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: Связывание классов

Студент гр. 3341 _____ Мальцев К.Л.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы

Разработка шаблонного класса управления игрой и соответствующего класса отображения игры для обеспечения гибкости и модульности в реализации игрового процесса.

Задание

Создать шаблонный класс управления игрой. Данный класс должен содержать ссылку на игру. В качестве параметра шаблона должен указываться класс, который определяет способ ввода команда, и переводящий введенную информацию в команду. Класс управления игрой, должен получать команду для выполнения, и вызывать соответствующий метод класса игры.

Создать шаблонный класс отображения игры. Данный класс реагирует на изменения в игре, и производит отрисовку игры. То, как происходит отрисовка игры определяется классом переданном в качестве параметра шаблона.

Реализовать класс считывающий ввод пользователя из терминала и преобразующий ввод в команду. Соответствие команды введенному символу должно задаваться из файла. Если невозможно считать из файла, то управление задается по умолчанию.

Реализовать класс, отвечающий за отрисовку поля.

Примечание:

Класс отслеживания и класс отрисовки рекомендуется делать отдельными сущностями. Таким образом, класс отслеживания инициализирует отрисовку, и при необходимости можно заменить отрисовку (например, на GUI) без изменения самого отслеживания

После считывания клавиши, считанный символ должен сразу обрабатываться, и далее работа должна проводить с сущностью, которая представляет команду.

Для представления команды можно разработать системы классов или использовать перечисление enum.

Хорошей практикой является создание “прослойки” между считыванием/обработкой команды и классом игры, которая сопоставляет команду и вызываемым методом игры. Существуют альтернативные решения без явной “прослойки”

При считывания управления необходимо делать проверку, что на все команды назначена клавиша, что на одну клавишу не назначено две команды, что на одну команду не назначено две клавиши.

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. Наблюдатель:

- GameObserver: Подписывается на события в игре и инициализирует нужную отрисовку.

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

9.1 CommandComparator: сопоставляет команды пользователя и ключи для CLIListener, чтобы иметь возможность обработать команду одним из handle методов CLIListener-а

10. CLIDisplayer: Отображает игру.

10.1 CLIGameMapDisplayer: Класс отображения игровой карты.

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 игровой наблюдатель зафиксировал действие после, которого игра должна отобразиться, то 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 игрового контроллера будет обеспечивать последовательность раундов, поддержку взаимодействия между игроком и противником, а также сохранение состояния.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

https://github.com/drimmovka/Seabattle