# Example 1. 5-dice ver1

|  |  |
| --- | --- |
| ver0 | original description |
| ver1 | finding class, attribute and method candidates |

## Original description

5-dice, where a user places a bet 1-100 points, throws 5 dice and if their faces are all same, he wins 5 000 \* his bet. If there are 4 same he wins 400 \* the bet, with 3 same faces 30\*the bet and with 2 same faces the win is 2 \* the bet. If there are no same faces, the player loses the bet. It is not possible to lose more points in one game than what the player has placed. When the game is played the first time the user has an initial pot of 100 points, which can be used to place a bet. The points are saved from each game. Each game either adds points to the pot or the bet is lost. When the pot is gone (= 0), the user can restart the whole game and get a new initial 100 points pot.

## 1st iteration, check nouns

**5-dice, where a user places a bet 1-100 points, throws 5 dice and if their faces are all same, he wins 5 000 \* his bet. If there are 4 same he wins 400 \* the bet, with 3 same faces 30\*the bet and with 2 same faces the win is 2 \* the bet. If there are no same faces, the player loses the bet. It is not possible to lose more points in one game than what the player has placed. When the game is played the first time the user has an initial pot of 100 points, which can be used to place a bet. The points are saved from each game. Each game either adds points to the pot or the bet is lost. When the pot is gone (= 0), the user can restart the whole game and get a new initial 100 points pot.**

|  |  |  |  |
| --- | --- | --- | --- |
| noun | basic form, synonym, duplicate value, notes | value or collections | possible class, object, attribute |
| **5-Dice** |  | one value, name of the game, title | game’s title |
| **game** |  | object, one run of the game inside the program | game, possible class, knows the rules of the game and keeps track of the game’s data |
| **dice** |  | Object, model a basic dice | Dice, possible class, |
| face | faces, | Type:Integer | Possible attribute of dice class |
| **point** | Points, | Type:integer | Possible Attribute of player class |
| **pot** |  | Type: Integer | Possible attribute of game class |
| **Initial pot** |  | Type: Integer, Value=100 | Possible attribute of game class |
| **player, user** |  | Object, | Player, possible class, |
| **same\_faces** | user’s input, type integer |  | game compares the number to magic number, possibly knows the guess or all the guesses or the number of guesses |
| **face\_values** | result of checking | text in user interface |  |

## 2nd iteration, check verbs

**5-dice, where a user places a bet 1-100 points, throws 5 dice and if their faces are all same, he wins 5 000 \* his bet. If there are 4 same he wins 400 \* the bet, with 3 same faces 30\*the bet and with 2 same faces the win is 2 \* the bet. If there are no same faces, the player loses the bet. It is not possible to lose more points in one game than what the player has placed. When the game is played the first time the user has an initial pot of 100 points, which can be used to place a bet. The points are saved from each game. Each game either adds points to the pot or the bet is lost. When the pot is gone (= 0), the user can restart the whole game and get a new initial 100 points pot.**

|  |  |  |
| --- | --- | --- |
| verb | subject – object (who does, who is the target) | possible action, function, method |
| Bet = Place | Player’s action, places a value between 1-100 | Adds an initial value to the pot |
| Throw | Player’s action, | Generate random values for the dices |
| Add | Game’s action | Adds value to the pot |
| Remove | Game’s action | Removes value from the pot |
| Save | Game’s action | Saves the previous games’ bets and points for the user. |
| Play | Game’s action | Starts the game |
| pass | pass as an argument to a function (game’s method) |  |
| restart | loop to play the game (generate new game initial pot = 0, players points = 100) | driving program  game needs to be re-initialized, restarted |

## 3rd iteration, specification ver1

### Description

1. 5-Dice is a class that has
   1. attributes (it knows):
      1. title, tells which rules a game object follows
      2. points
         1. type: integer
         2. value: how much you have points
      3. Initial pot how much points has placed in one round
      4. Dices Machine trhoe 5 dices and tell them faces.
         1. type: list
         2. value: 5 numbers between 1-6

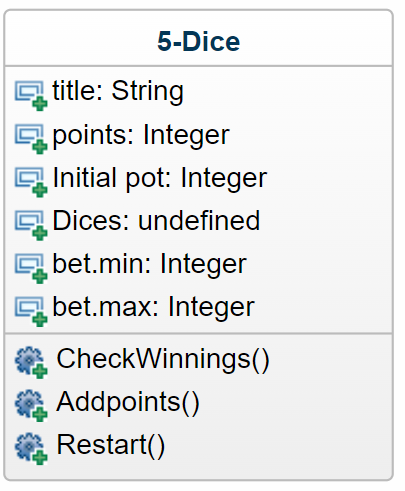
1.15. bet min and max (how much player can bet

1.15.1 types: integers

1.15.2 min bet is 1 point and max is 100 or how much player have points.

* 1. methods
     1. \_\_init\_\_, initialize all attributes
     2. restart, reset the initial pot
     3. checkWinnigs,
        1. How many same faces dices have
        2. returns the initial pot (how much player won)

### Generated UML class diagram (<https://app.genmymodel.com/>) @ver1



### Code generated from the UML diagram @ver1

class 5Dice(object):

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

self.title = ""

self.points = 0

self.initial\_pot = 0

self.dices =

self.bet.min = 0

self.bet.max = 0

# Start of user code -> properties/constructors for 5-Dice class

# End of user code

def CheckWinnings(self):

# Start of user code protected zone for CheckWinnings function body

raise NotImplementedError

# End of user code

def Addpoints(self):

# Start of user code protected zone for Addpoints function body

raise NotImplementedError

# End of user code

def Restart(self):

# Start of user code protected zone for Restart function body

raise NotImplementedError

# End of user code

# Start of user code -> methods for 5-Dice class

# End of user code

# Start of user code -> functions/methods for object-oriented programming package

# End of user code

### Completed code @ver1

import random

class 5Dice:

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

self.title = "5dice"

self.points = 0

self.initial\_pot = 0

self.dices =

self.bet.min = 0

self.bet.max = 0

def CheckWinnings(self):

def Addpoints(self):

def Restart(self):