**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В. И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Объектно-ориентированное программирование»**

**Тема: Связывание классов**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Мальцев К.Л. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## **Цель работы**

Разработать класс игры с реализацией игрового цикла, включающего смену ходов между пользователем и компьютерным врагом. Реализовать управление состоянием игры, возможность сохранения и загрузки прогресса, а также методы для начала новой игры и выполнения ходов.

## **Задание**

Создать класс игры, который реализует следующий игровой цикл:

Начало игры

Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.

В случае проигрыша пользователь начинает новую игру

В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

Примечание:

Класс игры может знать о игровых сущностях, но не наоборот

Игровые сущности не должны сами порождать объекты состояния

Для управления самое игрой можно использовать обертки над командами

При работе с файлом используйте идиому RAII.

## **Выполнение работы**

Структура Проекта

Проект состоит из нескольких ключевых классов, каждый из которых выполняет свою уникальную роль в игре. Вот основные компоненты структуры:

1. Action: Это интерфейс, определяющий метод execute, который должен реализовать каждый класс действия.

2. Действия (Actions):

- ApplyBombing: Реализует действие применения способности «бомбордировка».

- ApplyDoubleDamage: Реализует действие применения способности «двойной урон».

- ApplyScanner: Реализует действие применения способности «сканер».

- InformAboutReadiness: Реализует действие сообщение о готовности игрока.

- Attack: Реализует действие атаки по координатам.

- Load: Реализует действие загрузки игры по имени файла.

- PlaceShipsRandomly: Реализует действие расстновки кораблей в случайном порядке для игрока.

- Save: Реализует действие сохранения игры.

- SetSettings: Реализует действия выставления настроек – является первым действием, которое может вызвать игрок не считая команды загрузки.

- Stop: Реализует действие остановки игровой сессии (без сохранения)

4. Игровой контроллер:

- GameController: Контролирует процесс игры, взаимодействует с Game, View (CLIListener и CLIDisplayer) и View Model карты.

5. Модель карты:

- GameMapViewModel: Предоставляет доступ к свойствам игровой карты и интерфейсу для обновления информации на ней.

6. Данные игры:

- GameData: Хранит состояние игры, включая информацию об игроках и настройки. В классе переопределны методы ввода и вывода для реализации сохранения игрового процесса.

7. Обработчик файла сохранения:

- GameDataHandler: Отвечает за сохранение и загрузку данных игры.

7. Игра:

- Game: Основной класс, который объединяет все части, управляет ходом игры, отслеживает состояние и взаимодействует с игроком через команды от GameController.

8. CLIListener: Слушает команды от игрока и возвращает действие, основанное на введенной команде.

9. CLIDisplayer: Отображает игру.

Как работает проект

1. Инициализация:

- В GameController конструктора передается объект Game.

- Он инициализирует приватное поле \_game, которое будет использоваться для управления игровой логикой.

2. Запуск игры:

- Метод play() запускает главный цикл игры.

- В начале каждого нового игрового процесса отображает сообщение "Game started".

3. Исполнение игровых процессов:

- Вызывается метод runGame(), который ведет к основному игровому циклу.

- Проверяется, находится ли игра в состоянии "running". Если нет, цикл завершается.

4. Отображение информации:

- Если игра не в фазе kSettingSettings, отображается игровая информация (карты, состояние боевых единиц и т.д.).

5. Получение действий от пользователя:

- Входит в подцикл для получения команды от пользователя через \_listener.getAction().

- Если команда недействительна, выбрасывается исключение ListenerException, и пользователю отображается ошибка.

6. Исполнение команды:

- После получения команды execute() вызывается на объекте action.

- В случае ошибки при выполнении команды (например, не правильные параметры) выводится сообщение об ошибке и выполнение переходит к следующей итерации цикла.

7. Проверка состояния игры:

- Если команда остановки была выполнена, цикл завершается.

- Вызывается метод tryToAddAbilitiesForUser() для добавления способностей игрока.

8. Проверка выигрыша:

- Если игрок выиграл раунд, происходит сброс противника и увеличение счета.

- Выводится сообщение о выигранном раунде.

9. Ход противника:

- Если текущий ход противника, вызывается метод enemyMakeMove(), и отображается информация о текущей игре.

10. Конец игры:

- Проверяется, завершилась ли игра. В случае завершения вызовется метод stop() и выведется сообщение "Game over(".

11. Отображение информации о работе игры:

- Метод displayGameInfo() будет вызван, чтобы показать текущий счет, доступные способности пользователя и карты игры.

Таким образом, GameController управляет игровым процессом, взаимодействует с пользователем и отображает необходимую информацию, обеспечивая при этом обработку ошибок и управление состоянием игры.

В целом, проект построен по паттерну MVC (Model-View-Controller), что позволяет четко разделять логику игры, взаимодействие с пользователем и представление данных.

Разработанный программный код см. в приложении А.

## **Выводы**

Созданный класс игры будет обеспечивать последовательность раундов, поддержку взаимодействия между игроком и противником, а также сохранение состояния.

# **Приложение А Исходный код программы**

https://github.com/drimmovka/Seabattle