# 5-dice ver3

|  |  |
| --- | --- |
| ver0 | original description |
| ver1 | finding class, attribute and method candidates |
| ver2 | description refining the use of 5-dice |
| ver3 | refining the logic, attributes and methods. |

## 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 |
| **bet\_value** |  | intger  Value: 1-100 |  |
| **dice** |  | List with 5 random integers | Dice, possible class, |
| matching\_faces |  | Type:Integer  How many same faces are trhown | Possible attribute of dice class |
| **points** | Points, | Type:integer | Possible Attribute of player class |
| **pot** |  | Type: Integer | Possible attribute of game class |
| **Initial pot** |  | Type: Integer, Value=0  how much is pot on current round | Possible attribute of game class |
| **player, user** |  | Object, | Player, possible class,  that has Methods throw and place bet, and attributes points |

## 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 bet value to initial pot |
| throw | Player’s action, | Generate random values for the dices |
| calc\_points | Game’s action | Calculates how much player won or lost |
| match | Game’s action | Calculates how many same faces have trhown |
| change\_points | Game’s action | gives or takes points from user |
| restart | loop to play the game (generate new game initial pot = 0, players points = 100) | driving program  game needs to be re-initialized, restarted |
| bet | User bets, Game takes the bet | method for user input getting the bet |

3rd iteration, specification ver1,ver2 and ver3

@ver3 description

5-Dice, is a game that is played with 5 dices. At the beginning player places a bet between 1-100. Then game trhow 5 dices and check how many of the has same faces. If 2 same faces player win 2\*bet, if 3 same faces player win 30\*bet, if 4 same faces player win 400\*bet and if all the dices has same face player win 5000\*bet. Player can quit anytime he wants, and game ends when player have 0 points.

Description

ver3 description added with changes in ver2

Attributes that are removed with @ver3:

Bet

Attributes that are added with @ver3:

initial\_pot

name, id, points -> player class

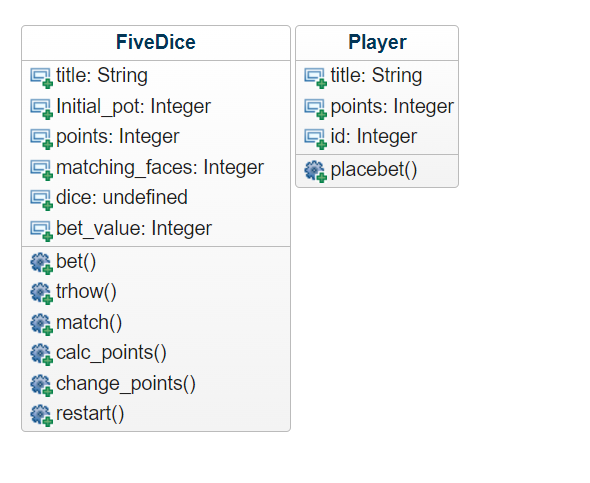
Methods that are added with @ver3

bet, change\_points

place\_bet -> player class

Description

1. Double dice is a class that has:
   1. Attributes
      1. title
         1. Type sting, Title of the game
      2. bet\_value
         1. Type integer holds value how 1-100
      3. points
         1. Type integer, starting points for player
         2. Value 100
      4. dice
         1. Type list, with 5 random integers between 1-6
      5. matching\_faces
         1. Type integer, how many same values in dice
      6. initial\_pot
         1. Type integer, how much is bet in one round
   2. Methods
      1. bet
         1. inputs players value 1-100
      2. trhow
         1. puts five random integers between 1-6 to dice
      3. match
         1. calculates how many matching faces in dicec
      4. calc\_points
         1. calculates how much player won or lost
      5. change\_points
         1. change player points based on calc\_points
      6. restart
         1. reserts player points to 100
2. Player is class that has:
   1. Attributes
      1. name
         1. Stores the name/nicname of the player
      2. id
         1. Stores an integer value of player id
      3. points
         1. Stores the points the plaer receives during a game
   2. Methods
      1. place a bet
         1. Insets the desired bet points between 1-100



Generated UML class diagram (<https://app.genmymodel.com/>) @ver3

Code generated from the UML diagram @ver3

class FiveDice(object):

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

self.title = ""

self.points = 0

self.Initial\_pot = 0

self.dices =

self.bet.max = 0

self.dicevalue = 0

self.points = 0

self.matching\_faces = 0

self.dice =

self.bet\_value = 0

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

# End of user code

def bet(self):

# Start of user code protected zone for bet function body

raise NotImplementedError

# End of user code

def trhow(self):

# Start of user code protected zone for trhow function body

raise NotImplementedError

# End of user code

def match(self):

# Start of user code protected zone for match function body

raise NotImplementedError

# End of user code

def calc\_points(self):

# Start of user code protected zone for calc\_points function body

raise NotImplementedError

# End of user code

def change\_points(self):

# Start of user code protected zone for change\_points 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 FiveDice class

# End of user code

class Player(object):

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

self.title = ""

self.points = 0

self.id = 0

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

# End of user code

def placebet(self):

# Start of user code protected zone for placebet function body

raise NotImplementedError

# End of user code

def (self):

# Start of user code protected zone for function body

raise NotImplementedError

# End of user code

# Start of user code -> methods for Player class

# End of user code

class dict(object):

pass

# Start of user code -> properties/constructors for dict class(interface)

# End of user code

# Start of user code -> methods for dict class(interface)

# End of user code

class Dices(object):

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

self.title = ""

self.id =

self.nopat = None

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

# End of user code

# Start of user code -> methods for Dices class

# End of user code

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

# End of user code

Complete code @ver3

from random import randint

class FiveDice:

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

self.title = "5-Dice"

self.bet\_value = 0

self.points = 100

self.dice = [0, 0, 0, 0, 0]

self.matching\_faces = 0

self.initial\_pot = 0

def throw(self):

for i in range(0, len(self.dice)):

self.dice[i] = randint(1, 6)

def bet(self):

bet\_value = input(f"\nPlace a bet between 1 and 100 points: ")

if not (bet\_value.strip().isnumeric() and 1 <= int(bet\_value.strip()) <= 100):

print("Wrong type or Wrong interval!")

self.bet()

else:

self.bet\_value = int(bet\_value)

return self.bet\_value

self.bet\_value = int(bet\_value)

return self.bet\_value

def match(self):

self.matching\_faces = 0

for item in self.dice:

if self.dice.count(item) > self.matching\_faces:

self.matching\_faces = self.dice.count(item)

else:

continue

return self.matching\_faces

def calc\_points(self):

self.initial\_pot = 0

match self.matching\_faces:

case 1:

self.initial\_pot -= self.bet

print("No matches, bet lost")

case 2:

self.initial\_pot = 2\*self.bet\_value

case 3:

self.initial\_pot = 30\*self.bet\_value

case 4:

self.initial\_pot = 400\*self.bet\_value

case 5:

self.initial\_pot = 5000\*self.bet\_value

return self.initial\_pot

def change\_points(self):

self.points += self.initial\_pot - self.bet\_value

return self.points

def restart(self):

self.points = 100