Module: Town.py

# src/town.py

import random

from skills import skills

from items import items\_for\_sale, get\_item\_by\_name

class Town:

    def \_\_init\_\_(self):

        self.skills\_available = self.randomize\_skills()

        self.items\_for\_sale = self.randomize\_items\_with\_stock()

        self.visitors = []

    def randomize\_skills(self):

        available\_skills = [skill for category in skills.values() for skill in category.keys()]

        return random.sample(available\_skills, min(5, len(available\_skills)))

    def randomize\_items\_with\_stock(self):

        items = random.sample(items\_for\_sale, min(5, len(items\_for\_sale)))

        items\_with\_stock = [{\*\*item, "stock": random.randint(1, 5)} for item in items]

        return items\_with\_stock

    def add\_visitor(self, player):

        self.visitors.append(player)

        print(f"{player.name} has entered the town.")

    def remove\_visitor(self, player):

        self.visitors.remove(player)

        print(f"{player.name} has left the town.")

    def display\_skills(self):

        print("\nAvailable skills:")

        for skill\_name in self.skills\_available:

            skill\_info = skills[skill\_name]

            print(f"{skill\_name}: {skill\_info['description']}")

    def display\_items(self):

        print("\nItems for sale:")

        for item in self.items\_for\_sale:

            print(f"{item['name']} - {item['price']} gold (Stock: {item['stock']})")

    def learn\_skill(self, player, skill\_name):

        if skill\_name in self.skills\_available:

            player.learn\_skill(skill\_name)

        else:

            print(f"Skill {skill\_name} is not available in this town.")

    def buy\_item(self, player, item\_name):

        for item in self.items\_for\_sale:

            if item["name"] == item\_name:

                if item["stock"] > 0:

                    if player.money >= item["price"]:

                        player.buy\_item(item\_name)

                        item["stock"] -= 1

                        return True

                    else:

                        print(f"{player.name} does not have enough gold to buy {item\_name}.")

                        return False

                else:

                    print(f"{item\_name} is out of stock.")

                    return False

        print(f"Item {item\_name} is not available for sale in this town.")

        return False

    def sell\_item(self, player, item\_name):

        for item in player.inventory:

            if item["name"] == item\_name:

                for town\_item in self.items\_for\_sale:

                    if town\_item["name"] == item\_name:

                        town\_item["stock"] += 1

                        player.sell\_item(item\_name)

                        return

                self.items\_for\_sale.append({"name": item\_name, "price": item["price"], "stock": 1})

                player.sell\_item(item\_name)

                return

        print(f"{item\_name} not found in inventory.")

// End module

The Town class is a well-structured representation of a town in your game. It effectively encapsulates the key features and functionalities of a town, making it reusable and easy to manage. Here's a breakdown of its strengths:

**Strengths:**

* **Clear Responsibilities:** The class focuses on the town's specific attributes and actions, such as managing available skills, items for sale, and visitors. This clear separation of concerns makes the code more organized and maintainable.
* **Data Encapsulation:** The town's data (skills, items, visitors) is private, accessed only through methods. This promotes data integrity and prevents accidental modification.
* **Methods for Interaction:** The class provides methods for common town interactions, like learn\_skill, buy\_item, and sell\_item. This makes it easy for other parts of the game to interact with the town.
* **Randomization:** The randomize\_skills and randomize\_items\_with\_stock methods add a dynamic element to the town, ensuring that each town feels unique.
* **Visitor Management:** The add\_visitor and remove\_visitor methods allow the town to track who is currently present, which could be useful for various game mechanics.

**Potential Improvements:**

* **More Town-Specific Features:** You could expand the class to include more town-specific features, such as:
  + **Quests:** A system for managing quests given to players in the town.
  + **Reputation:** A mechanism to track the player's reputation in the town, affecting interactions.
  + **NPCs:** A way to represent and interact with non-player characters in the town.
* **Event Handling:** Consider adding a system for handling events that occur in the town, such as festivals, rumors, or random encounters.
* **Persistence:** If you want towns to persist between game sessions, you could implement a mechanism to save and load town data.

Overall, the Town class is a solid foundation for representing towns in your game. By adding more features and functionality, you can make it even more robust and engaging.