

**PROFESSIONAL TRAINING REPORT**  
**at**  
**Sathyabama Institute of Science and Technology**  
**(Deemed to be University)**

Submitted in partial fulfillment of the requirements for the award  
of Bachelor of Technology Degree in Information Technology

By  
**SANTHOSH KUMAR M**

**38120075**



**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**SCHOOL OF COMPUTING**  
**SATHYABAMA INSTITUTE OF SCIENCE AND**  
**TECHNOLOGY**  
**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,**  
**CHENNAI – 600119. TAMILNADU**

**JANUARY 2020**

SCHOOL OF COMPUTING

**BONAFIDE CERTIFICATE**

This is to certify that this Project Report is the Bonafide work of who carried out the project entitled “LUDO GAME USING PYTHON” under our supervision from August 2020 to November 2020.

Internal Guide

Dr. Brumancia M.E., Ph.D.,

Head of the Department

Dr. R. Subhashini M.E., Ph.D.,

---

Submitted for Viva voce Examination held on \_\_\_\_\_

**Internal Examiner**

**External Examiner**

# DECLARATION

I hereby declare that the Project Report entitled “**LUDO GAME USING PYTHON**” done by me under the guidance of **E.Brumancia M.E., Ph.D.**, at Sathyabama Institute of Science and Technology (Deemed to be University), Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600119 is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology

DATE: 20/01/2020

Place: Chennai

SIGNATURE OF THE CANDIDATE

## **ACKNOWLEDGEMENT**

I am pleased to acknowledge our sincere thanks to Board of management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey our thanks to **Dr. T. SASIKALA M.E., Ph.D., Dean, School of Computing** and **Dr. R. SUBHASHINI M.E., Ph.D., Head of the Department, Department of Information Technology** for providing us the necessary support and details at the right time during the progressive reviews.

I would like to express our sincere and deep sense of gratitude to our Project **E.BRUMANCIA M.E., Ph.D.,** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express our thanks to all Teaching and Non-teaching staff members of the Department of **INFORMATION TECHNOLOGY** who were helpful in many ways for the completion of the project.



**RETECH**  
SOLUTIONS PVT. LTD.  
An ISO 9001:2015 certified company

INT: RE20S058

**TO WHOMSOEVER IT MAY CONCERN**

This is to certify that **SANTHOSH KUMAR M** student of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY, BTECH/IT** has undergone the Internship Training through online in our concern entitled **PYTHON** from **12<sup>th</sup> Aug 2020 to 26<sup>th</sup> Aug 2020** in relevant departments related to their academic studies.

During the above period, the performance was good & we wish great success in all your future endeavours.

*J. Meru*

*Technology defines the world. We are the people who are behind the technology*

**Authorised Signature**



No-31, 1<sup>st</sup> Floor, Alagesan Street, Tambaram West, Chennai-45.

Office : 044-42077204 | +91-8122266653 [www.retechsolutions.in](http://www.retechsolutions.in)



06/12/2020

**SANTHOSH KUMAR M**

has successfully completed

**Programming for Everybody (Getting Started with Python)**

an online non-credit course authorized by University of Michigan and offered through Coursera

A handwritten signature in black ink, appearing to read 'Chuck', followed by a horizontal line.

Charles Severance  
Clinical Professor, School of Information  
University of Michigan

**COURSE  
CERTIFICATE**



Verify at [coursera.org/verify/JFJBZAUM77EH](https://coursera.org/verify/JFJBZAUM77EH)  
Coursera has confirmed the identity of this individual and  
their participation in the course.

---

# CERTIFICATE OF COMPLETION

This is to certify that

**SANTHOSH KUMAR M**

successfully completed

5 Studies in the

## Python Course

This course covers the fundamentals of  
Python for software development.

Certificate URL: [pragate.com/course\\_certificate/f3317d12qf48p9](https://pragate.com/course_certificate/f3317d12qf48p9)



## ABSTRACT

Ludo Game project is written in Python. The project file contains python scripts (game.py, run.py, painter.py, recorder.py). This is a simple console based strategy board game which is very easy to understand 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, the player has to enter details such as name and select color (red, green, yellow and blue). the player can also start the game within two players if he/she wants.

After starting the game, a console based ludo board appears, other rules are the same. 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, it displays a dice with the number. The system keeps on rolling until there's a possible pawn to move. All the game movements are performed automatically. Whenever the player tries to end the game, there's an option available to save the unfinished game so that he/she can continue it later. A simple console GUI is provided for the easy gameplay. The gameplay design is so simple that user won't find it difficult to use and understand.

The Ludo Game project is composed in Python. The job data has python manuscripts (Ludo.py, Ludo-Game.py) as well as photo data. This is a simple GUI based method board game which is really understandable and make use of. Speaking about the gameplay, all the playing policies are the same much like real time LUDO. This is a GUI based 2D multiplayer game. The game also consists of a high score module where the winner is assigned specific points based on the ranking and is stored in a database. The individual score is parsed from the variable to the SQL database. After starting the game, a GUI Ludo board shows up, other rules coincide. Initially, the player needs to chance. The main thing in this GUI based game is that the player simply has to push "Roll" to chance. On top of the board, it displays a dice with the number. The needs to keep on rolling up until there's a possible pawn to relocate. All the game motions are to be done by hand by the player. The 2D GUI is designed using Tkinter python libraries

## **AIM AND SCOPE**

### **AIM**

The simple Ludo Game project is written in Python. The job data contains python manuscripts (game.py, run.py, painter.py, recorder.py). This is a simple console-based approach parlor game which is extremely understandable and makes use of. Speaking about the gameplay, all the playing regulations coincide similar to what we play in real-time ludo. Below initially, the individual needs to select players i.e either human or computer systems. After selecting human, the gamer needs to go into details such as name and pick shade (red, green, yellow as well as blue). the player can likewise begin the game within two players if he/she desires.

### **SCOPE**

After starting the game, a console based ludo board appears, other rules are the same. 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, it displays a dice with the number. The system keeps on rolling until there's a possible pawn to move. All the game movements are performed automatically. Whenever the player tries to end the game, there's an option available to save the unfinished game so that he/she can continue it later. A simple console GUI is provided for the easy gameplay. The gameplay design is so simple that user won't find it difficult to use and understand.

## **1. INTRODUCTION**

### **1.1 GENERAL**

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves. Crop yield prediction is one of the challenging problems in precision agriculture, and many models have been proposed and validated so far. This problem requires the use of several datasets since crop yield depends on many different factors such as climate, weather, soil, use of fertilizer, and seed variety. This indicates that crop yield prediction is not a trivial task; instead, it consists of several complicated steps.

### **1.2 OUTLINE OF THE PROJECT**

Machine Learning studies consist of different challenges when aiming to build a high-performance predictive model. It is crucial to select the right algorithms to solve the problem at hand, and in addition, the algorithms and the underlying platforms need to be capable of handling the volume of data. This project mainly focuses on creating a crop recommendation system using Machine learning algorithm to predict which crop can give a better yield with the conditions of a particular state or country. By applying Random forest algorithm in Machine learning model to train a dataset containing details like rainfall, pH value, humidity and temperature, we can recommend the crop suitable for a particular area

## **4 USER INTERFACE**

The user interface (UI) is the point of human-computer interaction and communication in a device. This can include display screens, keyboards, a mouse and the appearance of a desktop. It is also the way through which a user interacts with an application or a website

## **4 PYTHON WEB FRAMEWORK**

A Web framework is a collection of packages or modules which allow developers to write Web applications or services without having to handle such low-level details as protocols, sockets or process/thread management. The Python web framework used in this project is Flask.

## **SOFTWARES USED**

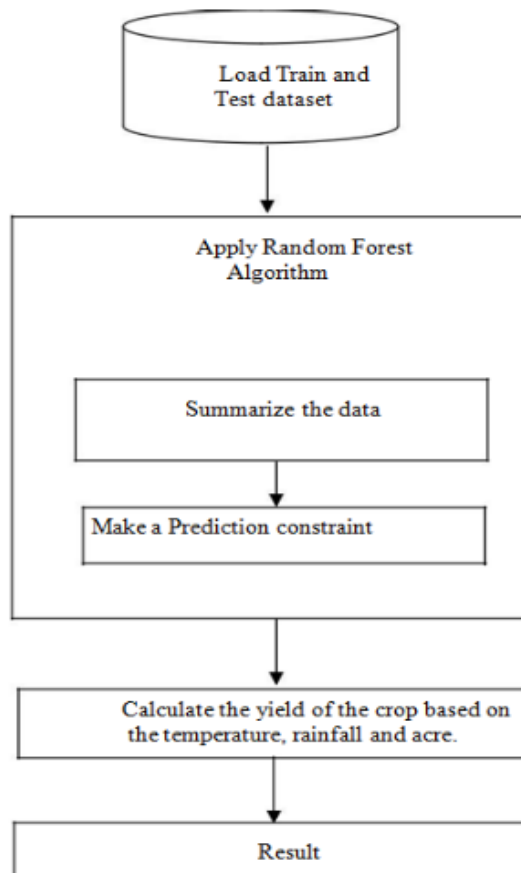
### **3.2.1 CODE EDITORS USED**

#### ***VISUAL STUDIO CODE***

Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and MacOS. It includes support for debugging, embedded Git control and GitHub, syntax highlighting, intelligent code completion, snippets, and code refactoring.

## 4 PROJECT DESCRIPTION

The simple Ludo Game project is written in Python. The job data contains python manuscripts (game.py, run.py, painter.py, recorder.py). This is a simple console-based approach parlor game which is extremely understandable and makes use of. Speaking about the gameplay, all the playing regulations coincide similar to what we play in real-time ludo. Below initially, the individual needs to select players i.e either human or computer systems. After selecting human, the gamer needs to go into details such as name and pick shade (red, green, yellow as well as blue). the player can likewise begin the game within two players if he/she desires.



**FIG 3.1 DATA FLOW**

## **The rules of the game:**

### **The dice**

A six-sided dice is used. With it, you have to hit six to move a piece out of the starting area, while all other throws give the right to move a piece the number of fields the eyes show. Extra throws are given when hitting six (no limit to how many times in a row).

### **At the start**

Players take turns alternating with the dice. At the stroke of 6, you have the right to throw a piece out on the board. In the first round, you have three attempts to beat a 6. Players turn in turn in a clockwise direction (player 1 to 4).

### **The course of the game**

A 6's entitles you to an extra roll. The player moves the number of squares corresponding to the eyes of the dice. If you land in a field where there is already a piece, the first arrived piece must return to starting field. If, on the other hand, there are two pieces on the field, then the last to arrive has to return to the starting field. You have to move a piece (if there is one that can be moved), even if it means you have to back to start. If you cannot move forward, you must stand for the next turn.

### **Fields**

On the playing board there are several special areas and fields.

- The starting area is where the four pieces start. To take a piece out of the starting area, you have to hit a six. If you have all your pieces in the starting area, you get up to three strokes with the dice before the tour is passed.
- The target area is where the four pieces should end and can get to the goal. Each player has their own target area, and no other player is allowed to take their pieces in there. To reach the end of the target area (the goal), it must hit precisely - otherwise you have to move your piece in the opposite direction by the amount you have leftover.
- Globus fields protect the pieces from being knocked home. If an opponent's piece lands on a protected piece, it is hit home. However, there is an exception to the colored globe fields. For example, only red chips can be protected in the red globe field, regardless of the number of chips in the field. If you have two pieces standing on the opponent's colored globe, they can both be knocked home.

- Star fields act as shortcuts that can bring the pieces faster to the target area. If a piece lands on a star, it must be moved to the next star. If it is the star in front of the target area that lands on, the chip is moved directly to the goal.

### The winner

The one who first gets all 4 pieces in the goal is the winner. (But you choose if the game ends there or if the other players still has to fight) Goal are the center. When you have entered the colored fields you can not be hit home. The piece has to be move precisely into the goal otherwise the piece are struck back the exact number of eyes there are to many.

### ABOUT PROJECT

Simple 2D Ludo Game project is written in Python. The project file contains python scripts (Ludo.py, Ludo-Game.py) and image files. This is a simple GUI based strategy board game that is very easy to understand and use. Talking about the gameplay, all the playing rules are the same just like we play in real-time ludo. This is a simple 2D multiplayer game.

After starting the game, a GUI ludo board appears, other rules are the same. First, the player has to roll the dice. The main thing in this simple GUI based game is that the player just has to press “Roll” to roll the dice. At the top of the board, it displays a dice with the number. The has to keep on rolling until there’s a possible pawn to move. All the game movements are to be performed manually by the player. A simple 2D GUI is provided for easy gameplay. The gameplay design is so simple that the user won’t find it difficult to use and understand. For the gaming environment, the project uses various images and GIF files to provide real-time experience.

In order to run the project, you must have installed [Python](#), on your PC. This is a simple 2D GUI Based game written for the beginners. Simple 2D Ludo Game in Python project with source code is free to download.

### ABOUT LUDO:

also consists of a high score module where the winner is assigned specific points based on the ranking and is stored in a database. The individual score is parsed from the variable to the SQL database. After starting the game, a GUI Ludo board shows up, other rules coincide. Initially, the player needs to chance. The main thing in this GUI based game The Ludo Game project is composed in Python. The job data has python manuscripts (Ludo.py, Ludo-Game.py) as well as photo data. This is a simple GUI based method board game which is really understandable and make use of. Speaking

about the gameplay, all the playing policies are the same much like real time LUDO. This is a GUI based 2D multiplayer game. The game is that the player simply has to push “Roll” to chance. On top of the board, it displays a dice with the number. The needs to keep on rolling up until there’s a possible pawn to relocate. All the game motions are to be done by hand by the player. The 2D GUI is designed using Tkinter python libraries. The gameplay and GUI is designed in such a way that the user has no difficulty in locating and comprehending the contents of the gameplay. The GUI utilizes various photos as well as GIF documents to provide real-time experience. At the end of the game a GUI High score table is displayed for semi-quantitative analysis of the player’s performance.

### **1. Python**

The game is developed using Python. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

### **2 .MySQL**

MySQL is a fast, easy-to-use efficient Relational Database Management System. MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages. It uses a standard form of well-known SQL data language. It works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc. It supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB). It is customizable. The open-source GPL license allows

programmers to modify the MySQL software to fit their own specific environments.

### **3 Tkinter**

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter outputs the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

The Tk interface is located in a binary module named `_tkinter`. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter. In addition to the Tk interface module, Tkinter includes a number of Python modules. The two most important modules are the Tkinter module itself, and a module called Tkconstants. The former automatically imports the latter, so to use Tkinter, all you need to do is to import one module.

### **Ludo**

Ludo is a board game played by 2-4 players. Each player is assigned a specific color and given four pieces. In this board game 2 to 4, players race their tokens from start to finish according to the dice rolls. Various variations are seen in the way people play

Ludo. A Ludo board is in the shape of a cross, each arm being divided into three adjacent columns of six squares. The middle squares form the home column for each color and cannot be landed upon by other colors. The middle of the cross forms a large square which is the 'home' area and which is divided into 4 home triangles, one of each color. At each corner are colored areas where the pieces are placed to begin. The typical colors seen in Ludo boards are bright yellow, green, red, and blue. Each player is assigned a color and has four tokens of matching color. The players put their tokens in the starting circle. The movement of the tokens is determined by the dice. During game play the token moves clockwise from the starting square around the perimeter of the board, and up the player's home column to the finishing square. Players are not allowed to move a token out of the starting area unless they roll a 6 on the dice. In this

game, players have to carefully play their token or they run the risk of sending it back to the starting point to start all over again. When a player 1's token lands on square that is already occupied by Player 2's token of different color, Player 2's token is returned to its starting point.

## **2. REVIEW OF LITERATURE**

**2.1 Artificial Intelligence: Game Techniques Ludo** - A Case Study by Veenus Chhabra<sup>1</sup> and Kuldeep, M.Tech Student Department of CSE, NGF College of Engineering & Technology, Palwal,. In this paper the artificial intelligence ludo provides environment for playing to the artificial agents. Its main purpose lies in the area of artificial intelligence. At one side, Ludo is instead a very simple game and is fully observable but on the other side it contains a few challenges due to the stochastic and multi agent environment. Therefore, it offers a good balance between simplicity and complexity and is able to attract a wide audience and not only professionals.

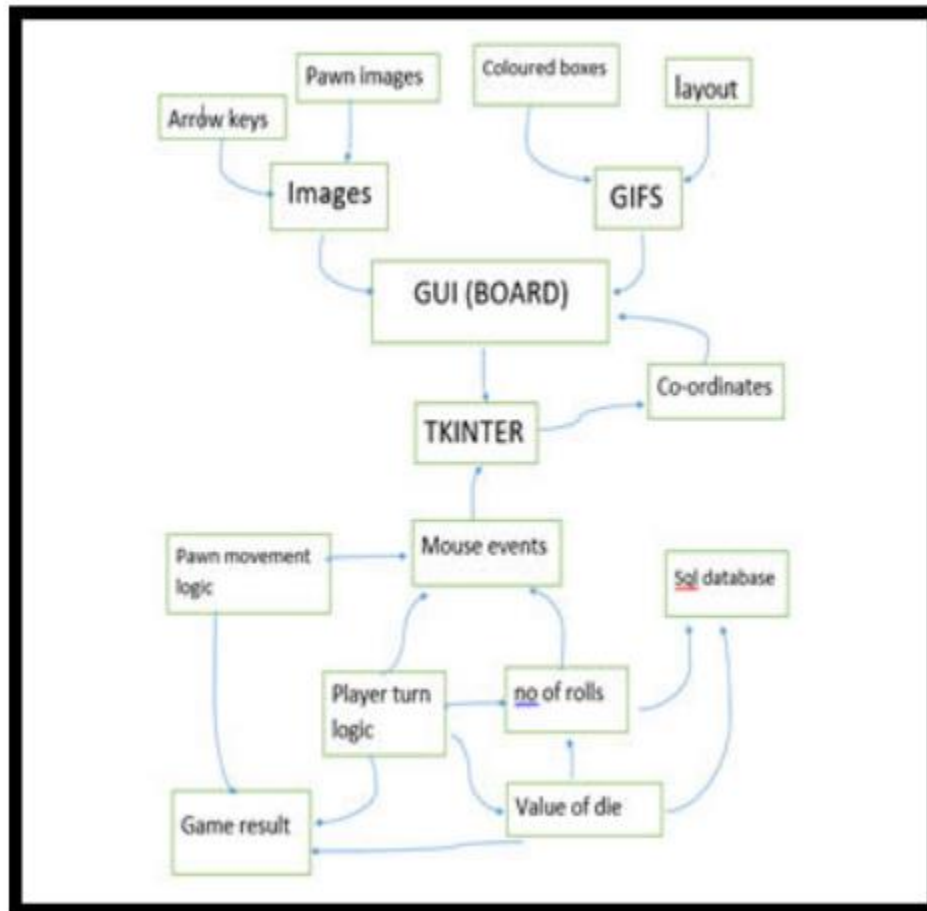
### **2.2 TD ( $\lambda$ )**

based Ludo player and also implemented a Q-learning based Ludo player using reinforcement learning by Georgios N. Yannakakis and Carla P. Gomes. This paper explains a thorough description of Game Theory and its applications. He explained about AI and its four flagships, briefing about the terminology used currently in games and also gives connection between search algorithms with hard combinatorial problems. He grouped this connection under three things planning, duality and randomization.

**2.3 Complexity Analysis and Playing Strategies** for Ludo and its Variant Race Games by Faisal Alvi, Member IEEE, Moataz Ahmed. In this paper the game is set up by designing classes representing entities involved in a typical Ludo game. Initially they tested the game setup by running numerous games with all four players selecting the random strategy. This was done to ensure that our game setup is correct and that each random player wins approximately equal number of times. They found that each random player wins  $25.0 \pm 1.0\%$  of the games and that this performance stabilized at nearly

5000 games. Running a higher number of games did not reduce the variation in results significantly, suggesting that 5000 games was an adequate number to observe reliable trends in strategy evaluation. The tests were also repeated for each strategy separately and identical results were obtained.

### 3. DESIGN METHODOLOGY OF PROPOSED WORK



#### 3.1 Description of Game Board Architecture

- **Board GUI:** The interface of the Ludo game.
- **Images and GIFS:** Contains image files and gifs for the interface.
- **Pawn movement logic:** It contains the main game logic.
- **Console:** It displays value of die, co-ordinates and number of rolls

**3.2 Existing System** The current Ludo game is offline based multi-player game. Multiple players can play concurrently. The game interface is built in 2D format. The interface is an interactive window where the user's manipulations of pawns are handled. A console window is created as soon as the game starts and the console window

maintains the records of current player's turn, value of dice and number of rolls and disclaimer after every turn. The player can interact with the game simply by using mouse events. The GUI is developed using Tkinter and various image files are used to create the interface. The interface is divided into various sections and each sections lead up to different functionalities. The limitations with the existing system is that the game has no dedicated storage allocated to store the user's information and performance statistics. The game has no information regarding the user so every time the game is executed it is like the game is running for the first time.

**3.3 Proposed System** To overcome the limitations of the existing system a database is created and a database connection is setup between the python scripts and the database. The user's information and performance statistics like individual's win score is retrieved from python script and the values are passed through SQL workbench to the database. The stored data is monitored and updated every time a game is played and now the game has information about the users. A high score table is displayed in a window at the end of each game where user's name and his individual score can be viewed.

To overcome the problem of lack of storage of game information is to describe an external SQL database and parse variable values into the records of the table in the database. So the player's score values can be stored in a database in their respective rows and the scores can be retrieved and updated the next time the game is opened. A window is created to display the high score of all the individual players. A.

**Ludo game logic** A class is created to develop game logic and various methods are created to control pa contains numerous methods like turn, roll, clear, move check kill etc. First, the player's turn is determined. Then roll function is called to that player's color and the player can roll the die. Then the value of the die is checked player is asked to roll the die again. Else move method is called and the control of the die is sent to next color. The move method enables the user to manipulate the pawns when the method is called in roll method. Left Click is a method for mouse click events and it is called every time left click is clicked on valid co used to move the pawns by selecting on the valid pawns.

**B. Ludo Board Interface setup** To create the board first all the necessary image files are initialized. For each color different co-ordinates are set in the board. The pawn images are also given set of rules where it cannot lie on illegal co-ordinates. Mouse events are ensured at the time. A separate frame right next to board is created for user to roll the die by clicking on a button. A random number is defined every time the roll button is clicked. The interface opens up with a start the game. Then the game interface is called. At the end of the game once the user has won, it displays which user has won the game.

**C. Database Connectivity** To establish a connection with a SQL database, a platform needs to be installed so that the code can interact easily with the database. SQL workbench is installed to allow the easy interaction

imported which can setup the connection with the database. First an Object is created in sql class. Then the host, user id and password of the database account is passed as a parameter in connect method. The database is also chosen. When the database is chosen, a table is created and linked with the code. A SQL command is executed every game is finished, the command increments the winner's color's score value by 5 and the table is displayed in descending order.

### **3.5 Features of Proposed System**

- All the player's data will be stored and won't be deleted once the game is terminated
- The user can view their individual scores.
- High score table is displayed to compare the score with other players.
- A separate database is created for storing of user data
- The data stored cannot be deleted from the game.
- A dedicated window to display the stored data.

## **4.1 SYSTEM REQUIREMENTS**

- Operating System: Windows/Mac
- RAM: 4GB
- Processor: 64x 1.0Ghz processor
- ROM: 8GB

## 4.2 Software Requirements

- Python 3 or higher
- Python Package Installer
- JetBrainsPyCharm IDE
- MySQL Server
- MySQL workbench
- Sublime text Editor
- Tkinter Python Libraries

## 4.3 Tools and Platform PyCharm

IDE PyCharm provides smart code completion, code inspections, on-the-fly error highlighting and quick-fixes, along with automated code refactoring's and rich navigation capabilities. **Features**

- Intelligent Code Editor
- Smart Code Navigation
- Fast and safe Refactorings.
- Debugging, Testing and Profiling
- VCS, Deployment and Remote Development
- Database Tools MySQL Workbench MySQL Workbench is a graphical tool for working with **MySQL**. **MySQL Workbench** provides an easy to use interface for performing the many tasks involved when working with databases. It integrates SQL development, administration, database design, creation and maintenance into one visual integrated development environment. MySQL Workbench is similar to SQL Server's SSMS, which is used for administering SQL Server

## Tkinter

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter outputs the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task. The Tk interface is located in a binary module named `_tkinter`. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter. In addition to the Tk interface module, Tkinter includes a number of Python modules. The two most important modules are the Tkinter module itself, and a module called Tkconstants. The former automatically imports the latter, so to use Tkinter, all you need to do is to import one module:

## Sublime Editor Sublime

Text is a proprietary cross-platform source code editor with a Python application programming interface (API). It natively supports many programming languages and markup languages, and functions can be added by users with plugins, typically community-built and maintained under free-software licenses. • Multiple selections • Command palette • Powerful API and package ecosystem • Customize anything • Split editing • Instant Project Switch • Cross Platform • Performance

## **5. IMPLEMENTATION AND RESULTS**

**5.1 IMPLEMENTATION** The game will be opened by executing the python script. Then the user will be asked to start the game by a prompt screen. Then the game interface is shown in a window and players can start playing the game.

**5.3 TESTING** In unit testing, the program modules that make up the system are tested individually. Unit testing focuses to locate errors in the working modules that are independent to each other. This testing is also used to ensure the integrity of the data stored. The various routines were checked by passing the inputs and the corresponding output is tested. Test cases used in the project

## **RESULT**

The Results shows that we can attain an accurate crop yield prediction using the Random Forest algorithm. Random Forest algorithm achieves a largest number of crop yield models with a lowest models. It is suitable for massive crop yield prediction in agricultural planning. This makes the farmers to take the right decision for right crop such that the agricultural sector will be developed by innovative ideas.

When the game is opened, the user prompts the game to be started. Then the game interface is displayed and by default it is red's turn. The player must roll the die by clicking on the button in the top right. Until the value is 6, no pawns can be moved. Once it is 6, the same player gets another chance to roll and that value can be used to move the pawn to that many checks ahead. Then the control of the die shifts to the next color and same process goes on. When a player's pawn overlaps the other pawn, the latter pawn is sent to home box and the former pawn's players gets an another chance to roll the die. The one who ensures all the pawns get inside the center wins the game. As soon as the game is won, A dialog box appears showing who won the game and another window is opened which displays the score of each player's after the recent game. This is done by setting up a database connection with a table. The game's winner's score will be incremented by 5 in the table and the table will be displayed in descending order.

## CONCLUSION AND FUTURE WORK

### 5.1 CONCLUSION

This project implements a LUDO game in 2D GUI format. This application allows the user to play the game by simply opening the python script and prompting the game to start. The game interface is very simple and is easy to understand by the user. Since the game is built in 2D the game performance is fast and efficient . The game can also store user's score information by setting up a database connection from the python script. The individual's scores are displayed in a new window called highscore window where the players can compare each other's score.

. Ludo Game project is written in Python. The project file contains python scripts (game.py, run.py, painter.py, recorder.py). This is a simple console based strategy board game which is very easy to understand 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, the player has to enter details such as name and select color(red, green, yellow and blue). the player can also start the game within two players if he/she wants.

After starting the game, a console based ludo board appears, other rules are the same. 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, it displays a dice with the number. The system keeps on rolling until there's a possible pawn to move. All the game movements are performed automatically. Whenever the player tries to end the game, there's an option available to save the unfinished game so that he/she can continue it later. A simple console GUI is provided for the easy gameplay. The gameplay design is so simple that user won't find it difficult to use and understand.

In order to run the project, you must have installed [Python](#), on your PC.

### 5.2 FUTURE WORK

As of now, the game can only store data in a database which can be accessed offline. The database can be set up to different host and the game can be played from any system but the score in the table will be updated in the database. The table can also be extended by creating more attributes like number of rolls, Probability of winning of each

players. The table can also be made to view by accessing specific IP address

This project describes crop yield prediction ability of the algorithm. In future we can determine the efficient algorithm based on their accuracy metrics that will help to choose an efficient algorithm for crop yield prediction

### **5.3 REFERENCES**

1. <https://getbootstrap.com/>
2. <https://www.w3schools.com/html/default.asp>
3. [https://www.w3schools.com/html/html\\_css.asp](https://www.w3schools.com/html/html_css.asp)
4. <https://jupyter.org/try>

## APPENDIX

```
C:\WINDOWS\ip.py.exe

choose option
0 - start new game
1 - continue game
2 - run (review) recorded game
> 0

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

Enter name for player
> SANTHOSH

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

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
> 1

Game start with 2 players:
SANTHOSH(green)
computer(yellow)
```







## B.SOURCE CODE

### CLI.PY

```
from .game import Player, Game
from .painter import present_6_die_name
from .recorder import RunRecord, MakeRecord
from os import linesep

class CLIGame():

    def __init__(self):
        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
        param allawed_input can be list of allowed values
        param str_len is two sized tuple if min and max
        """
        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(["choose option",
                        "0 - start new game",
                        "1 - continue game",
                        "2 - run (review) recorded game"])
    choice = self.validate_input(text, int, (0, 1, 2))
    return choice

def prompt_for_file(self, mode="rb"):
    """return file descriptor"""
    text = "Enter filename (name of the record)"
    while True:
        filename = self.validate_input(text, str)
        try:
            file_descr = open(filename, mode=mode)
            return file_descr
        except IOError as e:
            print(e)
            print("Try again")

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"])
    choice = self.validate_input(text, int, (0, 1))
    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"])
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:
        # show available colours
        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:
        # only one colour left
        colour = available_colours.pop()
    player = Player(colour, name, self.prompt_choose_pawn)
elif choice == 0:
    # automatically assign colours
    colour = available_colours.pop()
    player = Player(colour)
self.game.add_palyer(player)

def prompt_for_players(self):
    """put all players in the game"""
    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):
    """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."
    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):
    """it used game attributes to print info"""
    pawns_id = [pawn.id for pawn in self.game.allowed_pawns]
    # nicer print of dice
    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 run_recorded_game(self):
    """get history of game (rolled_value
    and index's allowed pawn) from
    record_runner in order to replay game"""
    self.load_recorded_players()
    self.print_players_info()
    self.prompt_to_continue()
    for rolled_value, index in self.record_runner:
        self.game.play_turn(index, rolled_value)
        self.print_info_after_turn()
        self.print_board()
        self.prompt_to_continue()
        self.print_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()

```

```

        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)
        self.print_players_info()
        self.print_info_after_turn()
        self.print_board()

    def record_players(self):
        """save players on recorder"""
        for player in self.game.players:
            self.record_maker.add_player(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()
            self.record_runner = RunRecord(file_descr)
            file_descr.close()
        for player in self.record_runner.get_players(
            self.prompt_choose_pawn):
            self.game.add_palyer(player)

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

    def play_game(self):
        """mainly calling play_turn
        Game's method while game finished
        """
        try:
            while not self.game.finished:
                self.game.play_turn()
                self.print_info_after_turn()
                self.print_board()
                self.record_maker.add_game_turn(
                    self.game.rolled_value, self.game.index)
                self.prompt_to_continue()
            print("Game finished")
            self.print_standing()
            self.offer_save_game()
        except (KeyboardInterrupt, EOFError):

```

```

        print(linesep +
              "Exiting game. " +
              "Save game and continue same game later?")
        self.offer_save_game()
        raise

def offer_save_game(self):
    """offer user save game"""
    if self.does_user_want_save_game():
        file_descr = self.prompt_for_file(mode="wb")
        self.record_maker.save(file_descr)
        file_descr.close()
        print("Game is saved")

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 self.game.finished:
                print("Could not continue.",
                      "Game is already finished",
                      linesep + "Exit")
            else:
                self.prompt_to_continue()
                self.play_game()
        elif choice == 2: # review played game
            self.run_recorded_game()
    except (KeyboardInterrupt, EOFError):
        print(linesep + "Exit Game")

if __name__ == '__main__':
    CLIGame().start()

```

## GAME.PY

```

from collections import namedtuple, deque
import random
from .painter import PaintBoard

```

```

# Thanks to Angel Angelov

```

```
# This is piece or a token in ludo game
# Simple class has only index, colour and id attributes
Pawn = namedtuple("Pawn", "index colour id")
```

```
class 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
        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
        # initialize four pawns with
        # id (first letter from colour and index (from 1 to 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)

    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)
            else:
                index = self.choose_pawn_delegate()
        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):
    #fn1353c
    # get dict of start position for every colour
    Board.COLOUR_START = {
        colour: 1 + index * Board.COLOUR_DISTANCE for
        index, colour in enumerate(Board.COLOUR_ORDER)}
    # get dict 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

    # dict 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_possition = {}

    # painter is used to visually represent
    # the board and position of the pawns
    self.painter = PaintBoard()

    # pool means before start1353
```

```

self.board_pool_position = (0, 0)

def set_pawn(self, pawn, position):
    """save 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 True of False"""
    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):
    """check if pawn can outside board colour size"""
    common_poss, private_poss = self.pawns_possiotion[pawn]
    if private_poss + rolled_value > self.BOARD_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 is entering in own squares
        private_poss += rolled_value - (end - common_poss)
        common_poss = end
    else:
        # pawn will be still in common square
        common_poss += rolled_value
        if common_poss > self.BOARD_SIZE:
            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_ossition[pawn]
    if private_poss == self.BOARD_COLOUR_SIZE:
        return True
    return False

def get_pawns_on_same_postion(self, pawn):
    """return list of pawns on same position"""
    position = self.pawns_ossition[pawn]
    return [curr_pawn for curr_pawn, curr_postion in
            self.pawns_ossition.items()
            if position == curr_postion]

def paint_board(self):
    """painter object expect dict of
    key - occupied positions and
    value - list of pawns on that position
    """
    positions = {}
    for pawn, position in self.pawns_ossition.items():
        common, private = position
        if not private == Board.BOARD_COLOUR_SIZE:
            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():
    """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()
# is game finished
self.finished = False
# last rolled value from die (dice)
self.rolled_value = None
# player who last rolled die
self.curr_player = None
# curr_player's possible pawn to move
self.allowed_pawns = []
# curr_player's chosen pawn to move
self.picked_pawn = None
# chosen index from allowed pawn
self.index = None
# jog pawn if any
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):
    """if has available colour on boards"""
    used = [player.colour for player in self.players]
    available = set(self.board.COLOUR_ORDER) - set(used)
    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:
        self.players.rotate(-1)
    return self.players[0]

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):
            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 = []
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):
    """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)
        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):
    """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 = []
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

```

## PAINTER.PY

```
from copy import deepcopy
from os import linesep
```

[illegible]





```

'yellow': [(0, (6, 14), (6, 19), (8, 14), (8, 19)],
'blue': [(0, (6, 71), (6, 76), (8, 71), (8, 76)],
'red': [(0, (24, 71), (24, 76), (26, 71), (26, 76)],
'green': [(0, (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:
            # private squares
            row, column = CODE_COLOUR_SQUARES[colour][private_poss]
        elif common_poss == 0:
            # pool
            row, column = CODE_POOL_PLACES[colour][pawn.index]
            offset = 0 # we do not need from offset in the pool only in squares
        else:
            # common squares
            row, column = CODE_COMMON_SQUARES[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):
        """expect dict of
        key - occupied positions and
        value - list of pawns on that 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):
        """nicer print of die and
        name of the player

```

```

"""
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

```

### record.py

```

import pickle
from .game import Player

class RunRecord():
    """provide recoded 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)
        self.players = data[0]
        self.game_history = data[1]

    def get_players(self, func=None):
        """
        return Player object
        recreated from a list
        func is callable which player
        may need for choice delegation
        """
        res = []
        for colour, name, is_computer in self.players:
            if is_computer:
                player = Player(colour)
            else:
                player = Player(colour, name, func)
            res.append(player)

```

```

        return res

    def get_game_history(self):
        return self.game_history

    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 = []
        self.game_history = []

    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:
            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)

```

### **run.py**

```

#!/usr/bin/env python3

from ludo.cli import CLIGame

```

```
CLIGame().start()
```

# THANK YOU