# 5-dice ver4

|  |  |
| --- | --- |
| 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. |
| ver4 | Tkinter implementation, class inheritance and composition |

## 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, refining the logic, attributes and methods for ver2

@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.

## 4th iteration, Tkinter implementation, class inheritance and composition for ver3

@ver4 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.

### Attributes that are removed with @ver4:

points

### Attributes that are added with @ver4:

dice\_faces, player\_points -> renamed points, rules

### Methods that are removed with @ver4

bet,

### Methods that are added with @ver4

show\_rules

1. class FiveDice:
   1. Attributes
      1. title
         1. Stores the title of the game
      2. dice\_faces
         1. Stores Unicode characters of the dice
      3. bet\_value
         1. Stores the bet value of the player
      4. dice
         1. Stores the integer representations of the generated 5 dice
      5. matching\_faces
         1. Stores the value for how many same faces come in a single dice throw
      6. initial\_pot
         1. Stores the beginning value for the pot, set to zero
      7. rules
         1. Store the Five Dice game rules
   2. Methods
      1. throw
         1. Generates five dice and retrieves the unicode characters for the corresponding face
      2. match
         1. Calculates how many same faces came with dice thrown
      3. calc\_points
         1. Calculates how many points a player wins or loses
      4. change\_points
         1. Changes player points based on calc\_points
      5. show\_rules
         1. Shows the Five Dice game rules
2. class Player
   1. Attributes
      1. player\_name
         1. Stores the player’s name
      2. player\_points
         1. Stores the points of the player
      3. player\_count
         1. Stores the amount of players generated by the player class
      4. player\_id
         1. Stores an id for the player instance
3. class GamePlay
   1. Attributes
      1. game\_play
         1. Runs the game
      2. Restart
         1. Restarts the game

Graphical user interface, text

Description automatically generated

Generated UML class diagram (<https://app.genmymodel.com/>) @ver4

Code generated from the UML diagram @ver4

class Player(object):

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

self.player\_name = ""

self.player\_points = 0

self.player\_count = 0

self.player\_id = 0

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

# End of user code

# Start of user code -> methods for Player class

# End of user code

class FiveDice(object):

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

self.title = ""

self.dice\_faces = None

self.bet\_value = 0

self.player\_points = 0

self.dice = None

self.matching\_faces = 0

self.initial\_pot = 0

self.rules = ""

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

# End of user code

def throw(self):

# Start of user code protected zone for throw 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 show\_rules(self):

# Start of user code protected zone for show\_rules function body

raise NotImplementedError

# End of user code

# Start of user code -> methods for FiveDice class

# End of user code

class GamePlay(object):

pass

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

# End of user code

def game\_play(self):

# Start of user code protected zone for game\_play 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 GamePlay class

# End of user code

# Start of user code -> functions/methods for FiveDice package

# End of user code

Complete code @ver4

# File : fivedice.py

# Authors : Sebastian Sopola, Uras Ayanoglu, Jerry Karkainen

# Description: This is game where you play with 5 dices

# --------------------------------------------------------------------------------------------------------------------------------------

# Import necessary libraries.

from tkinter import messagebox

import tkinter as tk

import tkinter.ttk as ttk

from PIL import Image,ImageTk

from pathlib import Path

from random import randint

from minigames.game\_components import GamePlay, Player

# --------------------------------------------------------------------------------------------------------------------------------------

class FiveDice(tk.Frame, GamePlay, Player):

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

parent.update()

self.width = parent.winfo\_width()

self.height = parent.winfo\_height()

'''initializes Window's attributes'''

super().\_\_init\_\_(master=parent)

self.parent = parent

self.title = "5-Dice"

self.dice\_faces = ['\u2680', '\u2681', '\u2682', '\u2683', '\u2684', '\u2685']

self.bet\_value = 0

self.player\_points = 100

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

self.matching\_faces = 0

self.initial\_pot = 0

self.rules = """

5-Dice Game Rules:

5-Dice, is a game that is played with 5 dices. Player starts the game with 100 points.

At the beginning of the game, the player needs to place their bet which is always between 1 and 100 points.

Then five dice are thrown and game will check how many of the dice have the same faces.

1. If 2 same faces come, player wins 2 \* the bet s/he placed,

2. If 3 same faces come, player wins 30 \* the bet s/he placed,

3. If 4 same faces come, player wins 400 \* the bet s/he placed,

4. If all of the dice have same faces, player wins 5000 \* the bet s/he placed.

- Overall points for the game cannot be negative and in case this happens the points are set to zero.

And the game ends.

- If the player wants to continue playing, they can restart the game by pressing the restart button.

- If the player wants to quit the game, they can press the quit button.

- The points are saved for each game.

"""

game\_label = ttk.Label(self, text=self.title, font=("Helvetica", 40))

game\_label.grid(row=0, column=0, sticky=tk.NSEW)

# Text box for bet amount

bet\_label = ttk.Label(self, text='Your Bet:')

bet\_label.grid(row=3, column=0, sticky=tk.E)

self.bet\_entry = ttk.Entry(self, width=10)

self.bet\_entry.grid(row=3, column=1, sticky=tk.NSEW)

# Player Points Label

self.points\_label = ttk.Label(self, text=f"Points: {self.player\_points}")

self.points\_label.grid(row=0, column=7, sticky=tk.NE)

# dice frame

dice\_label\_frame = ttk.LabelFrame(self)

self.dice\_labels = []

for i in range(5):

dice\_label = ttk.Label(dice\_label\_frame, font=("Helvetica", 100))

dice\_label.pack(side=tk.LEFT)

self.dice\_labels.append(dice\_label)

dice\_label\_frame.grid(row=1, column=0, columnspan=6, sticky=tk.NSEW)

self.winning\_label = ttk.Label(self, text="", font=("Helvetica 20"))

self.winning\_label.grid(row=2, column=0, sticky=tk.NSEW)

# Buttons for betting and continuing the game

bet\_button = ttk.Button(self, text="Bet", command=self.game\_play)

bet\_button.grid(row=4, column=0,sticky=tk.NSEW)

#continue\_button = ttk.Button(self, text="Continue")

#continue\_button.grid(row=4, column=1, sticky=tk.NSEW)

# Pop up window for the rules

rules\_button = ttk.Button(self, text='Rules', command=self.show\_rules)

rules\_button.grid(row=5, column=4, sticky=tk.NSEW)

# Quit button

close\_button = ttk.Button(self, text='Quit', command=self.\_\_close)

close\_button.grid(row=6, column=4, sticky=tk.NSEW)

def throw(self):

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

self.dice[i] = self.dice\_faces[randint(0, 5)]

self.update\_dice\_labels()

def game\_play(self):

self.bet\_value = self.bet\_entry.get()

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

ttk.tkinter.messagebox.showinfo(title="Wrong input ", message="Please use just a numbers\nand place a bet between 1-100")

elif int(self.bet\_value) > self.player\_points:

ttk.tkinter.messagebox.showinfo(title="Bet value ", message="You can't bet more than your balance!")

else:

self.throw()

self.match()

self.calc\_points()

self.change\_points()

if self.player\_points == 0:

self.restart()

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 -= int(self.bet\_value)

case 2:

self.initial\_pot = 2\*int(self.bet\_value)

case 3:

self.initial\_pot = 30\*int(self.bet\_value)

case 4:

self.initial\_pot = 400\*int(self.bet\_value)

case 5:

self.initial\_pot = 5000\*int(self.bet\_value)

return self.initial\_pot

def change\_points(self):

if self.initial\_pot < 0:

self.winning\_label.config(text = f"You lost {abs(self.initial\_pot)} points!")

self.player\_points += self.initial\_pot

self.points\_label.config(text = f"Points: {self.player\_points}")

else:

self.winning\_label.config(text = f"You won {self.initial\_pot} points!")

self.player\_points += self.initial\_pot - int(self.bet\_value)

self.points\_label.config(text = f"Points: {self.player\_points}")

def restart(self):

if messagebox.askyesno("Restart", "You dont have points anymore\nDo you want to play again?"):

self.player\_points = 100

self.points\_label.config(text = f"Points: {self.player\_points}")

else:

self.parent.destroy()

def \_\_close(self):

'''asking if closing is intended'''

if messagebox.askyesno("Close", "Do you want to close the Double Dice game?"):

self.parent.destroy()

def show\_rules(self):

rules = ttk.tkinter.messagebox.showinfo(title="Rules", message=self.rules)

return rules

def update\_dice\_labels(self):

for i in range(5):

self.dice\_labels[i].config(text=str(self.dice[i]), foreground="black")

if \_\_name\_\_ == "\_\_main\_\_":

app = FiveDice()

app.mainloop()