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: Уровни абстракции, управление игроком

Студент гр. 1304 Шаврин А.П.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Изучить уровни абстракции с помощью ООП в языке C++. Применить полученные знания в реализации управления игроком в игре. Научиться выстраивать архитектуру проекта с возможностью дальнейшего расширения без изменения исходного кода.

Задание.

Реализовать набор классов отвечающих за считывание команд пользователя, обрабатывающих их и изменяющих состояния программы (начать новую игру, завершить игру, сохраниться, управление игроком, и.т.д.). Команды/клавиши определяющие управление должны считываться из файла.

Требования:

1. Реализован класс/набор классов обрабатывающие команды
2. Управление задается из файла (определяет какая команда/нажатие клавиши отвечает за управление. Например, w - вверх, s - вниз, и.т.д)
3. Реализованные классы позволяют добавить новый способ ввода команд без изменения существующего кода (например, получать команды из файла или по сети). По умолчанию, управление из терминала или через GUI, другие способы реализовывать не надо, но должна быть такая возможность.
4. Из метода считывающего команду не должно быть “прямого” управления игроком

Примечания:

5. Для реализации управления можно использовать цепочку обязанностей, команду, посредника, декоратор, мост, фасад

Выполнение работы.

1. Сначала было создано перечисление *GameControllCommand* всех возможных команд.
2. Затем был создан класс *GameControllCommandsConverter*, имеющий два приватных поля: *dict_commands* типа *std::map <std::string, sf::Keyboard::Key>* и *dict_commands_name* типа *std::map <std::string, GameControllCommands>*.

Поле *dict_commands* – словарь, где ключ - строковое значение клавиши (‘W’, ‘D’ и т.д.), а значение - соответствующее ключу клавиша типа *sf::Keyboard::Key* (*sf::Keyboard::W, sf::Keyboard::D* и т.д.). Приватный метод *createCommandDictionary* заполняет словарь.

Поле *dict_commands_name* – словарь, где ключ - строковое значение комманды (‘Up’, ‘Right’ и т.д.), а значение – соответствующий элемент перечисления типа GameControllCommands (*MovePlayerUp, MovePlayerRight* и т.д.). Приватный метод *createCommandNameDictionary* заполняет словарь.

Метод c*onvertCommand* получает на вход строку (‘W’ – строковой вид кнопки) и возвращает значение данной кнопки в виде *sf::Keyboard::Key*, используя поиск по *dict_commands*.

Метод *convertCommandName* получает на вход строку (‘Up’ – строковой вид команды) и возвращает значение данной команды в виде *enum GameControllCommands*, используя поиск по *dict_commands_name*.

Методы *isCorrectCommand* и *isCorrectCommandName* получают на вход строковой вид команды/названия команды и возвращают *true*, если данное значение можно сконвертировать к необходимому типу, иначе *false*.

3. Далее был написан класс *GameControllCommandsCreator*, который имеет три приватных поля: *default_commands* типа *std::map <GameControllCommands, sf::Keyboard::Key>* , *file_commands* типа *std::map <GameControllCommands, sf::Keyboard::Key> и converter* типа *GameControllCommandsConverter**.

Метод *createDefaultCommands*, заполняет словарь *default_commands* значениями команд по умолчанию.

Метод *setCommand* получает на вход строку из файла строго определенного типа *(“Up = W”).* По полученной строке метод заполняет словарь *file_commands* командами пользователя, предварительно проверяя все возможные ошибки входной строки и конвертируя данные с помощью конвертера.

Метод *isRepeatCommand* получает на вход *sf::Keyboard::Key* и проверяет, была ли уже установлена данная клавиша на другую команду в уже созданном словаре.

Метод *isFullCommands* получает на вход словарь команд и проверяет, присутствуют ли в нем все необходимые ключи.

Метод *getCommands* возвращает собранный по результатам преобразований словарь команд.

Метод *getDefaultCommands* возвращает словарь команд по умолчанию.

Данный класс унаследован от класса *GameElement<Log *>*, что бы иметь возможность отправлять логи об ошибках через медиатор (прописано в родительском классе).

4. Затем был написан класс *GameControllCommandsReader,* который считывает файл, в котором задаются параметры управления игрой.

Данный класс имеет два приватных поля: *file* типа *std::ifstream* и *commands_creator* типа *GameControllCommandsCreator.* Метод *readCommands* считывает файл построчно и каждую строку передает в *converter.setCommand*, тем самым устанавливая значения команд. Затем данный метод возвращает словарь команд, получая его из *commands_creator*, при помощи метода *getCommands,* если ошибок при заполнении словаря не возникло, в противном же случае вызывается метод *getDefaultCommand*s.

Данный класс унаследован от класса *GameElement<Log *>*, что бы иметь возможность отправлять логи об ошибках через медиатор (прописано в родительском классе).

5. После был написан класс *CommandAdapter,* имеющий одно приватное поле *commands* типа *std::map<GameControllCommands, sf::Keyboard::Key>*.

В конструкторе класса происходит создание обьекта класса *GameControllCommandsReader* и заполнение словаря *commands.*

Метод *adapt* принимает на вход клавишу типа *sf::Keyboard::Key* и возвращает соответствующий элемент перечисления (ключ), удобный для *CommandReaderMediator*.

Данный класс принимает в конструкторе указатель на медиатор посылающий логи и устанавливает его у класса *GameControllCommandReader.*

6. Был модернизован класс *KeyboardCommandReader*. Ему было добавлено приватное поле *adapter* типа *CommandAdapter*. Теперь данный класс посылает через медиатор не саму кнопку, а соответствующую ей команду типа *GameControllCommands*, которую получает от *adapter*.

7. После был модернизован класс *CommandReaderMediator*, осуществляющий запуск соответствующего действия в игре, в зависимости от команды.

Ему было добавлено приватное поле *mediator_commands* типа *std::map <GameControllCommands, std::function<void(CommandReaderMediator*)>>*. Затем был модернизован метод *send*, получающий на вход сообщение типа *GameControllCommands*. Теперь сравнивание полученного сообщения происходит со значениями команд в словарях, а при совпадении в необходимый метод соответствующего контроллера передается значение по этому ключу.

Зависимости классов приведены на UML диаграмме ниже:

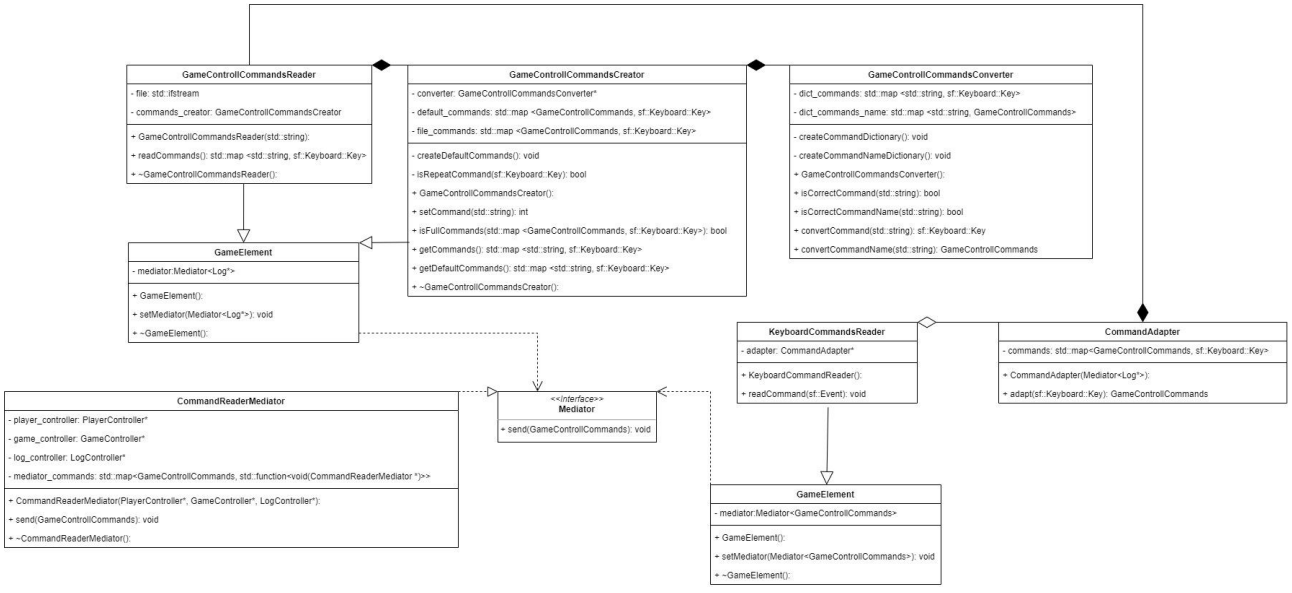

Рисунок 1: UML диаграмма классов.

Выводы.

Изучены уровни абстракции с помощью ООП в языке C++. Полученные знания применены в реализации управления игроком в игре. Научились выстраивать архитектуру проекта с возможностью дальнейшего расширения без изменения исходного кода.