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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Объектно-ориентированное программирование»

Тема: Управление, разделение на уровни абстракции

Студент гр. 0381 Прохоров Б. В.

Преподаватель Жангиров Т. Р.

Санкт-Петербург

2021

Цель работы.

Изучить основы объектно-ориентированного программирования на языке C++.

Задание.

Необходимо организовать управление игрой (номинально через CLI). При управлении игрой с клавиатуры должна считываться нажатая клавиша, после чего происходит перемещение игрок или его взаимодействия с другими элементами поля.

Требования:

- Реализовать управление игрой. Считывание нажатий клавиш не должно происходить в классе игры, а должно происходить в отдельном наборе классов.
- Клавиши управления не должны жестко определяться в коде. Например, это можно определить в отдельном классе.
- Классы управления игрой не должны напрямую взаимодействовать с элементами игры (поле, клетки, элементы на клетках)
- Игру можно запустить и пройти.

Выполнение работы.

Чтобы выполнить задание, нужно было реализовать 6 классов: Controller, *Button, DownButton, LeftButton, RightButton, UpButton* и немного модифицировать классы *Presenter, Model*.

Класс *Controller*, представляющий собой класс управления игрой, осуществляет считывание клавиш и исполнение команд. Класс управления игрой не взаимодействует напрямую с элементами игры.

Класс *Button* является интерфейсом классов кнопок. В нём определён метод запуска.

Было добавлено перечисление *Direction* для того, чтобы клавиши управления не определялись в коде жёстко.

Классы кнопок: *DownButton, LeftButton, RightButton, UpButton*, являются наследниками класса *Button*, в каждом переопределён метод запуска. Каждая кнопка имеет поле для хранения указателя на объект класса *Model* для того, чтобы у неё была возможность обращаться к бизнес-логике. Значение поля кнопка получает через конструктор.

Классу *Model*, представляющему собой класс бизнес-логики, необходимо было добавить 4 метода: *down(), left(), right(), up()* для того, чтобы кнопки не делали всю работу самостоятельно, а лишь перенаправляли запрос бизнес-логике.

В класс *Presenter*, представляющий собой шаблонный класс игры, было внедрено управление через класс *Controller* (было через *switch*).

Реализован паттерн Singleton – см. класс Controller и приложение А. Он нужен для гарантии того, что у нас будет существовать только один контроллер.

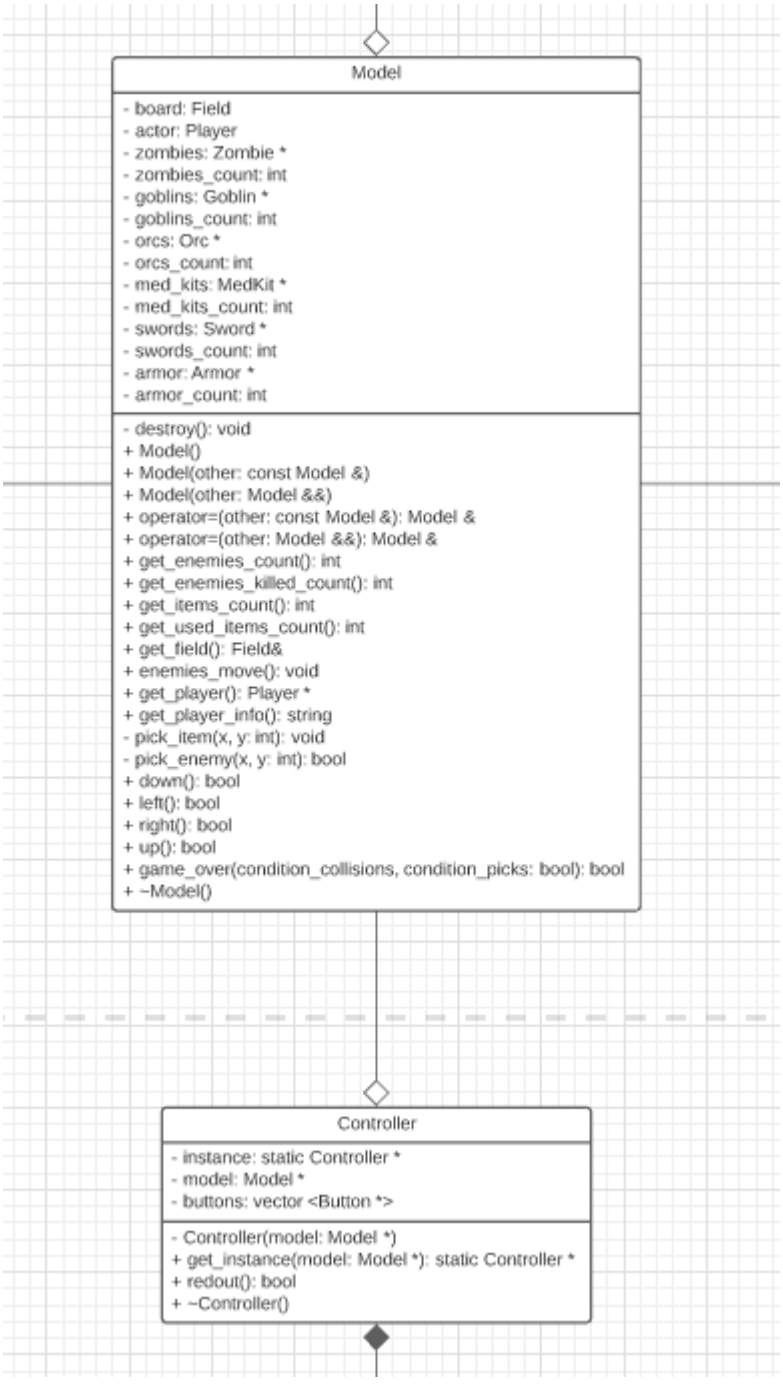
Реализован паттерн Command – см. класс Controller и приложение А. Он позволяет нам заворачивать запросы или простые операции в отдельные объекты.

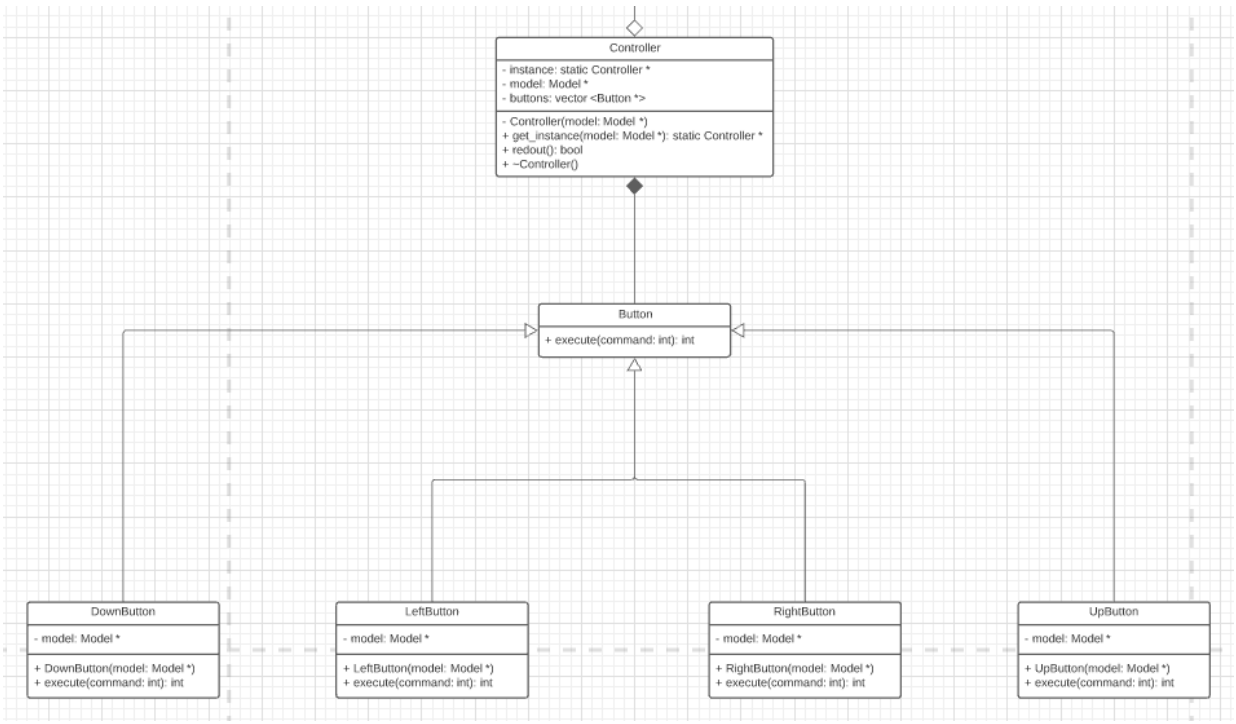
Выводы.

Был получен навык разделения приложения на несколько уровней абстракции, что способствует обеспечению поддерживаемости и расширяемости. Были изучены паттерны Singleton и Command и то, как их грамотно использовать на практике.

ПРИЛОЖЕНИЕ А

UML ДИАГРАММА





Ссылка: http://surl.li/audlr