**PROJECT NAME: CONSOLE BASED LUDO GAME**

**CODE:** (comments in green colour)

**painter.py:**

from copy import deepcopy #creates new object & recursively adds copies of nested objects present in original elements.

from os import linesep #used to separate lines in windows

# board template (matrix)

BOARD\_TMPL = [['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '#', ' ', '|', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'Y', 'E', 'L', 'L', 'O', 'W', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', ' ', 'V', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'B', 'L', 'U', 'E', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', '-', '-', '>', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'], ['#', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '#'], ['#', '-', '-', '-', '-', '-', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '-', '-', '-', '-', '-', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '-', '-', '-', '-', '-', '#'], ['#', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', 'X', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '#'], ['#', '-', '-', '-', '-', '-', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '-', '-', '-', '-', '-', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '-', '-', '-', '-', '-', '#'], ['#', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '#'], ['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '<', '-', '-', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'G', 'R', 'E', 'E', 'N', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'R', 'E', 'D', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '^', ' ', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', '-', '-', '-', '-', '-', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '|', ' ', '#', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '|', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'], ['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']]

‘’’In the below List of two sized tuples, The content of tuple correspond with address of matrix denoted via BOARD\_TMPL. While list index correspond with pawn sharing position with each other from board class’’’

CODE\_COMMON\_SQUARES = [(), # 0 index not used(14, 2), (14, 8), (14, 14), (14, 20), (14, 26), (14, 32), (14, 38),(12, 38), (10, 38), (8, 38), (6, 38), (4, 38), (2, 38), (2, 44), (2, 50), (4, 50), (6, 50), (8, 50), (10, 50), (12, 50), (14, 50),(14, 56), (14, 62), (14, 68), (14, 74), (14, 80), (14, 86), (16, 86),(18, 86), (18, 80), (18, 74), (18, 68), (18, 62), (18, 56), (18, 50),(20, 50), (22, 50), (24, 50), (26, 50), (28, 50), (30, 50), (30, 44),(30, 38), (28, 38), (26, 38), (24, 38), (22, 38), (20, 38), (18, 38),(18, 32), (18, 26), (18, 20), (18, 14), (18, 8), (18, 2), (16, 2)]

‘’’In the below dictionary, tuple correspond with address of matrix BOARD\_TMPL, color correspond to pawn colour and the index of colour's list correspond with pawn private (final) position’’’

CODE\_COLOUR\_SQUARES = {

'yellow': [(), (16, 8), (16, 14), (16, 20), (16, 26), (16, 32), (16, 38)],

'blue': [(), (4, 44), (6, 44), (8, 44), (10, 44), (12, 44), (14, 44)],

'red': [(), (16, 80), (16, 74), (16, 68), (16, 62), (16, 56), (16, 50)],

'green': [(), (28, 44), (26, 44), (24, 44), (22, 44), (20, 44), (18, 44)]

}

‘’’In the below dictionary, the tuple correspond with address of matrix BOARD\_TMPL, color correspond to pawn colour and the index of colour's list correspond with pawn initial position’’’

CODE\_POOL\_PLACES = {

'yellow': [(), (6, 14), (6, 19), (8, 14), (8, 19)],

'blue': [(), (6, 71), (6, 76), (8, 71), (8, 76)],

'red': [(), (24, 71), (24, 76), (26, 71), (26, 76)],

'green': [(), (24, 14), (24, 19), (26, 14), (26, 19)]

}

class PaintBoard():

def \_\_init\_\_(self): #init is for constructor and self is used to access attributes of a class

self.board\_tmpl\_curr = deepcopy(BOARD\_TMPL)

#adding the copies of nested objects from BOARD\_TMPL

def \_place\_pawn(self, pawn, position, offset): # Used to initialise the position

common\_poss, private\_poss = position #assigning the initial values

colour = pawn.colour.lower() #Assigning initial values

if private\_poss > 0: # condition to check if the pawn is in private squares

row, column = CODE\_COLOUR\_SQUARES[colour][private\_poss]

elif common\_poss == 0: # Condition to check whether the pawn is in its own pool

row, column = CODE\_POOL\_PLACES[colour][pawn.index]

offset = 0 # we do not need offset in the pool we only need it in squares

else: # condition when pawn is in common squares

row, column = CODE\_COMMON\_SQUARES[common\_poss]

if offset > 0: #condition when pawn is outside the pool

self.board\_tmpl\_curr[row - 1][column + offset] = pawn.id[1]

else: # condition to move the pawn across the board

self.board\_tmpl\_curr[row - 1][column - 1] = pawn.id[0]

self.board\_tmpl\_curr[row - 1][column] = pawn.id[1]

# condition to move the pawn across the board

def \_place\_pawns(self, position\_pawns): #Function to position the pawns in the board

for position, pawns in position\_pawns.items(): #Outer loop to access the position

for index, pawn in enumerate(pawns): #assigning index to the pawn

self.\_place\_pawn(pawn, position, index)#assigning the value

def paint(self, position):

'''expect dictionary of key - occupied positions and value - list of pawns on that position'''

self.board\_tmpl\_curr = deepcopy(BOARD\_TMPL)

#adding the copies of nested objects from BOARD\_TMPL

self.\_place\_pawns(position)

board\_paint = [''.join(row\_list) for row\_list in self.board\_tmpl\_curr]

board\_paint\_str = linesep.join(board\_paint)

return board\_paint\_str

#Defining the current situation of the pawn on the board

def present\_6\_die\_name(number, name): #nicer print of die and name of the player

hor\_line = 9 \* '-' #Creating the die pattern

sps = 37 \* ' ' #Creating space on the die

hor\_line = sps + hor\_line #combining the above 2 patterns

matrix = [['| |', '| # |', '| |'],

['| |', '| # # |', '| |'],

['| # |', '| # |', '| # |'],

['| # # |', '| |', '| # # |'],

['| # # |', '| # |', '| # # |'],

['| # # # |', '| |', '| # # # |']] #Displaying the numbers on the die via patterns

matrix = [[sps + cell for cell in row] for row in matrix] #Adding numbers on the die

die = matrix[number - 1] #number on the die

die[1] = die[1] + " " + name

s = linesep.join([hor\_line] + die + [hor\_line]) #Displaying the final representation of the number

return s

**game.py:**

from collections import namedtuple, deque # Creating tuple subclass with named fields

import random #to generate random numbers

from .painter import PaintBoard #Importing elements of the previous file(painter.py)

Pawn = namedtuple("Pawn", "index colour id") #creating a sub class named pawn

class Player():

'''Player Knows (holds) his pawns, also know his colour and choose which pawn to move if more than one are possible'''

def \_\_init\_\_(self, colour, name=None, choose\_pawn\_delegate=None):

'''choose\_pawn\_delegate is callable. if choose\_pawn\_delegate is not None it is called with argument list of available pawns to move and expect chosen index from this list if it is None (means computer) random index is chosen '''

self.colour = colour #Assigning default values

self.choose\_pawn\_delegate = choose\_pawn\_delegate #Assigning default values

self.name = name #Assigning default values

if self.name is None and self.choose\_pawn\_delegate is None:

#Condition when no information is provided by the user

self.name = "computer" #Default values

self.finished = False

#for the loop & variable below initialize 4 pawns with id(first letter from colour and index(from 1-4))

self.pawns = [Pawn(i, colour, colour[0].upper() + str(i))

for i in range(1, 5)]

def \_\_str\_\_(self):

return "{}({})".format(self.name, self.colour) #Returning the name and colour of pawn

def choose\_pawn(self, pawns):

#Delegate choice to choose\_pawn\_delegate func attribute if it is not None

if len(pawns) == 1:

index = 0

elif len(pawns) > 1:

if self.choose\_pawn\_delegate is None:

index = random.randint(0, len(pawns) - 1)

#Generating a random number between the given arguments

else:

index = self.choose\_pawn\_delegate() #getting value from user

return index

class Board():

'''Knows where are pawns. Pawns are assigned with position numbers. Can move (change position number) pawn. Knows other things like what distance pawn must past to reach end. It just board. It does not know rules of the game. '''

# common (shared) squares for all pawns

BOARD\_SIZE = 56

# save (private) positions (squares) for each colour

# This is squares just before pawn finished

BOARD\_COLOUR\_SIZE = 7

COLOUR\_ORDER = ['yellow', 'blue', 'red', 'green']

‘’’distance between two neighbour colours(The distance from start square of one colour to start square of next colour)’’’

COLOUR\_DISTANCE = 14

def \_\_init\_\_(self): # get dictionary of start position for every colour

Board.COLOUR\_START = {

colour: 1 + index \* Board.COLOUR\_DISTANCE for

index, colour in enumerate(Board.COLOUR\_ORDER)}

# get dictionary of end position for every colour

Board.COLOUR\_END = {

colour: index \* Board.COLOUR\_DISTANCE

for index, colour in enumerate(Board.COLOUR\_ORDER)}

Board.COLOUR\_END['yellow'] = Board.BOARD\_SIZE

# dictionary where key is pawn and value is two size tuple holds position

# Position is combination of common (share) square and coloured (private) square.

self.pawns\_possiotion = {} # painter is used to visually represent

self.painter = PaintBoard() # the board and position of the pawns

self.board\_pool\_position = (0, 0) # pool means position before start of the game

def set\_pawn(self, pawn, position): #save position of the pawn

self.pawns\_possiotion[pawn] = position

def put\_pawn\_on\_board\_pool(self, pawn): #when dice rolls a 6 and the pawn is out of the pool

self.set\_pawn(pawn, self.board\_pool\_position)

def is\_pawn\_on\_board\_pool(self, pawn): #return True of False

return self.pawns\_possiotion[pawn] == self.board\_pool\_position

def put\_pawn\_on\_starting\_square(self, pawn): #Positioning the pawn on the first square

start = Board.COLOUR\_START[pawn.colour.lower()] #starting the game

position = (start, 0) #Initial position of the pawn

self.set\_pawn(pawn, position)

def can\_pawn\_move(self, pawn, rolled\_value):

#check if pawn can move outside board and colour size

common\_poss, private\_poss = self.pawns\_possiotion[pawn]

if private\_poss + rolled\_value > self.BOARD\_COLOUR\_SIZE:

#condition to check position > colour size

return False

return True

def move\_pawn(self, pawn, rolled\_value):

#change pawn position, check if pawn reach his color square

common\_poss, private\_poss = self.pawns\_possiotion[pawn]

end = self.COLOUR\_END[pawn.colour.lower()]

if private\_poss > 0: # pawn is already reached own final squares

private\_poss += rolled\_value

elif common\_poss <= end and common\_poss + rolled\_value > end:

#pawn entering in own square

private\_poss += rolled\_value - (end - common\_poss)

common\_poss = end

else: # pawn will be still in common square

common\_poss += rolled\_value #Adding number on the dice to current position

if common\_poss > self.BOARD\_SIZE: #condition to find the position pawn should be at

common\_poss = common\_poss - self.BOARD\_SIZE

position = common\_poss, private\_poss

self.set\_pawn(pawn, position)

def does\_pawn\_reach\_end(self, pawn): #if pawn must leave game

common\_poss, private\_poss = self.pawns\_possiotion[pawn]

if private\_poss == self.BOARD\_COLOUR\_SIZE: #if the pawn has reached the final square

return True

return False

def get\_pawns\_on\_same\_postion(self, pawn): #return list of pawns on same position

position = self.pawns\_possiotion[pawn] #assigning values to the variable

return [curr\_pawn for curr\_pawn, curr\_postion in self.pawns\_possiotion.items()

if position == curr\_postion]

def paint\_board(self):

#painter object expect dictionary of key - occupied positions and value - list of pawns on that position

positions = {} #creating empty dictionary

for pawn, position in self.pawns\_possiotion.items():

common, private = position #checking where the pawn is situated

if not private == Board.BOARD\_COLOUR\_SIZE:

positions.setdefault(position, []).append(pawn) #moving the pawn new position

return self.painter.paint(positions)

class Die():

MIN = 1 #minimum number a die can roll

MAX = 6 #maximum number a die can roll

@staticmethod #creating a decorator

def throw():

return random.randint(Die.MIN, Die.MAX) # generating a random number between 1 and 6

class Game():

#Knows the rules of the game. Knows for example what to do when one pawn reach another or pawn reach end or player roll six and so on

def \_\_init\_\_(self):

self.players = deque()

self.standing = []

self.board = Board()

self.finished = False # is game finished

self.rolled\_value = None # last rolled value from die (dice)

self.curr\_player = None # player who last rolled die

self.allowed\_pawns = [] # curr\_player's possible pawn to move

self.picked\_pawn = None # curr\_player's chosen pawn to move

self.index = None # chosen index from allowed pawn

self.jog\_pawns = [] # jog pawn if any

def add\_palyer(self, player): #Adding players to the game

self.players.append(player)

for pawn in player.pawns:

self.board.put\_pawn\_on\_board\_pool(pawn) # putting the no of players(pawns) on the board

def get\_available\_colours(self): #if has available colour on boards

used = [player.colour for player in self.players] #used pawn colour

available = set(self.board.COLOUR\_ORDER) - set(used) # available pawn colour

return sorted(available)

def \_get\_next\_turn(self):

#Get next player's turn. It is underscore because if called outside the class will break order

if not self.rolled\_value == Die.MAX: # if 6 is not rolled on the die

self.players.rotate(-1)

return self.players[0] #pawn wouldn’t move out of the pool

def get\_pawn\_from\_board\_pool(self, player): #when pawn must start

for pawn in player.pawns:

if self.board.is\_pawn\_on\_board\_pool(pawn): #condition to check if pawn still in pool

return pawn

def get\_allowed\_pawns\_to\_move(self, player, rolled\_value):

#return all pawns of a player which rolled value from die allowed to move the pawn

allowed\_pawns = [] #creating an empty list

if rolled\_value == Die.MAX: #if we get a 6 on the die

pawn = self.get\_pawn\_from\_board\_pool(player) #pawn comes out of the pool

if pawn:

allowed\_pawns.append(pawn) #moving the pawn head

for pawn in player.pawns:

if not self.board.is\_pawn\_on\_board\_pool(pawn) and\

self.board.can\_pawn\_move(pawn, rolled\_value):

allowed\_pawns.append(pawn) #if pawn is already out of the pool, moving it ahead

return sorted(allowed\_pawns, key=lambda pawn: pawn.index) #returning the desired value

def get\_board\_pic(self): #getting the picture of the board

return self.board.paint\_board()

def \_jog\_foreign\_pawn(self, pawn):

pawns = self.board.get\_pawns\_on\_same\_postion(pawn) #when 2 pawns on the same position

for p in pawns:

if p.colour != pawn.colour: #if the pawns are of different colour

self.board.put\_pawn\_on\_board\_pool(p)

#putting the pawn back in its pool, after being captured

self.jog\_pawns.append(p)

def \_make\_move(self, player, pawn):

'''tell the board to move pawn. After move ask board if pawn reach end or jog others pawn. Check if pawn and player finished.'''

if self.rolled\_value == Die.MAX and\

self.board.is\_pawn\_on\_board\_pool(pawn):

self.board.put\_pawn\_on\_starting\_square(pawn)

self.\_jog\_foreign\_pawn(pawn)

#if the die rolls to 6, pawn is out of the pool and it is sent to the function jog\_foriegn\_pawn

return

self.board.move\_pawn(pawn, self.rolled\_value)

if self.board.does\_pawn\_reach\_end(pawn): #reached the final position

player.pawns.remove(pawn) #remove the pawn from the board

if not player.pawns:

self.standing.append(player)

self.players.remove(player)

if len(self.players) == 1:

self.standing.extend(self.players)

self.finished = True #game finished

else:

self.\_jog\_foreign\_pawn(pawn) #if not then keep playing

def play\_turn(self, ind=None, rolled\_val=None):

'''this is main method which must be used to play game.Method ask for next player's turn, roll die, ask player to choose pawn, move pawn. ind and rolled\_val are suitable to be used when game must be replicated (recorded) ind is chosen index from allowed pawns '''

self.jog\_pawns = [] #creating an empty list

self.curr\_player = self.\_get\_next\_turn() # giving the turn to the next player

if rolled\_val is None:

self.rolled\_value = Die.throw() #throw die again

else:

self.rolled\_value = rolled\_val #the value on the die will be taken into consideration

self.allowed\_pawns = self.get\_allowed\_pawns\_to\_move(self.curr\_player, self.rolled\_value)

#moving the pawn ahead with the given number of steps on the die

if self.allowed\_pawns:

if ind is None:

self.index = self.curr\_player.choose\_pawn(self.allowed\_pawns) #running the old pawn

else:

self.index = ind

self.picked\_pawn = self.allowed\_pawns[self.index] #running the pawn with the given index

self.\_make\_move(self.curr\_player, self.picked\_pawn) #moving the pawn

else:

self.index = -1

self.picked\_pawn = None #default values

**recorder.py:**

import pickle #used to serialize an object

from .game import Player # importing Player from game.py

class RunRecord(): #provide recorded game data iterating over instance yield rolled\_value and index

def \_\_init\_\_(self, file\_obj):

self.file\_obj = file\_obj

data = pickle.load(self.file\_obj) #loading the data from the recorded game

self.players = data[0]

self.game\_history = data[1] #assigning the values to variables

def get\_players(self, func=None):

#return Player object recreated from a list function is callable which player may need for choice delegation

res = [] #creating an empty list

for colour, name, is\_computer in self.players: #inputing choices from the computer

if is\_computer:

player = Player(colour)

else:

player = Player(colour, name, func) #if not computer, take values from user

res.append(player) #input the values provided

return res

def get\_game\_history(self):

return self.game\_history #returning the history of the game

def \_\_iter\_\_(self):

return iter(self.game\_history)

class MakeRecord(): #save game data as a nested list which is saved with pickle

def \_\_init\_\_(self):

self.players = [] #creating empty list

self.game\_history = [] #creating empty list

def add\_player(self, player\_obj): #Accept Player object and it save NOT as object rather as a list

if player\_obj.choose\_pawn\_delegate is None: #if user chooses computer or no input

is\_computer = True

else:

is\_computer = False

self.players.append((player\_obj.colour,player\_obj.name, is\_computer))

def add\_game\_turn(self, rolled\_value, index):

self.game\_history.append((rolled\_value, index))

def save(self, file\_obj): #list of lists with players and game history

pickle.dump([self.players, self.game\_history], file\_obj)

**cli.py:**

from .game import Player, Game #importing player, Game from game.py

from .painter import present\_6\_die\_name #importing \_6\_die\_name from painter.py

from .recorder import RunRecord, MakeRecord #import RunRecord, MakeRecord from recorder.py

from os import linesep #using line separation function from os module

class CLIGame(): #creating class

def \_\_init\_\_(self): #setting default/previous values

self.prompt\_end = "> "

self.game = Game() # used for nicer print

self.prompted\_for\_pawn = False # saving game data

self.record\_maker = MakeRecord() # getting game data

self.record\_runner = None

def validate\_input(self, prompt, desire\_type, allawed\_input=None, error\_mess="Invalid Option!", str\_len=None):

‘’’loop while receive correct value parameter allowed\_input can be list of allowed value parameter str\_len is two sized tuple if min and max’’’

prompt += linesep + self.prompt\_end #setting the value of prompt

while True:

choice = input(prompt) #accessing the input

if not choice:

print(linesep + error\_mess) #printing error message due to incorrect input

continue

try:

choice = desire\_type(choice)

except ValueError:

print(linesep + error\_mess) #printing error message

continue

if allawed\_input:

if choice in allawed\_input: #if the input is from the given choices

break

else:

print("Invalid Option!") #error message due to incorrect input

continue

elif str\_len:

min\_len, max\_len = str\_len

if min\_len < len(choice) < max\_len: #setting the range of length of choice

break

else:

print(linesep + error\_mess) #error message due to incorrect input

else:

break

print()

return choice

def get\_user\_initial\_choice(self): #asking for user to input his choice

text = linesep.join(["choose option",

"0 - start new game",

"1 - continue game",

"2 - run (review) recorded game"]) #printing the choices

choice = self.validate\_input(text, int, (0, 1, 2)) #validating the input from user

return choice

def prompt\_for\_file(self, mode="rb"): #return file descriptor ;rb: read and binary

text = "Enter filename (name of the record)" #previous filename that you want to view

while True:

filename = self.validate\_input(text, str) #condition if a valid filename

try:

file\_descr = open(filename, mode=mode) #opening the game in the given file

return file\_descr

except IOError as e: #condition if wrong filename is provided

print(e)

print("Try again") #error message

def does\_user\_want\_save\_game(self): #return True if user want to save game or False

text = linesep.join(["Save game?",

"0 - No",

"1 - Yes"]) #printing the choices

choice = self.validate\_input(text, int, (0, 1)) #validating the input

return choice == 1

def prompt\_for\_player(self):

#get player attributes from input, initial player instance and add player to the game

available\_colours = self.game.get\_available\_colours()

text = linesep.join(["choose type of player",

"0 - computer",

"1 - human"]) #printing the choices

choice = self.validate\_input(text, int, (0, 1)) #validating the input

if choice == 1:

name = self.validate\_input("Enter name for player",str, str\_len=(1, 30)) #input player name

available\_options = range(len(available\_colours)) #only the unchosen colours left

if len(available\_options) > 1: # show available colours

options = ["{} - {}".format(index, colour)

for index, colour in

zip(available\_options, available\_colours)] #generating the available colour tuple

text = "choose colour" + linesep

text += linesep.join(options)

choice = self.validate\_input(text, int, available\_options) #validating the input

colour = available\_colours.pop(choice) #removing the chosen colour from the tuple

else: # only one colour left

colour = available\_colours.pop()

player = Player(colour, name, self.prompt\_choose\_pawn)

elif choice == 0:

colour = available\_colours.pop() # automatically assign colours

player = Player(colour)

self.game.add\_palyer(player) #last player

def prompt\_for\_players(self):

counts = ("first", "second", "third", "fourth last") #put all players in the game

text\_add = "Add {} player"

for i in range(2): #to add players if space available

print(text\_add.format(counts[i]))

self.prompt\_for\_player()

print("Player added") #printing the message

text = linesep.join(["Choose option:",

"0 - add player",

"1 - start game with {} players"]) #printing the choices

for i in range(2, 4):

choice = self.validate\_input(text.format(str(i)), int, (0, 1)) #validating the input

if choice == 1:

break

elif choice == 0:

print(text\_add.format(counts[i])) #if user wants to add more players

self.prompt\_for\_player()

print("Player added") #displaying the message

def prompt\_choose\_pawn(self):

#used when player (human) has more than one possible pawn to move. This method is pass as a callable during player instantiation

text = present\_6\_die\_name(self.game.rolled\_value, str(self.game.curr\_player))

text += linesep + "has more than one possible pawns to move." #assigning the message

text += " Choose pawn" + linesep #assigning the message

pawn\_options = ["{} - {}".format(index + 1, pawn.id)

for index, pawn

in enumerate(self.game.allowed\_pawns)]

# if there are more than one pawn possible for movement

text += linesep.join(pawn\_options)

index = self.validate\_input(text, int, range(1, len(self.game.allowed\_pawns) + 1)) #validate input

self.prompted\_for\_pawn = True

return index - 1

def prompt\_to\_continue(self):

text = "press Enter to continue" + linesep #printing the message

input(text) #taking ‘enter’ command as input

def print\_players\_info(self):

word = "start" if self.game.rolled\_value is None else "continue"

#condition to start or continue game

print("Game {} with {} players:".format(word, len(self.game.players))) #printing the game info

for player in self.game.players:

print(player) #printing player information

print()

def print\_info\_after\_turn(self):

pawns\_id = [pawn.id for pawn in self.game.allowed\_pawns]#used game attributes to print info

message = present\_6\_die\_name(self.game.rolled\_value,str(self.game.curr\_player))

# nicer print of dice

message += linesep

if self.game.allowed\_pawns:

message\_moved = "{} is moved. ".format(self.game.picked\_pawn.id)

#assigning message that which pawn has been moved

if self.prompted\_for\_pawn:

self.prompted\_for\_pawn = False

print(message\_moved)

return

message += "{} possible pawns to move.".format(" ".join(pawns\_id))

#assigning the desired messages

message += " " + message\_moved

if self.game.jog\_pawns: #if pawn needs to be moved

message += "Jog pawn " #assigning message

message += " ".join([pawn.id for pawn in self.game.jog\_pawns])

else: #if no possible movement available

message += "No possible pawns to move." #assigning the given message

print(message) #printing the above mentioned appropriate message

def print\_standing(self):

standing\_list = ["{} - {}".format(index + 1, player)

for index, player in enumerate(self.game.standing)]

#returning a tuple with a counter and value

message = "Standing:" + linesep + linesep.join(standing\_list) #assigning the given message

print(message) #printing the appropriate message

def print\_board(self):

print(self.game.get\_board\_pic()) #printing the picture of the board

def run\_recorded\_game(self):

#get history of game (rolled\_value & index's allowed pawn) from record\_runner in order to replay game

self.load\_recorded\_players()

self.print\_players\_info() #printing players information

self.prompt\_to\_continue() #continuing the game

for rolled\_value, index in self.record\_runner:

self.game.play\_turn(index, rolled\_value) #accessing the rolled value and index

self.print\_info\_after\_turn()

self.print\_board()

self.prompt\_to\_continue() #continuing the game

self.print\_board() #printing the board

def continue\_recorded\_game(self):

#move forward the game by calling play\_turn method to the moment where game was interrupted.

self.load\_recorded\_players()

self.record\_players() #calling the function that has the record of the players

for rolled\_value, index in self.record\_runner:

self.game.play\_turn(index, rolled\_value)

self.record\_maker.add\_game\_turn( self.game.rolled\_value, self.game.index)

#checking if the rolled value is greater than the recorded value

self.print\_players\_info()

self.print\_info\_after\_turn() #printing the information after turn completion

self.print\_board()

def record\_players(self): #save players on recorder

for player in self.game.players:

self.record\_maker.add\_player(player) #adding the player

def load\_recorded\_players(self):

#get recorded (save) players from recorder and put them in game.

if self.record\_runner is None:

file\_descr = self.prompt\_for\_file() # using recorded file to access the data

self.record\_runner = RunRecord(file\_descr) #running the data

file\_descr.close()

for player in self.record\_runner.get\_players(self.prompt\_choose\_pawn): #to add more players

self.game.add\_palyer(player)

def load\_players\_for\_new\_game(self): #new game

self.prompt\_for\_players()

self.print\_players\_info() #printing the information of the players

self.record\_players()

def play\_game(self):

#mainly calling play\_turn Game's method while game finished

try:

while not self.game.finished: #if game is not finished

self.game.play\_turn()

self.print\_info\_after\_turn() #printing the board info after the turn

self.print\_board()

self.record\_maker.add\_game\_turn(self.game.rolled\_value, self.game.index)

self.prompt\_to\_continue() #function to continue the game

print("Game finished") #printing the message

self.print\_standing()

self.offer\_save\_game() #saving the game

except (KeyboardInterrupt, EOFError): #after pressing a different key on the keyboard

print(linesep +

"Exiting game. " +

"Save game and continue same game later?") #printing the choices

self.offer\_save\_game()

raise

def offer\_save\_game(self): #offer user save game after the game is played

if self.does\_user\_want\_save\_game():

file\_descr = self.prompt\_for\_file(mode="wb") #write and binary

self.record\_maker.save(file\_descr) #saving the game played

file\_descr.close()

print("Game is saved") #printing the message

def start(self): #main method, starting cli

print()

try:

choice = self.get\_user\_initial\_choice()

if choice == 0: # start new game

self.load\_players\_for\_new\_game()

self.play\_game()

elif choice == 1: # continue game

self.continue\_recorded\_game() #if there is a prior unfinished game

if self.game.finished: #if no unfinished game is found

print("Could not continue.",

"Game is already finished",

linesep + "Exit") #printing the output statement

else: #if there is an unfinished game

self.prompt\_to\_continue() #the unfinished game continues

self.play\_game()

elif choice == 2: # review played game

self.run\_recorded\_game()

except (KeyboardInterrupt, EOFError): #to exit the game press any key except enter

print(linesep + "Exit Game")

if \_\_name\_\_ == '\_\_main\_\_': #condition to start the game

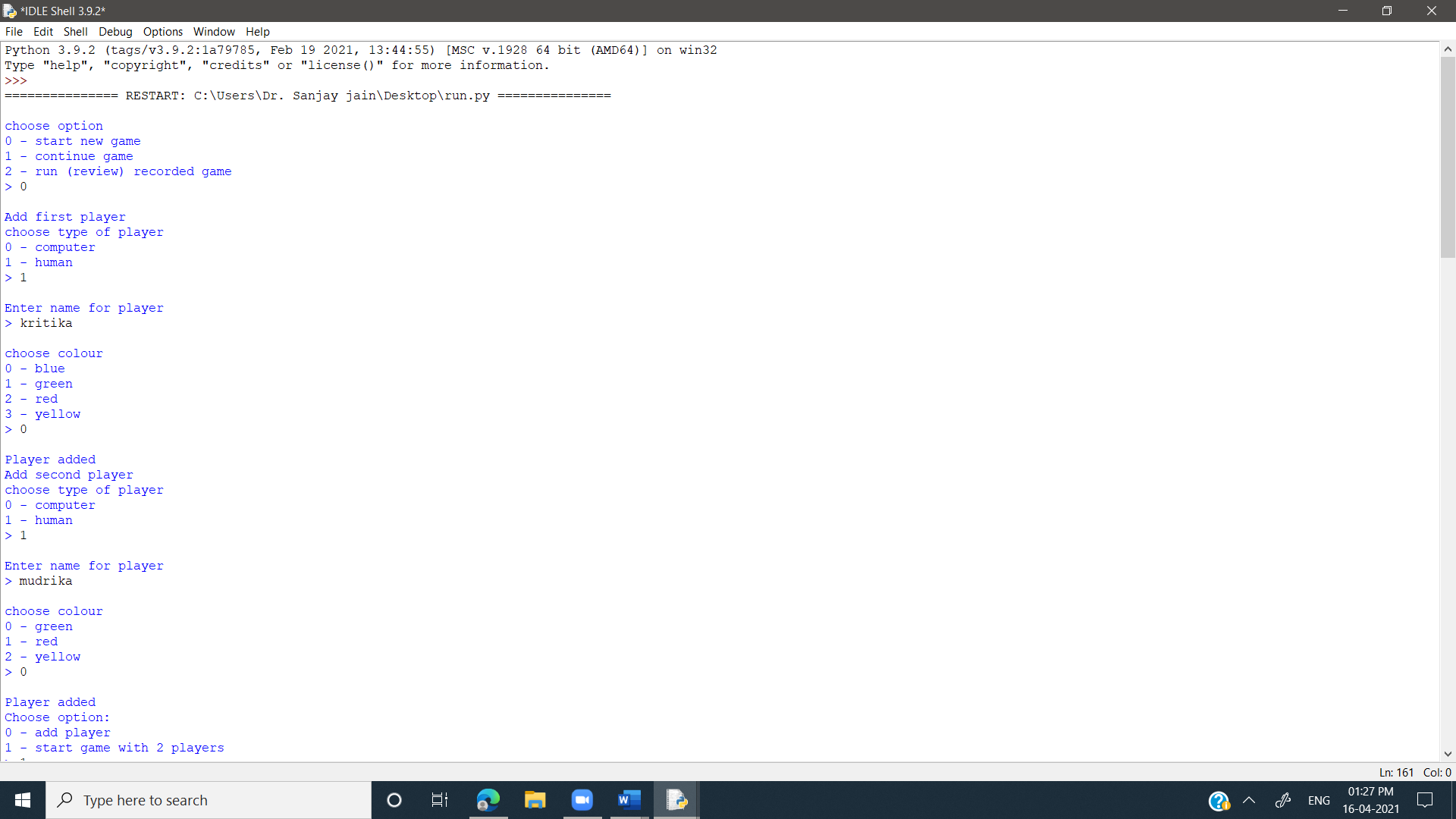
CLIGame().start()

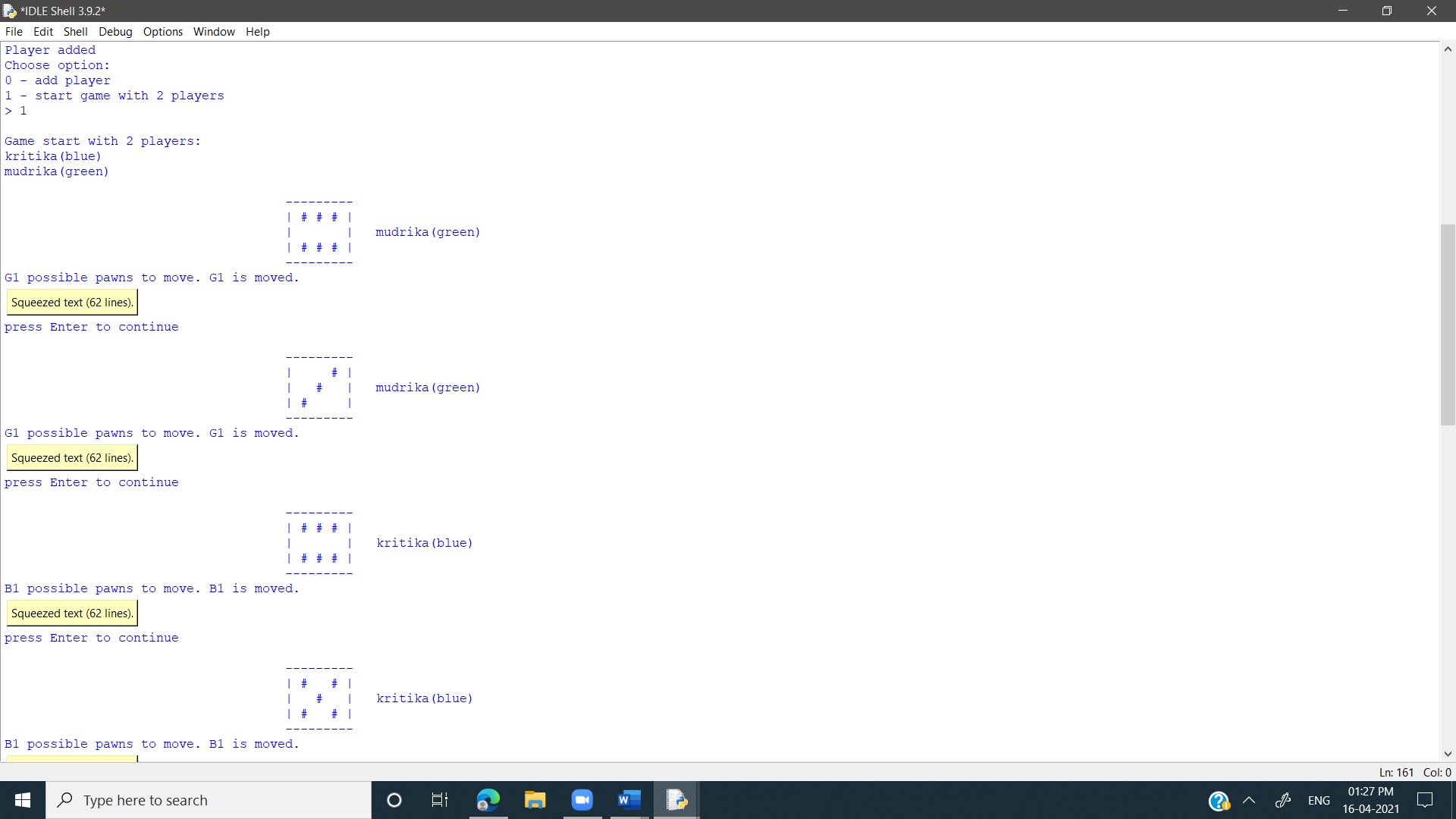
**run.py:**

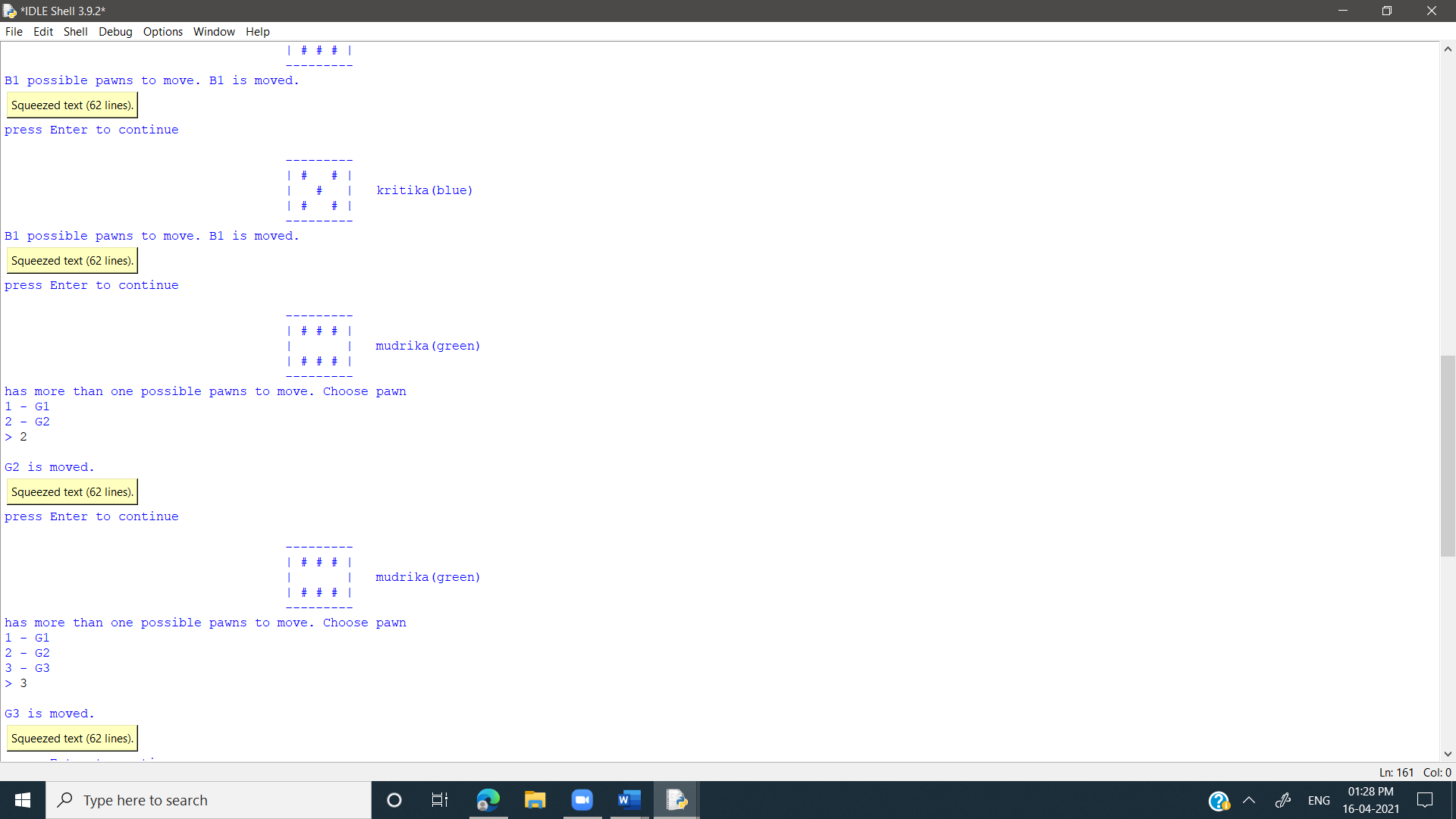
from ludo.cli import CLIGame #importing elements from cli.py

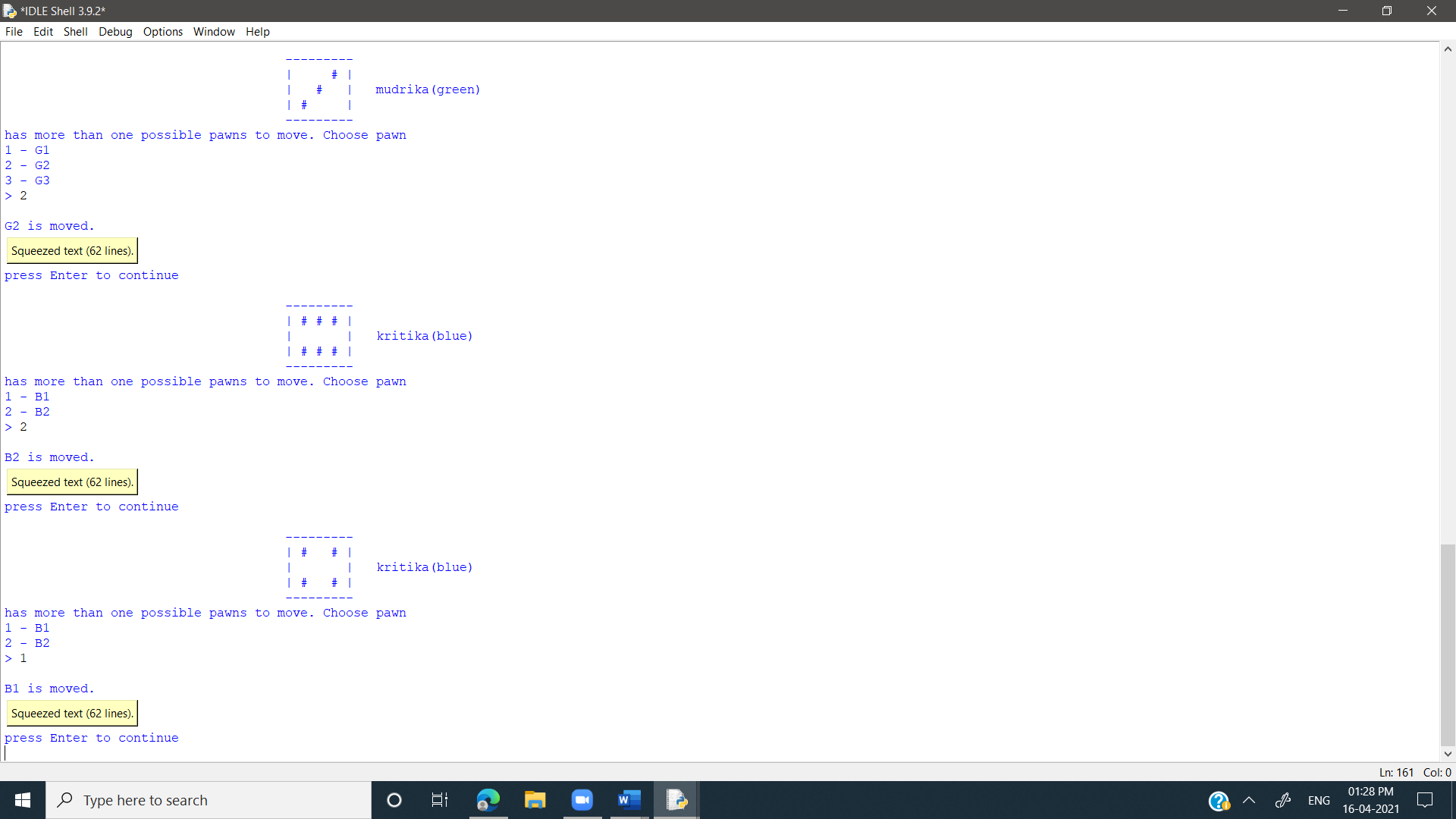
CLIGame().start() #Running the ludo game

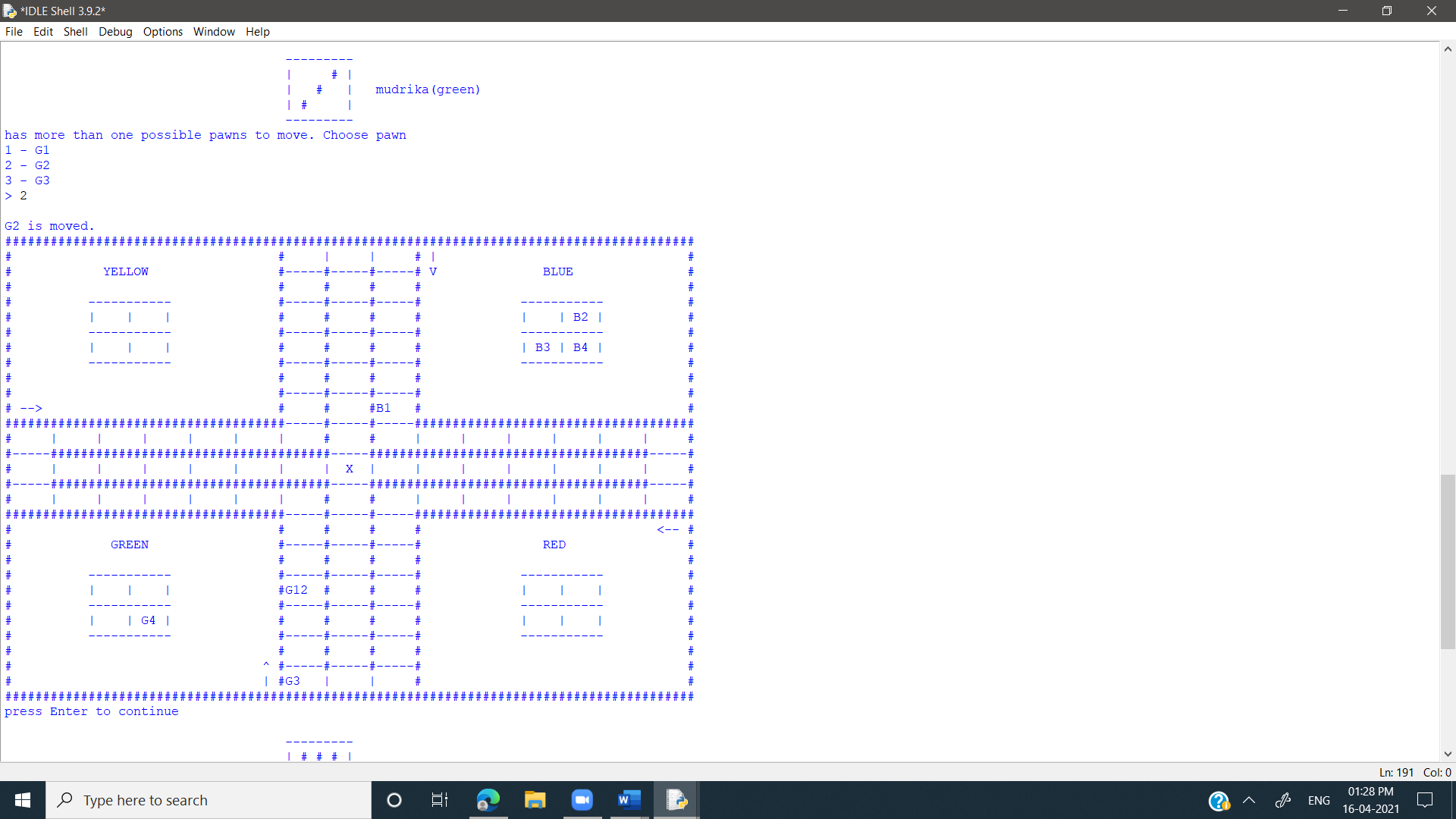
**SNAPSHOT OF THE OUTPUT:**











**\*\*\*\*\*\*\*\*\*\*\***