**GenAI Assignment 3**

2241199

Anvesha Singh

**You**

help me write a code for a bot to play yahtzee

**Gemini**

The complexity of your Yahtzee bot code depends on the desired functionality. Here are two approaches:

**1. Basic Yahtzee Bot:**

This bot simulates rolling dice and allows selecting dice to re-roll. It won't make strategic decisions but provides a foundation for further development.

Here's a possible Python implementation:

Python

import random

def roll\_dice(num\_dice):

"""Simulates rolling dice and returns a list of results."""

return [random.randint(1, 6) for \_ in range(num\_dice)]

def play\_turn():

dice = roll\_dice(5)

print("You rolled:", dice)

re\_roll = input("Select dice to re-roll (e.g., 1,3): ")

re\_roll = [int(x) for x in re\_roll.split(",")]

for i in re\_roll:

dice[i-1] = roll\_dice(1)[0]

print("After re-roll:", dice)

# Example usage

play\_turn()

This code defines functions to simulate dice rolls and a turn where you can choose dice to re-roll.

**2. Strategic Yahtzee Bot (Advanced):**

This bot requires more effort and involves implementing strategies to maximize scoring. Two popular approaches exist:

* **Optimal Strategy Bot:** This bot uses pre-calculated expected values for different roll combinations and scoring options. It requires significant processing power and data pre-processing. You can find an example using JavaScript on GitHub<https://github.com/sorenchr/yahtzeebot>
* **Machine Learning Bot:** An AI bot can be trained through self-play or reinforcement learning to learn optimal strategies. This approach is complex but can be very effective. You can find an example using JAX library on GitHub<https://github.com/smo-key/Yahtzee-AI>

**Remember:**

These are just starting points. You can customize the code to fit your specific needs, like adding opponent logic, scorekeeping, or a user interface.

**You**

## make a bot that can play yahtzee with me and keep track of the scorecard

**Gemini**

**Here's a Python code for a Yahtzee bot that plays with you and keeps track of the scorecard:**

**Python**

**import random**

**class Yahtzee:**

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

**self.scorecard = {**

**"upper\_section": [None] \* 6, # Tracks scores for 1s to 6s**

**"lower\_section": {**

**"three\_of\_a\_kind": None,**

**"four\_of\_a\_kind": None,**

**"full\_house": None,**

**"small\_straight": None,**

**"large\_straight": None,**

**"yahtzee": None,**

**"chance": None**

**},**

**"bonus": 0,**

**"grand\_total": 0**

**}**

**self.dice = []**

**def roll\_dice(self, num\_dice):**

**"""Simulates rolling dice and returns a list of results."""**

**self.dice = [random.randint(1, 6) for \_ in range(num\_dice)]**

**return self.dice**

**def select\_dice\_to\_re\_roll(self):**

**"""Prompts user to select dice for re-roll."""**

**re\_roll\_str = input("Select dice to re-roll (e.g., 1,3): ")**

**return [int(x) for x in re\_roll\_str.split(",")]**

**def re\_roll(self):**

**"""Re-rolls selected dice."""**

**re\_roll\_dice = self.select\_dice\_to\_re\_roll()**

**for i in re\_roll\_dice:**

**self.dice[i-1] = self.roll\_dice(1)[0]**

**def calculate\_score(self, category):**

**"""Calculates score based on the chosen category and current dice."""**

**dice\_counts = [0] \* 7 # Initialize dice count (1 to 6)**

**for die in self.dice:**

**dice\_counts[die] += 1**

**if category in self.scorecard["upper\_section"]:**

**# Upper section (score based on matching number)**

**return dice\_counts[category] \* category if dice\_counts[category] >= category else 0**

**elif category == "three\_of\_a\_kind":**

**# Check for 3 of a kind**

**for num, count in enumerate(dice\_counts[1:]):**

**if count >= 3:**

**return sum(self.dice)**

**return 0**

**elif category == "four\_of\_a\_kind":**

**# Check for 4 of a kind**

**for num, count in enumerate(dice\_counts[1:]):**

**if count >= 4:**

**return sum(self.dice)**

**return 0**

**elif category == "full\_house":**

**# Check for full house (3 of a kind and a pair)**

**has\_three = False**

**has\_two = False**

**for count in dice\_counts[1:]:**

**if count == 3:**

**has\_three = True**

**elif count == 2:**

**has\_two = True**

**return 25 if has\_three and has\_two else 0**

**elif category == "small\_straight":**

**# Check for small straight (1, 2, 3, 4) or (2, 3, 4, 5)**

**return 30 if any(sorted(self.dice)[i:i+4] == [x for x in range(i+1, i+5)] for i in range(2)) else 0**

**elif category == "large\_straight":**

**# Check for large straight (1, 2, 3, 4, 5) or (2, 3, 4, 5, 6)**

**return 40 if any(sorted(self.dice) == [x for x in range(1, 6)] or sorted(self.dice) == [x for x in range(2, 7)]) else 0**

**elif category == "yahtzee":**

**# Check for Yahtzee (all dice same)**

**return 50 if all(count == dice\_counts[0] for count in dice\_counts[1:]) else 0**

**elif category == "chance":**

**# Chance (sum of all dice)**

**return sum(self.dice)**

**def choose\_and\_fill\_category(self):**

**"""Prompts user to choose a category and fills the scorecard."""**