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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «ООП»

Тема: Создание классов, конструкторов и методов классов

Студент гр. 0382 _____ Азаров М.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы.

Научится проектировать программы в ООП стиле, для того чтобы они были более гибкими , модифицируемыми и имели возможность повторного использования.

Задание.

Игровое поле представляет из себя прямоугольную плоскость разбитую на клетки. На поле на клетках в дальнейшем будут располагаться игрок, враги, элементы взаимодействия. Клетка может быть проходимой или непроходимой, в случае непроходимой клетки, на ней ничего не может располагаться. На поле должны быть две особые клетки: вход и выход. В дальнейшем игрок будет появляться на клетке входа, а затем выполнив определенный набор задач дойти до выхода.

При реализации класса поля запрещено использовать контейнеры из stl

Требования:

- Реализовать класс поля, который хранит набор клеток в виде двумерного массива.
- Реализовать класс клетки, которая хранит информацию о ее состоянии, а также того, что на ней находится.
- Создать интерфейс элемента клетки.
- Обеспечить появление клеток входа и выхода на поле. Данные клетки не должны быть появляться рядом.
- Для класса поля реализовать конструкторы копирования и перемещения, а также соответствующие операторы.
- Гарантировать отсутствие утечки памяти.

Потенциальные паттерны проектирования, которые можно использовать:

- Итератор (Iterator) - обход поля по клеткам и получение косвенного доступа к ним
- Строитель (Builder) - предварительное конструирование поля с необходимым параметрами. Например, предварительно задать кол-во непроходимых клеток и алгоритм их расположения

Выполнение работы.

Для реализации программы и выполнения программы были созданы следующие классы :

Класс ***ICell***:

Является интерфейсом для любой ячейки поля. Любая ячейка поля должна наследоваться от интерфейса либо от потомка интерфейса. Все методы чистые виртуальные функции.

Методы класса:

- ***putItem()*** - помещает *предмет* (объект с интерфейсом ***IItem)*** на выбранную клетку.
- ***topItem()*** - возвращает *предмет* без удаления из клетки.
- ***popItem()*** - возвращает *предмет* с удалением из клетки.

- ***standOnCreature()*** - помещает *существо* (объект с интерфейсом ***ICreature)*** на выбранную клетку.
- ***getCreature()*** - возвращает *существо* без удаления из клетки.
- ***leaveCreature() -*** возвращает *существо* с удалением из клетки.

- ***getTypeCell() -*** возвращает тип клетки в виде ***enam TypeCell*** , используется в классе ***ViewField*** для определения что за клетка находится под интерфейсом.
- ***clone() -*** возвращает копию данного объекта, метод необходим для глубокого копирования объекта ***Field***.
- ***~ICell()*** - виртуальный деструктор, нужен для того чтобы все наследуемые объекты удалялись коректно.

Класс ***NormalCell***:

Класс обычной клетки наследуемой от интерфейса ***ICell*** , которая может содержать *предмет* или *существо* , быть либо проходимой , либо нет .

Методы класса:

- ***NormalCell()*** - конструктор класса , инициализирует параметр проходимости ***m_passable***.
- Реализация виртуальных методов интерфейса ***ICell.***
- ***_universal_clone() -*** защищенный метод шаблон , помогает копировать текущий объект в методе ***clone()*** и может использоваться для реализации ***clone()*** в наследниках. Копирует клетку с учетом проходимости и содержащихся *предметов* и *существ* .

Класс ***StartCell***:

Класс клетки старта. Наследуется от класса ***NormalCell***. Пока не сильно отчается от ***NormalCell,*** так как не реализован до конца.

Методы класса:

- ***StartCell()*** - конструктор класса , использует делегирующий конструктор класса ***NormalCell***.
- Реализует методы ***clone(), getTypeCell()*** . Реализация остальных методов интерфейса ***ICell*** наследуется от ***NormalCell***.

Класс ***EndCell***:

Аналогично ***StartCell.***

Класс ***Field:***

Касс поля . По сути в архитектуре программы является *моделью*(MVC). Состоит из клеток подчиняющихся интерфейсу ***ICell***.

Также объявляется что класс ***BuilderField*** является дружественным классом(для того чтобы ***BuilderField*** мог построить объект).

Методы класса:

- ***Field()*** - конструктор класса , инициализирует значения ширины и высоты поля, создает двухмерный массив указателей на объекты с интерфейсом ***ICell***, по сути именно этот массив является полем. Является приватным , для того чтобы исключить возможность создавать объект напрямую, так как этим занимается класс ***BuilderField***.
- ***_init_arr_cells()*** - защищенный метод , создает двухмерный массив указателей на объекты с интерфейсом ***ICell***, используется в конструкторе класса и операторе копирования.

- Реализация конструктора и оператора копирования с глубоким копированием .
- Реализация конструктора и оператора перемещения с передачей ресурсов .
- ***getHeight()*** - геттер высоты *рабочего поля*(без рамки).
- ***getWidth()*** - геттер ширины *рабочего поля*(без рамки).
- ***getCell()*** - константный геттер клетки по определенным координатам.

Класс ***BuilderField:***

Реализация паттерна Строитель. Класс отвечает за создание объекта типа ***Field***.

Методы класса:

- ***createEmptyField()*** - создает *пустое поле* (двухмерный массив указателей на ***ICell***, сами указатели указывают на ***NULL***) , и рамку по периметру из непроходимых клеток. Ширина и высота поля задаются в параметрах функции, причем ширина и высота *активной области* (т.е. без учета рамки).
- ***buildStartCell()*** - создает начальную клетку по преданным координатам(если по этим координатам ничего нет), если координаты не заданы поставит в случайное свободное место. Также проверяется чтобы рядом не находилась клетка типа ***EndCell*** (в противном случае выбрасывается исключение).
- ***buildEndCell()*** - аналогично как и для ***buildStartCell()*** , только проверяется что нет рядом клеток типа ***StartCell.***
-

- ***_checkNear()*** - приватный метод , который проверяет есть ли рядом возле переданных координат переданный тип клетки. Используется в ***buildStartCell()*** и ***buildEndCell()*** .
- ***_checkXY()*** - приватный метод , который проверяет есть ли по переданным координатам переданный тип клетки. Используется в ***_checkNear()***.
- ***buildImpassableCell()*** - создает на поле непроходимую клетку по переданным координатам. Также проверяется чтоб по этим координатам ничего не было (иначе выбрасывается исключение).
- ***buildRandomImpassableCells()*** - строит относительное количество непроходимых клеток (в параметрах задается процентное отношение непроходимых клеток от общего количества рабочих клеток) и расставляет их на случайные свободные места.
- ***getField()*** - заполняет все пустые места в поле обычными проходимыми ***NormalCell*** и возвращает созданное поле .

Класс ***IItem***:

Интерфейс *предметов* которые можно будет положить на клетку. Будет реализовывается в дальнейших лаб. работах

Методы:

- ***clone()*** - возвращает копию данного объекта, метод используется в копирования клеток.

Класс ***ICreature***:

Интерфейс *существ* которые можно будет поставить на клетку.

Будет реализовывается в дальнейших лаб. работах.

Методы:

- ***clone()*** - возвращает копию данного объекта, метод используется в копирования клеток.

Класс ***ViewField***:

Класс отображения модели (поля), пока реализован примитивно для более удобного тестирования модели.

Методы:

- ***rendering()*** - обрисовывает модель.

Файлы не относящиеся к файлам классов:

- ***My_Exception.h*** - файл содержит объявление структуры ***enam My_Exception***, которая используется для созданий исключений . В дальнейшем возможно будет реализована работа исключений через наследование от класса exception.
- ***unq_p.h*** - простое переименование умных указателей ***std::unique_ptr*** и массивов(одномерных и двухмерных) указателей, для более комфортного написания кода.

UML диаграмма классов:

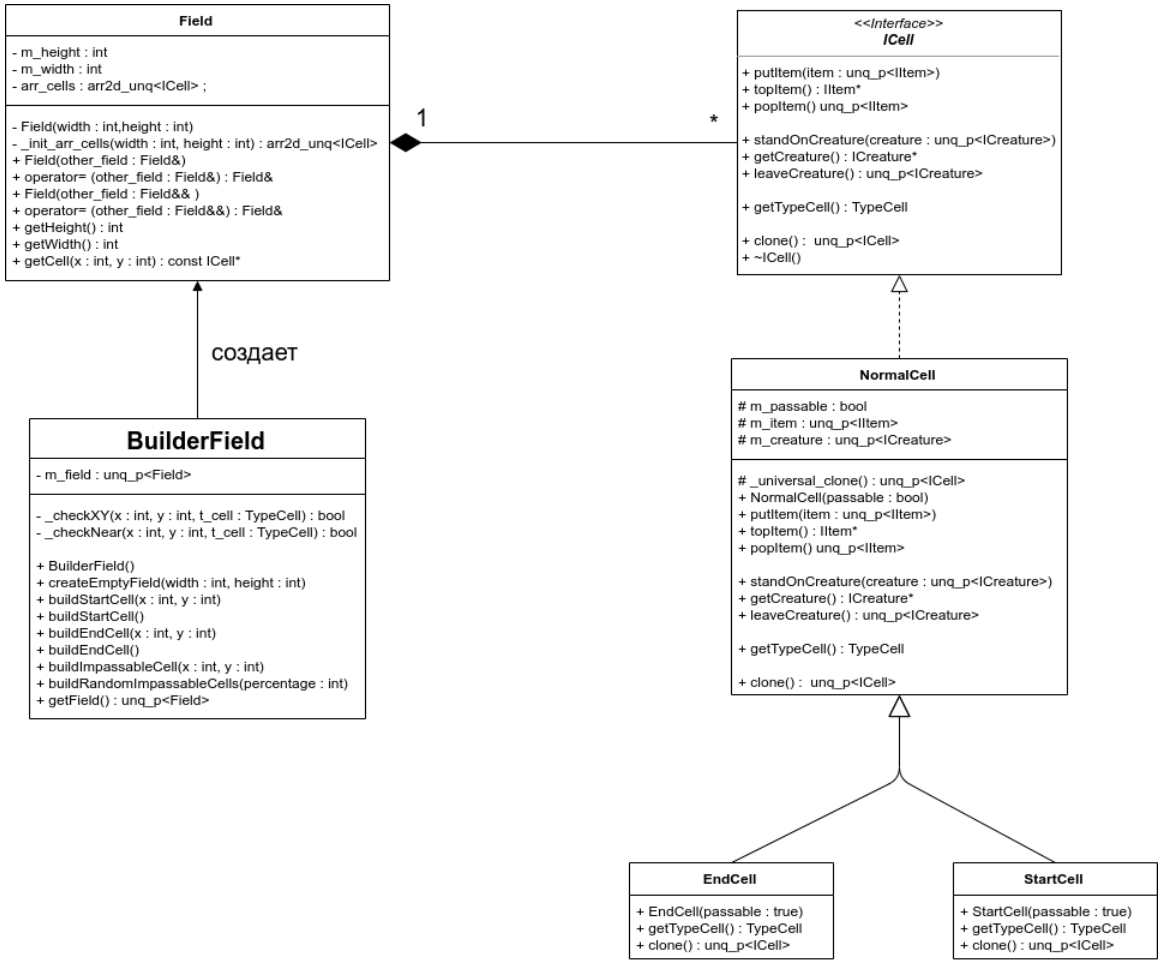

Рисунок 1: UML диаграмма классов

Тестирование.

main.ccp(тело функции main):

```
BuilderField builder;
unq_p<Field> field1,f2;

//строительство
builder.createEmptyField(20,10);
builder.buildRandomImpassableCells(20);
builder.buildStartCell();
builder.buildEndCell();
```

```
builder.buildEndCell();
field1 = builder.getField();

//отображение
ViewField  view(field1.get());
view.rendering();
```

Результат:

Рисунок 2: Результат тестирования

Программа работает правильно.

Также утечек памяти не обнаружено:

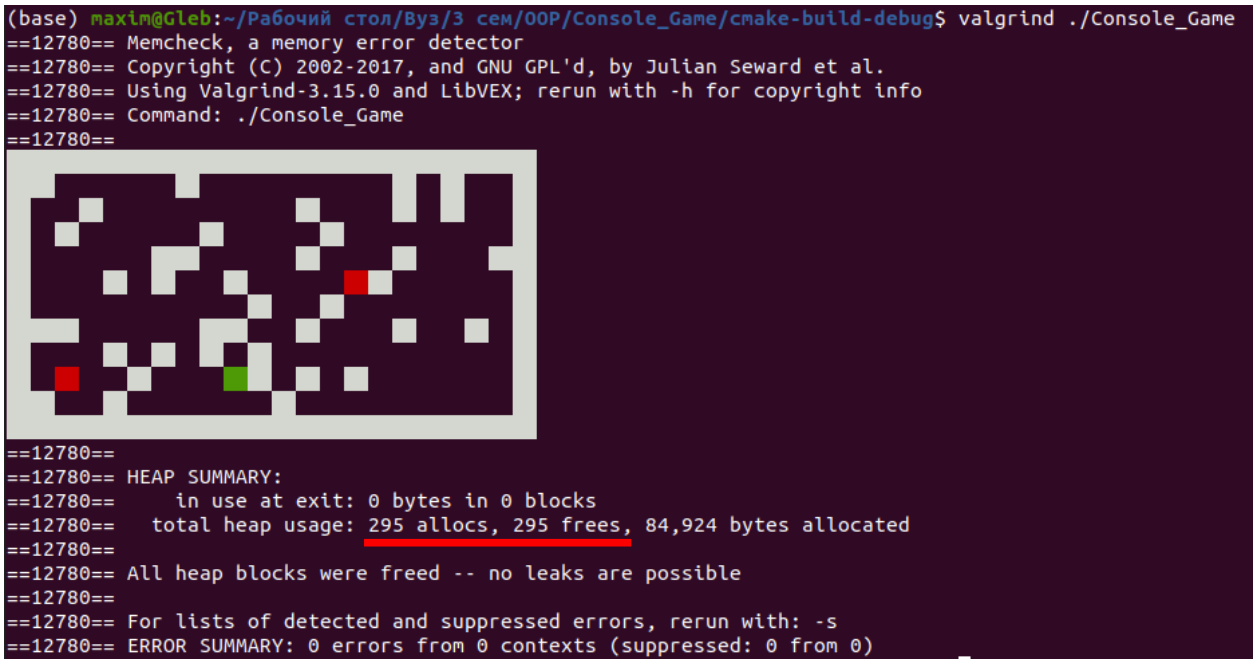

Рисунок 3: Исследование на утечки памяти

Выводы.

Был получен опыт в проектировать программы в ООП стиле, для ее большей гибкости , модифицируемости и возможность повторного использования.

Было изучены основные принципы ООП, также был получен опыт работы с умными указателями и полиморфизмом.

Разработана программа, выполняющая поставленное задание.