# Analysis

## Decomposition

#### **Method**: Main game

The main game method runs the entire program:

1. Instantiate a game\_settings class that manage the game rules
2. Instantiate a game\_users class
3. Print the rules and introduction to the screen
4. Let the user choose his main operations:
   1. Add a user
   2. Play a game
   3. View the players records (number of wins / losses per user)
   4. Or hidden option to change the game settings (rules)
5. Upon user selection the main game either:
   1. Command its game\_users instance to add a user
   2. Or let the user to select players and play the game and report the winner and loser to the game \_users instance.
   3. Or print the user records (number of wins / losses per user)
   4. Or command the game\_settings to let the user change the settings

The following diagram illustrates the main game workflow:



#### **Method**: Introduction

The introduction method is a simple method that prints to the screen an introduction and the game rules.

#### **Class**: Game Settings

The game settings is an advance class that maintains all the game settings. These include:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Settings | Variable name | Variable data type | Default value (initialized) | Minimum value | Comment |
| The cube sides | cube\_size | integer | 6 | 2 | Settings to 2 helped to simulate / test double |
| The number of turns in a game | num\_turns | Integer | 5 | 0 | Settings to 0 helped to simulate / test draw events |
| The bonus for even number | Bonus[0] | Integer | 10 | - |  |
| The bonus for odd number | Bonus[1] | Integer | -5 | - |  |
| The number of players | num\_players | integer | 2 | 2 | Changing the number of users have significant implications! Warning is displayed |
| Number of dice | num\_of\_dice | Integer | 0 | 0 |  |
| Number of dice  after double number was rolled) | double\_num\_of\_dice | Integer | 1 | 0 |  |
| Number of dice  after game ended at draw | draw\_num\_of\_dice | Integer | 1 | 1 |  |

The class has a ‘change settings’ method that enables the user to change the settings.

The following diagram illustrates the workflow of GameSettings.change\_settings:



#### **Class**: Game

The game class runs a game.

The class maintain the following variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable name** | **Data type** | **Comment & description** | **Initialized** |
| players | List of string | Each string in the list is a username. | Input to \_\_init\_\_ of the class |
| s | GameSettings | Instance of GameSettings class | to \_\_init\_\_ of the class |
| scores | List of integers | List of scores.   For each integer in the range [0..len(players)]  Score[i] is the score of players[i] | Initialize to 0 for all players |

It also holds the score (points) of each player, i.e. a list of scores.

The purpose of this class is to run a game between selected players.

The Game class has the following methods:

#### **Method**: Game.run

This is the main method of the Game class and its purpose is to run a game between selected players.

The method:

1. Iterates for each turn in the game between 1 and game\_settings.num\_terns and:
   1. Prints the current turn number
   2. Iterates over all the players and for each player
      1. Prints the current player
      2. Calls the GAME.PLAY method
      3. Captures the score from the PLAY method
2. If the number of players is 2
   1. And if the score is a draw the method
      1. Prints the draw score and the draw rules
      2. Iterates over all the players and for each player
         1. Prints the current player
         2. Calls the GAME.PLAY method (now with different number of dice)
   2. Finally, the method prints the final score and report the winner and the loser

The following diagram illustrates the workflow of this method:



#### **Method**: Game.Play

This method is called whenever a user rolls one or more cubes.

The method has three inputs:

* The player number (representing Player ‘P’)
* The number of cubes (dice) to roll (N)
* A Boolean flag to indicate if a bonus method should be called (towards the end of this method)

The method mimic rolling N dice by player P by calling N times the random.randint

It adds the number of points that were rolled to the score of player P

It may call the Game.Bonus method depends on the input Boolean flag (see above)

Last, it validates the each player score is not negative, and if so, it set it to be 0.

#### **Method**: Game.Bonus

1. This method is called after each player has rolled its two dice
2. It receives the last roll (points) of each of the cubes
3. It has three cases:
   1. Double (both cubes has the same number of points)
      1. Call the Play method (reclusively!) with the ‘double number of cubes’ settings and indicating that bonus should not be called again.
   2. Odd (see even)
   3. Even

In both the odd and the even, this method add the odd/even bonus to the score

#### **Class**: Game users

This class manage and stores the users.

The class maintain the following variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable name** | **Data type** | **Comment & description** | **Initialized** |
| file\_name | String (path) | A path to a pickle file | Initialize in \_\_init\_\_ to be:  'game\_storage.pkl' There is no method to change the file name or its location (directory). |
| s | Dictionary | See description of the dictionary below | Initialized from a pickle file in \_\_init\_\_ or to an empty dictionary is pickle file does not exists |

The dictionary contains: For each user

* Username
* Passwords
* Number of games that the user win
* Number of games that the user lost

It maintains this information in a nested dictionary data type that looks like:

Dictionary[username][‘password’] -> password  
Dictionary[username][‘win’] -> the number of games that username won: integer initialize to 0  
Dictionary[username][‘lost’] ->the number of games that username lost: integer initialize to 0

It stores the dictionary in a [pickle file](https://docs.python.org/3/library/pickle.html)

I chose pickle file because:

1. It is very simple and quick to implement
2. It is a binary file and hence the password cannot be easily seen by other users.
3. I considered JSON file but I found it more complicated

I have learned a lot about pickle and json files from the following references:

<https://docs.python.org/3/library/pickle.html>  
<https://wiki.python.org/moin/UsingPickle>  
<https://pythontips.com/2013/08/02/what-is-pickle-in-python/>

Game users contains the following methods:

#### **Method**: Game users.\_\_init\_\_

This method searches for a file with the name ‘game\_storage.pkl’. If such a file exists, it initialize the dictionary (see above) from this pickle file.

#### **Method**: Game users.Store

This method stores the dictionary to a pickle file ‘game\_storage.pkl’.

This method is called whenever the ‘dictionary’ changes

#### **Method**: Game users.ADD\_USER

This method is called to add a user

It lets the user to input his username. It validates that the username matches the username policy

The username policy that I have implemented demands that username

* Must contain at least 3 characters
* Must not exceed 30 characters
* Must not contain spaces

It then lets the user to input his password. It validates that the password matches the password policy

The password policy that I have implemented is very similar to the username policy, it demands that a password

* Must contain at least 3 characters
* Must not exceed 30 characters
* Must not contain spaces

The method then adds the username and its password to the dictionary and initialize the number of wins and losses of this new user to be 0.

Finally, as the dictionary has changed, it calls the store method.

The following diagram illustrates the workflow of the add\_user method



#### **Method**: Game users.select\_players

This method is called to select N players to play in a new game.

It receives an input of the number of players to select (for the game).

It lets the user input his username and validates that:

* The username exists in the GAME USERS dictionary
* The username was not selected already
* The user has insert a string, otherwise the method returns with no players selected.

It then lets the user 3 attempts to input his password and validates that the password matches. If the password does not matches, after three attempts, the method returns with no players selected.

The following diagram illustrates the workflow of the select\_player method



#### **Method**: Game users.user\_win

This method is called to increment the number of times that a user wins the game

It changes the dictionary accordingly and store the dictionary

#### **Method**: Game users.user\_loss

This method is called to increment the number of times that a user loss the game

It changes the dictionary accordingly and store the dictionary

#### **Method**: Game users.user\_records

This method iterates over all the users and foreach user it prints a line with the username, the number of times that the user won a game and the number of times that the user loss a game. i.e. it prints the content of the dictionary of this class except the passwords.

#### **Method**: Game users.number\_of\_users

This method returns the size of the dictionary, i.e. the number of users that are available.

#### **Additioanl Method**

There are also the following methods:

* **query\_yes\_no** – the content copied from <https://stackoverflow.com/questions/3041986/apt-command-line-interface-like-yes-no-input>
* **encrypt** and **decrypt** – currently not complete. Now these methods simply return the input. Yet in the future they need to encrypt and decrypt passwords.
* **select\_number –** a method to that enable the user to select an integer number within a range of numbers

#### **Unit tests - test\_game()**

TBD

#### **References**

The main source is: <https://docs.python.org/3/tutorial/>

I used it for almost each of the statements that I wrote

In addition, I found the following source very useful:

* <https://www.tutorialspoint.com/python/python_basic_operators.htm>

Additional sources that I used:

* <https://stackoverflow.com/questions/4841436/what-exactly-does-do-in-python>
* <https://stackoverflow.com/questions/159720/what-is-the-naming-convention-in-python-for-variable-and-function-names>
* <https://www.pythoncentral.io/how-to-generate-a-random-number-in-python/>
* <https://www.pythonforbeginners.com/basics/getting-user-input-from-the-keyboard>
* <https://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/io.html>
* <https://www.hackerearth.com/practice/python/getting-started/input-and-output/tutorial/>
* <https://pypi.org/project/getch/>
* <https://stackoverflow.com/questions/983354/how-do-i-make-python-to-wait-for-a-pressed-key>

I tried the msvcrt.getch from above, but could not bring it to work... hence I used (revert to) the python 'input' method.

* <https://www.programiz.com/python-programming/keyword-list>
* <https://introcs.cs.princeton.edu/python/appendix_precedence/>
* <https://www.programiz.com/python-programming/examples/odd-even>
* <http://interactivepython.org/runestone/static/pip2/Functions/Returningavaluefromafunction.html>
* <https://stackoverflow.com/questions/9202224/getting-command-line-password-input-in-python>
* <https://www.pythonforbeginners.com/files/reading-and-writing-files-in-python>
* <https://www.tutorialspoint.com/python/python_strings.htm>
* <https://www.programiz.com/python-programming/dictionary>
* <https://stackoverflow.com/questions/1047318/easiest-way-to-persist-a-data-structure-to-a-file-in-python>
* <https://docs.python.org/3/library/pickle.html>