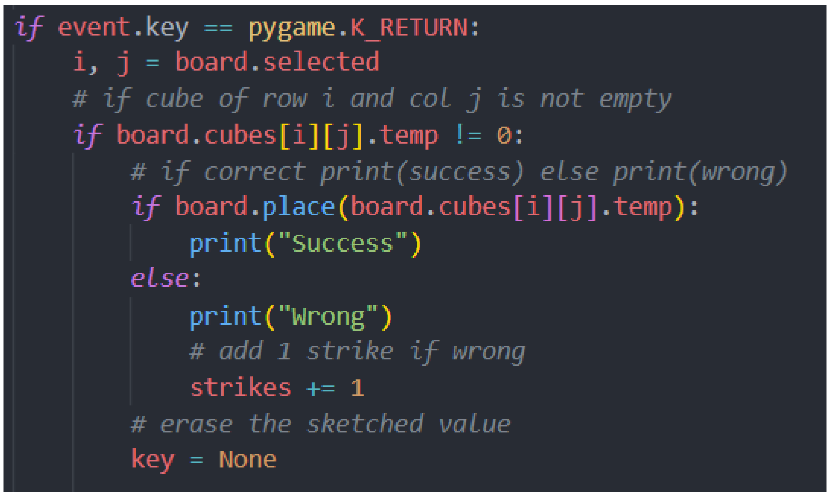
```
while run:

    play_time = round(time.time() - start)
```

- This shows while True, the program will run
- The play_time variable shows the time spent running the program.



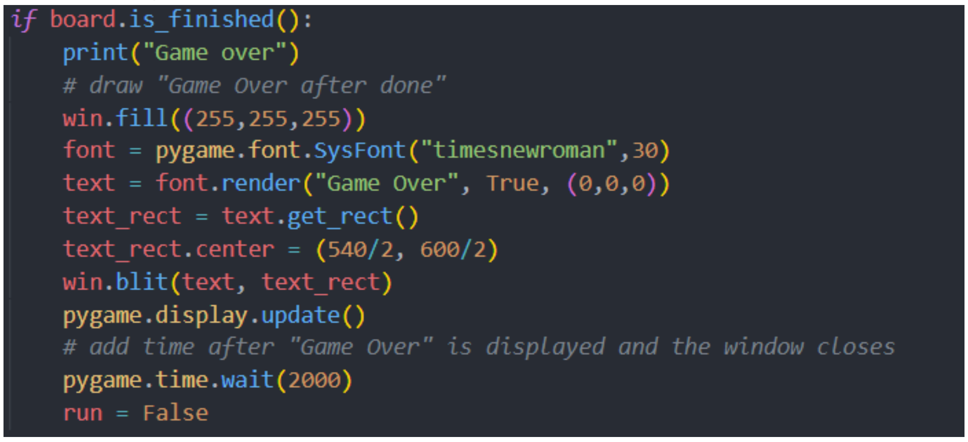
```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        run = False
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_1:
            key = 1
        if event.key == pygame.K_KP1:
            key = 1
        if event.key == pygame.K_2:
            key = 2
        if event.key == pygame.K_KP2:
            key = 2
        if event.key == pygame.K_3:
            key = 3
        if event.key == pygame.K_KP3:
            key = 3
        if event.key == pygame.K_4:
            key = 4
        if event.key == pygame.K_KP4:
            key = 4
        if event.key == pygame.K_5:
            key = 5
        if event.key == pygame.K_KP5:
            key = 5
        if event.key == pygame.K_6:
            key = 6
        if event.key == pygame.K_KP6:
            key = 6
        if event.key == pygame.K_7:
            key = 7
        if event.key == pygame.K_KP7:
            key = 7
        if event.key == pygame.K_8:
            key = 8
        if event.key == pygame.K_KP8:
            key = 8
        if event.key == pygame.K_9:
            key = 9
        if event.key == pygame.K_KP9:
            key = 9
        if event.key == pygame.K_DELETE:
            board.clear()
            key = None
        if event.key == pygame.K_BACKSPACE:
            board.clear()
            key = None
```

- The allocation of keys.



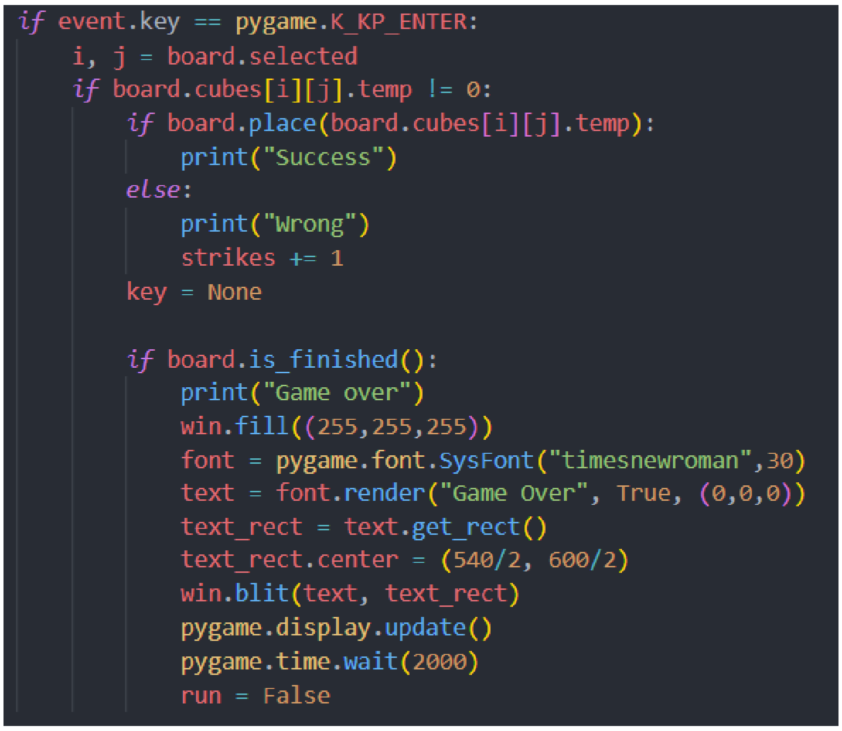
```python
if event.key == pygame.K_RETURN:
    i, j = board.selected
    # if cube of row i and col j is not empty
    if board.cubes[i][j].temp != 0:
        # if correct print(success) else print(wrong)
        if board.place(board.cubes[i][j].temp):
            print("Success")
        else:
            print("Wrong")
            # add 1 strike if wrong
            strikes += 1
        # erase the sketched value
        key = None
```

- It shows what the return key does.
- If the selected box has a sketch, it will validate if the number entered is correct to be placed there or not.
- If it is correct, it will turn the sketch to a real value into the board and print "Success"
- If it is not correct, it will clear that cube, erase the sketched value, and add a strike and print "Wrong".



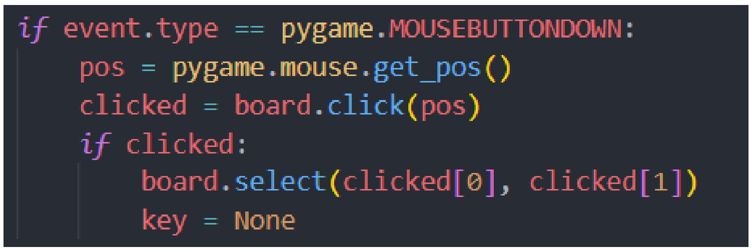
```python
if board.is_finished():
    print("Game over")
    # draw "Game Over after done"
    win.fill((255,255,255))
    font = pygame.font.SysFont("timesnewroman",30)
    text = font.render("Game Over", True, (0,0,0))
    text_rect = text.get_rect()
    text_rect.center = (540/2, 600/2)
    win.blit(text, text_rect)
    pygame.display.update()
    # add time after "Game Over" is displayed and the window closes
    pygame.time.wait(2000)
    run = False
```

- If the board is done, it will print "Game over" and a game over window will show.
- After 2 seconds, the program will exit.



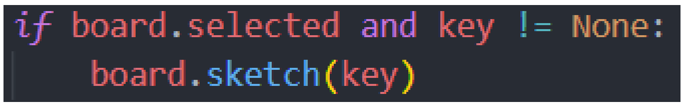
```python
if event.key == pygame.K_KP_ENTER:
    i, j = board.selected
    if board.cubes[i][j].temp != 0:
        if board.place(board.cubes[i][j].temp):
            print("Success")
        else:
            print("Wrong")
            strikes += 1
        key = None

        if board.is_finished():
            print("Game over")
            win.fill((255,255,255))
            font = pygame.font.SysFont("timesnewroman",30)
            text = font.render("Game Over", True, (0,0,0))
            text_rect = text.get_rect()
            text_rect.center = (540/2, 600/2)
            win.blit(text, text_rect)
            pygame.display.update()
            pygame.time.wait(2000)
            run = False
```

- Same as the above but using Numpad.



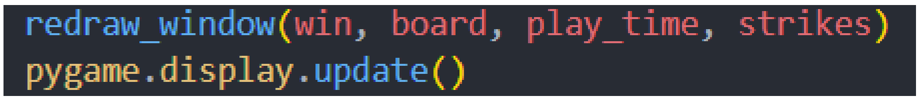
```python
if event.type == pygame.MOUSEBUTTONDOWN:
    pos = pygame.mouse.get_pos()
    clicked = board.click(pos)
    if clicked:
        board.select(clicked[0], clicked[1])
        key = None
```

- Shows the clicked cube using mouse.



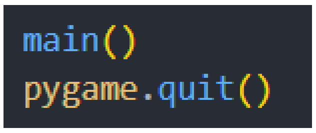
```python
if board.selected and key != None:
    board.sketch(key)
```

- Sketch the value clicked by the user but not entered yet.



```python
redraw_window(win, board, play_time, strikes)
pygame.display.update()
```

- Redraw the window and update the window.



```python
main()
pygame.quit()
```

- Call the main() function and the Sudoku window will be shown.
- After the board is all done, the game window will close and exit the program.