МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Объектно-ориентированное программирование»

Тема: Шаблонные классы, управление.

Студент гр. 0381 _____ Дзаппала Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Определить набор правил для игры в виде классов. Также определить класс игры, который должен быть прослойкой между бизнес-логикой и командами управления.

Задание.

Необходимо определить набор правил для игры в виде классов (например, какие задачи необходимо выполнить, чтобы он мог выйти с поля; какое кол-во врагов и вещей должно быть на поле, и.т.д.). Затем определить класс игры, которое параметризуется правилами. Класс игры должен быть прослойком между бизнес-логикой и командами управления, то есть непосредственное изменение состояния игрой должно проходить через этот класс.

Требование:

- Созданы шаблонные классы правил игры. В данном случае параметр шаблона должен определить конкретные значения в правилах.
- Создан шаблонный класс игры, который параметризуется конкретными правилами. Класс игры должен проводить управление врагами, передачей хода, передавать информацию куда переместить игрока, и.т.д.

Потенциальные паттерны проектирования, которые можно использовать:

- *Компоновщик (Composite) - выстраивание иерархии правил*
- *Фасад (Facade) - предоставления единого интерфейса игры для команд управления*
- *Цепочка обязанностей (Chain of Responsibility) - обработка поступающих команд управления*
- *Состояние (State) - отслеживание состояние хода / передача хода от игрока к врагам*

Посредник (Mediator) - организация взаимодействия элементов бизнес-логики

Выполнение работы.

Для начала, был реализован класс со статистикой игры — GameStats. Сделано это для того, чтобы эти данные были не в перемешку в классе Game, хотя и это возможно, но так вся статистика хранится в одном месте. Класс этот содержит поля: указатель на Игрока (Player*), кол-во убитых врагов(size_t) и время начала игры (std::chrono::steady_clock::time_point). Конструктор принимает указатель на Player, что бы мы могли инициализировать переменную игрока, которая следит за текущим состоянием игрока. Методы: GetStartTime() - геттер начала времени, чтобы в будущем можно было считать разницу во времени; startTime() - метод для начала отчета времени; kill() - метод, который инкрементирует переменную убитых врагов; getHowMuchKills() - метод, который возвращает кол-во убитых врагов.

Было создано два класса с условиями игры — TimeEvent и EnemyKillerEvent. Оба класса являются шаблонными, и принимают целочисленную переменную. В каждом из них объявлен предикат, для проверки условия игры. Так, например, предикат TimeEvent принимает ссылку на переменную std::chrono::seconds, которая содержит в себе дельту времен, и если это время больше или равно шаблонной переменной, то игра заканчивается, игрок проиграл, и не успел.

Класс Game стал шаблонным, принимающим в шаблон два «класса» с условиями для игры. В классе появились три новых поля: под статистику (GameStats), и два шаблонных указателя. Также были определены два метода, возвращающие bool значения, для проверки наших условий. Метод checkTimeEvent() проверяет условие со временем (в случае, если время уже прошло (то есть уже больше), игрок проигрывает и окно закрывается), checkEKE() проверяет условие с убийством врагов (если нынешнее кол-во убитых врагов равно условию EnemyKillerEvent, то игрок выиграл, и окно

закрывается). Эти методы вызываются в методе Game::HaveFun(), там же, где и все обновления объектов и «рендер».

Выводы.

Было изучено, как работать с шаблонами, а также реализованы условия для работы игры. Класс игры уже был реализован.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ