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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Объектно-ориентированное программирование»
Тема: Создание классов, конструкторов и методов

Студент гр. 1383 _____ Куликов М.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2022

Цель работы.

Реализовать прямоугольное игровое поле, состоящее из клеток. Клетка - элемент поля, которая может быть проходима или нет (определяет, куда может стать игрок), а также содержит какое-либо событие, которое срабатывает, когда игрок становится на клетку. Для игрового поля при создании должна быть возможность установить размер (количество клеток по вертикали и горизонтали). Игровое поле должно быть зациклено по вертикали и горизонтали, то есть если игрок находится на правой границе и идет вправо, то он оказывается на левой границе (аналогично для всех краев поля).

Реализовать класс игрока. Игрок - сущность, контролируемая пользователем. Игрок должен иметь свой набор характеристик и различный набор действий (например, разные способы перемещения, попытка избежать событие, и так далее).

Задание.

Реализовать класс игрового поля

Для игрового поля реализовать конструктор с возможностью задать размер и конструктор по умолчанию (то есть конструктор, который можно вызвать без аргументов)

Реализовать класс интерфейс события (в данной лабораторной это может быть пустой абстрактный класс)

Реализовать класс клетки с конструктором, позволяющим задать ей начальные параметры.

Для клетки реализовать методы реагирования на то, что игрок перешел на клетку.

Для клетки реализовать методы, позволяющие заменять событие. (То есть клетка в ходе игры может динамически меняться)

Реализовать конструкторы копирования и перемещения, и соответствующие им операторы присваивания для игрового поля и при необходимости клетки

Реализовать класс игрока минимум с 3 характеристиками. И соответствующие ему конструкторы.

Реализовать перемещение игрока по полю с проверкой допустимости на переход по клеткам.

Выполнение работы.

Был реализован набор классов для выполнения данной лабораторной работы: Field, Cell, Player, CellEvent, FieldView, Controller. Далее приводится описание этих классов, их полей и методов.

CellEvent – интерфейс событий, которые содержатся на клетке. На данный момент имеет только одну виртуальную функцию void interact (Player* hero), которая срабатывает, когда игрок попадает на клетку.

Cell – класс клетки, хранящейся на поле.

Было создано перечисление enum Cell_type {Wall, Free, Start, Finish} для обозначения типа клетки.

Поля класса Cell:

Cell_type type; - тип клетки

Cell_Event *event; - событие, содержащееся на клетке (для стены или начальной клетки – nullptr)

int player_on_cell_flag; - флаг присутствия игрока на клетке

Методы класса Cell:

void set_cell_type(Cell_type other_type); - метод для изменения типа клетки. Создан для динамического изменения клетки по ходу игры.

void player_out_of_cell(); - метод, указывающий, что игрок покинул данную клетку. Позволяет FieldView понять, где игрок больше не находится.

void player_on_cell(Player *hero); - метод, указывающий, что игрок попал на данную клетку. Также запускает взаимодействие с событием.

Cell_type get_cell_type(); - метод, возвращающий тип клетки. Позволяет FieldView понять, каким символом обозначать клетку.

void set_event(Cell_Event *other_event); - метод, позволяющий изменить событие на клетке. Создан для динамического изменения клетки по ходу игры.

Cell_Event *get_event(); - метод, позволяющий получить событие, содержащееся на клетке. Создан для доступа к полю события клетки.

int is_player_on_cell(); - метод, позволяющий узнать, есть ли на клетке игрок. Позволяет FieldView понять, где игрок находится.

explicit Cell(Cell_type other_type = Free, Cell_Event *other_event = nullptr, int player_flag = 0); - конструктор клетки со значениями по умолчанию. При создании поля все клетки свободны и не содержат игрока.

Также были созданы конструкторы копирования и перемещения и соответствующие им операторы присваивания.

Player – класс игрока.

Для игрока были создано 3 поля для хранения его характеристик:

int player_health; - здоровье игрока.

int player_damage; - урон игрока.

int agility; - ловкость игрока. Определяет шанс избежать событие.

Методы класса Player:

explicit Player(int health = 10, int damage = 1, int agility = 1); - конструктор игрока со значениями по умолчанию.

void set_player_damage(int new_damage); - метод, позволяющий изменить урон игрока.

void set_player_health(int new_health); - метод, позволяющий изменить здоровье игрока.

void set_player_agility(int new_agility); - метод, позволяющий изменить ловкость игрока.

int avoid_event(); - метод, позволяющий избежать событие. Чем больше ловкость игрока – тем больше шанс избежать событие.

Field – класс поля игры.

MAP_SIZE – стандартное значение для высоты и ширины поля, равное 10.

Поля класса Field:

Cell **map; - двумерный массив, содержащий в себе клетки, по которым передвигается игрок.

int field_height; - высота поля.

int field_width; - ширина поля.

int player_x_coord; - координата x клетки, на которой сейчас игрок.

int player_y_coord; - координата y клетки, на которой сейчас игрок.

Player *hero; - указатель на игрока.

Cell *current_player_cell; - указатель на клетку, на которой сейчас игрок.

Методы класса Field:

Field(int height = MAP_SIZE, int width = MAP_SIZE, Player *hero = new Player(), int player_x_coord = 0, int player_y_coord = 0); - конструктор поля с аргументами по умолчанию. По умолчанию игрок появляется в клетке с координатами (0,0).

Cell **get_field(); - метод, позволяющий получить массив клеток поля. Позволяет FieldView понять, как изменилось поле после действий игрока и отразить это в выводе в терминал.

int get_height(); - метод, позволяющий получить высоту поля.

int get_width(); - метод, позволяющий получить ширину поля.

void move_player(int delta_x_coord, int delta_y_coord); - метод для передвижения игрока. Изменяет после вызова поля int player_x_coord , int player_y_coord и Cell *current_player_cell. Также меняет флаги

player_on_cell_flag в нужных клетках на нужные значения. Реализована цикличность карты за счет проверки координат новой клетки, на которую хочет пойти игрок.

~Field(); - деструктор, очищающий массив клеток поля.

FieldView – класс, преобразующий поле и выводящий его в терминал.

Поля класса FieldView:

Field *game_field; - текущее поле, которое нужно вывести в терминал.

char **output_arr; - массив символов, который выводится на экран, где F – свободная клетка, P – Клетка с игроком, W – непроходимая клетка.

int output_arr_width; - ширина массива символов

int output_arr_height; - высота массива символов

Методы класса Field:

explicit FieldView(Field *other_field); - конструктор, принимающий игровое поле в качестве аргумента.

void show_field(); - метод, выводящий в терминал массив символов, обозначающих поле

Field* get_field(); - метод, возвращающий текущее игровое поле.

~FieldView(); - деструктор, очищающий массив символов.

Controller – класс, запускающий игру и принимающий посимвольные команды от пользователя.

Поля класса Controller:

FieldView* fieldview_obj; - объект класса FieldView, который взаимодействует с полем и выводит его.

Методы класса Controller:

Controller(); - конструктор, не имеющий аргументов.

void set_field_size(int width, int height); - метод, позволяющий задать размер поля. Если ширина или высота меньше 3 или высота или ширина больше 20, то высота и ширина поля становится равна 10 по умолчанию.

void start_game(); - метод, запускающий игру. Посимвольно считывает команды пользователя, где w - передвижение вверх по полю, a - передвижение влево по полю, s - передвижение вниз по полю, d - передвижение вправо по полю, e – выход из игры.

~Controller(); - деструктор, очищающий объект типа FieldView.

Тестирование.

Начальное состояние поля размером 10x10:

Состояние поле после трех шагов вправо:

Состояние поле после шага влево, шага вниз, шага вправо:

Выводы.

Был создан набор классов, набор полей и методов для них, что в совокупности удовлетворило целям лабораторной работы и создало игровое поле из клеток, содержащих события, по которым передвигается игрок.

ПРИЛОЖЕНИЕ А

UML - ДИАГРАММА