Англичанинов Александр Сергеевич

**Questscape**

Questscape - это текстовая RPG с визуальными элементами, где игроки исследуют фэнтезийный мир, сражаются с монстрами, выполняют квесты и развивают своих персонажей. Каждое решение игрока влияет на развитие персонажа, который имеет уникальные характеристики и навыки.

Реализованный функционал:

Интерфейс и Визуальные Элементы:

Игровая форма:

* Интерактивная форма, где игроки могут взаимодействовать с игровым миром через текстовое поле.

Визуальный кубик:

* На основной форме располагается визуальный 20-гранный кубик для бросков.
* Предоставляет анимацию броска и выводит результаты броска, определяя исход событий в игре.

Окно характеристик персонажа:

* Подробная информация о характеристиках персонажа, включая здоровье, ману, опыт и навыки.
* Графическое представление персонажа.

Игровой Процесс и Вариативность:

Исследование и Квесты:

* Возможность исследования разнообразных локаций, каждая из которых имеет свою уникальную атмосферу и задания.

Развитие и Персонализация:

* Возможность выбора класса персонажа и развитие его навыков.
* Апгрейд и улучшение характеристик персонажа.

История и Сюжет:

* Сюжетная линия.
* Взаимодействие с npc.

Дополнительные Функции:

Выбор темы:

* Возможность выбора между светлой и темной темой для настройки внешнего вида приложения.

Запись хроники в txt файл:

* Эта функция позволяет игрокам сохранить свои уникальные приключения и историю персонажа в текстовом формате, описывая ваши действия, решения и встречи в игре.

Классы и методы (с описанием):

Класс Dialog - это пользовательское диалоговое окно, наследуемое от QDialog. Этот класс предназначен для создания и настройки интерфейса для взаимодействия пользователя с программой.

Методы класса Dialog:

* \_\_init\_\_(self, \*args, \*\*kwargs): Конструктор класса, инициализирует диалоговое окно и устанавливает его начальные параметры, включая стиль, оконные флаги, анимацию открытия и эффект тени.
* initUi(self): Метод для инициализации пользовательского интерфейса. Создает виджеты, такие как кнопки закрытия и переключения режимов, а также устанавливает макеты для чата и других элементов интерфейса.
* toggle\_mode(self): Метод для переключения режима (дневной/ночной). Этот метод позволяет пользователю изменять тему интерфейса между дневным и ночным режимами.
* set\_day\_mode(self): Метод для установки дневного режима интерфейса. Он изменяет стиль окна и передает команду дочерним виджетам установить дневный режим.
* set\_night\_mode(self): Метод для установки ночного режима интерфейса. Он изменяет стиль окна и передает команду дочерним виджетам установить ночный режим.
* animate\_open(self): Метод для анимации открытия окна. Создает анимацию изменения прозрачности окна при его открытии.
* sizeHint(self): Метод, возвращающий предпочтительный размер окна.

Этот класс предоставляет основу для создания и настройки диалогового окна в PyQt, а также управления его внешним видом и поведением.

Класс ChatWindow - это окно для отображения сообщений и взаимодействия пользователя с игровым процессом.

Методы класса ChatWindow:

* \_\_init\_\_(self, hero, stats\_window): Конструктор класса, инициализирует окно чата и устанавливает его начальные параметры, такие как размер, виджеты, стиль и игровые переменные. Принимает объект hero и stats\_window для взаимодействия с героем и окном статистики.
* animate\_open(self): Метод для анимации открытия окна. Создает анимацию изменения прозрачности окна при его открытии.
* reset\_button\_opacity(self): Метод для сброса прозрачности кнопки после анимации.
* custom\_keyPressEvent(self, event): Переопределенный метод для обработки нажатия клавиш в текстовом поле. Отслеживает клавишу "Enter" и "Ctrl+Enter" для отправки сообщений.
* process\_enter\_key(self, event): Метод для обработки нажатия клавиши "Enter" в текстовом поле.
* set\_day\_mode(self): Метод для установки дневного режима интерфейса.
* set\_night\_mode(self): Метод для установки ночного режима интерфейса.
* send\_message(self): Метод для отправки сообщения. Обрабатывает текст из текстового поля, отображает сообщение в окне чата и взаимодействует с игровым процессом.
* game\_process(self, input): Метод для обработки игрового процесса. В зависимости от текущего состояния игры (уровня), обрабатывает ввод пользователя и взаимодействует с игровой логикой.
* fight(self, input): Метод для обработки боевого процесса в игре. Определяет действия игрока и врага в бою.
* display\_location(self, location): Метод для отображения информации о текущей локации игрока.
* story(self, input): Метод для обработки сюжетных событий в игре.
* ceate\_hero(self, input): Метод для создания персонажа.
* roll(self): Метод для выполнения броска кубика и обработки его результатов.
* print\_text(self, message, color="black"): Метод для вывода текста в окне чата. Он добавляет сообщения в очередь и обеспечивает плавное отображение сообщений.
* process\_message\_queue(self): Метод для обработки очереди сообщений и их отображения в окне чата.
* scroll\_to\_bottom(self): Метод для прокрутки текста в окне чата вниз.
* set\_button\_style(self, button): Метод для установки стиля кнопки.
* sizeHint(self): Метод, возвращающий предпочтительный размер окна.

Класс ChatWindow предоставляет пользователю интерфейс для взаимодействия с игрой, отправки сообщений и отображения игровой информации.

Класс Sword (Меч):

* \_\_init\_\_(self, name, damage): Инициализирует объект меча с указанным именем и диапазоном урона.

Класс Hammer (Молот):

* \_\_init\_\_(self, name, damage): Инициализирует объект молота с указанным именем и диапазоном урона.

Класс Staff (Посох):

* \_\_init\_\_(self, name, damage): Инициализирует объект посоха с указанным именем и диапазоном урона.

Класс Hero (Герой):

* \_\_init\_\_(self): Инициализирует объект героя с начальными характеристиками и оборудованием.
* attack(self, dice, select\_index, enemy): Выполняет атаку героя по выбранной цели с использованием заклинания.
* take\_damage(self, damage): Уменьшает здоровье героя на указанное количество урона.
* load\_spells\_from\_database(self, table\_name): Загружает данные заклинаний из базы данных.
* add\_random\_spells(self, num\_spells, hero\_class): Добавляет случайные заклинания герою на основе его класса.
* add\_spell(self): Добавляет новое заклинание герою, соответствующее его уровню.
* add\_curr\_spell(self, spell\_name): Добавляет указанное заклинание герою.
* get\_spell\_usage(self, index): Возвращает описание использования заклинания по его индексу.
* level\_up(self): Повышает уровень героя и предоставляет ему новое заклинание.
* spell\_complexity(self, spell): Возвращает характеристики сложности заклинания.
* info(self): Возвращает информацию о герое, включая имя, класс, расу, характеристики, оружие, деньги, уровень и опыт.
* stats(self): Возвращает статистику героя, включая класс, расу и характеристики.
* stats\_weapon(self): Возвращает информацию об оружии героя, включая название, тип, урон и вес.
* format\_Money(self, coins): Возвращает форматированное представление количества монет в виде платинов, золота, серебра и бронзы.

Класс Enemy (Враг):

Этот класс представляет общую структуру для всех врагов в игре. Враги имеют общие атрибуты, такие как имя, здоровье, урон, броня и опыт.

Методы класса:

* \_\_init\_\_(self, name, health=50, damage=5, armor=5, exp=100): Этот метод инициализирует объект класса Enemy с указанными атрибутами (имя, здоровье, урон, броня, опыт).
* take\_damage(self, damage): Метод, который вычитает урон, принятый врагом, и уменьшает его здоровье. Если здоровье становится отрицательным, оно устанавливается в 0.
* is\_alive(self): Метод, который возвращает True, если у врага осталось здоровье, и False, если он мертв.

Класс Goblin (Гоблин):

Этот класс представляет гоблинов, одного из видов врагов в игре. Он наследует все атрибуты и методы из класса Enemy и переопределяет некоторые из них.

Методы класса:

* \_\_init\_\_(self, name="Гоблин", health=60, damage=10, armor=0, exp=200): Метод инициализации объекта класса Goblin, который вызывает инициализатор родительского класса Enemy с определенными значениями по умолчанию.
* attack(self, hero, dice): Метод, который определяет, как гоблин атакует игрока в зависимости от значения "dice" (результат случайного броска). Метод возвращает описание атаки гоблина.
* escape\_attack(self, hero, dice): Метод, который определяет, как гоблин атакует игрока, когда тот пытается уйти от схватки. Метод возвращает описание атаки гоблина при попытке побега.

Класс Ogre (Огр):

Этот класс представляет огров, другого вида врагов в игре. Он также наследует все атрибуты и методы из класса Enemy и переопределяет некоторые из них.

Методы класса:

* \_\_init\_\_(self, name="Огр", health=300, damage=35, armor=15, exp=1000): Метод инициализации объекта класса Ogre, который вызывает инициализатор родительского класса Enemy с определенными значениями по умолчанию.
* attack(self, hero, dice): Метод, который определяет, как огр атакует игрока в зависимости от значения "dice". Метод возвращает описание атаки огра.
* escape\_attack(self, hero, dice): Метод, который определяет, как огр атакует игрока при попытке побега. Метод возвращает описание атаки огра при попытке побега.

Класс Robber (Разбойник):

Этот класс представляет разбойников, еще одного вида врагов в игре. Он также наследует все атрибуты и методы из класса Enemy и переопределяет некоторые из них.

Методы класса:

* \_\_init\_\_(self, name="Разбойник", health=150, damage=20, armor=5, exp=500): Метод инициализации объекта класса Robber, который вызывает инициализатор родительского класса Enemy с определенными значениями по умолчанию.
* attack(self, hero, dice): Метод, который определяет, как разбойник атакует игрока в зависимости от значения "dice". Метод возвращает описание атаки разбойника.
* escape\_attack(self, hero, dice): Метод, который определяет, как разбойник атакует игрока при попытке побега. Метод возвращает описание атаки разбойника при попытке побега.

Класс StatsWindow:

Этот класс создает окно с информацией о герое в игре. Он позволяет отображать характеристики героя, включая имя, изображение, статистику здоровья, маны, опыта, а также другую информацию.

Методы класса:

* \_\_init\_\_(self, hero): Конструктор класса StatsWindow, который инициализирует объект окна с информацией о герое. Принимает объект hero (герой) в качестве аргумента.
* initUI(self): Инициализирует пользовательский интерфейс окна. Устанавливает заголовок окна, геометрию и разметку. Создает и настраивает виджеты для отображения информации о герое, такие как имя, изображение, полосы здоровья, маны и опыта.
* update\_stats(self): Обновляет информацию о герое, включая показатели здоровья, маны и опыта. Также обновляет изображение героя, его имя и текстовые метки с дополнительной информацией.
* set\_day\_mode(self): Устанавливает дневной режим для окна, изменяя цветовую схему элементов интерфейса, такие как фон и текст.
* set\_night\_mode(self): Устанавливает ночной режим для окна, изменяя цветовую схему элементов интерфейса, такие как фон и текст.

Класс DiceApp:

Этот класс представляет приложение для броска 20-гранного кубика с графическим интерфейсом пользователя (GUI). Он предоставляет пользователю возможность бросить кубик и отображает результат с анимацией.

Методы класса:

* \_\_init\_\_(self, chat\_window): Конструктор класса DiceApp, который инициализирует объект приложения. Принимает chat\_window в качестве аргумента, представляющего окно чата.
* initUI(self): Инициализирует пользовательский интерфейс приложения. Устанавливает размер и заголовок окна, создает разметку (layout) и добавляет виджеты, такие как метка для отображения изображения кубика и кнопка для броска кубика.
* start\_animation(self): Запускает анимацию броска кубика. Выключает кнопку броска, изменяет стиль кнопки для отображения анимации и начинает анимацию.
* update\_animation(self): Обновляет анимацию броска кубика. Выбирает случайное изображение кубика и отображает его с анимацией вращения. Постепенно увеличивает интервал таймера для замедления анимации. По завершении анимации, отображает окончательное изображение кубика и передает результат в окно чата.

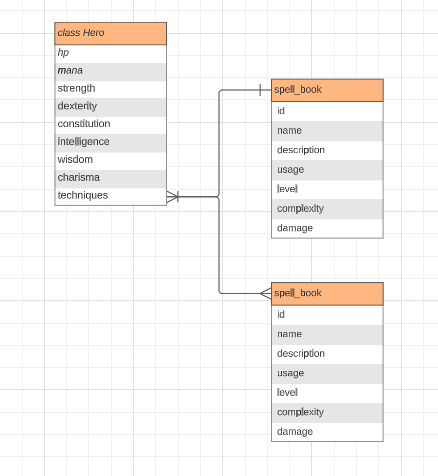
Класс RotatableLabel:

Этот класс представляет метку (label), которую можно вращать. Используется для отображения изображения кубика с возможностью вращения.

Методы класса RotatableLabel:

* \_\_init\_\_(self, parent=None): Конструктор класса RotatableLabel, который инициализирует объект метки. Принимает parent (родительский элемент) в качестве аргумента.
* angle(self): Геттер для свойства angle, возвращает текущий угол вращения.
* angle(self, value): Сеттер для свойства angle, устанавливает угол вращения и применяет трансформацию к изображению метки для вращения.

...



База данных:

Таблица 1: spell\_book

Id (индекс)

name (название)

description (описание)

usage (использование)

level (уровень)

complexity (сложность)

damage (урон)

Таблица 2: sword\_book

Id (индекс)

name (название)

description (описание)

usage (использование)

level (уровень)

complexity (сложность)

damage (урон)