# DoubleDice ver4

|  |  |
| --- | --- |
| ver0 | Original description |
| ver1 | Finding class, attribute and method candidates |
| ver2 | Description refining the use of DoubleDice |
| ver3 | Refining the logic, attributes and methods. |
| ver4 | Tkinter implementation, class inheritance and composition |

# Original Description

# Double dice, a game of betting the result of throwing two dice. First, a user places his or her bet, 1...100 points and makes the guess: the sum is less, equal or bigger than 7 e.g. 1+6, 2+5, 3+4. The program throws the dice. And, if the sum is less or bigger and the player guessed it correctly he wins the bet \* 100 points. If the sum is 7 and the user had guessed that he wins the bet \* 1000 points. But if the sum was bigger when the guess was smaller or wise versa, the user loses bet \* 10 points. And if the guess was equal and the sum is not 7 he loses bet \* 100 points. The points are saved from each game. There are no negative points, but the points are set to zero.

# 1st iteration, check nouns

Double dice, a game of betting the result of throwing two dice. First, a user places his or her bet, 1...100 points and makes the guess: the sum is less, equal or bigger than 7 e.g. 1+6, 2+5, 3+4. The program throws the dice. And, if the sum is less or bigger and the player guessed it correctly he wins the bet \* 100 points. If the sum is 7 and the user had guessed that he wins the bet \* 1000 points. But if the sum was bigger when the guess was smaller or wise versa, the user loses bet \* 10 points. And if the guess was equal and the sum is not 7 he loses bet \* 100 points. The points are saved from each game. There are no negative points, but the points are set to zero.

|  |  |  |  |
| --- | --- | --- | --- |
| noun | basic form, synonym, duplicate value, notes | value or collections | possible class, object, attribute |
| Double dice |  | one value, name of the game, title | game’s title |
| game |  | object, on run of the game inside the program | game, possible class, knows the rules of the game and keeps track of the game’s data |
| result | sum | one value, sum of two dice numbers. It is a sum of collection though. | attribute to dice |
| dice |  | object, gets thrown. | dice, possible class, makes the result to be guessed. |
| user, player |  | object, has points | possible class, can own points |
| points |  | one value, the amount of “score” you can get or lose by betting | attribute to user |
| program |  | program to drive the game object |  |
| bet |  | value of points that is put into the game | possible attribute to the game class |
| guess |  | value if the result is going to be </>/= of 7 | possible attribute to the game |
| zero |  | Minimum amount of score user can have | possible attribute to user as a minimum score threshold |

# 2nd iteration, check verbs.

Double dice, a game of betting the result of throwing two dice. First, a user places his or her bet, 1...100 points and makes the guess: the sum is less, equal or bigger than 7 e.g. 1+6, 2+5, 3+4. The program throws the dice. And, if the sum is less or bigger and the player guessed it correctly he wins the bet \* 100 points. If the sum is 7 and the user had guessed that he wins the bet \* 1000 points. But if the sum was bigger when the guess was smaller or wise versa, the user loses bet \* 10 points. And if the guess was equal and the sum is not 7 he loses bet \* 100 points. The points are saved from each game. There are no negative points, but the points are set to zero.

|  |  |  |
| --- | --- | --- |
| verb | subject-object(who does, who is the target) | possible action, function, method |
| is | Double dice is a game |  |
| betting | User bets game takes the bet | method for user input getting the bet |
| throw | game throws the dice | method for initializing the result |
| makes the guess | user makes guess if result is </>/= of 7 | method for initializing the guess |
| wins | user wins points, if the guess is correct | possible method for the game |
| loses | user loses points, if the guess is wrong | possible method for the game |
| saved | points are saved from each game | possible method for the game |
| are set to zero | points are set to zero if all points are lost, no negative values | possible method for the game |

# 3rd iteration, refining the logic, attributes and methods for ver2

### @ver3 description

Double dice, is a game that is played with two dice. At the beginning of the game, two dice are thrown but are not shown to the player. Player starts the game with 100 points. After the dice are thrown, the player needs to place their bet which is always between 1 and 100 points. Then the player is asked if the sum of the dice is less, equal, or bigger than 7. After being asked, the player makes their guess by selecting one of the options listed, "less", "equal" or "greater". If the player's guess and the sum are in the same range, the player gets its bet \* 100 points, e.g., player guesses "less" and the sum of the dice comes under 7. If the player guesses "equal", and the sum of the dice is 7 then the player gets its bet \* 1000 points but if the sum of the dice is not 7, then the player loses its bet \* 100 points. If the sum of the dice and the player's guess are in opposite ranges, then the player loses its bet \* 10. Overall points for the game cannot be negative and in case this happens the points are set to zero. The points are saved for each game.

## Description

ver3 description added with changes in ver2

### Attributes that are removed with @ver3:

Roll, value, points

### Attributes that are added with @ver3:

title, rules, dice\_sum, both\_dice, player\_guess, player\_points, initial\_points

name, id, points 🡪 Player class

### Methods that are added with @ver3

place\_bet, make\_guess à Player class

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

### Attributes that are removed with @ver4:

name, id, points 🡨 Player class

### Attributes that are added with @ver4:

dice\_dots, game\_round\_result

player\_name, player\_points, player\_count, player\_id 🡪 Player Class

### Methods that are removed with @ver4

guess, place\_bet, make\_guess, save\_points

### Methods that are added with @ver4

bet, result, show\_rules

game\_play, restart à GamePlay class

# Description

1. Double dice is a class that has:
   1. Attributes
      1. title
         1. @ver3 Type: String, Title of the game.
      2. dice1
         1. Type: Integer, values between 1 and 6
      3. dice2
         1. Type: Integer, values between 1 and 6
      4. both\_dice
         1. Type: Tuple, stores the values of dice1 and dice2
      5. dice\_dots
         1. @ver4 Type: List, Stores Unicode characters of the dice
      6. dice\_sum
         1. Type: Integer, values between 2 and 12
      7. bet\_value
         1. Type: Integer, values from 0 to 100
         2. @ver3 attribute name changed from betvalue to bet\_value
      8. player\_guess
         1. @ver3 Type: String, options: less, equal or greater
      9. win
         1. Type: Boolean, true if player won the bet
      10. game\_round\_result
          1. @ver4 Type: String, options: “You win”, “You lose”
      11. Initial\_points
          1. Type: Integer, starting points for the player which is 100.
      12. player\_points
          1. Type: Integer, the amount of points player has
          2. @ver3 attribute name changed from points to player\_points
      13. rules
          1. @ver3 Type: string, shows the rules for the game.
   2. Methods
      1. throw
         1. Assigns random numbers from 1 to 6 for each dice
      2. bet
         1. Inputs players bet value [1 - 100]
      3. sum
         1. Adds values of dice1 and dice2 and returns the sum.
      4. result
         1. Compares the sum and guess of the player and calculates the new points.
      5. save\_points
         1. Saves the result to points
      6. restart
         1. Restarts the game
      7. run
         1. Runs the game
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, application

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\_count = 0

self.player\_points = 0

self.player\_name = ""

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 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

class DoubleDice(object):

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

self.title = ""

self.dice1 = 0

self.dice2 = 0

self.both\_dice = None

self.dice\_dots = None

self.dice\_sum = 0

self.bet\_value = 0

self.player\_guess = ""

self.win = False

self.game\_round\_result = ""

self.initial\_points = 0

self.player\_points = 0

self.rules = ""

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

# End of user code

def throw(self):

# Start of user code protected zone for throw function body

return 0

# End of user code

def bet(self):

# Start of user code protected zone for bet function body

return 0

# End of user code

def guess(self):

# Start of user code protected zone for guess function body

return ""

# End of user code

def sum(self):

# Start of user code protected zone for sum function body

return 0

# End of user code

def result(self):

# Start of user code protected zone for result function body

return False

# End of user code

def show\_rules(self):

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

return ""

# End of user code

# Start of user code -> methods for DoubleDice class

# End of user code

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

# End of user code

# Completed code @ver4

# File doubledice.py

# Authors : Sebastian Sopola, Uras Ayanoglu, Jerry Karkainen

# Description: This is a dice game called that checks player guess if it is less, equal or greater than 7 against outcomes of the dice.

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

# Import necessary libaries

from tkinter import messagebox

import tkinter as tk

import tkinter.ttk as ttk

from PIL import Image,ImageTk

from pathlib import Path

import random

from minigames.game\_components import GamePlay, Player

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

class DoubleDice(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 = "Double Dice"

self.dice1 = 0

self.dice2 = 0

self.both\_dice = (self.dice1, self.dice2)

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

self.dice\_sum = 0

self.bet\_value = 0

self.player\_guess = tk.StringVar()

self.win = False

self.game\_round\_result = None

self.initial\_points = 100

self.player\_points = 100

self.rules = """

Double Dice Game Rules:

- Double dice, is a game that is played with two dice. Player starts the game with 100 points.

At the beginning of the game, two dice are thrown but are not shown to the player.

After the dice are thrown, the player needs to place their bet which is always between 1 and 100 points.

- Then the player is asked if the sum of the dice is less, equal, or bigger than 7.

After being asked, the player makes their guess by selecting one of the options listed, "less", "equal" or "greater".

1. If the player's guess and the sum are in the same range, the player gets its bet \* 100 points.

(e.g., player guesses "less" and the sum of the dice comes under 7.)

2. If the player guesses "equal", and the sum of the dice is 7 then the player gets its bet \* 1000 points

but if the sum of the dice is not 7, then the player loses its bet \* 100 points.

3. If the sum of the dice and the player's guess are in opposite ranges, then the player loses its bet \* 10.

(e.g., player guesses "less" and the sum of the dice come over 7.)

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

The points are saved for each game.

"""

# Display Game Title

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

game\_label.grid(row=0, column=0, columnspan=3, sticky=tk.NSEW)

# Label that shows the points

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

self.points\_label.grid(row=0, column=4, sticky=tk.NE)

# Display Dice

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

self.dice\_label.grid(row=1, column=0, columnspan=3, sticky=tk.NSEW)

# Display Result

self.result\_label = ttk.Label(self, font=("Helvetica", 40))

self.result\_label.grid(row=1, column=3, columnspan=3, sticky=tk.NSEW)

# Spinbox for setting the points

self.points\_slider = ttk.Spinbox(self, from\_=1, to=100)

self.points\_slider.grid(row=3, column=1, columnspan=2, sticky=tk.NSEW)

# Buttons for the bet, less, equal and greater options

bet\_button = ttk.Button(self, text='Bet', command=self.bet)

bet\_button.grid(row=3, column=3, sticky=tk.NSEW)

#style = ttk.Style()

#style.configure("TRadiobutton", background=self.cget("background"), foreground=self.cget("foreground"), indicatorsize=15)

#less\_button = ttk.Button(self, text='Less', command=self.run)

#less\_button.grid(row=3, column=1, sticky=tk.NSEW)

less\_button = ttk.Radiobutton(self, text='Less', variable=self.player\_guess, value='less', command=self.game\_play)

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

#equal\_button = ttk.Button(self, text='Equal', command=self.run)

#equal\_button.grid(row=3, column=2, sticky=tk.NSEW)

equal\_button = ttk.Radiobutton(self, text='Equal', variable=self.player\_guess, value='equal', command=self.game\_play)

equal\_button.grid(row=4, column=2, sticky=tk.NSEW)

#greater\_button = ttk.Button(self, text='Greater', command=self.run)

#greater\_button.grid(row=3, column=3, sticky=tk.NSEW)

greater\_button = ttk.Radiobutton(self, text='Greater', variable=self.player\_guess, value='greater', command=self.game\_play)

greater\_button.grid(row=4, column=3, 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)

# Close button

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

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

# End of user code

def throw(self):

# Start of user code protected zone for throw function body

dice\_dots = {1: '\u2680', 2: '\u2681', 3: '\u2682', 4: '\u2683', 5: '\u2684', 6: '\u2685'}

self.dice1 = random.randint(1, 6)

self.dice2 = random.randint(1, 6)

self.both\_dice = (self.dice1, self.dice2)

self.dice\_label.configure(text=f'{dice\_dots[self.dice1]} {dice\_dots[self.dice2]}')

self.dice\_label.grid()

return self.both\_dice

# End of user code

def points\_label\_change(self):

# Start of user code protected zone for points\_label\_change function body

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

self.points\_label.grid()

# End of user code

def bet(self):

# Start of user code protected zone for bet function body

self.bet\_value = self.points\_slider.get()

try:

not (1 <= int(self.bet\_value) <= 100)

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

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

raise ValueError

except ValueError:

ttk.tkinter.messagebox.showinfo(title="Wrong type or Wrong Interval!", message="Please place a bet between 1 and 100 points")

else:

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

ttk.tkinter.messagebox.showinfo(title="Less, Equal or Greater", message=f"Now you should click one of the options below\n'Less, Equal or Greater'")

print(self.bet\_value)

return self.bet\_value

# End of user code

def sum(self):

# Start of user code protected zone for sum function body

self.dice\_sum = self.both\_dice[0] + self.both\_dice[1]

return self.dice\_sum

# End of user code

def result(self):

self.bet\_value = int(self.points\_slider.get())

# Start of user code protected zone for result function body

if self.dice\_sum == 7:

if self.player\_guess.get() == 'equal':

self.game\_round\_result = "You win!"

self.win = True

self.player\_points += (self.bet\_value \* 1000 - self.bet\_value)

else:

self.game\_round\_result = "You lose!"

self.win = False

self.player\_points -= (self.bet\_value \* 10 + self.bet\_value)

elif self.dice\_sum < 7:

if self.player\_guess.get() == 'less':

self.game\_round\_result = "You win!"

self.win = True

self.player\_points += (self.bet\_value \* 100 - self.bet\_value)

else:

self.game\_round\_result = "You lose!"

self.win = False

self.player\_points -= (self.bet\_value \* 10 + self.bet\_value)

elif self.dice\_sum > 7:

if self.player\_guess.get() == 'greater':

self.game\_round\_result = "You win!"

self.win = True

self.player\_points += (self.bet\_value \* 100 - self.bet\_value)

else:

self.game\_round\_result = "You lose!"

self.win = False

self.player\_points -= (self.bet\_value \* 10 + self.bet\_value)

return self.win

# End of user code

def restart(self):

# Start of user code protected zone for restart function body

self.dice1 = 0

self.dice2 = 0

self.both\_dice = (self.dice1, self.dice2)

self.dice\_sum = 0

self.bet\_value = 0

self.player\_guess = tk.StringVar()

self.player\_points = self.initial\_points

# End of user code

def game\_play(self):

self.throw() # returns game.both\_dice --> tuple (dice1, dice2)

self.sum() # returns the sum of the dices --> int value

self.result()

self.result\_label.configure(text=f"{self.game\_round\_result}")

self.result\_label.grid()

self.points\_label\_change()

self.update()

if self.player\_points <= 0:

msg\_box = messagebox.showinfo('Info screen', message=f"Your overall points are negative now.We're setting it back to 0.")

exit\_box = tk.messagebox.askquestion('Exit Application', 'Would you like to play again?')

if exit\_box == 'yes':

tk.messagebox.showinfo('Info screen', 'You will now return back to the game and you can place a new bet. Good luck!')

self.restart()

self.points\_label\_change()

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):

"""Shows the rules of the game"""

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

return rules

# Start of user code -> methods for DoubleDice class

# End of user code

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

app = DoubleDice()

app.mainloop()

# Start of user code -> functions/methods for minigames lab1 Double dice package

# End of user code