**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: **Логическое разделение классов**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8382 |  | Мирончик П.Д. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2020

**Задание**

Разработать и реализовать набора классов для взаимодействия пользователя с юнитами и базой. Основные требования:

* Должен быть реализован функционал управления юнитами
* Должен быть реализован функционал управления базой

|  |  |
| --- | --- |
| Выполнены все основные требования к взаимодействию | 6 баллов |
| Добавлен функционал просмотра состояния базы | 3 балла |
| Имеется 3+ демонстрационных примера | 1 балл |
| *\*Реализован паттерн “Фасад” через который пользователь управляет программой* | *1 балл* |
| *\*Объекты между собой взаимодействуют через паттерн “Посредника”* | *3 балла* |
| *\*Для передачи команд используется паттерн “Команда”* | *3 балла* |
| *\*Для приема команд от пользователя используется паттерн “Цепочка обязанностей”* | *3 балла* |
| **Кол-во баллов за основные требования** | **10 баллов** |
| **Максимальное кол-во баллов за лаб. работу** | **20 баллов** |

# Ход работы

**Описание основных классов.**

**Описание из лабораторной работы №2:**

*GameBoard* – корень приложения. Хранит информацию о клетках доски (*Cell*), привязанных к доске объектах (*GameObject*), подписчиках на изменения поля (*BoardListener*). К экземпляру *GameBoard* привязывается *GameController* и *MouseTracker*. GameBoard отвечает за рассылку уведомлений об изменении игрового поля (перемещение/добавление/удаление юнитов), передачу действий пользователя (мышь и клавиатура) игровым объектам, обработку корректного удаления/добавления объектов, отрисовку поля и вызов функций отрисовки у подписанных объектов. Добавление и удаление объектов возможно только через *GameController*.

*Cell* – элемент сетки игры, клетка. Содержит информацию о ландшафте в клетке, положении клетки а также объектах, находящихся в данной клетке.

*GameController* – мост между доской и объектами. Содержит методы для создания объектов поля (юнитов и нейтральных объектов), добавления и удаления элементов с поля (вызывая затем соответствующие методы в *GameBoard*, если вызов корректен: например, при добавлении элемента необходимо убедиться, что в целевой клетке отсутствует объект). При необходимости взаимодействия объектов поля между собой (например нанесение урона) действие также проходит через *GameController*.

*MouseTracker* – как следует из названия, класс предназначен для отслеживания действий пользователя при помощи мыши. На текущий момент единственным классом, использующим *MouseTracker,* является *GameBoard*. Данный класс позволяет отслеживать перемещения мыши в удобном формате, отслеживая смещения мыши относительно последней позиции и нажатия левой клавишей мыши.

*GameObject* – базовый класс для всех объектов поля. Отвечает за хранение своего состояния (привязан ли к доске) и позиции ячейки, в которой он находится в данный момент. *GameObject* предоставляет ряд полезных интерфейсов (*BoardListener*, слушатели состояния привязки) и обязательных к реализации абстрактных методов (отрисовка, обработка нажатий клавиатуры и мыши).

*Unit* – базовый класс для юнитов: объектов, которыми может манипулировать пользователь. Обладает такими характеристиками, как: здоровье, скорость, атака. Может перемещаться по полю.

*Neutral* – базовый класс для нейтральных юнитов. Пользователь не может влиять на нейтральные юниты. Каждый *Neutral* обладает радиусом действия. Если *Unit* попадает в зону действия, на него накладывается определенный эффект, который наследуется от *NeutralEffect*.

*Terrain* – класс ландшафта. Каждой клетке поля (*Cell*) устанавливается определенный тип ландшафта. Terrain обладает следующими возможностями: отрисовка, возможность накладывать эффекты на объекты типа *Unit*.

*Effect* – эффект, который накладывается на объекты типа *Unit*. Имеет возможность изменять любые свойства объекта. По сути эффекты – основной способ взаимодействия с юнитами.

*TerrainEffect* – класс, являющийся наследником *Effect*. По большей части это вспомогательный класс для других эффектов ландшафта. Он отслеживает положение *Unit*-a, к которому привязан, и, если нет нейтральных объектов подходящего типа, в радиус действия которых попадает целевой юнит, то эффект снимается.

**Классы, дополнительно затронутые в лабораторной работе №3:**

*BoardView* – помощник для класса *GameBoard*. Содержит объект *GridDrawer* (см. описание далее), *Viewport* (см. описание далее) и вспомогательные функции для расчета размеров и позиции точек в зависимости от текущего состояния *Viewport*-а.

*CellDrawer* – достаточно часто использующийся вспомогательный класс. Предоставляет простой интерфейс для рисования ячеек на поле в соответствии с текущим состоянием доски: размер ячейки и ее позиция основывается на состоянии Viewport-а, который необходимо передать вместе с позицией ячейки для отрисовки. У *CellDrawer* имеется 2 подкласса: *ColorCellDrawer* (заполняет клетку цветом) и *TextureCellDrawer* (заполняет клетку заданной текстурой).

*GridDrawer* – вспомогательный класс, который рисует координатную сетку поля, основываясь на текущем состоянии *Viewport*.

*HealthDrawer* – класс, позволяющий рисовать здоровье юнитов. Привязывается к конкретному юниту и рисует полоску здоровья, основываясь на показателях здоровья юнита, его положения а также состояния *Viewport*-а.

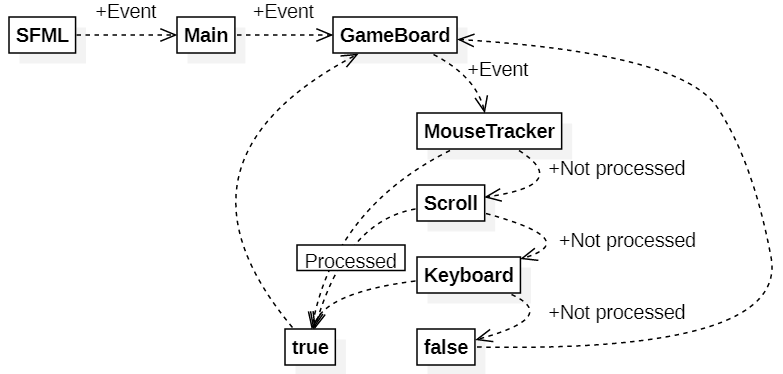
*ShapeDrawer* – класс, позволяющий рисовать на координатной сетке фигуры, заданные клетками. На вход передается список координат клеток, которые включены в фигуру, также имеется возможность задать цвет заливки и цвет границы. Используется для отображения дистанции эффектов нейтральных объектов.

*Viewport* – хранит информацию о текущем состоянии видимой области: ее размер и координаты верхнего левого угла. Предоставляет функции для перевода реальных координат (реальной позиции курсора относительно верхнего левого угла экрана) в игровые координаты. Игровые координаты – это высота/ширина с соотношением, совпадающим с соотношением реального окна, однако подобранные таким образом, чтобы игровое поле вписывалось в них с некоторым отступом (для более удобного управления).

*Textures* – синглтон, хранящий экземпляры всех текстур, используемых в программе.

# Обработка событий

События приходят от фреймворка SFML в виде объектов Event. В функции main перед отрисовкой программы все доступные события (т.е. все события произошедшие между предыдущим и текущим фреймами) считываются и передаются на обработку в класс GameBoard.



GameBoard, используя паттерн цепочка обязанностей, обрабатывает полученное событие:

* Проверка на событие мыши – нажатие клавиши или перемещение. Происходит внутри класса MouseTracker.
* Проверка на прокрутку колеса мыши – изменяется масштаб Viewport-а.
* Проверка на нажатие клавиатуры. Если была нажата клавиша, GameBoard итерирует по всем привязанным объектам, повторяя ту же цепочку обязанностей для них. Если очередной объект возвращает true, обработка заканчивается и GameBoard также возвращает положительный результат (true). При этом приоритет имеют объекты, имеющие фокус.

Если событие не было обработано, возвращается false.

Отдельно стоит указать, как происходит обработка события после его попадания в MouseTracker. Есть 3 типа событий, которые могут произойти:

* Нажатие кнопки мыши – контроллер состояния кнопки переводится в режим PRESSED и запоминаются координаты нажатия. MouseTracker предоставляет методы для получения текущего состояния кнопок мыши.
* Поднятие кнопки мыши – контроллер переводится в состояние IDLE, у подписчика (в данном случае экземпляра GameBoard) вызывается метод onClickLeft.
* Перемещение мыши – MouseTracker запоминает новую позицию мыши и вызывает у подписчика метод onMove, передавая туда смещение относительно последней позиции мыши.

Дальнейшая обработка происходит при помощи GameBoard:

При перемещении мыши возможно 2 варианта:

1. Левая кнопка мыши зажата – производится перемещение Viewport-а (изменение его верхнего левого угла относительно игровых координат)

2. Левая кнопка не зажата – действие игнорируется.

Правая кнопка мыши на текущий момент не обрабатывается.

Если пользователь отпустил кнопку мыши (левую), возможно 2 варианта:

1. Пользователь переместил курсор между событиями onDown и onUp (нажатием и поднятием кнопки мыши) – Viewport был перемещен и событие в дальнейшем не обрабатывается.

2. Пользователь не перемещал курсор – это был целенаправленный клик по точке экрана. В этом случае вычисляется клетка, по которой пользователь кликнул, и по тому же паттерну Цепочка обязанностей событие обрабатывается списком объектов. Приоритет имеют объекты, содержащие фокус.

# Особенности лабораторной работы

**Должен быть реализован функционал управления юнитами.**

Каждый юнит может перемещаться в пределах поля, при этом на каждой клетке поля может находиться не более одного материального объекта (это, например, юниты и нейтральные объекты). Для этого используется обработка нажатий мыши (можно запустит exe файл и потыкать).

**Должен быть реализован функционал управления базой.**

База является тем же юнитом, однако обладает некоторыми особенностями – она не может перемещаться (обладает скоростью, равной нулю) и атаковать. Если база имеет фокус (пользователь нажал на нее мышью), пользователь может создавать юнитов. Для этого он должен выбрать юнита (цифрами от 1 до 6), навести мышь на клетку поля, куда необходимо установить юнита, и нажать левой кнопкой мыши.

**Выполнены все основные требования к взаимодействию.**

см. выше

**Добавлен функционал просмотра состояния базы.**

Как юнит, база имеет запас здоровья (повышенный по сравнению с обычными юнитами). Полоска здоровья отображается поверх изображения базы – это можно считать состоянием, которое показано всегда.

**Имеется 3+ демонстрационных примера.**

Как и раньше, можно создать множество примеров, просто перемещая юнитов.

**Реализован паттерн “Фасад” через который пользователь управляет программой.**

Класс GameBoard является тем самым фасадом, делегирую практически весь процесс обработки другим классам.

**Объекты между собой взаимодействуют через паттерн “Посредника”.**

Плохо представляю, как точно вписать этот паттерн в программу, однако можно привести в пример и другие реализации. Например, MouseTracker является прекрасным примером – сам по себе он не реализует никакой логики, просто сообщая GameBoard-у о произошедших событиях.

Отчасти объекты поля также реализуют данный паттерн. Каждый объект имеет свою позицию, и при ее изменении отправляет уведомление в GameBoard, который в свою очередь производит перемещение объекта из одной ячейки (Cell) в другую.

**Для передачи команд используется паттерн “Команда”.**

В проекте имеется ряд абстрактных классов-команд – CellClickBehaviour, UnitAttackBehaviour и UnitMoveBehaviour.

Каждый Unit имеет реализацию данных классов и производит обработку нажатий мыши с их помощью. Весь процесс обработки нажатия выглядит следующим образом:

Unit передает управление экземпляру CellClickBehaviour. Все.

Реализованы 3 базовых класса поведения:

BaseUnitAttackBehaviour – определяет возможности атаки юнита. Имеет методы:

bool attack(sf::Vector2i &cell) - ищет юнита в переданной клетке и вызывает attack(Unit), если юнит был найден

bool attack(Unit& other) - проверяет возможность атаковать юнита (позволяет ли дальность атаки) и атакует, если есть возможность.

BaseUnitMoveBehaviour - определяет возможности перемещения юнита. Переопределяемые функции можно посмотреть в сооответствующем классе.

BaseUnitClickBehaviour – определяет процесс обработки нажатия. Если объект не имеет фокуса, то перехватывает нажатие лишь в случае, когда нажатие произведено на клетку с целевым юнитом (к которому привязано поведение). Если фокус установлен на юните, то по цепочке обязанностей вызывает BaseUnitAttackBehaviour и BaseUnitMoveBehaviour, после чего убирает фокус с юнита.

**Для приема команд от пользователя используется паттерн “Цепочка обязанностей”.**

Многократно описывался выше.

# Пути к классам

**BaseUnitAttackBehaviour -** \include\GAME\engine\behaviour\BaseUnitAttackBehaviour.hpp

**BaseUnitClickBehaviour -** \include\GAME\engine\behaviour\BaseUnitClickBehaviour.hpp

**BaseUnitMoveBehaviour -** \include\GAME\engine\behaviour\BaseUnitMoveBehaviour.hpp

**BlackHole -** \include\GAME\engine\units\BlackHole.hpp

**BlackHoleEffect -** \include\GAME\engine\units\BlackHole.hpp

**BoardListener** - \include\GAME\engine\BoardListener.hpp

**BoardView** - \include\GAME\engine\graphics\BoardView.hpp

**Cell** - \include\GAME\engine\Cell.hpp

**CellClickBehaviour** - \include\GAME\engine\behaviour\CellClickBehaviour.hpp

**CellDrawer** - \include\GAME\engine\graphics\CellDrawer.hpp

**Chancel** - \include\GAME\engine\units\Chancel.hpp

**ChancelEffect** - \include\GAME\engine\units\Chancel.hpp

**Effect** - \include\GAME\engine\Effect.hpp

**EffectsComparator** - \include\GAME\engine\Effect.hpp

**EffectsSet** - \include\GAME\engine\Effect.hpp

**GameBoard** - \include\GAME\engine\GameBoard.hpp

**GameController** - \include\GAME\engine\GameController.hpp

**GameObject** - \include\GAME\engine\GameObject.hpp

**GridDrawer** - \include\GAME\engine\graphics\GridDrawer.hpp

**GroundTerrain** - \include\GAME\engine\terrains\GroundTerrain.hpp

**Heal** - \include\GAME\engine\units\Heal.hpp

**HealthDrawer** - \include\GAME\engine\graphics\HealthDrawer.hpp

**Home** - \include\GAME\engine\units\Home.hpp

**LavaTerrain** - \include\GAME\engine\terrains\LavaTerrain.hpp

**MouseTracker** - \include\GAME\engine\MouseTracker.hpp

**Neutral** - \include\GAME\engine\Neutral.hpp

**NeutralEffect -** \include\GAME\engine\NeutralEffect.hpp

**SeaTerrain** - \include\GAME\engine\terrains\SeaTerrain.hpp

**ShapeDrawer** - \include\GAME\engine\graphics\ShapeDrawer.hpp

**Stone** - \include\GAME\engine\units\Stone.hpp

**Terrain** - \include\GAME\engine\Terrain.hpp

**Unit** - \include\GAME\engine\Unit.hpp

**UnitAttachBehaviour** - \include\GAME\engine\behaviour\UnitAttachBehaviour.hpp

**UnitMoveBehaviour** - \include\GAME\engine\behaviour\UnitMoveBehaviour.hpp

**Viewport** - \include\GAME\engine\graphics\Viewport.hpp

# Запуск приложения

Проект собирается при помощи VisualStudio2017 и, насколько я знаю, не требует дополнительных разрешений/установки библиотек. Для запуска можно использовать дебажную сборку, находящуюся в ${ProjectRoot}/Debug/SimpleGame.exe. Программа использует дополнительные библиотеки (SFML), однако они находятся внутри проекта, так что приложение должно запуститься корректно.

# Вывод

При выполнении данной лабораторной работы были изучены паттерны проектирования (Фасад, Посредник, Команда, Цепочка обязанностей), особенности и различные стандартные механизмы языка c++ (например, умные указатели), особенности реализации взаимодействия с событиями (клавиатура/мышь) а также значительно расширена кодовая база приложения в сравнении с первой лабораторной работой.

Написание этой части проекта помогло обнаружить и попытаться понять множество нюансов языка, выявить для себя основные плюсы и, в основном, минусы проектов на c++.