Disciplinary

Engineering Computing

CS3001

Course Project

Sheikh Muhammed Tadeeb (AU19B1014)

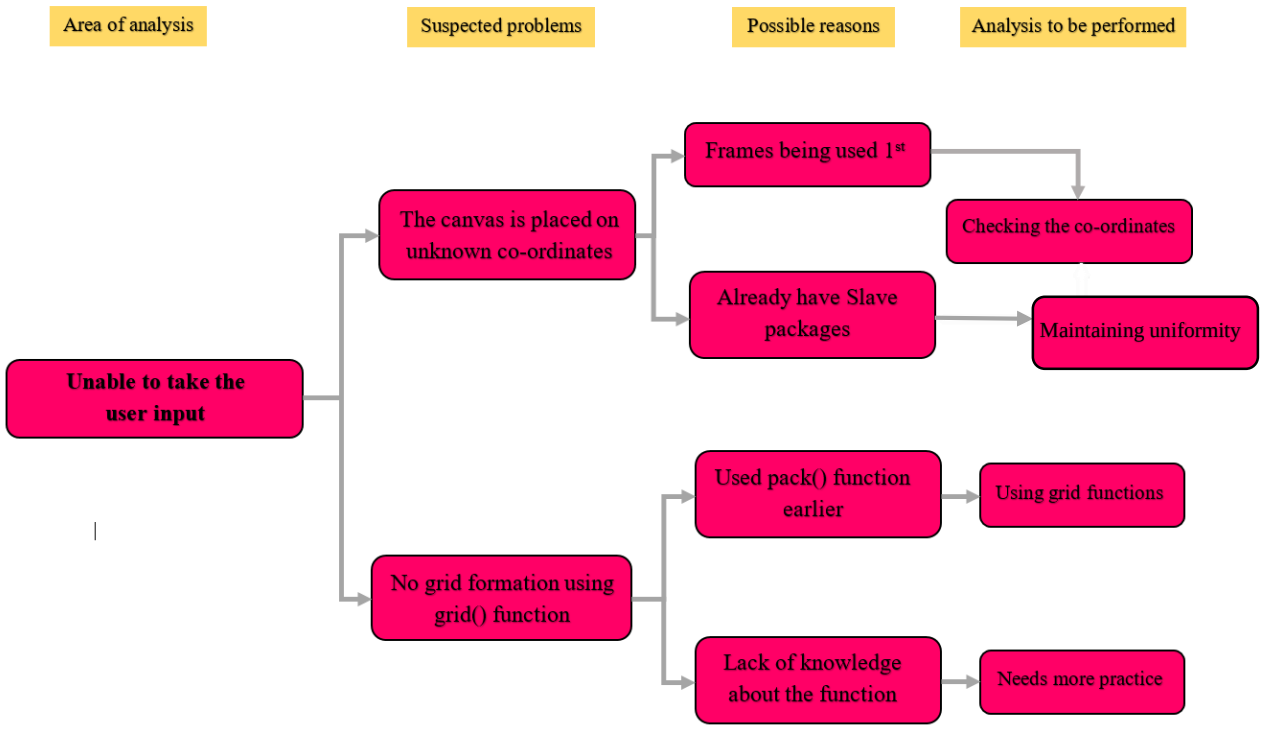
❖ Problem Statement:

Designing board game to develop better insights about computational thinking & problem solving strategy.

❖ Time Chart:

Note: All things were uploaded to GitHub simultaneously	27 Aug , 2020	28 Aug , 2020	29 Aug , 2020	30 Aug , 2020	31 Aug , 2020	1 Sep , 2020	2 Sep , 2020	3 Sep , 2020
Tasks	**Thursday**	**Friday**	**Saturday**	**Sunday**	**Monday**	**Tuesday**	**Wednesday**	**Thursday**
1) Confirm the Game and make sure the main tasks	✓							
2) Considering more details about system architecture(GUI)		✓						
3) Writing Logic, algorithm and initial code for GUI & Game.			✓	✓				
4) Finishing the GUI and Placement algorithm					✓			
5) Writing user input program at specified location using mouse click						✓	✓	
6) Documentation , presentation and video creation.								✓

❖ Issue Tree:



❖ Game Description:

Habibi (حبيبي)Time pass is an abstract strategy board game. It is traditionally a two player game played with Habibi Digits (2 and 3) on a Habibi board. It can be played using the 20×20 Habibi board or the 10×10 Habibi board.

The main objective of "Habibi (حبيبي)Time" pass is you must try to create a row of 5 or more digits in any direction (horizontal, vertical, diagonal) to win. It is a straightforward yet fun game.

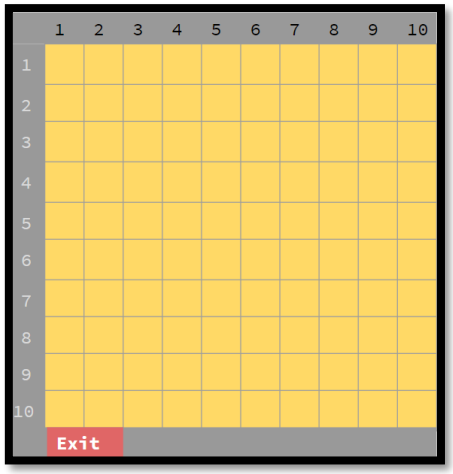
- **Game Rules: -**
  1. It is traditionally played with Habibi Digits (2 and 3) on a 20×20 or 10×10 Habibi board.

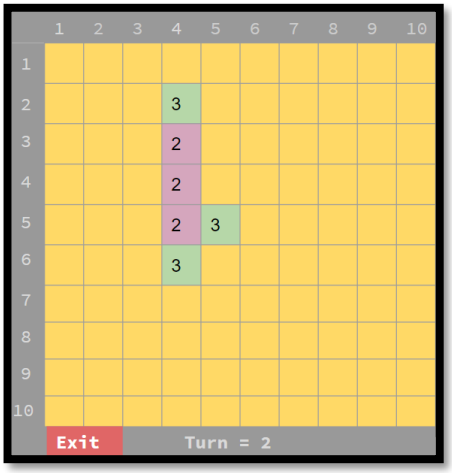
  2. Players alternatively place a digit assigned to them (either 2 or 3) on a cell.

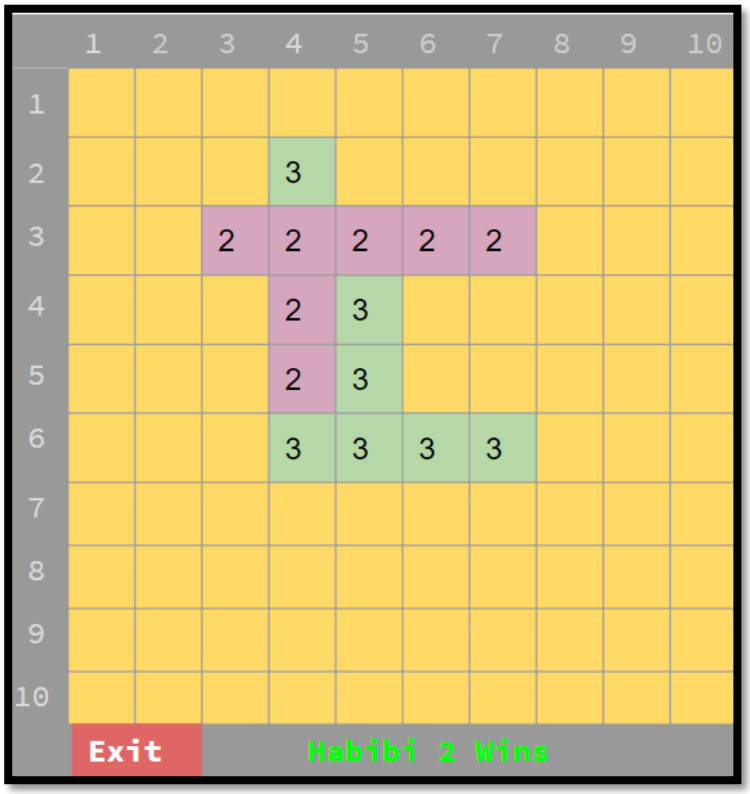
  3. The winner is the first player to form an unbroken chain of five digits horizontally, vertically, or diagonally to get a sum of (10 or 15) depending upon his chosen number.

  4. So both the players should try to place their number in such a manner that their opponent aren't able to make 5 digits in a row, column or diagonally.

❖ Initial Plan:







❖ Decomposition:

1. Creating a canvas to display things on.
2. Adding title to it.
3. Adding game ‘Label’ on it.
4. Creating Buttons (like exit and Game information).
5. How to place all the above things to a place of our choice on canvas.
6. Adding Grid over the canvas.
7. Adding functionalities to the grid (like displaying something using mouse click or taking inputs from the user at a specific location).
8. Adding a Restart button to clear everything on the existing list and starting fresh canvas to play on.

❖ Pattern recognition:

The implementation of this game is based on translation of rows, columns and diagonals into sequences of strings in lists of 5 lengths because some patterns require checking 5 consecutive positions to determine whether patterns exist.

The translation of sequences occurs four directions for each position on the board – downward vertically, rightward horizontally, left downward diagonally and right downward diagonally (see figure 1, shown in sequence of length 5). Both players, essentially, need to be aware of some threat patterns their opponent constructed, but also to construct threat patterns for their opponent. Threat patterns refer to those patterns, in which if they are not blocked on time, will result in a loss very soon.

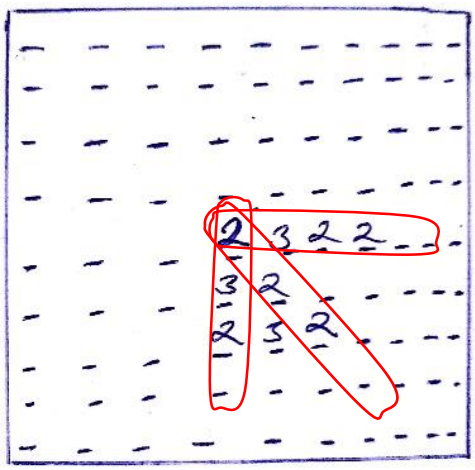
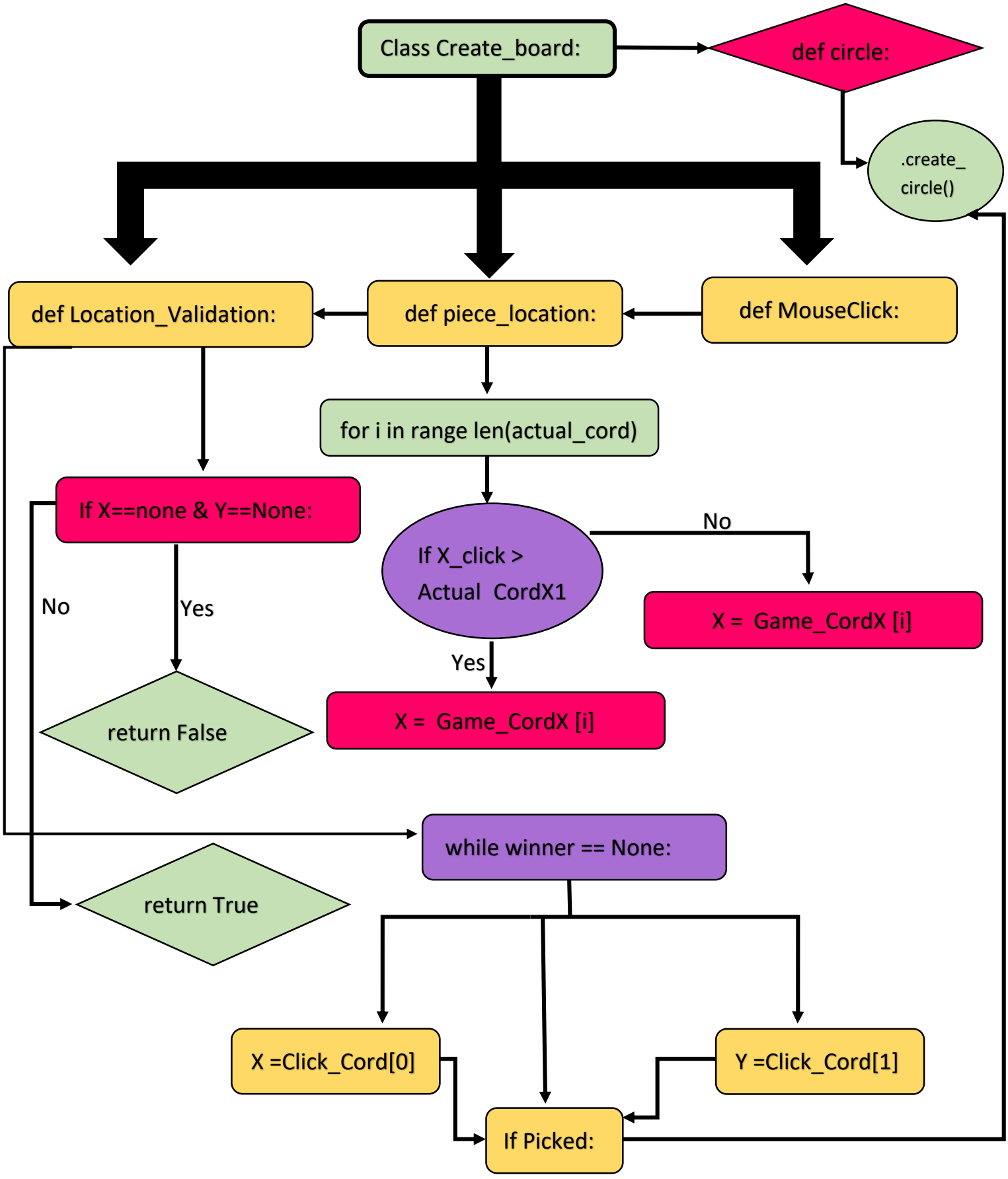
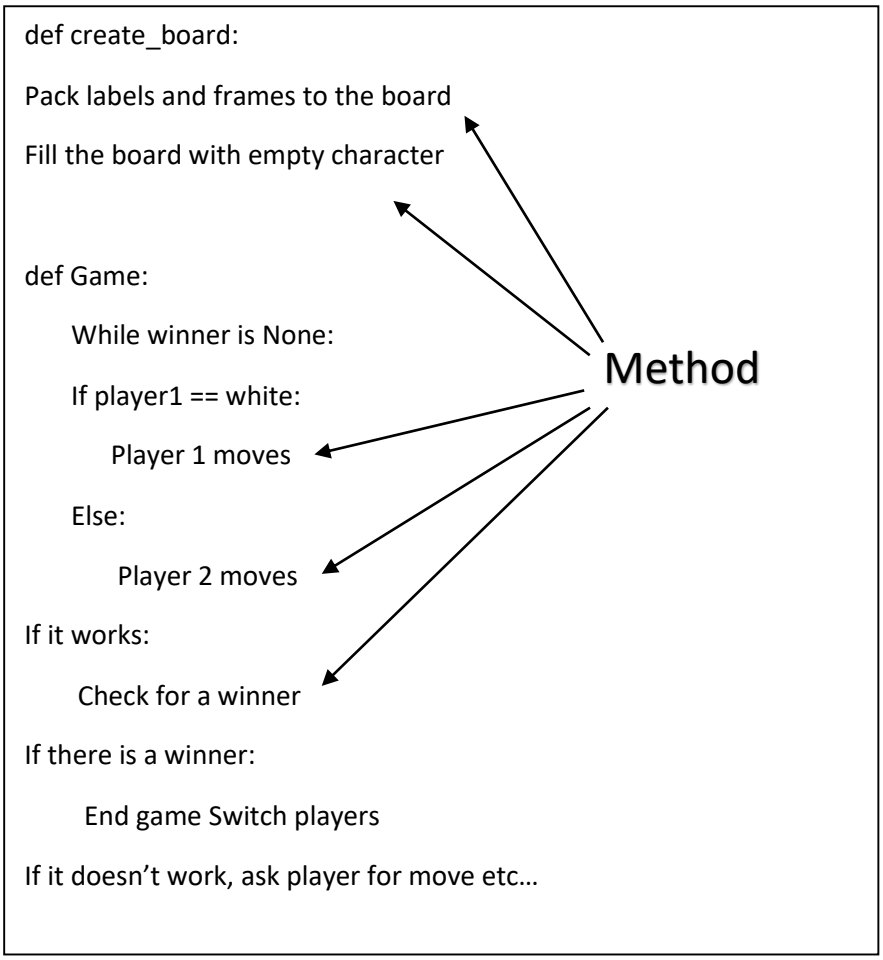


Fig 1

❖ **Algorithm:**


Class Create_board:
def circle:
.create_ circle()
def Location_Validation:
def piece_location:
def MouseClick:
for i in range len(actual_cord)
If X==none & Y==None:
If X_click > Actual CordX1
No
X = Game_CordX [i]
No
Yes
Yes
return False
X = Game_CordX [i]
while winner == None:
return True
X =Click_Cord[0]
Y =Click_Cord[1]
If Picked:

❖ Pseudocode:



❖ Design & Implementation:

Modules	Usage
From tkinter *	from Tkinter import * imports every exposed object in Tkinter into your current namespace. import Tkinter imports the "namespace" Tkinter in your namespace.
From tkinter .messagebox	Python Tkinter – Message Box Widget is used to display the message boxes in the python applications. This module is used to display a message using provides a number of functions.

Functions & Methods	Usage
.minsize & .maxsize	To define minimum & maximum size of canvas.
.title	For specifying GUI title
Frame & Label	To specify frames and labels in canvas.

.pack	To insert Labels & Frame in canvas
Canvas	To define the configurations of canvas
Close_window,Game_rule, checkeredre,start_program	User defined functions for doing different tasks
Button	Use to generate button on canvas.

❖ Reflection and Conclusions

- **Original Goals**
  My original plans for this project is implementing a game which takes user input on a specified location of his/her choice on canvas board. The inputs should be in the form of numbers (2 and 3) inside the grid. Each player will be assigned one number out of these two numbers. The algorithm should check for 5 similar numbers in a row or a column. If any number is present in a row/column of 5 the player assigned to that number wins. Along with this the user has many options like restarting the game, checking the game rules and exiting the game.

- **Accomplishments and Realization of the Plans**
  I think ¾ of my original goals have been accomplished. I have originally divided the project into several tasks. I have also realized that representing the board, as a list in a node, is definitely not the best way to keep track of the game state, I had to do a lot of index checking when translating sequences of list for pattern recognition. So far, in the project, I have made the program to perform simple user inputs inside the grid in the form of dots, as well as all functionalities associated with buttons. Because of the limited schedule I was given in the project, I did not get the chance to implement two tasks i.e.
  1) Detection a winning state.
  2) To place numbers 2 and 3 on the grid has not been accomplished yet.

- **Future Plan**
  My future plan of this project is to implement the above two points into the program for performance and strategy improvement purposes. I would like to collect statistics on the computing time difference between each player turn. I would also like to configure the winning GUI in my program.

❖ Code:

Note GitHUB Link for code: https://github.com/smTadeeb/Board_game-

```
from tkinter import *
import tkinter .messagebox

myInterface = Tk()

myInterface.minsaize(500,500)
myInterface.maxsize(700,698)

label_info = Label(text="Habibi (حبيبي)Time pass",font= "Calibri 18 bold",
bg = "orange" ,fg = "green")     ## Defined label of canvas
label_info.pack(side =TOP , anchor = "n" ,fill= X)

myInterface.title("Tadeeb's GUI")
## Defined title of canvas

f1 = Frame(myInterface,borderwidth = 5 ,bg= "goldenrod", relief = GROOVE)
f1.pack(side = BOTTOM, anchor = "s" , fill = X)

f2 = Frame(myInterface,borderwidth = 2 ,bg= "goldenrod", relief = GROOVE)
f2.pack(side = LEFT, anchor = "e" , fill = Y)


L1 = Label(f2 , text = "grid size\n\
          \n10*10 ",bg = "goldenrod",font="Calibri 14 bold italic",)
L1.pack(side = LEFT , anchor = "w" ,fill= Y)

L2 = Label(f1, text="Player1:- Habibi digit 2",font="Calibri 14 bold
italic",bg = "goldenrod" )
L2.pack( side = TOP , anchor = "s" ,fill= X,pady=3,padx=3)

L3 = Label(f1, text="Player2:- Habibi digit 3",font="Calibri 14 bold
italic",bg = "goldenrod" )
L3.pack( side = TOP , anchor = "s" ,fill= X,pady=3,padx=3)

can_widget = Canvas(myInterface, width=500, height=500, background= "gold")
# Created Canvas

can_widget.pack(fill = Y)

def close_window():
    """
    Asks the user if they really want to exit
    """
    if tkinter.messagebox.askokcancel("Exit", "Do you really wish to
Exit?"):
        myInterface.destroy()


def game_rules():
    """
    Displays game rules
    """
```

```
    messagebox.showinfo("Game Rules" , "1. It is traditionally played with
Habibi Digits (2 and 3) on a 20×20 or 10×10 Habibi board.\n\
                        \n2. Players alternatively place a digit assigned
to them (either 2 or 3) on an cell.\n\
                        \n3. The winner is the first player to form an
unbroken chain of five digits horizontally, vertically, or diagonally to
get a sum of (10 or 15) depending upon his chosen number.\n\
                        \n4. So both the players should try to place their
number in such a manner that their opponent aren't able to make 5 digits in
a row,column or diagonally.")

def checkered(canvas, line_distance):
    """
    Used to make the 10*10 grid on canvas
    """

for x in range(line_distance,500,line_distance):
       canvas.create_line(x, 0, x, 500)

for y in range(line_distance,500,line_distance):
       canvas.create_line(0, y, 500, y)

def restart_program():
    """Restarts the current program.
    Note: this function does not return. Any cleanup action (like
    saving data) must be done before calling this function."""
    python = sys.executable
    os.execl(python, python, * sys.argv)
    return restart_program()


b1 = Button(f1,fg="red",text="Exit",padx = "8",pady = "8",command =
close_window)
b1.pack(side = TOP ,anchor ="s")
b2 = Button(f1,fg="Green",text="Restart",padx = "8",pady = "8",command =
restart_program)
b2.pack(side = RIGHT ,anchor = "se" )

b3 = Button(f1,fg="Blue",text="How to Play",padx = "5",pady = "5",command =
game_rules)
b3.pack(side = LEFT ,anchor = "sw" )

checkered(can_widget,50)
#myInterface.mainloop()

#Board Size
Board_Size = int(16.2)
Frame_Gap = 35
width = 500
height = 500

def create_circle(x, y, radius, fill = "", outline = "black", width = 1):
    can_widget.create_oval(x - radius, y - radius, x + radius, y + radius,
fill = fill, outline = outline, width = width)       # Use for creating
circles

def Value_Check_int(Value):
```

```
    try:
        Value = int(Value)
    except ValueError:
        return "string"
    else:
        return "int"

def MouseClick(event):
    global Click_Cord
X_click = event.x
    Y_click = event.y
    Click_Cord = Piece_Location(X_click, Y_click)
    print(Click_Cord)

can_widget.bind("<Button-1>", MouseClick)

Click_Cord = [None, None]

def Piece_Location(X_click, Y_click):
# Checks for the location/co-ordinate where the click has been made
    X = None
    Y = None
    for i in range(len(Actual_CordX1)):

        if X_click > Actual_CordX1[i] and X_click < Actual_CordX2[i]:
            X = Game_CordX[i]

        if Y_click > Actual_CordY1[i] and Y_click < Actual_CordY2[i]:
            Y = Game_CordY[i]

    return X, Y

def Location_Validation():

    if X == None or Y == None:
# Validates whether a block exist on a perticular place or not
        return False

    elif board[Y - 1][X - 1] == 0:
        return True

#Board
Board_Size = Board_Size - 1
Board_X1 = 500 / 10
Board_Y1 = 500 / 10
Board_GapX = (500 - Board_X1 * 2) / Board_Size
Board_GapY = (500 - Board_Y1 * 2) / Board_Size

#Chess Piece
Chess_Radius = (Board_GapX * (9 / 10)) / 2

#Cord List
Black_Cord_PickedX = []
Black_Cord_PickedY = []
White_Cord_PickedX = []
White_Cord_PickedY = []
```

```
#Click Detection Cord
Game_CordX = []
Game_CordY = []
Actual_CordX1 = []
Actual_CordY1 = []
Actual_CordX2 = []
Actual_CordY2 = []

#2D Board List
board = []

#2D list for gameboard
for i in range(Board_Size + 1):
    board.append([0] * (Board_Size + 1))

Unfilled = 0
Black_Piece = 1
White_Piece = 2

#Fills Empty List
for z in range(1, Board_Size + 2):

    for i in range(1, Board_Size + 2):
        Game_CordX.append(z)
        Game_CordY.append(i)
        Actual_CordX1.append((z - 1) * Board_GapX + Board_X1 -
Chess_Radius)
        Actual_CordY1.append((i - 1) * Board_GapY + Board_Y1 -
Chess_Radius)
        Actual_CordX2.append((z - 1) * Board_GapX + Board_X1 +
Chess_Radius)
        Actual_CordY2.append((i - 1) * Board_GapY + Board_Y1 +
Chess_Radius)

#Turn
Turn_Num = 1
Turn = "black"
Winner = None

#Game Code
while Winner == None:
    can_widget.update()

    X = Click_Cord[0]
    Y = Click_Cord[1]

    Picked = Location_Validation()

    if Picked:
        create_circle(Board_X1 + Board_GapX * (X - 1), Board_Y1 +
Board_GapY * (Y - 1), radius = Chess_Radius, fill = Turn)

        if Turn_Num % 2 == 1:
            White_Cord_PickedX.append(X)
            White_Cord_PickedY.append(Y)
            board[Y - 1][X - 1] = 2
            Turn = "black"
```

```
        elif Turn_Num % 2 == 0:
            Black_Cord_PickedX.append(X)
            Black_Cord_PickedY.append(Y)
            board[Y - 1][X - 1] = 1
            Turn = "white"

myInstance.mainloop()
```

❖ Output: