**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Объектно ориентированное программирование»**

Тема: **Интерфейсы классов; взаимодействие классов; перегрузка операций**

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

Санкт-Петербург

2020

**Задание**

Разработать и реализовать набор классов:

* Класс базы
* Набор классов ландшафта карты
* Набор классов нейтральных объектов поля

Класс базы должен отвечать за создание юнитов, а также учитывать юнитов, относящихся к текущей базе. Основные требования к классу база:

* База должна размещаться на поле
* Методы для создания юнитов
* Учет юнитов, и реакция на их уничтожение и создание
* База должна обладать характеристиками такими, как здоровье, максимальное количество юнитов, которые могут быть одновременно созданы на базе, и.т.д.

Набор классов ландшафта определяют вид поля. Основные требования к классам ландшафта:

Должно быть создано минимум 3 типа ландшафта

* Все классы ландшафта должны иметь как минимум один интерфейс
* Ландшафт должен влиять на юнитов (например, возможно пройти по клетке с определенным ландшафтом или запрет для атаки определенного типа юнитов)
* На каждой клетке поля должен быть определенный тип ландшафта

Набор классов нейтральных объектов представляют объекты, располагаемые на поле и с которыми могут взаимодействие юнитов. Основные требования к классам нейтральных объектов поля:

* Создано не менее 4 типов нейтральных объектов
* Взаимодействие юнитов с нейтральными объектами, должно быть реализовано в виде перегрузки операций
* Классы нейтральных объектов должны иметь как минимум один общий интерфейс

|  |  |
| --- | --- |
| Выполнены основные требования к классу база | 2 балла |
| Выполнены основные требования к набору классов ландшафта | 2 балла |
| Выполнены основные требования к набору классов нейтр. объектов | 2 балла |
| Добавлено взаимодействие юнитов | 1 балла |
| Имеется 3+ демонстрационных примера | 1 балл |
| Взаимодействие через перегрузку операторов | 2 балла |
| *\*Для хранения информации о юнитах в классе базы используется паттерн “Компоновщик”/ Использование “Легковеса” для хранения общих характеристик юнитов* | *2 балла* |
| *\*Для наблюдения над юнитами в классе база используется паттерн “Наблюдатель”* | *2 балла* |
| *\*Для взаимодействия ландшафта с юнитам используется паттерн “Прокси”* | *3 балла* |
| *\*Для взаимодействия одного типа нейтрального объекта с разными типами юнитов используется паттерн “Стратегия”* | *3 балла* |
| **Кол-во баллов за основные требования** | **10 баллов** |
| **Максимальное кол-во баллов за лаб. работу** | **20 баллов** |

# Ход работы

**Описание основных классов**

*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.

# Особенности лабораторной работы

**Основные требования**

Класс базы: *Home*. Наследуется от класса *Unit*, что позволяет иметь такую характеристику как здоровье, размещать базу на поле а также взаимодействовать с юнитами. База может создавать любые юниты: для этого надо выбрать объект базы, выбрать юнит, который нужно установить (нажав цифру от 1 до 6) и левым кликом установить юнит в свободную ячейку.



Максимальное количество юнитов, которое может быть создано базой, устанавливается константой *Home::MAX\_UNITS\_COUT*.

Создано три вида ландшафта: земля, море и лава.

Земля (*GroundTerrain*) – базовый тип ландшафта. Юнит, находящийся на земле, сохраняет свои базовые характеристики.

Море (*SeaTerrain*) – запрещает юнитам атаковать (фактически устанавливая дальность атаки равную 0) и уменьшает скорость вдвое.

Лава (*LavaTerrain*) – наносит юнитам урон, который экспоненциально увеличивается с количеством пройденных по лаве клеток.

Нейтральные объекты представлены четырьмя различными классами:

Камень (*Stone*) – запрещает юнитам проходить по клетке, где находится.



Аптека (*Heal*) – лечит юнитов, находящихся в радиусе действия, в конце каждого хода.



Тотем (*Chancel*) – удваивает скорость и увеличивает в полтора раза дальность атаки юнитов, находящихся в радиусе действия тотема.



Черная дыра (*BlackHole*) – уничтожает юнитов, находящихся в радиусе действия, в конце каждого хода



PS: Поскольку нейтральные объекты накладывают эффекты (*ChancelEffect*, *BlackHoleEffect*, *BlackHoleEffect*), взаимодействие с юнитами можно считать реализованным при помощи переопределения методов. Пункт из задания требует реализации при помощи перегрузки операций, однако неясно, какое именно взаимодействие предполагается.

Поскольку класс *Unit* использует *EffectsSet*, для которого используется *EffectsComparator*, переопределяющий оператор *operator(const T&, const T&)*, а эффекты добавляются в *EffectsSet*, при взаимодействии юнитов и нейтральных объектов косвенно используется данный оператор, что формально можно считать выполнением требования задания.

**Выполнение лабораторной работа по пунктам из таблицы:**

* Выполнены основные требования к классу база – см. выше
* Выполнены основные требования к набору классов ландшафта – см. выше
* Выполнены основные требования к набору классов нейтр. объектов – см. выше
* Добавлено взаимодействие юнитов – юниты не могут пересекаться друг с другом: на одной клетке может находиться только один юнит.
* Имеется 3+ демонстрационных примера – имеется графический интерфейс, с помощью которого можно создать неограниченное количество примеров (класс базы позволяет создавать юнитов, юниты могут перемещаться)
* Взаимодействие через перегрузку операторов – фактически все взаимодействие тем или иным образом реализовано через перегрузку операторов, начиная от обработки пользовательских действий и заканчивая переопределением методов класса *Effect* для изменения параметров юнитов.
* Для хранения информации о юнитах в классе базы используется паттерн “Компоновщик”/ Использование “Легковеса” для хранения общих характеристик юнитов – используется паттерн Легковес. Все текстуры, используемые в игре, хранятся в классе Textures, который является синглтоном и, соответственно, существуют в единственном экземпляре на протяжении всей работы программы.
* Для наблюдения над юнитами в классе база используется паттерн “Наблюдатель” – юниты сами по себе являются лишь контейнерами и способны менять определенные параметры (например, свое положение на карте). При изменении положения вызывается метод *BoardListener::onObjectMoved* у всех подписчиков класса *GameBoard*, что соответствует паттерну.
* Для взаимодействия ландшафта с юнитам используется паттерн “Прокси” – ландшафт накладывает на юнитов эффекты, поэтому такие эффекты можно считать частью взаимодействия ландшафта и юнитов (особенно учитывая то, что все эффекты ландшафта наследуются от одного класса *TerrainEffect*). Можно было бы реализовать прокси класс, который просто прокидывал бы определенные методы классу юнита, однако я не нашел подходящему применению такого паттерна в общем виде. Тем не менее эффект *LavaEffect* косвенно использует паттерн прокси – он не напрямую наносит урон юниту, а делает это через *GameController*. Более подходящим вариантом реализации паттерна может стать эффект черной дыры, который вызывает метод удаления юнита у *GameController*-а, в своб очередь запрашивающего удаления у *GameBoard*. В какой-то мере каждый эффект является прокси, поскольку изменение параметров объекта – это последовательный прогон параметра через все эффекты, примененные к юниту.
* Для взаимодействия одного типа нейтрального объекта с разными типами юнитов используется паттерн “Стратегия” – эффекты идеально вписываются в данный паттерн. Когда *Unit* попадает в область действия *Neutral*-а, к *Unit*-у добавляется эффект. Фактически это и является стратегией, поскольку вместо непосредственного изменения параметров юнита действия делегируются другому объекту.

# Пути к классам

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

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

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

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

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

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

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

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

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

# Запуск приложения

Проект собирается при помощи VisualStudio2017 и, насколько я помню, 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, скорее всего, приложение запустится корректно.

# Uml диаграмма

Файл Uml диаграммы называется uml.png.

# Вывод

При выполнении данной лабораторной работы были изучены паттерны проектирования (Компоновщик, Легковес, Наблюдатель, Прокси, Стратегия), особенности и различные стандартные механизмы языка c++ (например, умные указатели) 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++.