Software Development Project Report

Ludo Game in Python

Submitted by -

Priyanshu Srivastava(22101036)

Dhyanendra Tripathi(221010218)

Arpit Kumar Sinha(221010211)

Introduction

This **Ludo Game** project is written in Python.This project contains four python scripts which are runner.py , game.py , painter.py and run.py. The code has been broken in four scripts in order to increase simplicity and to break down the main project into four main sections.This is a simple console based strategy board game which is very easy to interact and use. Talking about the gameplay, all the playing rules are the same just like we play in real time ludo. Here at first, the user has to select players i.e either human or computer. After selecting human as a player, the player has to enter details such as name and select colour(red, green, yellow and blue). The player can also start the game within two players if he/she wants.

After starting the game, a ludo board appears on the console. Now, the rules of the game are the same as the standard ludo game. First, the computer or the player has to roll the dice. The main thing in this console based game is that the player just has to press “Enter Key” to roll the dice. At the top of the board, the code displays a dice face like GUI representing a random number generated by the code .The game loop keeps on running till it finds a possible move or it feels a dilemma when more than one possibility of moving pieces exists. All the game movements are performed by the code in real time. The board design is simple, resembling the original board used in ludo. The interface between the code and the program is so simple that users will understand it easily.

Methodology

Inbuilt modules used:

i)Copy Module: For copying objects that are mutable or contain mutable items, a copy is sometimes needed so one can change one copy without changing the other.In order to make these copies, we use the copy module. In the case of deep copy, a copy of the object is copied into another object. It means that **any changes** made to a copy of the object **do not reflect in the original object.**

ii)os.linesep: The os module contains different separator constants that indicate different file separators used in different operating systems.The os.linesep indicates the character used by the operating system to terminate lines.

iii)random:random.random() function generates random floating numbers in the range[0.1, 1.0).

iv)collections(namedtuple): The collection Module in Python provides different types of containers. A Container is an object that is used to store different objects and provide a way to access the contained objects and iterate over them.A NamedTuple returns a tuple object with names for each position which the ordinary tuples lack.

v)collections(deque): Deque (Doubly Ended Queue) is the optimised list for quicker append and pop operations from both sides of the container.

Function description:

choose_pawn( ): chooses the pawn to move for the computer when it has more than one option

set_pawn( ): saves the position of pawn

put_pawn_on_board_pool( ): puts the pawn in the unopened section(board pool)

is_pawn_on_board_pool( ):checks if there are pawns in the unopened section

put_pawn_on_starting_square( ): takes a pawn from the board pool and puts it on the starting square

move_pawn( ): changes the pawn position, checks if pawn reached its colour space

does_pawn_reach_end( ): checks if the pawn has passed to home

get_pawns_on_same_position( ): returns list of pawns on the same position

paint_board( ):puts the pawns on the required positions as per their position on the board

add_player( ): Adds new player and puts the pawns on the board pool

get_available_colours( ): returns the list of available colours on the board

_get_next_turn( ): gets next player's turn, cannot be called outside class

get_pawn_from_board_pool( ): when starting a new pawn

get_allowed_pawns_to_move( ): return all pawns of a player which rolled value from die allowed to move the pawn

get_board_pic( ): calls the function to paint the board

_jog_foreign_pawn( ): checks if pawns of different players are in the same position

_make_move( ): tells the board to move the pawn, after the move asks the board if the pawn reaches the end or jogs others pawn. Checks if pawn and player finished

play_turn( ): This is the main method which must be used to play the game. Method asks for next player's turn, roll die, ask player to choose pawn, move pawn.

validate_input( ): validates the input done by the user and displays error message if the input doesn't qualify the requirements

prompt_for_players( ): takes input for player type(computer or human)

prompt_choose_pawn( ): asks user if to choose pawn if there multiple moves possible by different pawns

print_info_after_turn( ): prints which pawn was moved in the last move and also prints which are the possible pawns to move

print_standing( ):prints the final standings of the participants at the finish of the game

play_game( ): plays the game, printing the board and asking input unless the game finishes i.e. finish pointer updates to true

start( ): starts the game calling the functions to take input the players info

_place_pawn( ): updates the change the position of a pawn on the board after each turn by changing its position in the board template

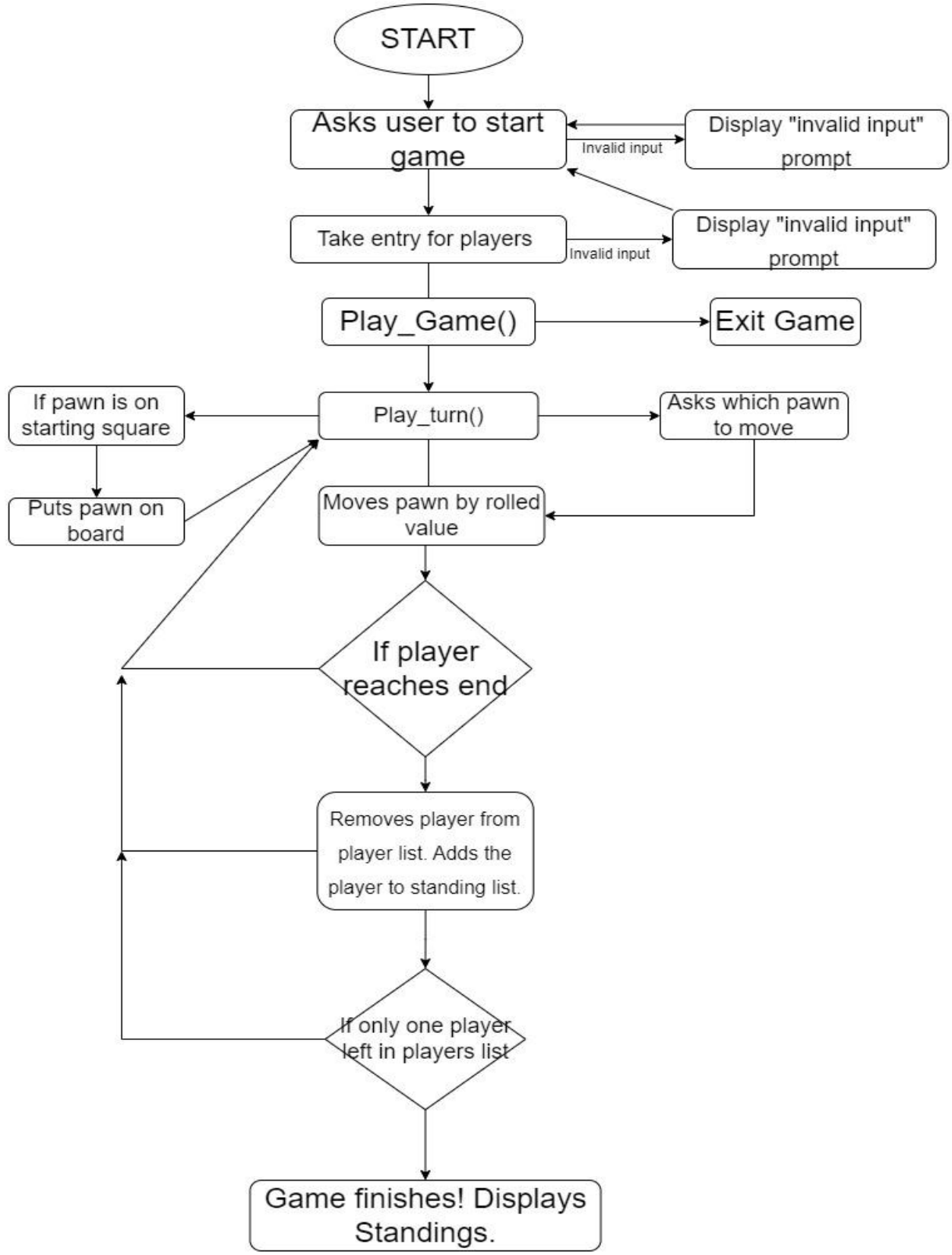
paint( ): reflects the changes made by _place_pawn( ) on the board

present_6_die_name( ): it is the function on which the die is based, it designs the die and changes its face after each turn according to the random number generated by the class Die( ) present in game.py

prompt_to_continue( ): asks the user to press enter to continue the game

print_board( ): prints the board after each turn

Flowchart of the code:



Code

This Ludo Game project contains 4 separate scripts as follows:

1)**runner.py**

```
from .game import Player, Game
from .painter import present_6_die_name
from os import linesep
class rungame():
    def __init__(self):
        self.prompt_end = "> "
        self.game = Game()
        self.prompted_for_pawn = False

    def validate_input(self, prompt, desire_type, allawed_input=None,error_mess="Invalid
Option!", str_len=None):
        prompt += linesep + self.prompt_end
        while True:
            choice = input(prompt)
            if not choice:
                print(linesep + error_mess)
                continue
            try:
                choice = desire_type(choice)
            except ValueError:
                print(linesep + error_mess)
                continue
            if allawed_input:
                if choice in allawed_input:
                    break
                else:
                    print("Invalid Option!")
                    continue
            elif str_len:
                min_len, max_len = str_len
                if min_len < len(choice) < max_len:
                    break
                else:
                    print(linesep + error_mess)
            else:
                break
        print()
        return choice

    def get_user_initial_choice(self):
```

```
        text = linesep.join(["Enter 0 to start new game"])
        choice = self.validate_input(text, int, (0))
        return choice

    def prompt_for_player(self):
        available_colours = self.game.get_available_colours()
        text = linesep.join(["choose type of player",
                             "0 - computer",
                             "1 - human"])
        choice = self.validate_input(text, int, (0, 1))

        if choice == 1:
            name = self.validate_input("Enter name for player",
                                       str, str_len=(1, 30))
            available_options = range(len(available_colours))
            if len(available_options) > 1:
                options = ["{} - {}".format(index, colour)
                           for index, colour in
                           zip(available_options,
                           available_colours)]
                text = "choose colour" + linesep
                text += linesep.join(options)
                choice = self.validate_input(text, int, available_options)
                colour = available_colours.pop(choice)
            else:
                colour = available_colours.pop()
            player = Player(colour, name, self.prompt_choose_pawn)
        elif choice == 0:
            colour = available_colours.pop()
            player = Player(colour)
        self.game.add_palyer(player)

    def prompt_for_players(self):
        counts = ("first", "second", "third", "fourth last")
        text_add = "Add {} player"
        for i in range(2):
            print(text_add.format(counts[i]))
            self.prompt_for_player()
            print("Player added")

        text = linesep.join(["Choose option:","0 - add player","1 - start game with {} players"])
        for i in range(2, 4):
            choice = self.validate_input(text.format(str(i)), int, (0, 1))
            if choice == 1:
```

```
            break
        elif choice == 0:
            print(text_add.format(counts[i]))
            self.prompt_for_player()
            print("Player added")

    def prompt_choose_pawn(self):
        text = present_6_die_name(self.game.rolled_value,
                                  str(self.game.curr_player))
        text += linesep + "has more than one possible pawns to move."
        text += " Choose pawn" + linesep
        pawn_options = ["{} - {}".format(index + 1, pawn.id)
                        for index, pawn
                        in enumerate(self.game.allowed_pawns)]
        text += linesep.join(pawn_options)
        index = self.validate_input(
            text, int, range(1, len(self.game.allowed_pawns) + 1))
        self.prompted_for_pawn = True
        return index - 1

    def prompt_to_continue(self):
        text = "press Enter to continue" + linesep
        input(text)

    def print_players_info(self):
        word = "start" if self.game.rolled_value is None else "continue"
        print("Game {} with {} players:".format(
              word,
              len(self.game.players)))
        for player in self.game.players:
            print(player)
        print()

    def print_info_after_turn(self):
        pawns_id = [pawn.id for pawn in self.game.allowed_pawns]
        message = present_6_die_name(self.game.rolled_value,
                                     str(self.game.curr_player))
        message += linesep
        if self.game.allowed_pawns:
            message_moved = "{} is moved. ".format(
                self.game.picked_pawn.id)
            if self.prompted_for_pawn:
                self.prompted_for_pawn = False
                print(message_moved)
```

```
            return
        message += "{} possible pawns to move.".format(
            " ".join(pawns_id))
        message += " " + message_moved
        if self.game.jog_pawns:
            message += "Jog pawn "
            message += " ".join([pawn.id for pawn in self.game.jog_pawns])
    else:
        message += "No possible pawns to move."
    print(message)

def print_standing(self):
    standing_list = ["{} - {}".format(index + 1, player) for index, player in
enumerate(self.game.standing)]
    message = "Standing:" + linesep + linesep.join(standing_list)
    print(message)

def print_board(self):
    print(self.game.get_board_pic())

def load_players_for_new_game(self):
    self.prompt_for_players()
    self.print_players_info()

def play_game(self):
    try:
        while not self.game.finished:
            self.game.play_turn()
            self.print_info_after_turn()
            self.print_board()
            self.prompt_to_continue()
        print("Game finished")
        self.print_standing()

    except (KeyboardInterrupt, EOFError):
        print(linesep +
              "Exiting game. ")
        raise

def start(self):
    print()
    try:
        choice = self.get_user_initial_choice()
        if choice == 0:  # start new game
```

```
            self.load_players_for_new_game()
            self.play_game()
        except (KeyboardInterrupt, EOFError):
            print(linesep + "Exit Game")
if __name__ == '__main__':
    rungame().start()
```

2)game.py

```
from collections import namedtuple, deque
import random
from .painter import PaintBoard
Pawn = namedtuple("Pawn", "index colour id")
class Player():
    def __init__(self, colour, name=None, choose_pawn_delegate=None):
        self.colour = colour
        self.choose_pawn_delegate = choose_pawn_delegate
        self.name = name
        if self.name is None and self.choose_pawn_delegate is None:
            self.name = "computer"
        self.finished = False
        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)

    def choose_pawn(self, pawns):
        if len(pawns) == 1:
            index = 0
        elif len(pawns) > 1:
            if self.choose_pawn_delegate is None:
                index = random.randint(0, len(pawns) - 1)
            else:
                index = self.choose_pawn_delegate()
        return index


class Board():
    bs = 56
    Colorpos = 7
    corder = ['yellow', 'blue', 'red', 'green']
    quarter = 14
    def __init__(self):
        Board.COLOUR_START = {colour: 1 + index * Board.quarter for index, colour in
enumerate(Board.corder)}
```

```
    Board.COLOUR_END = {
        colour: index * Board.quarter
        for index, colour in enumerate(Board.corder)}
    Board.COLOUR_END['yellow'] = Board.bs
    self.pawns_possiotion = {}
    self.painter = PaintBoard()
    self.board_pool_position = (0, 0)

def set_pawn(self, pawn, position):
    self.pawns_possiotion[pawn] = position

def put_pawn_on_board_pool(self, pawn):
    self.set_pawn(pawn, self.board_pool_position)

def is_pawn_on_board_pool(self, pawn):
    return self.pawns_possiotion[pawn] == self.board_pool_position

def put_pawn_on_starting_square(self, pawn):
    start = Board.COLOUR_START[pawn.colour.lower()]
    position = (start, 0)
    self.set_pawn(pawn, position)

def can_pawn_move(self, pawn, rolled_value):
    common_poss, private_poss = self.pawns_possiotion[pawn]
    if private_poss + rolled_value > self.Colorpos:
        return False
    return True

def move_pawn(self, pawn, rolled_value):
    common_poss, private_poss = self.pawns_possiotion[pawn]
    end = self.COLOUR_END[pawn.colour.lower()]
    if private_poss > 0:
        private_poss += rolled_value
    elif common_poss <= end and common_poss + rolled_value > end:
        private_poss += rolled_value - (end - common_poss)
        common_poss = end
    else:
        common_poss += rolled_value
        if common_poss > self.bs:
            common_poss = common_poss - self.bs
    position = common_poss, private_poss
    self.set_pawn(pawn, position)

def does_pawn_reach_end(self, pawn):
```

```
        common_poss, private_poss = self.pawns_possiotion[pawn]
        if private_poss == self.Colorpos:
            return True
        return False

    def get_pawns_on_same_postion(self, pawn):
        position = self.pawns_possiotion[pawn]
        return [curr_pawn for curr_pawn, curr_postion in self.pawns_possiotion.items() if position
== curr_postion]

    def paint_board(self):
        positions = {}
        for pawn, position in self.pawns_possiotion.items():
            common, private = position
            if not private == Board.Colorpos:
                positions.setdefault(position, []).append(pawn)
        return self.painter.paint(positions)


class Die():
    MIN = 1
    MAX = 6
    @staticmethod
    def throw():
        return random.randint(Die.MIN, Die.MAX)


class Game():
    def __init__(self):
        self.players = deque()
        self.standing = []
        self.board = Board()
        self.finished = False
        self.rolled_value = None
        self.curr_player = None
        self.allowed_pawns = []
        self.picked_pawn = None
        self.index = None
        self.jog_pawns = []

    def add_palyer(self, player):
        self.players.append(player)
        for pawn in player.pawns:
            self.board.put_pawn_on_board_pool(pawn)
```

```
def get_available_colours(self):
    used = [player.colour for player in self.players]
    available = set(self.board.corder) - set(used)
    return sorted(available)

def _get_next_turn(self):
    if not self.rolled_value == Die.MAX:
        self.players.rotate(-1)
    return self.players[0]

def get_pawn_from_board_pool(self, player):
    for pawn in player.pawns:
        if self.board.is_pawn_on_board_pool(pawn):
            return pawn

def get_allowed_pawns_to_move(self, player, rolled_value):
    allowed_pawns = []
    if rolled_value == Die.MAX:
        pawn = self.get_pawn_from_board_pool(player)
        if pawn:
            allowed_pawns.append(pawn)
    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)
    return sorted(allowed_pawns, key=lambda pawn: pawn.index)

def get_board_pic(self):
    return self.board.paint_board()

def _jog_foreign_pawn(self, pawn):
    pawns = self.board.get_pawns_on_same_postion(pawn)
    for p in pawns:
        if p.colour != pawn.colour:
            self.board.put_pawn_on_board_pool(p)
            self.jog_pawns.append(p)

def _make_move(self, player, pawn):
    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)
        return
```

```
        self.board.move_pawn(pawn, self.rolled_value)
        if self.board.does_pawn_reach_end(pawn):
            player.pawns.remove(pawn)
            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
        else:
            self._jog_foreign_pawn(pawn)

    def play_turn(self, ind=None, rolled_val=None):
        self.jog_pawns = []
        self.curr_player = self._get_next_turn()
        if rolled_val is None:
            self.rolled_value = Die.throw()
        else:
            self.rolled_value = rolled_val
        self.allowed_pawns = self.get_allowed_pawns_to_move(
            self.curr_player, self.rolled_value)
        if self.allowed_pawns:
            if ind is None:
                self.index = self.curr_player.choose_pawn(
                    self.allowed_pawns)
            else:
                self.index = ind
            self.picked_pawn = self.allowed_pawns[self.index]
            self._make_move(self.curr_player, self.picked_pawn)
        else:
            self.index = -1
            self.picked_pawn = None
```

3)painter.py

from copy import deepcopy #deep copy so that the changes done on the board in the game are not reflected on the original board
from os import linesep

BOARD_TMPL = [**board template list couldn't be pasted because of software problem***]

cmn_sqrs = [(),(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)]

locate_clr_sqr = {
   '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)]
}

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):
      self.board_tmpl_curr = deepcopy(BOARD_TMPL)

   def _place_pawn(self, pawn, position, offset):
      common_poss, private_poss = position
      colour = pawn.colour.lower()
      if private_poss > 0:
         row, column = locate_clr_sqr[colour][private_poss]
      elif common_poss == 0:
         row, column = CODE_POOL_PLACES[colour][pawn.index]
         offset = 0
      else:
         row, column = cmn_sqrs[common_poss]
      if offset > 0:
         self.board_tmpl_curr[row - 1][column + offset] = pawn.id[1]
      else:
         self.board_tmpl_curr[row - 1][column - 1] = pawn.id[0]
         self.board_tmpl_curr[row - 1][column] = pawn.id[1]

   def _place_pawns(self, position_pawns):
```

```
        for position, pawns in position_pawns.items():
            for index, pawn in enumerate(pawns):
                self._place_pawn(pawn, position, index)

    def paint(self, position):
        self.board_tmpl_curr = deepcopy(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


def present_6_die_name(number, name):
    hor_line = 9 * '-'
    sps = 37 * ' '
    hor_line = sps + hor_line
    matrix = [['|       |', '|   #   |', '|       |'],['|       |', '| #   # |', '|       |'],['|     # |', '|   #   |', '| #     |'],
              ['| #   # |', '|       |', '| #   # |'],['| #   # |', '|   #   |', '| #   # |'],['| # # # |', '|       |', '| # # # |']]
    matrix = [[sps + cell for cell in row] for row in matrix]
    die = matrix[number - 1]
    die[1] = die[1] + "   " + name
    s = linesep.join([hor_line] + die + [hor_line])
    return s
```

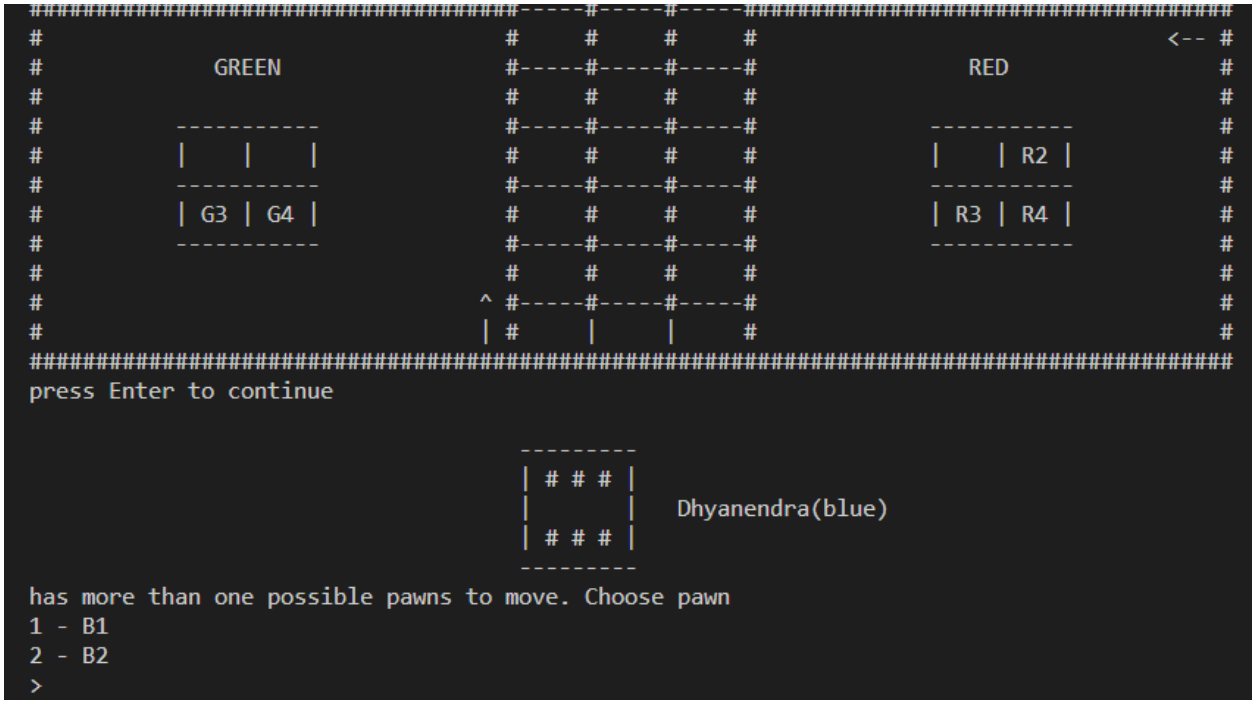
4)<u>run.py</u>: place this python file outside the file containing other three files

```
print(' '*37,end=' ')
print("Welcome to the LUDO GAME")
print(' '*35,end=' ')
print("SOFTWARE DEVELOPMENT PROJECT")
from ludo.runner import rungame
rungame().start()
```

Sample Outputs

```
                                    Welcome to the LUDO GAME
                                   SOFTWARE DEVELOPMENT PROJECT

Enter 0 to start new game
> 0

Add first player
choose type of player
0 - computer
1 - human
> 1

Enter name for player
> Dhyanendra

choose colour
0 - blue
1 - green
2 - red
3 - yellow
> 0

Player added
Add second player
choose type of player
0 - computer
1 - human
> 0

Player added
Choose option:
0 - add player
1 - start game with 2 players
> 0

Add third player
choose type of player
0 - computer
1 - human
> 0

Player added
Choose option:
0 - add player
```

```
Player added
Game start with 4 players:
Dhyanendra(blue)
computer(yellow)
computer(red)
computer(green)

                                  ---------
                                  | #   # |
                                  |       |   computer(yellow)
                                  | #   # |
                                  ---------
No possible pawns to move.
#############################################################################################
#                                  #     |     |     # |                                    #
#            YELLOW                #-----#-----#-----# V             BLUE                   #
#                                  #     #     #     #                                      #
#          -----------             #-----#-----#-----#             -----------            #
#          | Y1 | Y2 |             #     #     #     #             | B1 | B2 |            #
#          -----------             #-----#-----#-----#             -----------            #
#          | Y3 | Y4 |             #     #     #     #             | B3 | B4 |            #
#          -----------             #-----#-----#-----#             -----------            #
#                                  #     #     #     #                                      #
#                                  #-----#-----#-----#                                      #
# -->                              #     #     #     #                                      #
###################################-----#-----#-----#####################################
#     |     |     |     |     |     #     #     |     |     |     |     |     |     |     #
#-----#####################################-----#####################################-----#
#     |     |     |     |     |     |  X  |     |     |     |     |     |     |     |     #
#-----#####################################-----#####################################-----#
#     |     |     |     |     |     #     #     |     |     |     |     |     |     |     #
###################################-----#-----#-----#####################################
#                                  #     #     #     #                                 <-- #
#             GREEN                #-----#-----#-----#                RED                  #
#                                  #     #     #     #                                      #
#          -----------             #-----#-----#-----#             -----------            #
#          | G1 | G2 |             #     #     #     #             | R1 | R2 |            #
#          -----------             #-----#-----#-----#             -----------            #
#          | G3 | G4 |             #     #     #     #             | R3 | R4 |            #
#          -----------             #-----#-----#-----#             -----------            #
#                                  #     #     #     #                                      #
#                                ^ #-----#-----#-----#                                      #
#                                | #     |     |     #                                      #
#############################################################################################
press Enter to continue
```


GREEN
RED
<--
G3 | G4 |
R2 |
R3 | R4 |
press Enter to continue
Dhyanendra(blue)
has more than one possible pawns to move. Choose pawn
1 - B1
2 - B2
>


computer(yellow)
Y3 possible pawns to move. Y3 is moved.
YELLOW
BLUE
GREEN
RED
press Enter to continue
Game finished
Standing:
1 - Dhyanendra(blue)
2 - computer(red)
3 - computer(yellow)
4 - computer(green)

Conclusion

In today's modern world, most of the day to day things are digitalized so why not board games. This project focuses on making a basic application which can be used to play ludo game with players as humans or computers. In this project, different modules and functions are used to design the game board, the die, the pawns and their movement. The game of ludo can be played for having fun or passing time but this project definitely helped in improving the critical thinking and logic building of the project developers. An important plus point of this application is that it is very easy to use and is made by using minimal resources instead of superficial libraries.

We have put our utmost effort in completing this project to the best of our ability. We hope you would accept our project and hope to see your corrections and suggestions.
To conclude, we would like to thank our professor Mrs. Kavita Jaiswal, who gave us the opportunity to work on this project and have a great experience.