Creating game classes

# Rock, Paper, Scissors, (Lizard, Spock)

Create the following classes.

Each time when you add a method to the class, import the class into the Python console and test it out. Does it do what you expect it to do?

The code for each object is shown below, but see what you can do without looking at the code.

## PlayerObject

This class will allow the user to create instances of “rock”, “paper” and “scissors”, which could be extended to allow objects “lizard” and “spock”, or indeed you could choose your own objects and win cases.

The object should have:

* **Class attributes:**
  + allowable\_objects – a tuple of the object that can be created
  + win\_dict – a dictionary with keys being the allowable\_objects and values being a list of the objects which the key beats.
* **\_\_init\_\_** initialise the object with its name. Raise a ValueError if the name is not in allowable\_objects
* **Class method:**
  + **random\_object(cls)** return a random PlayerObject from the list of allowable objects
* **dunder methods (or magic methods) :** override these methods
  + **\_\_repr\_\_** returns a string representing the object if the user prints it
  + **\_\_eq\_\_(self, other)** return true if self and other are objects with the same name
  + **\_\_gt\_\_(self, other)** return true if other.name is in the list of object that self.name beats in win\_dict

## Player

This class stores and modifies general information about a Player. We will use it as the superclass for a HumanPlayer and a ComputerPlayer

Class methods:

* **\_\_init\_\_** initialise the player with a name given as an argument. Set the player attributes self.score to 0 and self.current\_object to None.
* **reset\_object** reset self.current\_object to None
* **win\_round** add one to self.score
* **\_\_repr\_\_** return a string that represents the player (name, score etc.)

## HumanPlayer

This is a subclass of Player and represents a human player.

It has only one method:

* **choose\_object** this will take a *choice* argument and set the current\_object to PlayerObject(*choice*)

## ComputerPlayer

This has two methods:

* **\_\_init\_\_** this function calls **super** to initialise the Player with the name “Computer”
* **choose\_object** this sets self.current\_object to a random PlayerObject using the PlayerObject.random\_object method.

## Game

The Game object keeps track of the overall game situation.

This includes the human and player objects, which become attributes of the game.

The Game also keeps track of the number of rounds played and the determines whether the maximum number of rounds has been reached.

Finaly the Game contains reporting methods that return strings showing the game status. These strings can be used by the Game interface (another object) to show to the user.

### Game attributes:

Game attributes are:

* *current\_round*: the number of the round being played
* *max\_rounds*: the maximum number of rounds that can be played
* *players*: a list containing the player objects
* *round\_result:* None if the round is yet to be played, otherwise draw, or won
* *round\_winner:* None or the player who has won the current round

### Game methods:

* **\_\_init\_\_:** set the attributes to default values
* **add\_human\_player:** add a HumanPlayer instance to the list of players
* **add\_computer\_player:** add a ComputerPlayer instance to the list of players
* **set\_max\_rounds:** validate and set *self.max\_rounds* to be an integer
* **find\_winner:** check that the players have both selected an object and compare the objects to see if there is a winner, or if the round has been drawn. Set *self.round\_result* and *self.max\_rounds* appropriately
* **next\_round:** set up the next round by incrementing *self.current\_round*, resetting *self.round\_result* and *self.round\_winner*, and calling the *reset\_object* method for each of the players
* **is\_finished:** return True/False if *self.current\_round* is greater than or equal to *self.max\_rounds*
* **reset:** reset the game so that current round is set to zero and player scores are all set to zero
* **report\_round:** returns a string message saying what each player choose, whether the round was drawn or won and if it was won, which player won
* **report\_score:** returns a string message saying what the current score is for each player
* **report\_winner:** compares the player scores and returns a string saying which player is the overall winner or returns a string saying that the game is drawn.

## ClInterface

The ClInterface (Command Line Interface) object is a object that will run the game from the command line. It will allow the user to input the information required to set up the game and input the player choices. Inputs will be sent to the Game object to process.

The interface will also print out reports on the games’ progress which are returned by the Game object.

### ClInterface attribute:

The only attribute is *self.game* which is an instance of the Game object.

### ClInterface methods:

* **\_\_init\_\_:** set self.game to be a new Game object
* **set\_up:** print a welcome message. Allow the user to choose two players as either human or computer players and add them to the game. The calls *self.input\_max\_rounds*
* **input\_max\_rounds:** gets an input and users *self.set\_max\_rounds* to set the maximum number of rounds
* **get\_choices:** cycles through *self.game.players* and uses the appropriate human or computer *choose\_object* method to choose the object.
* **run\_game:** loops through round of the game until *self.game\_is\_finished* is true.

In each round of the game:

* + run *self.get\_choices* to get set the player choices;
  + run *self.game.report\_round()* and *self.game.report\_score()* to get game status messages;
  + print the results.

When the game is finished run and print the message from *self.game.report\_winner*

* **run\_sequence**: runs the initial *self.set\_up* and *self.run\_game* and then loops round running *self.run\_game* until the user wants to quit.

# UML Class Diagrams

Graphical user interface, text, application

Description automatically generated

# Code Example

## PlayerObject

## Text Description automatically generated

## Player

Graphical user interface, text, application

Description automatically generated

## Human Player

Text

Description automatically generated

## Computer Player

Graphical user interface, text, application

Description automatically generated