**Министерство образования Российской Федерации**

**Нижегородский государственный технический университет им. Р. Е. Алексеева**

**Факультет**

**Кафедра**

**“Игра жанра Rouge-like”**

Техническое задание

на курсовою работу

Листов

Руководитель,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Исполнитель,

Студент гр. 19-ИВТ-2 Шишарин Р.А.

2020

**Содержание**

Введение …………………………………………………………………………….. стр. 3

Глава 1. Разработка технического задания………………………………………... стр. 4

Глава 2. Анализ предметной деятельности, уточнение спецификаций и

разработка системы………………………………………………………………… стр. 6

Глава 3. Проектирование интерфейса пользователя…………………………….. стр. 11

Глава 4. Проектирование классов приложения …………………………………. стр. 16

Заключение…………………………………………………………………………. стр. 33

Биография…………………………………………………………………………… стр. 34

Приложение А. Исходный код на С++……………………………………………. стр. 35

**Введение**

Компьютер быстро вошёл в нашу жизнь. Несколько лет назад персональные компьютеры были очень редким явлением из-за своей стоимости. Их стоимость была настолько большой, что только некоторые организации могли их себе позволить. Сейчас же персональные компьютеры есть практически в каждом доме, которые глубоко вошли в жизни каждого человека. С помощью компьютера можно не только работать и общаться, но и ещё развлекаться, проводить хорошо время. Одним из способов является игра в видеоигры.

В настоящее время существует миллионы игр различных жанров и видов. Одним из популярных жанров является Rogue-like. Восхождение данного жанра стала игра 1980 под названием “Rogue”. Хотя и до неё выходили подобные игры, такие как “Beneath Apple Manor” и “Sword of Fargoal”, именно “Rogue” стала образцом для бесчисленных подражаний. Главными особенностями данного жанра являются генерируемые случайным образом уровни и необратимость смерти персонажа – в случае гибели персонажа, игрок не мог загрузиться и продолжить прогресс, игроку приходилось начинать игру сначала.

**Глава 1. Разработка Технического Задания**

1 НАИМЕНОВАНИЕ РАЗРАБОТКИ

“Игра жанра Рогалик”

2 ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

Игра разрабатывается в качестве курсовой работы по курсу «» в соответствии учебного плана.

3 НАЗНАЧЕНИЕ

Основным назначением программы является

4 ТРЕБОВАНИЯ К ПРОГРАММНОЙ СИСТЕМЕ

4.1.1 Система должна обеспечивать выполнение следующих функций:

Ввод информации с помощью клавиатуры

Загрузка файлов в формате .png

4.2 ТРЕБОВАНИЯ К НАДЁЖНОСТИ

4.2.1 Предусмотреть контроль вводимой информации.

4.2.2 Предусмотреть блокировку некорректных действий пользователя при работе с системой.

4.3 ТРЕБОВАНИЯ К СОСТАВУ И ПАРАМЕТРАМ ТЕХНИЧЕСКИХ СРЕДСТВ

4.3.1 Программная система должна работать на персональных компьютерах типа IBM PC.

4.3.2 Минимальная аппаратная конфигурация:

Тип процесса\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pentium и выше.

Объём запоминающего устройства\_\_\_\_\_\_\_\_\_54 МБ и более.

4.4 Требования к информационной и программной совместимости

Система проектируется как платформо независимая программная система. Переносимость программы поддерживается за счёт переносимости С++-интерпретатора.

5 ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

5.1 Разрабатываемые программные модули должны быть самодокументированы, то есть тексты программ должны содержать все необходимые комментарии.

5.2 Разрабатываемая система должна включать справочную информацию.

5.3 В состав сопровождающей документации должны входить:

5.3.1 Пояснительная записка на 25-30 листах, содержащая описание разработки.

5.3.2 Руководство пользователя.

5.3.3 Тексты программ.

**ГЛАВА 2. АНАЛИЗ ПРЕДМЕТНОЙ ДЕЯТЕЛЬНОСТИ, УТОЧНЕНИЕ СПЕЦИФИКАЦИЙ И РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ**

Режим работы «Главное Меню»

Типичный ход событий

|  |  |
| --- | --- |
| Действия исполнителя | Отклик системы |
| 1. Пользователь выбирает режим работы в главном меню.   А) Пользователь нажал на кнопку «Вверх» или «Вниз».  Б) Пользователь нажал на кнопку «Enter». Если выбран режим «Новая Игра».  В) Пользователь нажал на кнопку «Enter». Если выбран режим «Выход». | 2А.Система изменяет выбранный раздел в главном меню.  2Б. Система начинает подготовку к игре. Смотрим раздел «Игра».  2В. Система завершает работу. |

Режим работы «Игра»

Типичный ход событий

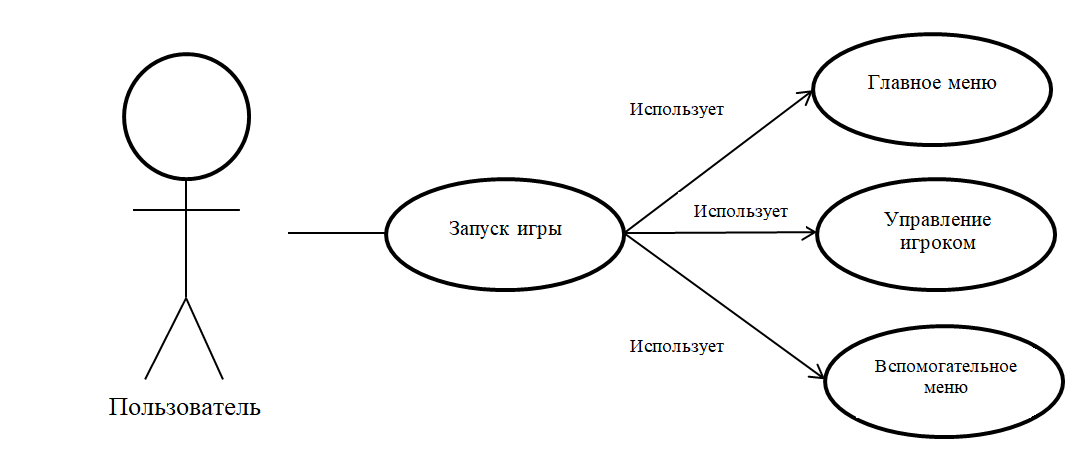
|  |  |
| --- | --- |
| Действия исполнителя | Отклик системы |
| 1. Пользователь нажимает кнопки на клавиатуре.   А) Пользователь нажимает на кнопки, которые являются «Стрелками».  Б) Пользователь нажал на кнопку «Escape». | 1. Система получает информацию ввода и производит действия.   А) Система начинает воспроизводить передвижение игрока в зависимости от нажатой кнопки. Если игровой персонаж управляемый игроком доходит до объекта перехода уровня, происходит перенос игрока на следующий уровень.  Б) Система запускает паузу в игре и, также запускает вспомогательное меню. Смотрим раздел «Вспомогательное меню». |

Режим работы «Вспомогательное меню»

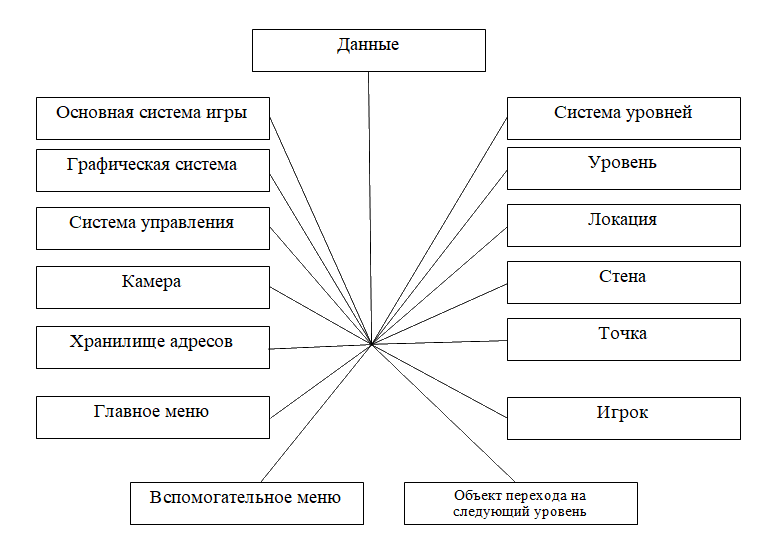
Типичный ход событий

|  |  |
| --- | --- |
| Действия исполнителя | Отклик системы |
| 1. Пользователь нажимает кнопки на клавиатуре.   А) Пользователь нажал на кнопку «Вверх» или «Вниз»  Б) Пользователь нажал на кнопку «Enter». Когда выбран раздел «Continue».  В) Пользователь нажал на кнопку «Enter». Когда выбран раздел «Exit».    Г) Пользователь нажал на кнопку «Escape». | 1. Система получает информацию ввода и производит действия.   А) Система изменяет выбранный раздел в вспомогательном меню.  Б) Система удаляет вспомогательное меню и продолжает игру. Смотрим раздел «Игра».  В) Система выключает игру и переходит в главное меню. Смотрим радел «Главное меню».  Г) Система удаляет вспомогательное меню и продолжает игру. Смотрим раздел «Игра». |

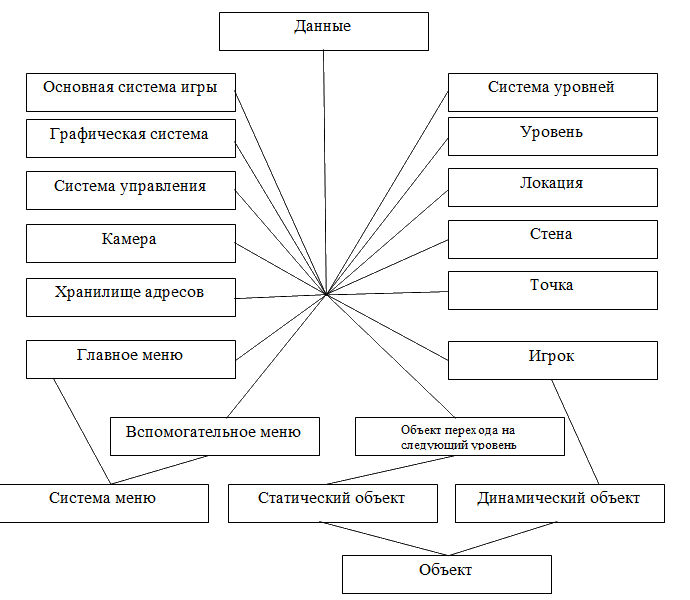
Таким образом получается один исходный вариант использования – запуск игры. Также три вспомогательных – главное меню, управление игроком и вспомогательное меню.



**Рис.1.** Диаграмма использования



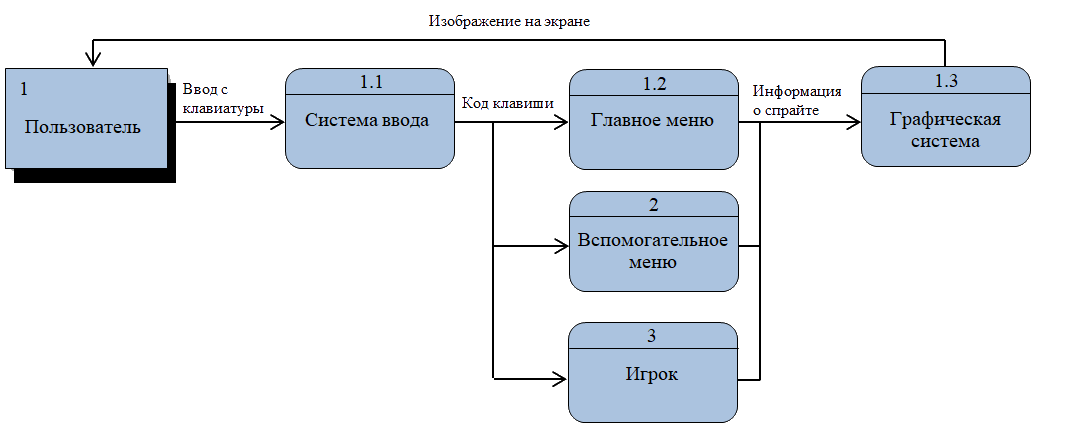
**Рис.2.** Диаграмма структуры данных



**Рис.3.** Полнаядиаграмма структуры данных

Из диаграммы использования следует, что пользователь взаимодействует с приложением с помощью ввода клавиатуры. Ввод клавиатуры принимает “Систма ввода”, и если она получила нужну клавишу, то в зависимости от состояния игры (Главное меню, пауза, игра) передаёт код клавиши следующим объектам: “Главное меню”, “Игрок”, “Вспомогательное меню”. Эти объекты после манипуляций с кодом клавиши отправляют информацию о спрайте объекту “Графическая система”, которая уже отрисовывает изображение на экране, которое видит пользователь.

Таким образом можно построить диаграмму потоков данны

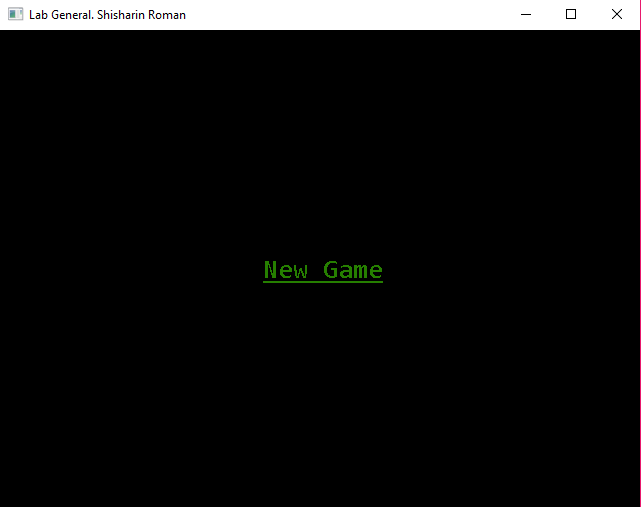


**Рис.4.** Диаграмма потоков данных

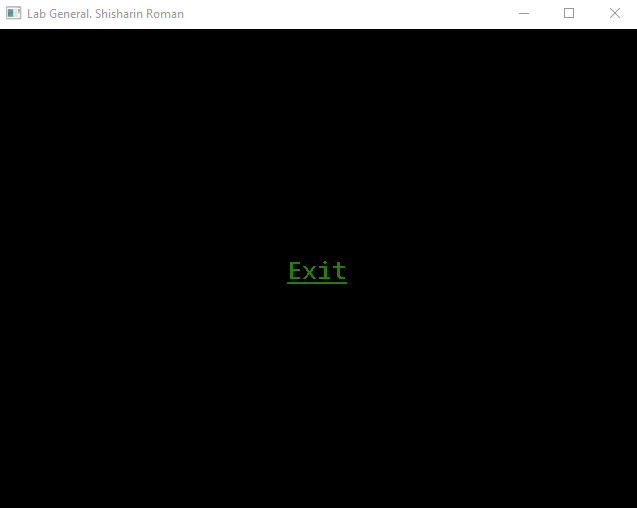
**ГЛАВА 3. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ**

При объектно-ориентированном подходе к разработке программного обеспечения проектирование программного продукта начинаем с разработки интерфейса пользователя. На этапе уточнения спецификации был определён сценарий работы игры.

В качестве языка программирования для реализации графического интерфейса был выбран C++. Также в состав сборки входит библиотека SFML v 2.5.1, для отрисовки спрайтов у объектов. В первую очередь игрока встречает главное меню игры, которое показывает выбранное изначально действие, которое может выбрать пользователь или поменять.

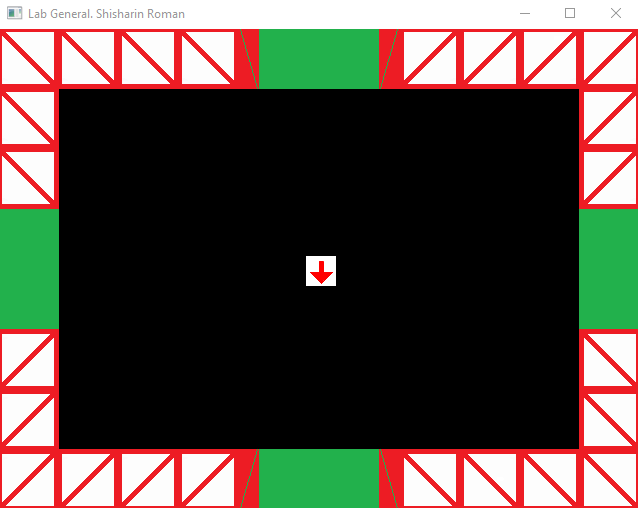


**Рис.5.** Главное меню при выборе действия “Начать игру”



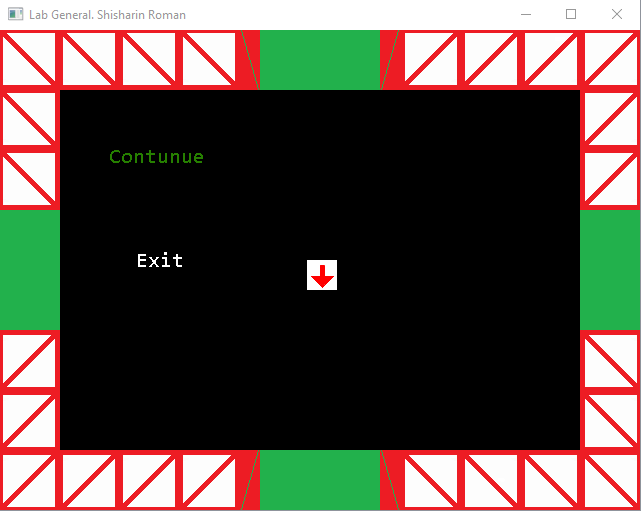
**Рис.6.** Главное меню при выборе действия “Выход”

Затем, когда игрок выбрал действие “Начать игру”, на экране начинает рисоваться изображение комнаты (Её стен) и игрового персонажа.



**Рис.7.** Отображение стартовой комнаты и игрового персонажа

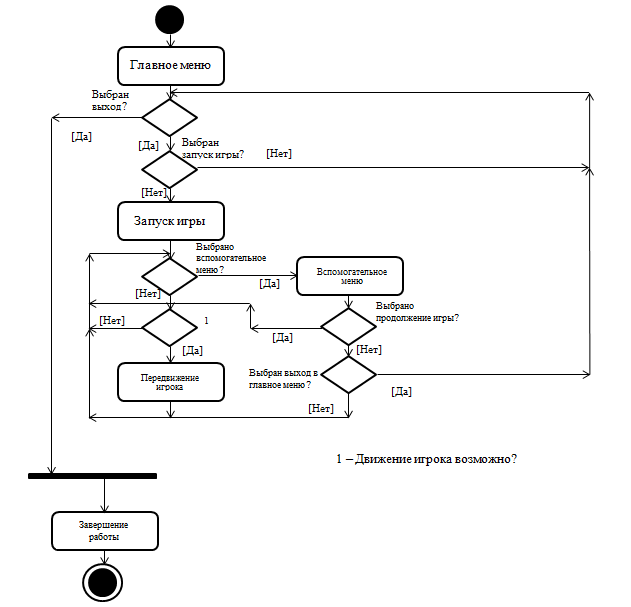
Также если во время игры, игрок решит выйти в главное меню или нажать на паузу, будет отображено вспомогательное подменю.



**Рис.** Отображение вспомогательного меню.

Библиотека SFML не поддерживает концепцию «Визуального программирования». Вся работа по отрисовки спрайтов выполняется в С++ сценарии.

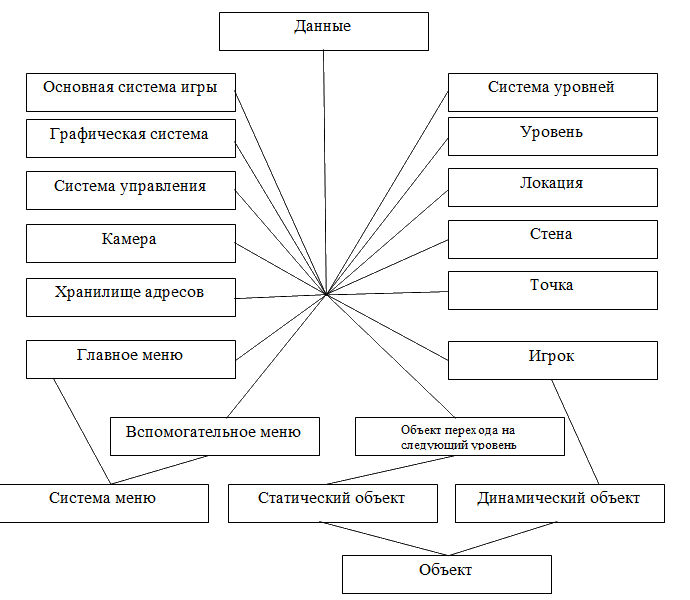
Создание экрана графического интерфейса происходит перед запуском “основного” цикла в экземпляре класса Game. Для реализации отрисовки используется экземпляр класса GraphicSystem. В GraphicSystem храниться массив структур данных, содержащую информацию по спрайту объекта. Таким образом можно довольно легко записывать внутрь массива информацию нужного спрайта при создании объекта, и также удалять при удалении соответственного объекта.



**Рис.8.** Диаграмма перехода состояний программной среды

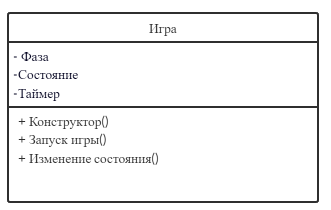
**ГЛАВА 5. ПРОЕКТИРОВАНИЕ КЛАССОВ ПРИЛОЖЕНИЯ**

Разработанные структуры данных и диаграммы использования позволяют подойти к этапу разработки классов. Ещё раз обратимся к структуре данных:



**Рис..** Диаграмма структуры данных

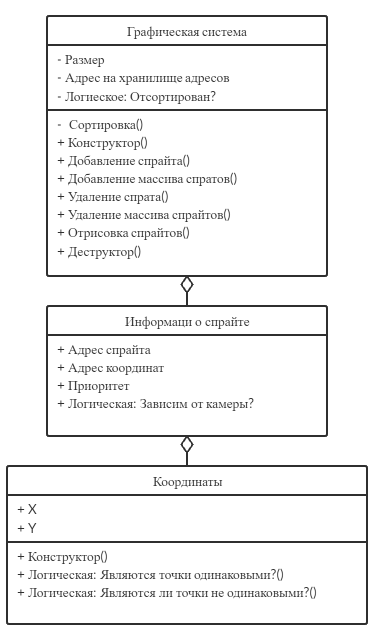
В начале разберём главный класс “Игра”, который отвечает за “основной цикл” игры. Основные атрибуты класса “Игра” определяются исходя из необходимых действий для контроля процессов запуска различных стадий игры: “Состояние”, “Фаза”, “Таймер”. Также у класса ”Игра” есть три метода: “Конструктор”, “Запуск игры”, “Изменение состояния”.



**Рис.** UML-модель класса “Игра”

Метод “Запуск игры” является “основным циклом” игры, в котором происходят все действия по созданию, работе и удалению других объектов. Атрибут “Состояние” необходимо для определения стадии в игре (главное меню, игра, пауза). Атрибут “Фаза” определяет, является ли это начавшимся состоянием или нет.

Так как у этой игры является графический интерфейс, то для его работы также необходим класс, который будет управлять графическим интерфейсом. Данную роль выполняет класс “Графическая система”. Задача данного класса хранить определённую информацию о спрайтах объектов для корректного отрисовывания их на экране. Для хранения информации о спрайте было решено в классе “Графическая система”, создать “агрегат” структуру “Информация о спрайте”. Таким образом основные атрибуты класса “Графическая система” являются: “Размер”, “Адрес на хранилище адресов”, “Логическское: Отсортирован?”. Основные методы класса, являются: “Сортировка”, “Конструктор”, “Добавление спрайта”, “Добавление массива спрайтов”, “Удаление спрайта”, “Удаление массива спрайтов”, “Отрисовка спрайтов”, “Деструктор”.

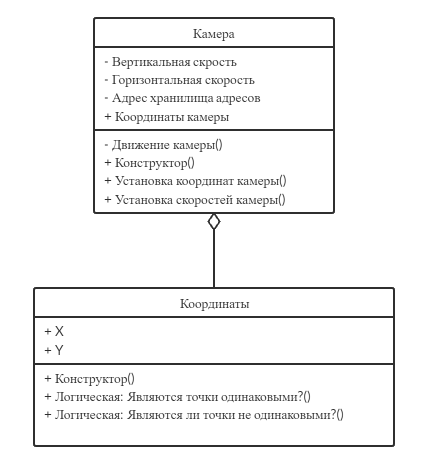


**Рис.9.** UML-модель класса “Графическая система” с “агрегатами”

Как можно заметить у структуры “Информация о спрайте” есть “агрегат” структура “Координаты”. Данная структура является базовой реализацией точки на координатной плоскости с методами сравнения с другими точками.

У класса “Графическая система” можно заметить атрибут “Логическая: Отсортирован?”. Данный атрибут показывает нуждается ли массив структур “Информация о спрайте” в сортировке по приоритету (атрибут структуры “Информация о спрайте”).

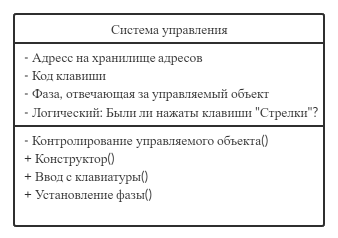
Для того чтобы спрайты рисовались корректно (В данном случае некоторые объекты могут отрисовываться только за экраном из-за координат), необходим объект, относительно которого будут отрисовываться спрайты. Данным объектом выступает класс “Камера”. Основные атрибуты класса “Камера”: “Вертикальная скорость”, ”Горизонтальная скорость”, ”Адрес хранилища адресов”, ”Координаты камеры”. Основные методы класса: “Движение камеры”, “Конструктор”, “Установка координат камеры”, “Установка скоростей камеры”.



**Рис.10.** UML-модель класса “Камера”

Метод класса “Установка скоростей камеры”, отвечает за правильное высчитывание горизонтальной и вертикальной скорости камеры во время игры, когда “Игрок” производит движение за определённую зону на экране.

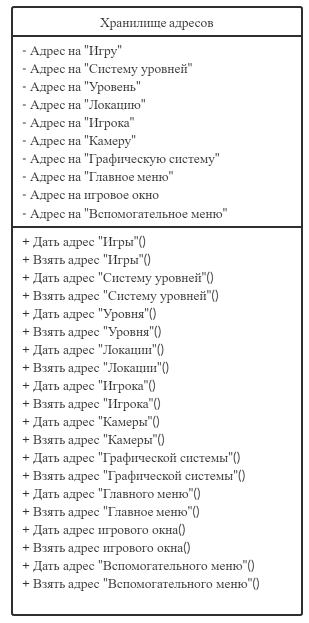
Кроме графического интерфейса у игры должен быть способ с помощью которого пользователь мог взаимодействовать с игрой. Данную функцию выполняет класс “Система управления”. Этот класс принимает ввод пользователя и определяет куда нужно отправить в какой объект направить данный ввод. Основными атрибутами для класса являются: “Адрес на хранилище адресов”, “Код клавиши”, “Фаза, отвечающая за управляемый объект”, “Логический: Были ли нажаты стрелки?”. Основными методами класса являются: “Контролирование управляемого объекта”, “Конструктор”, “Ввод с клавиатуры”, “Установление фазы”.



**Рис.11.** UML-модель класса “Система управления”

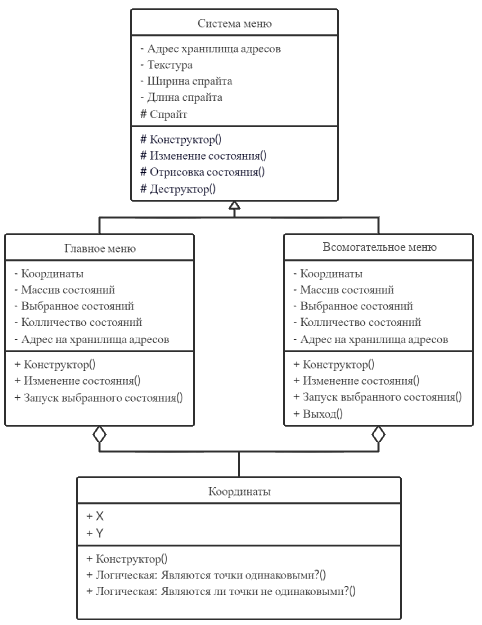
У данного класса есть атрибут “Логический: Были ли нажаты стрелки?”, данный атрибут используется в методе “Ввод с клавиатуры” и используется для определения нажаты ли кнопка являющаяся “стрелкой”. Также есть атрибут “Фаза, отвечающая за управляемый объект”, данный атрибут определяет стадию игры, в зависимости от которой переключается объект, который должен принимать ввод с клавиатуры (главное меню – “Главное меню”, игра – “Игрок”, пауза – “Вспомогательное меню”). Данная передача ввода осуществляется в методе класса “Контролирование управляемого объекта”.

Класс “Хранилище адресов”. Данный класс “связывает” различные объекты игры, тем самым упрощая взаимодействие между объектами. Атрибуты класса “Хранилище адресов”: “Адрес на “Игру””, “Адрес на ”Систему уровней””, “Адрес на ”Уровень””, “Адрес на ”Локацию””, “Адрес на ”Игрока””, “Адрес на ”Камеру””, “Адрес на ”Графическую систему””, “Адрес на ”Главное меню””, “Адрес на игровое окно”, “Адрес на ”Вспомогательное меню””. Методы класса “Хранилище адресов” являются базовыми методами взятия и отдачи адресов другим объектам.



**Рис.12.** UML-модель класса “Хранилище адресов”

Теперь, когда были рассмотрены все системные объекты игры, можно перейти к объектам меню. Данными объектами являются классы “Главное меню” и “Вспомогательное меню”. Так как основа меню - это список команд, из которых можно выбрать одну команду, то было решено создать дополнительный класс “Система меню”, который будет основой объектов меню. Класс “Система меню” является родительским классом для классов “Главное меню” и “Вспомогательное меню”. Основные атрибуты ”Система меню”: “Адрес хранилища адресов”, “Текстура”, “Ширина спрайта”, “Длина спрайта”, “Спрайт”. Основные методы класса “Система меню”: “Конструктор”, “Изменение состояния”, “Отрисовка состояния”, “Деструктор”. Атрибуты класса “Главное меню”: “Координаты”, “Массив состояний”, “Выбранное состояние”, “Количество состояний”, “Адрес на хранилище адресов”. Методы класса “Главное меню”: “Конструктор”, “Изменение состояния”, “Запуск выбранного состояния”. Атрибуты класса “Вспомогательное меню”: “Координаты”, “Массив состояний”, “Выбранное состояние”, “Количество состояний”, “Адрес на хранилище адресов”. Методы класса “Вспомогательное меню”: “Конструктор”, “Изменение состояния”, “Запуск выбранного состояния”, “Выход”.

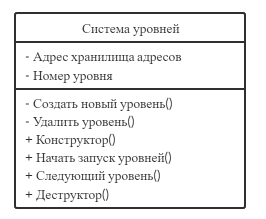


**Рис.13.** UML-модель класса “Главное меню”, “Вспомогательное меню” и

“Система меню”

Класс “Система уровней”. Этот класс является системой, которая работает с уровнями, т.е. ведёт подсчёт номера уровня, создаёт новый уровень и удаляет старый.

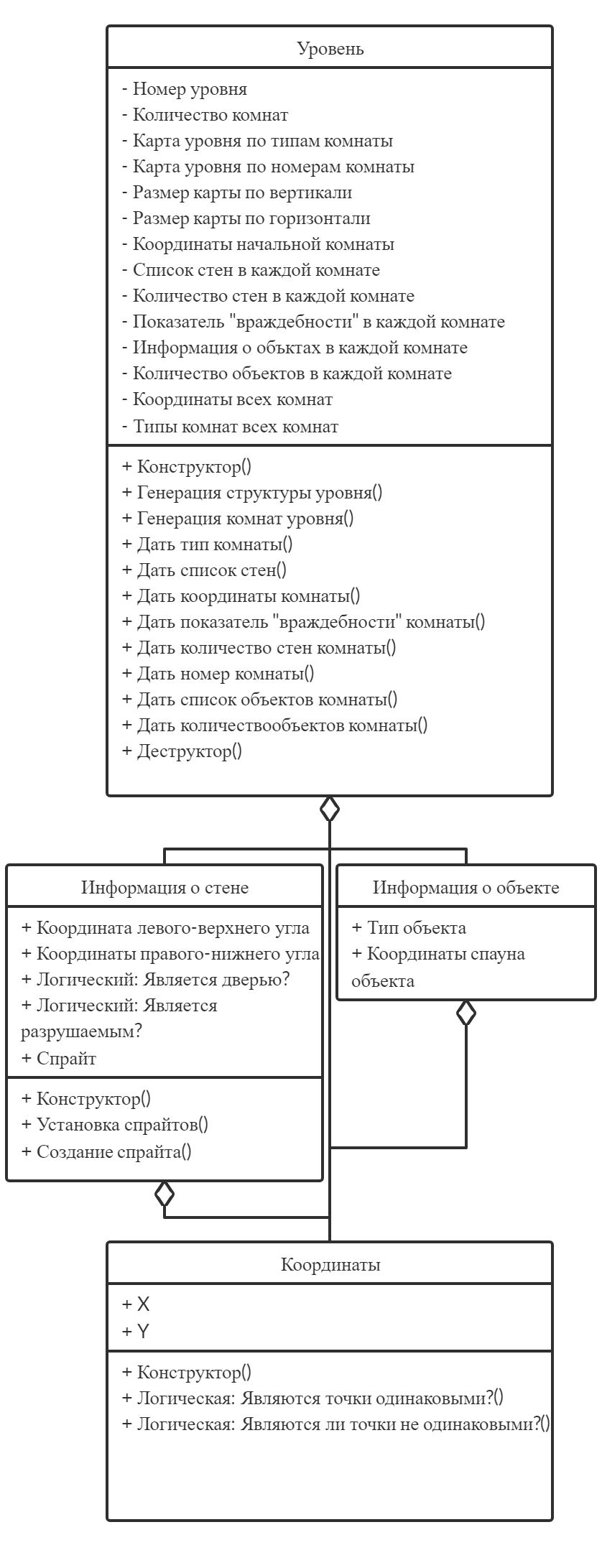
Основные атрибуты класса “Систему уровней”: “Адрес хранилища адресов”, ”Номер уровня”. Основные методы класса “Система уровней”: “Создать новый уровень”, “Удалить уровень”, “Конструктор”, “Начать запуск уровней”, “Следующий уровень”, “Деструктор”.



**Рис.14.** UML-модель класса “Система уровней”

Метод “Начать запуск уровней” запускается только для первого уровня, его цель задать начальные настройки для генерации уровня, для следующих уровней запускается метод “Следующий уровень”.

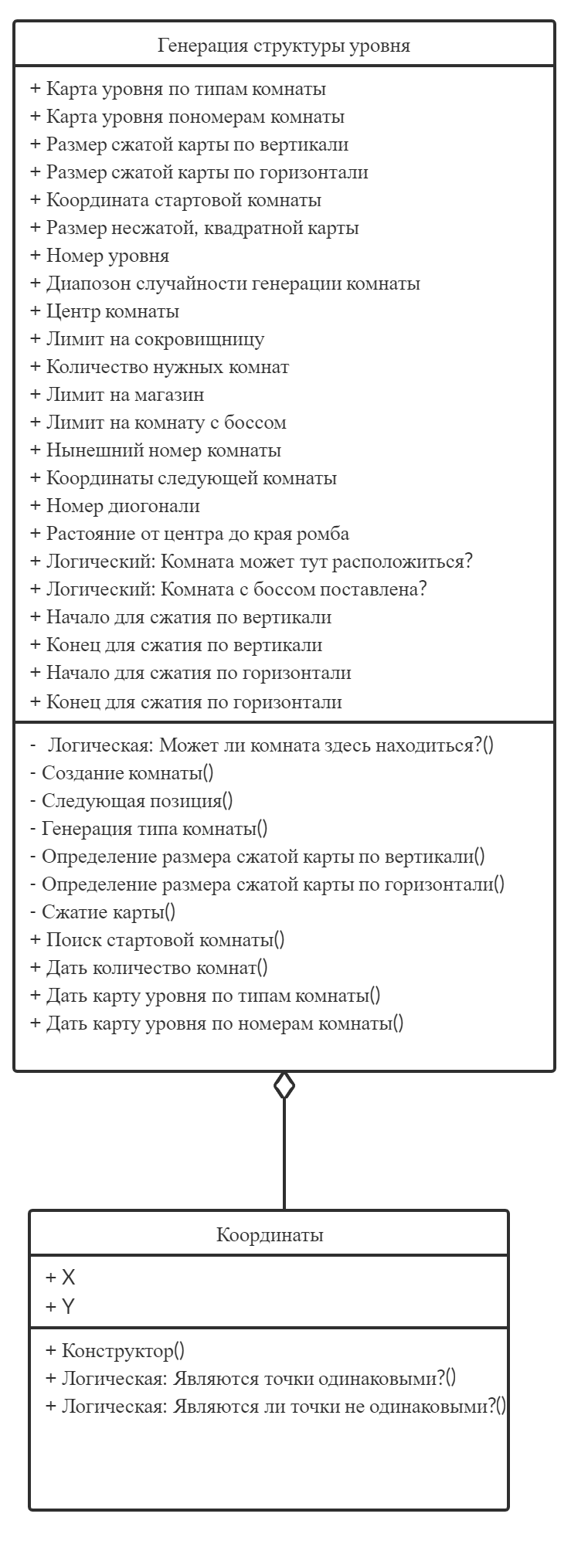
Для того чтобы хранить данные про уровень, был создан одноименный класс “Уровень”. Данный класс полную информацию о уровне и о каждой комнате включающейся в уровне. Также методы этого класса позволяют легко получить нужные данные о любой комнате. Основные атрибуты класса “Уровень”: “Номер уровня”, “Количество комнат”, “Карта уровня по типам комнат”, “Карта комнаты по номерам комнат”, “Размер карты по вертикали”, “Размер по горизонтали”, “Координаты начальной комнаты”, “Список стен в каждой комнате”, “Количество стен в каждой комнате”, “Показатель “враждебности” в каждой комнате”, “Информация о объектах в каждой комнате”, “Количество объектов в каждой комнате”, “Координаты всех комнат”, “Типы всех комнат”. Для облегчения разработки было решено создать две структуры “Информация об объекте” и “Информация о стене” для обобщения данных объектов, которые являются “агрегатами” класса “Уровень”. Основные методы класса “Уровень”: “Конструктор”, “Генерация структуры уровня”, “Генерация комнат уровня”, “Дать тип комнаты”, “Дать список стен комнаты”, “Дать координаты комнаты”, “Дать показатель “враждебности” комнаты”, “Дать количество стен комнаты”, “Дать номер комнаты”, “Дать список объектов комнаты”, “Дать количество объектов комнаты”, “Деструктор”.



**Рис.15.** UML-модель класса “Уровень” с “агрегатами”

Генерация уровня происходит за два этапа: первый этап – “Генерация структуры уровня”, второй этап – “Генерация комнат уровня”.

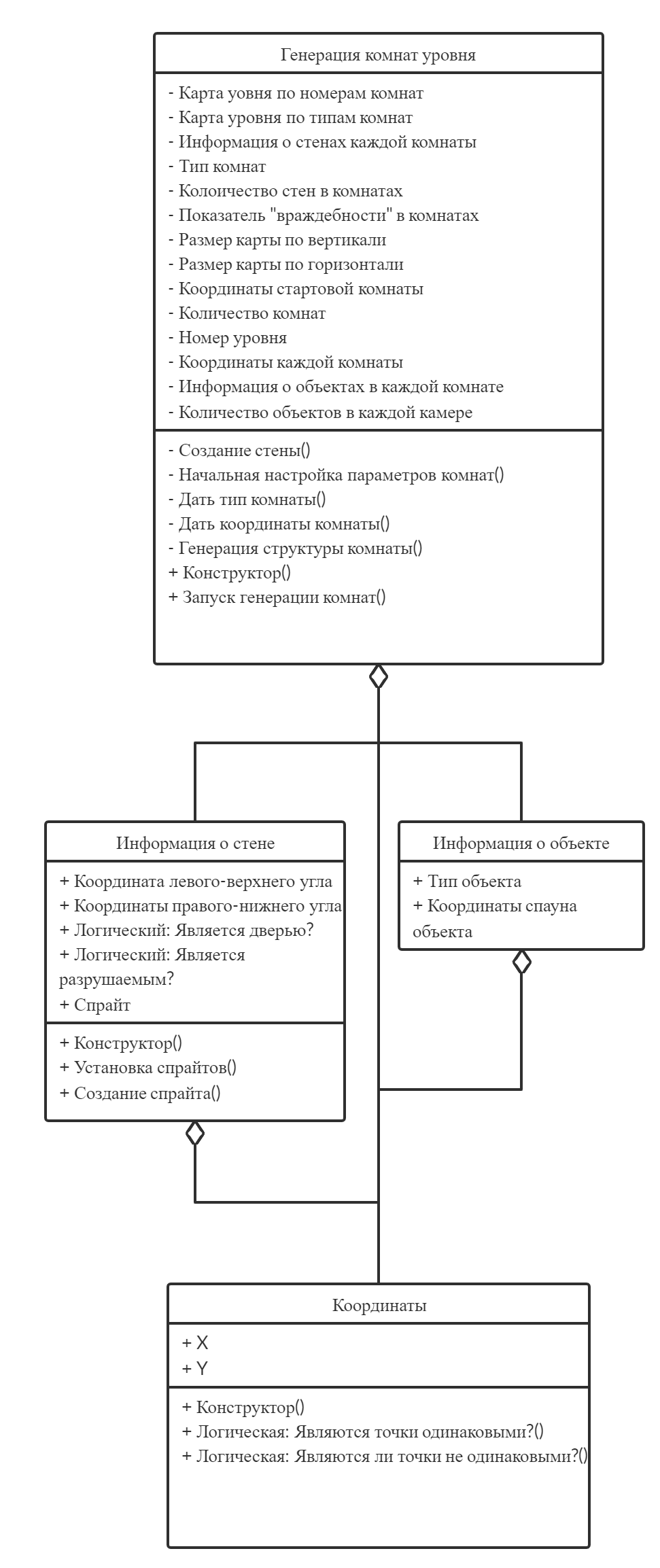
Для генерации структуры уровня используется класс “Генерация структуры уровня”. Основные атрибуты класса “Генерация структуры уровня”: “Карта по типам комнаты”, “Карта по номерам комнаты”, “Размер сжатой карты по вертикали”, “Размер сжатой карты по вертикали”, “Координата стартовой комнаты”, “Размер не сжатой карты”, “Номер уровня”, “Диапазон случайности генерации комнаты”, “Центр уровня”, “Лимит на количество сокровищниц”, “Лимит на количество магазинов”, “Лимит на количество комнат с боссом”, “Идентификатор комнат”, “Возможные координаты следующей комнаты”, “Номер диагонали”, “Расстояние от центра комнаты до угла ромба”, “Логический: Комната может тут находиться?”, “Логический: Комната с боссом поставлена?”, “Начало для сжатия карты по вертикали”, “Начало для сжатия карты по вертикали”, “Конец для сжатия карты по вертикали”, “Конец для сжатия карты по горизонтали”. Основные методы класса “Генерация структуры уровня”: “Логическая: Может комната тут находиться?”, “Создание комнаты”, “Следующая позиция”, “Генерация типа комнаты”, “Определение сжатой карты по вертикали”, “Определение сжатой карты по горизонтали”, “Сжатие карты”, “Поиск стартовой комнаты”, “Дать количество комнат”, “Дать карту уровня по типу комнат”, “Дать карту уровня по номерам комнат”.



**Рис.16.** UML-модель класса “Генерация структуры уровня” с “агрегатом”

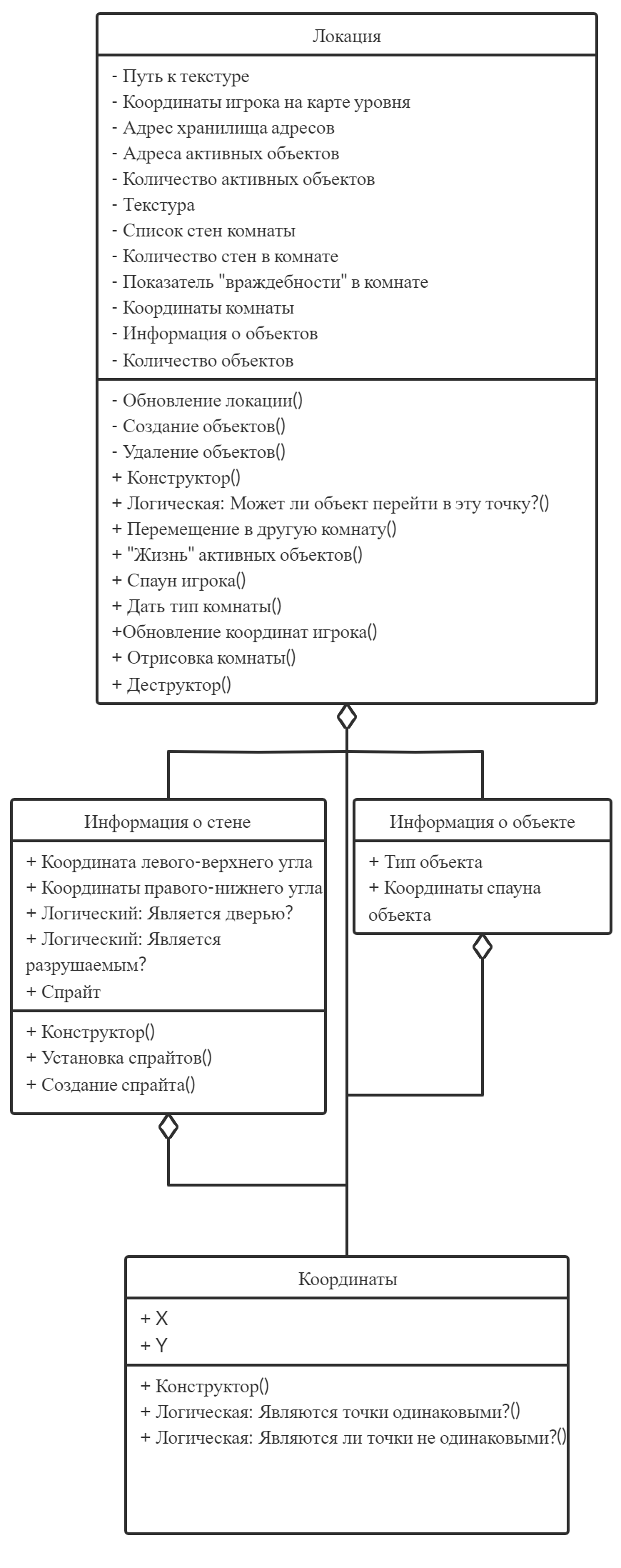
Изначально в данном классе создаётся заранее большая карта, для случайной генерации структуры. Когда структура полностью сгенерировалась, запускаются методы “Определение сжатой карты по вертикали”, “Определение сжатой карты по горизонтали” и “Сжатие карты”, т.е. происходит расчёт и “обрезание” пустых строк и столбцов в карте. Генерация структуры происходит от центра карты (Стартовой комнаты), постепенно происходит рассмотрение “клеток” карты, куда можно поставить комнату и соответственное заполнение “клеток” комнатами. Так как рассматриваются только “клетки” которые граничат с комнатами, выборщик позиции “вычерчивает” фигуру похожую на ромб.

Для второго этапа генерации был также выделен отдельный класс “Генерация комнат уровня”. Данный класс генерирует всю информацию в зависимости от их расположения и типа. Основные атрибуты класса “Генерация комнат уровня”: “Карта уровня по типу комнат”, “Карта уровня по номерам комнат”, “Информация о стенахв каждой комнате”, “Тип комнат”, “Количество стен в каждой комнате”, “Показатель “враждебности” комнат”, “Размер карты по вертикали”, “Размер карты по горизонтали”, “Координаты стартовой комнаты”, “Количество комнат”, “Номер уровня”, “Координаты каждой комнаты”, “Информация об объектах в каждой комнате”, “Количество объектов в каждой комнате”. Основные методы класса “Генерация комнат уровня”: “Создание стены”, “Начальная настройка параметров комнаты”, “Дать тип комнаты”, “Дать координаты комнаты”, “Генерация структуры комнаты”, “Конструктор”, “Запуск генерация комнат”.



**Рис.17.** UML-модель класса “Генерация комнат уровня” с “агрегатами”

Прохождение уровней в игре является “по комнатным”, т.е. в игре обрабатывается только одна комната в которой находиться игровой персонаж. Работать с классом-хранилищем “Уровень” в плане обработки событий, которые происходят в одной комнате не совсем логично, так как он хранит информацию обо всём уровне. Поэтому было решено создать класс “Локация”. Класс “Локация” берёт из класса “Уровень” только информацию об одной комнате, в которой находиться игрок, или в которую уже перемещается, и работает только с ней. Основные атрибуты класса “Локация”: “Путь к текстуре комнаты”, “Координаты игрока на карте уровня”, “Адрес хранилища адресов”, “Адрес активных объектов”, “Количество активных объектов”, “Текстура”, “Список стен в комнате”, “Количество стен в комнате”, “Показатель “враждебности” в комнате”, “Координаты в комнате”, “Информация об объектах”, “Количество объектов”. Основные методы класса “Локация”: “Обновление локации”, “Создание объектов”, “Удаление объектов”, “Конструктор”, “Логическая: Может ли объект перейти в эту точку?”, “Перемещение в другую комнату”, ““Жизнь” активных объектов”, “Создание игрока”, “Дать тип комнаты”, “Обновление координаты в комнате”, “Отрисовка комнаты”, “Деструктор”.



**Рис.18.** UML-модель класса “Локация” с “агрегатами”

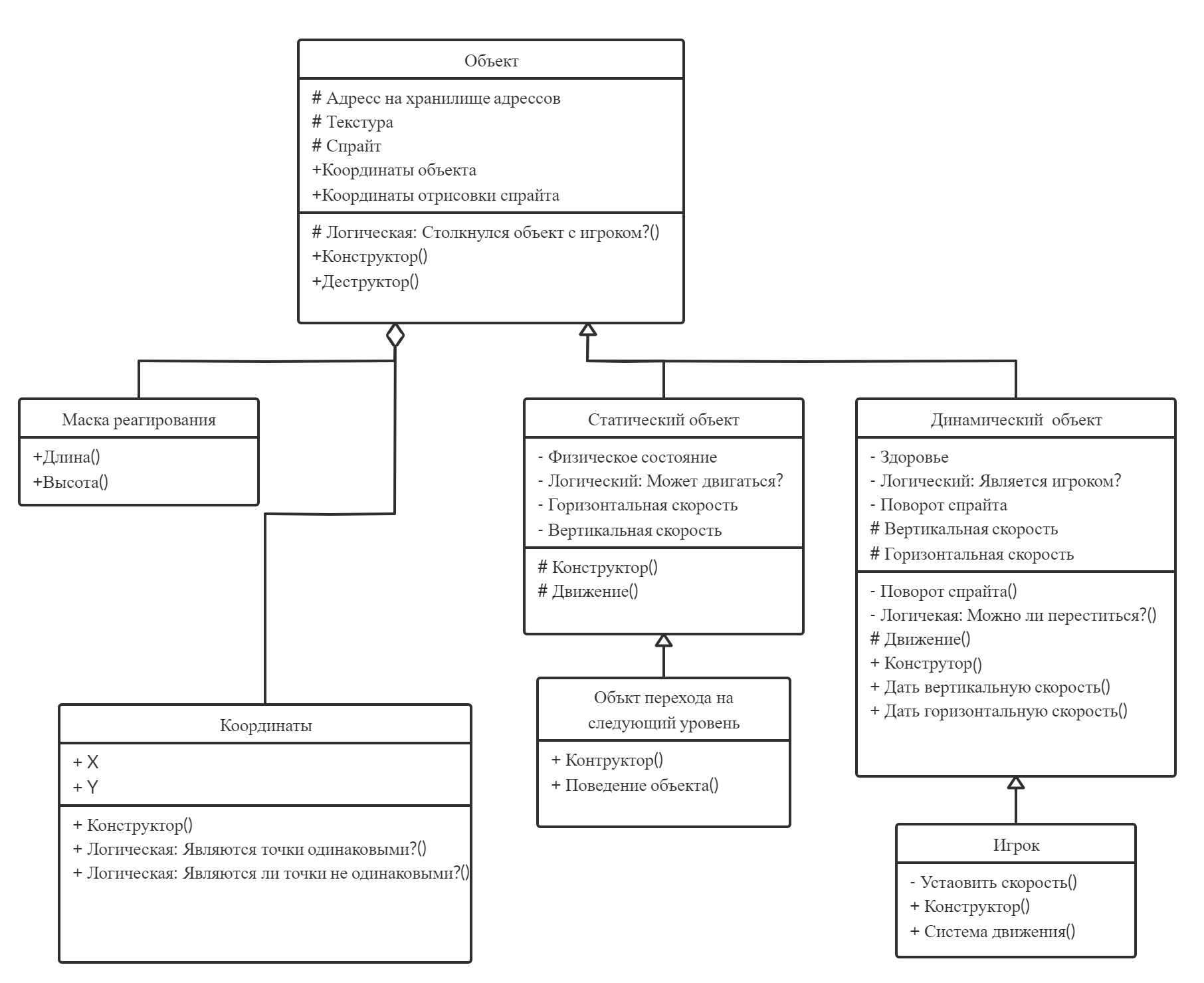
Теперь перейдём от классов, которые работают с уровнями, к игровым объектам. Игровые объекты являются целым семейством в отличии от других классов в этом проекте. Самый “главный” объект – это класс “Объект”. Данный класс хранит основную информацию о объекте (Координаты, координаты спрайта, маску реагирования) и также обрабатывает столкновения игровых объектов с игроком. Основные атрибуты класса “Объект”: “Адрес на хранилище адресов”, “Текстура”, “Спрайт”, “Координаты объекта”, “Координаты отрисовки спрайта”. Основные методы класса “Объект”: “Логическая: Столкнулся объект с игроком?”, “Конструктор”, “Деструктор”. Класс “Объект” является “классом-родителем” для двух классов: “Статический объект” и “Динамический объект”.

Основные атрибуты класса “Статический объект”: “Физическое состояние”, “Логический: Может двигаться?”, “Горизонтальная скорость”, “Вертикальная скорость”. Основные методы класса “Статический объект”: “Конструктор”, “Движение”. Класс “Статический объект” также является “классом-родителем” для класса “Объект перехода на следующий уровень”.

“Объект перехода на следующий уровень” – это объект который отвечает за прохождение уровня, данный объект активируется, когда игровой персонаж дотрагивается до него. Основные методы класса “Объект перехода на следующий уровень”: “Конструктор”, “Поведение объекта”. Метод “Поведение объекта” описывает, как объект взаимодействует с игроком и комнатой.

Класс “Динамический объект” описывает основу динамических объектов, которые имеют здоровье. Основные атрибуты класса “Динамический объект”: “Здоровье”, “Логический: Является игроком?”, “Поворот спрайта”, “Вертикальная скорость”, “Горизонтальная скорость”. Основные методы класса ‘Динамический объект”: “Поворот спрайта”, “Логическая: Можно ли переместиться?”, “Движение”, “Конструктор”, “Дать вертикальную скорость”, “Дать горизонтальную скорость”.

Класс “Динамический объект” является “классом-родителем” для класса “Игрок”. Класс “Игрок”- это игровой персонаж, которым управляет пользователь. Основные методы класса “Игрок”: “Установить скорость”, “Конструктор”, “Система движения”. Метод “Установить скорость” устанавливает скорость в зависимости от кода нажатой клавиши в одно из трёх состояний (ускорение, замедление, ничего).



**Рис.19.** UML-модель “древа классов игровых объектов” с “агрегатами”

**ЗАКЛЮЧЕНИЕ**

Результатом выполнения курсовой работы стало создание приложения на языке C++. Данное приложение работает в системе с предустановленным ПО С++-интерпретатора.

**БИБЛИОГРАФИЯ**

Используемая литература

1. Иванова Г.С. Технология программирования: Учебник для вузов. – 3-е изд., перераб. и доп. / Г.С. Иванова,

Интернет-ресурсы

2. Википедия. Свободная энциклопедия.

2.1 <https://ru.wikipedia.org/wiki/C%2B%2B> С++

2.2 [https://ru.wikipedia.org/wiki/Объектно-ориентированное\_программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) Объектно-ориентированное программирование

**ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ НА ЯЗЫКЕ PYTHON**

1. Сценарий запуска игры. main.cpp

/\* figure CourceWork.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: CourceWork.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Запускает игру \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"System//Game.h"

#include<iostream>

int main() {

classGame Game; // Создание игры

Game.GameStart(); // Запуск игры

return 0;

}

1. Объявление основной системы игры. Game.h

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Game.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classGame \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include<SFML/Graphics.hpp>

/\* classGame \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classGame - класс который является \*/

/\* главным в управлении всей игрой. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classGame {

private:

char\* State;

int Phase;

int timer;

public:

/\* classGame() - конструктор класса \*/

classGame();

/\* GameStart() - метод класса, который \*/

/\* запускает игру. \*/

void GameStart();

/\* ChangeState() - метод класса, который \*/

/\* изменяет \*/

void ChangeState(const char\* \_State);

};

1. Реализация основной системы игры. Game.cpp

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Game.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classGame \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//#include<cstring>

#include"Game.h"

#include"..\\Objects\\OtherObjects\\MainMenu.h"

#include"ControlSystem.h"

#include"..\\LevelSystem\\LevelSystem.h"

#include"GraphicSystem.h"

#include"IdList.h"

#include"..\\Objects\\OtherObjects\\Camera.h"

#include"..\\Objects\\OtherObjects\\SubMenu.h"

#include"..\\LevelSystem\\Location.h"

#include<iostream>

#include<iomanip>

#include<time.h>

classGame::classGame() {

State = new char[6];

State = (char\*)"Start";

Phase = 1;

timer = time(0);

srand(time(0));

rand();

return;

}

void classGame::ChangeState(const char\* \_State) {

Phase = 1;

State = (char\*)\_State;

return;

}

void classGame::GameStart() {

classIdList IdList;

IdList.setIdGame(this);

classControlSystem ControlSystem(&IdList);

sf::RenderWindow Window(sf::VideoMode(640, 480), "Lab General. Shisharin Roman");

IdList.setIdWindow(&Window);

classCamera Camera(&IdList);

Camera.setCoordinate(structPoint(0, 0));

IdList.setIdCamera(&Camera);

classGraphicSystem GraphicSystem(&IdList);

IdList.setIdGraphicSystem(&GraphicSystem);

classMainMenu\* IdMainMenu = nullptr;

classLevelSystem\* IdLevelSystem = nullptr;

classSubMenu\* IdSubMenu = nullptr;

while ((Window.isOpen()) && (strcmp(State,"Exit"))) { // Основной цикл игры

sf::Event event;

if (Window.pollEvent(event)) {

if (event.type == sf::Event::Closed)

Window.close();

}

if (Phase == 1) { // Если состояние только поменялось

if (!strcmp(State, "Start")) {

/\* Удаление лишних объектов \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (IdSubMenu) {

delete IdSubMenu;

IdSubMenu = nullptr;

}

if (IdLevelSystem) {

delete IdLevelSystem;

IdLevelSystem = nullptr;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Camera.setCoordinate(structPoint(0, 0)); //Установка камеры на начальное положение

/\* Создание нужных объектов \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (!IdMainMenu) {

IdMainMenu = new classMainMenu(&IdList);

IdList.setIdMainMenu(IdMainMenu);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ControlSystem.setPhase("Main Menu");

}

if (!strcmp(State, "Game")) {

/\* Удаление лишних объектов \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (IdMainMenu) {

delete IdMainMenu;

IdMainMenu = nullptr;

}

if (IdSubMenu) {

delete IdSubMenu;

IdSubMenu = nullptr;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Создание нужных объектов \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (!IdLevelSystem) {

IdLevelSystem = new classLevelSystem(&IdList);

(\*IdLevelSystem).Start();

IdList.setIdLevelSystem(IdLevelSystem);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ControlSystem.setPhase("Player");

}

if (!strcmp(State, "Pause")) {

/\* Создание нужных объектов \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (!IdSubMenu) {

IdSubMenu = new classSubMenu(&IdList);

IdList.setIdSubMenu(IdSubMenu);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ControlSystem.setPhase("Pause");

}

Phase = 2;

}

if (!strcmp(State, "Game")) {

classLocation\* IdLocation = static\_cast<classLocation\*>(IdList.getIdLocation());

(\*IdLocation).LiveObjects(); // Запуск повеения объектов

}

ControlSystem.KeyboardInput();

Camera.CameraSetMove();

Window.clear();

GraphicSystem.DrawTextures();

Window.display();

}

Window.close();

return;

}

1. Объявление системы управления. ControlSystem.h

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: classControlSystem.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classControlSystem \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

/\* classControlSystem \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classControlSystem - класс, который является \*/

/\* системой управления ввода с клавиатуры. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

class classControlSystem {

private:

void\* vIdList; // Указатель на класс IdList

int KeyCode; // Переменная принимающая код клавиши

const char\* Phase; //

bool ArrowPressed; // Переменная отвечающая за нажатие стрелок

/\* ObjectControl() - метод класса, который \*/

/\* передает управление объектам. \*/

void ObjectControl();

public:

/\* classControlSystem() - конструктор класса, который \*/

/\* принимает значения: указатель на класс IdList \*/

classControlSystem(void\* \_IdList);

/\* KeyboardInput() - метод класса, который \*/

/\* принимает ввод с клавиатуры. \*/

void KeyboardInput();

void setPhase(const char\* \_Phase);

};

1. Реализация системы управления. ControlSystem.cpp

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: ControlSystem.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classControlSystem \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<conio.h>

#include"ControlSystem.h"

#include"..\\Objects\\OtherObjects\\MainMenu.h"

#include"..\\Objects\\GameObjects\\MortalObjects\\Player\\Player.h"

#include"IdList.h"

#include"KeyCodes.h"

#include"..\\Objects\\OtherObjects\\SubMenu.h"

#include"Game.h"

classControlSystem::classControlSystem(void\* \_IdList) {

vIdList = \_IdList;

}

void classControlSystem::setPhase(const char\* \_Phase) {

Phase = \_Phase;

return;

}

void classControlSystem::KeyboardInput() {

KeyCode = 0; // Обнудение кода клавиши

ArrowPressed = false;

if (\_kbhit()) {

KeyCode = \_getch();

if (KeyCode == 224) { // Если были нажаты стрелки

ArrowPressed = true;

KeyCode = \_getch();

}

}

ObjectControl();

return;

}

void classControlSystem::ObjectControl() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

if (!strcmp(Phase, "Main Menu")) { // Если управляемый объект "Главное меню"

classMainMenu\* IdMainMenu = static\_cast<classMainMenu\*>((\*IdList).getIdMainMenu());

/\* Запуск определенных событий в завсимости от кода клавиши \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (ArrowPressed == true) {

(\*IdMainMenu).ChangeState(KeyCode);

}

if (KeyCode == VK\_ENTER) {

(\*IdMainMenu).EnterState();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

if (!strcmp(Phase, "Player")) { // Если управляемый объект "Игрок"

classPlayer\* IdPlayer = static\_cast<classPlayer\*>((\*IdList).getIdPlayer());

(\*IdPlayer).MoveSystem(KeyCode); // Передача кода клавиши

if (KeyCode == VK\_ESCAPE) {

classGame\* IdGame = static\_cast<classGame\*>((\*IdList).getIdGame());

(\*IdGame).ChangeState("Pause");

}

}

if (!strcmp(Phase, "Pause")) { // Если управляемый объект меню паузы

classSubMenu\* IdSubMenu = static\_cast<classSubMenu\*>((\*IdList).getIdSubMenu());

if (ArrowPressed) {

(\*IdSubMenu).ChangeState(KeyCode);

}

if (KeyCode == VK\_ENTER) {

(\*IdSubMenu).EnterState();

}

if (KeyCode == VK\_ESCAPE) {

(\*IdSubMenu).Exit();

}

}

return;

}

1. Объявление графической системы. GraphicSystem.h

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: GraphicSystem.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classGraphicSystem \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include<SFML/Graphics.hpp>

#include"Point.h"

/\* classGraphicSystem \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classGraphicSystem - класс который отвечаеть за \*/

/\* отрисовку спрайтов на экране. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classGraphicSystem {

private:

/\* structSpriteInformation \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* structSpriteInformation - структура в которой \*/

/\* храниться информация о спрайте. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct structSpriteInformation {

structPoint\* Coordinate; // Указатель на координаты спрайта

sf::Sprite\* Sprite;

int Priority; // Приоритет отрисовки спрайта

bool CameraDependent; // Зависит ли отрисовка спрайта от положения камеры

/\* structSpriteInformaation() - конструктор структуры. \*/

structSpriteInformation();

} \*SpriteInformation; // Массив информации о спайтах

int Size;

void\* vIdList; // Указатель на classIdList

bool Sort; // Отсортирован ли массив спрайтов по приоритету отрисовки

/\* SortArray() - метод класса, который сортирует \*/

/\* спрайты по приоритету отриовки. \*/

void SortArray();

public:

/\* classGraphicSystem() - конструктор класса, который \*/

/\* принимает значения: указатель на classIdList. \*/

classGraphicSystem(void\* \_IdList);

/\* putSprite() - метод класса который добавляет спрайт в массив \*/

void putSprite(sf::Sprite\* \_IdSprite, structPoint\* \_CoordinateSprite, int \_Priority = 0, bool \_CameraDependent = true);

/\* putSpriteArray() - метод класса которы добавляет массив \*/

/\* спрайтов в массив. \*/

void putSpriteArray(sf::Sprite\*\* \_IdSpriteArray, structPoint\*\* \_CoordinateSpriteArray, int \_Size, int \_Prioryte = 0, bool \_CameraDependent = true);

/\* deleteSprite() - метод класса который удаляет спрайт \*/

/\* из массива. \*/

void deleteSprite(sf::Sprite\* \_IdSprite);

/\* deleteSpriteArray() - метод класса, который удаляет \*/

/\* массив спрайтов из массива. \*/

void deleteSpriteArray(sf::Sprite\* \_IdSprite, int Size);

/\* DrawTextures() - метод класса, который рисует \*/

/\* спрайты. \*/

void DrawTextures();

};

1. Реализация графической системы. GraphicSystem.cpp

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: GraphicSystem.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description Реализация методов класса classGraphicSystem \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<cstring>

#include"GraphicSystem.h"

#include"..\\Objects\\OtherObjects\\Camera.h"

#include"IdList.h"

classGraphicSystem::structSpriteInformation::structSpriteInformation() {

Sprite = nullptr;

Coordinate = nullptr;

Priority = 0;

CameraDependent = false;

return;

}

classGraphicSystem::classGraphicSystem(void\* \_IdList) {

Size = 0;

SpriteInformation = new structSpriteInformation[40];

vIdList = \_IdList;

return;

}

void classGraphicSystem::putSprite(sf::Sprite\* \_IdSprite, structPoint\* \_CoordinateSprite, int \_Priority, bool \_CameraDependent) {

SpriteInformation[Size].Sprite = \_IdSprite;

SpriteInformation[Size].Coordinate = \_CoordinateSprite;

SpriteInformation[Size].Priority = \_Priority;

SpriteInformation[Size].CameraDependent = \_CameraDependent;

Size++;

Sort = false;

return;

}

void classGraphicSystem::putSpriteArray(sf::Sprite\*\* \_IdSpriteArray, structPoint\*\* \_CoordinateSpriteArray, int \_Size, int \_Priority, bool \_CameraDependent) {

for (int i = 0; i < \_Size; i++) {

SpriteInformation[Size].Sprite = \_IdSpriteArray[i];

SpriteInformation[Size].Coordinate = \_CoordinateSpriteArray[i];

SpriteInformation[Size].Priority = \_Priority;

SpriteInformation[Size].CameraDependent = \_CameraDependent;

Size++;

}

Sort = false;

return;

}

void classGraphicSystem::deleteSprite(sf::Sprite\* \_IdSprite) {

bool DeleteSprite = false;

for (int i = 0; i < Size; i++) {

if (SpriteInformation[i].Sprite == \_IdSprite) {

DeleteSprite = true;

}

if (i + 1 < Size) {

if (DeleteSprite == true) {

SpriteInformation[i] = SpriteInformation[i + 1]; // Смещение спрайтов для "закрытия" "удалённого" спрайта

}

}

}

Sort = false;

Size--;

return;

}

void classGraphicSystem::deleteSpriteArray(sf::Sprite\* \_IdSprite, int \_Size) {

bool DeleteSprite = false;

for (int i = \_Size; i < Size; i++) {

if (SpriteInformation[i - \_Size].Sprite == \_IdSprite) {

DeleteSprite = true;

}

if (DeleteSprite == true) {

SpriteInformation[i - \_Size] = SpriteInformation[i]; // Смещение спрайтов для "закрытия" "удалённых" спрайтов

}

}

Size -= \_Size;

Sort = false;

return;

}

void classGraphicSystem::DrawTextures() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

sf::RenderWindow\* IdWindow = static\_cast<sf::RenderWindow\*>((\*IdList).getIdWindow());

classCamera\* IdCamera = static\_cast<classCamera\*>((\*IdList).getIdCamera());

if (Sort == false) {// Если массив не сортирован

SortArray();

}

/\* Отрисовка спрайтов \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (int i = 0; i < Size; i++) {

if (SpriteInformation[i].CameraDependent) {

(\*SpriteInformation[i].Sprite).setPosition(((\*SpriteInformation[i].Coordinate).X - (\*IdCamera).CameraCoordinate.X), ((\*SpriteInformation[i].Coordinate).Y - (\*IdCamera).CameraCoordinate.Y));

(\*IdWindow).draw(\*(SpriteInformation[i].Sprite));

}

else {

(\*SpriteInformation[i].Sprite).setPosition((\*SpriteInformation[i].Coordinate).X, (\*SpriteInformation[i].Coordinate).Y);

(\*IdWindow).draw(\*(SpriteInformation[i].Sprite));

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return;

}

void classGraphicSystem::SortArray() {

int MinPriority;

int MinElement;

structSpriteInformation TempleSpriteInformation;

/\* Сортировка спрайтов по их приоритету отрисовки \*/

/\* (Спрайты идут в порядке возростания их \*/

/\* приоритетов). \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (int i = 0; i < Size; i++) {

MinPriority = 100;

for (int j = i; j < Size; j++) {

if (MinPriority > SpriteInformation[Size].Priority) {

MinPriority = SpriteInformation[Size].Priority;

MinElement = j;

}

}

TempleSpriteInformation = SpriteInformation[i];

SpriteInformation[i] = SpriteInformation[MinElement];

SpriteInformation[MinElement] = TempleSpriteInformation;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Sort = true; // Массив отсортирован

return;

}

1. Объявление хранилища адресов IdList.h

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: IdList.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classIdList \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

/\* classIdList \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classIdList - класс который хранит \*/

/\* указатели на различные важные \*/

/\* объекты. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct classIdList {

private:

/\* Указатели на важные объекты. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void\* IdGame;

void\* IdLevelSystem;

void\* IdLevel;

void\* IdLocation;

void\* IdPlayer;

void\* IdCamera;

void\* IdGraphicSystem;

void\* IdMainMenu;

void\* IdWindow;

void\* IdSubMenu;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public:

/\* classIdList() - конструктор класса. \*/

classIdList();

/\* Методы класса, которые принимают указатели \*/

/\* на объекты. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setIdGame(void\* Id);

void setIdLevelSystem(void\* Id);

void setIdLocation(void\* Id);

void setIdPlayer(void\* Id);

void setIdCamera(void\* Id);

void setIdMainMenu(void\* Id);

void setIdGraphicSystem(void\* Id);

void setIdWindow(void\* Id);

void setIdLevel(void\* Id);

void setIdSubMenu(void\* Id);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Методы класса, класса которые возвращают \*/

/\* указатели на объекты. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void\* getIdGame();

void\* getIdLevelSystem();

void\* getIdLocation();

void\* getIdPlayer();

void\* getIdCamera();

void\* getIdMainMenu();

void\* getGraphicSystem();

void\* getIdWindow();

void\* getIdGraphicSystem();

void\* getIdLevel();

void\* getIdSubMenu();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

};

1. Реализация хранилища адресов IdList.cpp

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: IdList.сpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classIdList \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"IdList.h"

classIdList::classIdList() {

IdGame = nullptr;

IdLevelSystem = nullptr;

IdLevel = nullptr;

IdLocation = nullptr;

IdPlayer = nullptr;

IdCamera = nullptr;

IdGraphicSystem = nullptr;

IdMainMenu = nullptr;

IdWindow = nullptr;

IdLevel = nullptr;

}

void classIdList::setIdGame(void\* Id) {

IdGame = Id;

return;

}

void classIdList::setIdLevelSystem(void\* Id) {

IdLevelSystem = Id;

return;

}

void classIdList::setIdLocation(void\* Id) {

IdLocation = Id;

return;

}

void classIdList::setIdPlayer(void\* Id) {

IdPlayer = Id;

return;

}

void classIdList::setIdCamera(void\* Id) {

IdCamera = Id;

return;

}

void classIdList::setIdMainMenu(void\* Id) {

IdMainMenu = Id;

return;

}

void classIdList::setIdGraphicSystem(void\* Id) {

IdGraphicSystem = Id;

return;

}

void classIdList::setIdWindow(void\* Id) {

IdWindow = Id;

return;

}

void classIdList::setIdLevel(void\* Id) {

IdLevel = Id;

return;

}

void classIdList::setIdSubMenu(void\* Id) {

IdSubMenu = Id;

return;

}

void\* classIdList::getIdGame() {

return IdGame;

}

void\* classIdList::getIdLevelSystem() {

return IdLevelSystem;

}

void\* classIdList::getIdLocation() {

return IdLocation;

}

void\* classIdList::getIdPlayer() {

return IdPlayer;

}

void\* classIdList::getIdCamera() {

return IdCamera;

}

void\* classIdList::getIdMainMenu() {

return IdMainMenu;

}

void\* classIdList::getGraphicSystem() {

return IdGraphicSystem;

}

void\* classIdList::getIdWindow() {

return IdWindow;

}

void\* classIdList::getIdGraphicSystem() {

return IdGraphicSystem;

}

void\* classIdList::getIdLevel() {

return IdLevel;

}

void\* classIdList::getIdSubMenu() {

return IdSubMenu;

}

1. Объявление системы меню SystemMenu.h

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: SystemMenu.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classSystemMenu \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"GraphicSystem.h"

/\* classSystemMenu \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classSystemMenu - родительский класс который \*/

/\* отвечает за основные действия меню. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classSystemMenu {

private:

void\* vIdList; // Указатель на classIdList

sf::Texture Texture;

int HeightSprite, WightSprite; // Длина и высота спрайта

protected:

sf::Sprite Sprite;

/\* classSystemMenu() - конструктор класса который \*/

/\* принимает значения: указатель на classIdList, \*/

/\* длина и высота спрайта. \*/

classSystemMenu(void\* \_IdList, int \_HeightSprite, int \_WightSprite);

/\* ChangeState() - метод класса который изменяет \*/

/\* выбранное состояние. \*/

void ChangeState(int KeyCode, int& SelectState);

/\* DrawSprite() - метод класса отрисовывание \*/

/\* спрайта. \*/

void DrawSprite(structPoint\* Coordinate, const char\* \_TextureName);

/\* ~classSystemMenu() - деструктор класса. \*/

~classSystemMenu();

};

1. Реализация системы меню. SystemMenu.cpp

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: SystemMenu.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classSystemMenu \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"KeyCodes.h"

#include"IdList.h"

#include"SystemMenu.h"

classSystemMenu::classSystemMenu(void\* \_IdList, int \_WightSprite, int \_HeightSprite) {

vIdList = \_IdList;

HeightSprite = \_HeightSprite;

WightSprite = \_WightSprite;;

return;

}

void classSystemMenu::ChangeState(int KeyCode, int& SelectState) {

if (KeyCode == VK\_UP) {

SelectState -= 1;

}

if (KeyCode == VK\_DOWN) {

SelectState += 1;

}

return;

}

void classSystemMenu::DrawSprite(structPoint\* Coordinate, const char\* \_TextureName) {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classGraphicSystem\* IdGraphicSystem = static\_cast<classGraphicSystem\*>((\*IdList).getGraphicSystem());

sf::Image Image;

Image.loadFromFile(\_TextureName);

Texture.loadFromImage(Image);

Sprite.setTexture(Texture);

Sprite.setTextureRect(sf::IntRect(1, 1, WightSprite, HeightSprite));

(\*IdGraphicSystem).putSprite(&Sprite, Coordinate, 0, false);

return;

}

classSystemMenu::~classSystemMenu() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classGraphicSystem\* IdGraphicSystem = static\_cast<classGraphicSystem\*>((\*IdList).getGraphicSystem());

(\*IdGraphicSystem).deleteSprite(&Sprite);

}

1. Объявление системы уровней. LevelSystem.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: LevelSystem.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classLevelSystem \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classLevelSystem \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classLevelSystem - класс, который является \*/

/\* системой управления уровнями. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classLevelSystem {

private:

void\* vIdList; // Указатель на структуру хранящую адресса

int NumberLevel; // Номер уровня

/\* CreateNewLevel() - метод класса, который \*/

/\* создаёт новый уровень. \*/

void CreateNewLevel();

/\* DeleteLevel() - метод класса, который \*/

/\* удаляет текущий уровень. \*/

void DeleteLevel();

public:

/\* classLevelSystem() - конструктор класса, \*/

/\* который принимает значения: указатель на \*/

/\* структуру хранящую адресса объектов. \*/

classLevelSystem(void\* \_IdList);

/\* Start() - метод класса, который начинает \*/

/\* создавать уровень. \*/

void Start();

/\* NextLevel() - метод класса, который \*/

/\* запускает удаление текущего уровня и \*/

/\* создаёт новый. \*/

void NextLevel();

/\* ~classLevelSustem() - деструктор класса \*/

~classLevelSystem();

};

1. Реализация системы уровня. LevelSystem.cpp

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: LevelSystem.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classLevelSystem \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"LevelSystem.h"

#include"..\\System\\IdList.h"

#include"Location.h"

#include"Level\\Level.h"

void classLevelSystem::CreateNewLevel() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classLevel\* IdLevel = new classLevel(); // Создание нового уровня

(\*IdList).setIdLevel(IdLevel);

(\*IdLevel).StartStructureGeneration(); // Запуск генерации структуры уровня

(\*IdLevel).StartRoomGeneration(); // Запуск генерации комнат уровня

classLocation\* IdLocation = new classLocation(vIdList, "images\\Wall\\Wall.png"); // Создание локации

(\*IdList).setIdLocation(IdLocation);

(\*IdLocation).SpawnPlayer(structPoint(320,120)); // Создание игрока в стартовой комнате

return;

}

void classLevelSystem::DeleteLevel() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classLocation\* IdLocation = static\_cast<classLocation\*>((\*IdList).getIdLocation());

classLevel\* IdLevel = static\_cast<classLevel\*>((\*IdList).getIdLevel());

delete IdLocation; // Удаление объекта просматривоаемого текущий уровень

delete IdLevel; // Удаление текущего уровня

NumberLevel++; // Увеличения номера уровня

return;

}

classLevelSystem::classLevelSystem(void\* \_IdList) {

vIdList = \_IdList;

NumberLevel = 1;

return;

}

void classLevelSystem::Start() {

CreateNewLevel();

return;

}

void classLevelSystem::NextLevel() {

DeleteLevel(); // Удаление текущего уровня

CreateNewLevel(); // Создание нового уровня

return;

}

classLevelSystem::~classLevelSystem() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classLocation\* IdLocation = static\_cast<classLocation\*>((\*IdList).getIdLocation());

classLevel\* IdLevel = static\_cast<classLevel\*>((\*IdList).getIdLevel());

delete IdLocation; // Удаление объекта просматривоаемого текущий уровень

delete IdLevel; // Удаление текущего уровня

return;

}

1. Объявление локации. Location.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Location.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classLocation \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"Level\\Wall\\Wall.h"

#include"ObjInformation.h"

/\* classLocation \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classLocation - класс, который работает с \*/

/\* комнатой из объекта classLevel. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classLocation {

private:

const char\* TextureName; // Путь к текущей текстуры уровня

structPoint PlayerLevelCoordinate; // Координаты игрока на карте уровня

void\* vIdList; // Указатель на структуру хранящую адресса

void\*\* ActiveObjectList;

int NumberActiveObject;

sf::Texture Texture; // Текстура

/\* Информация об комнате \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

structWall\* WallList; // Список стен

int NumberWall; // Кол-во стен

float\* EnemyCounter; // Показатель "враждебности"

structPoint RoomCoordinate; // Координаты комнаты

int RoomType; // Тип комнаты

structObjInformation\* ObjList;

int ObjNumber;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* UpdateLocation() - метод класса, который получает \*/

/\* информацию об комнате по координатам. \*/

void UpdateLocation(structPoint Coordinate);

/\* CreateObjects() - метод класса, который создает \*/

/\* объекты находящиеся в комнате. \*/

void CreateObjects();

/\* DeleteObjects() - метод класса, который удаляет \*/

/\* объекты находящиеся в комнате. \*/

void DeleteObjects();

public:

/\* classLocation() - конструктор класса, который \*/

/\* принимает значения: указатель на класс, хранящий \*/

/\* адресса объектов, и путь к текстуре. \*/

classLocation(void\* \_IdList, const char\* TextureName);

/\* CanObjectMoveToPoint() - метод класса, который \*/

/\* проверяет может объект переместиться в заданную \*/

/\* точку. \*/

/\* ObjPoint1 - левый верхний угол объекта. \*/

/\* ObjPoint2 - правый нижний угол объекта. \*/

int CanObjectMoveToPoint(structPoint ObjPoint1,structPoint ObjPoint2);

/\* MovingToAnotherRoom() - метод класса, который \*/

/\* перемещает игрока в другую комнату по направлению \*/

/\* Direction. \*/

void MovingToAnotherRoom(char Direction);

/\* LiveObjects() - метод класса, который запускает \*/

/\* объекты находящиеся в комнате. \*/

void LiveObjects();

/\* SpawnPlayer() - метод класса, который создает \*/

/\* игрока в заданных координатах. \*/

void SpawnPlayer(structPoint SpawnPoint);

/\* getRoomType() - метод класса, который возвращает \*/

/\* тип комнаты. \*/

int getRoomType();

/\* UpdatePlayerCoordinate() - метод класса, который \*/

/\* обновляет координаты игрока на карте уровня. \*/

void UpdatePlayerCoordinate();

/\* DrawRoom() - метод класса, который передает \*/

/\* информацию, что нужно нарисовать, объекту \*/

/\* classGraphicalSystem. \*/

void DrawRoom();

/\* ~classLocation() - деструктор класса. \*/

~classLocation();

};

1. Реализация локации Location.cpp

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Location.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classLocation \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"Location.h"

#include"Level\\Level.h"

#include"..\\System\\IdList.h"

#include"..\\Objects\\GameObjects\\MortalObjects\\Player\\Player.h"

#include"..\\Objects\\OtherObjects\\Camera.h"

#include"..\\System\\GraphicSystem.h"

#include"..\\Objects\\GameObjects\\ImmortalObjects\\PassNextLevelObject\\PassNextLevel.h"

classLocation::classLocation(void\* \_IdList, const char\* \_TextureName) {

vIdList = \_IdList;

TextureName = \_TextureName;

/\* Получение информации об стартовой комнате \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classLevel\* IdLevel = static\_cast<classLevel\*>((\*IdList).getIdLevel());

UpdateLocation((\*IdLevel).getRoomCoordinate(1));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ActiveObjectList = new void\* [60];

NumberActiveObject = 0;

return;

}

int classLocation::CanObjectMoveToPoint(structPoint ObjPoint1, structPoint ObjPoint2) {

int res = 0;

for (int i = 0; i < NumberWall; i++) {

if (!(((ObjPoint1.Y > WallList[i].Point2.Y) || (ObjPoint2.Y < WallList[i].Point1.Y)) ||

((ObjPoint1.X > WallList[i].Point2.X) || (ObjPoint2.X < WallList[i].Point1.X)))) { // Проверка колизии объекта и стен

if ((WallList[i].door)/\*&& ((\*EnemyCounter) == 0)\*/) { // Если стена является проходом

res = 1; // Стена является проходом

break;

}

else {

res = 2; // Стена является стеной

break;

}

}

}

return res;

}

void classLocation::SpawnPlayer(structPoint SpawnPoint) {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classPlayer\* IdPlayer = new classPlayer(SpawnPoint, vIdList); // Создание игрока

(\*IdList).setIdPlayer(IdPlayer);

PlayerLevelCoordinate = RoomCoordinate; // Обновление координат игрока на карте уровня

return;

}

void classLocation::UpdateLocation(structPoint Coordinate) {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classLevel\* IdLevel = static\_cast<classLevel\*>((\*IdList).getIdLevel());

classGraphicSystem\* IdGraphicSystem = static\_cast<classGraphicSystem\*>((\*IdList).getIdGraphicSystem());

(\*IdGraphicSystem).deleteSpriteArray(&(WallList[0].Sprite), NumberWall); // Удаление спрайтов стен прошлой комнаты

DeleteObjects();

/\* Получение информации об новой комнате \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int IdNewRoom = (\*IdLevel).getNumberRoom(Coordinate); // Номер комнаты

WallList = (\*IdLevel).getRoomWall(IdNewRoom); // Список стен комнаты

NumberWall = (\*IdLevel).getRoomNumberWall(IdNewRoom); // Кол-во стен

EnemyCounter = (\*IdLevel).getRoomEnemyCounter(IdNewRoom); // Показатель "враждебности"

RoomCoordinate = (\*IdLevel).getRoomCoordinate(IdNewRoom); // Координаты комнаты

RoomType = (\*IdLevel).getRoomType(IdNewRoom); // Тип комнаты

ObjList = (\*IdLevel).getObjList(IdNewRoom);

ObjNumber = (\*IdLevel).getObjNumber(IdNewRoom);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CreateObjects();

DrawRoom(); // Создание спрайтов стен следующей комнаты

return;

}

void classLocation::UpdatePlayerCoordinate() {

if (RoomType > 5) { // Только для комнат, которые состоят из более 1 "квадрата"

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classPlayer\* IdPlayer = static\_cast<classPlayer\*>((\*IdList).getIdPlayer());

/\* Изменение координат игрока на карте уровня в зависимости \*/

/\* от его координат в комнате. \*/

if (((\*IdPlayer).Coordinate.Y > 490) && (PlayerLevelCoordinate.Y == RoomCoordinate.Y)) {

PlayerLevelCoordinate.Y++;

}

if (((\*IdPlayer).Coordinate.Y < 470) && (PlayerLevelCoordinate.Y == RoomCoordinate.Y + 1)) {

PlayerLevelCoordinate.Y--;

}

if ((((\*IdPlayer).Coordinate.X > 650) && (PlayerLevelCoordinate.X == RoomCoordinate.X)) ||

(((\*IdPlayer).Coordinate.X > 10) && (PlayerLevelCoordinate.X == RoomCoordinate.X - 1)))

{

PlayerLevelCoordinate.X++;

}

if ((((\*IdPlayer).Coordinate.X < 630) && (PlayerLevelCoordinate.X == RoomCoordinate.X + 1)) ||

(((\*IdPlayer).Coordinate.X < -10) && (PlayerLevelCoordinate.X == RoomCoordinate.X))) {

PlayerLevelCoordinate.X--;

}

}

return;

}

void classLocation::MovingToAnotherRoom(char Direction) {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classPlayer\* IdPlayer = static\_cast<classPlayer\*>((\*IdList).getIdPlayer());

classCamera\* IdCamera = static\_cast<classCamera\*>((\*IdList).getIdCamera());

structPoint NewRoomCoordinate;

int VerEnter = 0;

int HorEnter = 0;

if (Direction == 'd') {

/\* Изменение и запись координат в зависимости \*/

/\* направления. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NewRoomCoordinate.Y = ++PlayerLevelCoordinate.Y;

NewRoomCoordinate.X = PlayerLevelCoordinate.X;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

UpdateLocation(NewRoomCoordinate);

/\* Вычисление разницы координат игрока на карте \*/

/\* уровня и координат комнаты для правильного \*/

/\* перемещения игрока в другую комнату. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

VerEnter = NewRoomCoordinate.Y - RoomCoordinate.Y;

HorEnter = NewRoomCoordinate.X - RoomCoordinate.X;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Перемещение игрока \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

(\*IdPlayer).Coordinate.X = 320 + 640 \* HorEnter;

(\*IdPlayer).Coordinate.Y = 80 + 480 \* VerEnter;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

if (Direction == 'u') {

/\* Изменение и запись координат в зависимости \*/

/\* направления. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NewRoomCoordinate.Y = --PlayerLevelCoordinate.Y;

NewRoomCoordinate.X = PlayerLevelCoordinate.X;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

UpdateLocation(NewRoomCoordinate);

/\* Вычисление разницы координат игрока на карте \*/

/\* уровня и координат комнаты для правильного \*/

/\* перемещения игрока в другую комнату. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

VerEnter = NewRoomCoordinate.Y - RoomCoordinate.Y;

HorEnter = NewRoomCoordinate.X - RoomCoordinate.X;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Перемещение игрока \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

(\*IdPlayer).Coordinate.X = 320 + 640 \* HorEnter;

(\*IdPlayer).Coordinate.Y = 400 + 480 \* VerEnter;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

if (Direction == 'l') {

/\* Изменение и запись координат в зависимости \*/

/\* направления. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NewRoomCoordinate.Y = PlayerLevelCoordinate.Y;

NewRoomCoordinate.X = --PlayerLevelCoordinate.X;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

UpdateLocation(NewRoomCoordinate);

/\* Вычисление разницы координат игрока на карте \*/

/\* уровня и координат комнаты для правильного \*/

/\* перемещения игрока в другую комнату. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

VerEnter = NewRoomCoordinate.Y - RoomCoordinate.Y;

HorEnter = NewRoomCoordinate.X - RoomCoordinate.X;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Перемещение игрока \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

(\*IdPlayer).Coordinate.X = 560 + 640 \* HorEnter;

(\*IdPlayer).Coordinate.Y = 240 + 480 \* VerEnter;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

if (Direction == 'r') {

/\* Изменение и запись координат в зависимости \*/

/\* направления. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NewRoomCoordinate.Y = PlayerLevelCoordinate.Y;

NewRoomCoordinate.X = ++PlayerLevelCoordinate.X;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

UpdateLocation(NewRoomCoordinate);

/\* Вычисление разницы координат игрока на карте \*/

/\* уровня и координат комнаты для правильного \*/

/\* перемещения игрока в другую комнату. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

VerEnter = NewRoomCoordinate.Y - RoomCoordinate.Y;

HorEnter = NewRoomCoordinate.X - RoomCoordinate.X;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Перемещение игрока \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

(\*IdPlayer).Coordinate.X = 80 + 640 \* HorEnter;

(\*IdPlayer).Coordinate.Y = 240 + 480 \* VerEnter;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

(\*IdCamera).setCoordinate(structPoint(0 + 680 \* HorEnter, 0 + 480 \* VerEnter)); // Смещение камеры

return;

}

int classLocation::getRoomType() {

return RoomType;

}

void classLocation::DrawRoom() {

sf::Sprite\*\* SpriteArray = new sf::Sprite \* [NumberWall]; // Создание массива для спрайтов стен

structPoint\*\* CoordinateArray = new structPoint \* [NumberWall]; // Создание масива координат стен

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classGraphicSystem\* IdGraphicSystem = static\_cast<classGraphicSystem\*>((\*IdList).getIdGraphicSystem());

sf::Image Image;

Image.loadFromFile(TextureName);

Texture.loadFromImage(Image);

for (int i = 0; i < NumberWall; i++) {

WallList[i].CreateSprite(&Texture); // Загрузка текстуры стене

SpriteArray[i] = &(WallList[i].Sprite); // Добавление спрайта в массив

CoordinateArray[i] = &(WallList[i].Point1); // Добавление координат в массив

}

(\*IdGraphicSystem).putSpriteArray(SpriteArray, CoordinateArray, NumberWall, 3, true); // Загрузка массива в графическую систему

/\* Удаление массивов \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

delete[] SpriteArray;

delete[] CoordinateArray;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return;

}

void classLocation::CreateObjects() {

for (int i = 0; i < ObjNumber; i++) {

if (!strcmp(ObjList[i].Type,"PassNextLevel")) {

classPassNextLevel\* TempleId = new classPassNextLevel(vIdList,ObjList[i].PointSpawn);

ActiveObjectList[NumberActiveObject] = TempleId; // Запись указателя на объект в массив

NumberActiveObject++;

}

}

return;

}

void classLocation::DeleteObjects() {

for (int i = 0; i < NumberActiveObject; i++) {

if (!strcmp(ObjList[i].Type, "PassNextLevel")) {

classPassNextLevel\* IdObject = static\_cast<classPassNextLevel\*>(ActiveObjectList[i]);

delete IdObject;

}

ActiveObjectList[i] = nullptr;

}

NumberActiveObject = 0;

return;

}

void classLocation::LiveObjects() {

for (int i = 0; i < NumberActiveObject; i++) {

if (!strcmp(ObjList[i].Type, "PassNextLevel")) {

classPassNextLevel\* IdObject = static\_cast<classPassNextLevel\*>(ActiveObjectList[i]);

(\*IdObject).ObjectBehavior(); // Запуск "поведения" объекта

}

}

return;

}

classLocation::~classLocation() {

DeleteObjects();

delete[] ActiveObjectList;

sf::Sprite\*\* SpriteArray = new sf::Sprite \* [NumberWall]; // Создание массива для спрайтов стен

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classGraphicSystem\* IdGraphicSystem = static\_cast<classGraphicSystem\*>((\*IdList).getIdGraphicSystem());

classPlayer\* IdPlayer = static\_cast<classPlayer\*>((\*IdList).getIdPlayer());

for (int i = 0; i < NumberWall; i++) {

SpriteArray[i] = &(WallList[i].Sprite); // Добавление спрайта в массив

}

(\*IdGraphicSystem).deleteSpriteArray(SpriteArray[0], NumberWall);

delete[] SpriteArray;

delete IdPlayer;

return;

}

1. Объявление уровня Level.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Level.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classLevel \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"Wall\\Wall.h"

#include"..\\ObjInformation.h"

/\* classLevel \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classLevel - класс, который хранит всю информацию \*/

/\* об уровне. Также запускает генерацию уровня \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classLevel {

private:

int NumberLevel; // Номер уровня

int RoomNumber; // Кол-во комнат

int\*\* LevelMap; // Карта уровня, отображающая типы комнат

int\*\* LevelMapId; // Карта уровня, отображающая номера комнат

int SizeHor, SizeVer; // Размеры карты

structPoint StartPos; // Координаты начальной комнаты

/\* Переменные для хранения общей информации по каждой комнате \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

structWall\*\* RoomListStructure; // Расположение стен

int\* NumberWallRoom; // Кол-во стен

float\* EnemyCounter; // Показатель "враждебности"

structObjInformation\*\* ObjList;

int\* ObjNumber;

structPoint\* RoomCoordinateList; // Координаты комнаты или верхнего-левого "квадрата", если комната состоит из нескольких квадратов

int\* RoomTypeList; // Тип комнаты

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public:

/\* classLevel() - конструктор класса. \*/

classLevel();

/\* StartStructureGeneration() - метод класса, который \*/

/\* начинает генерировать структуру уровня. \*/

void StartStructureGeneration();

/\* StartRoomGeneration() - метод класса, который начинает \*/

/\* генерировать комнаты. \*/

void StartRoomGeneration();

/\* getRoomType() - метод класса, который возвращает тип \*/

/\* комнаты по номеру комнаты \*/

int getRoomType(int Id);

/\* getRoomWall() - метод класса, который возвращает \*/

/\* информацию об стенах в комнате по номеру комнаты. \*/

structWall\* getRoomWall(int Id);

/\* getRoomCoordinate() - метод класса, который возвращает \*/

/\* координату комнаты по её номеру. \*/

structPoint getRoomCoordinate(int Id);

/\* getRoomEnemyCounter() - метод класса, который \*/

/\* возвращает ссылку на показатель "враждебности" комнаты \*/

/\* по её номеру. \*/

float\* getRoomEnemyCounter(int Id);

/\* getRoomNumberWall() - метод класса, который возвращает \*/

/\* кол-во стен комнаты по номеру комнаты. \*/

int getRoomNumberWall(int Id);

/\* getNumberRoom() - метод класса, который возвращает \*/

/\* номер комнаты по её координатам. \*/

int getNumberRoom(structPoint Point);

structObjInformation\* getObjList(int Id);

int getObjNumber(int Id);

/\* ~classLevel() - деструктор класса. \*/

~classLevel();

};

1. Реализация уровня Level.cpp

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Level.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classLevel \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<cmath>

#include<cstring>

#include"Level.h"

#include"LevelGeneration\\LevelGeneration.h"

using namespace std;

classLevel::classLevel() {

EnemyCounter = nullptr;

LevelMap = nullptr;

LevelMapId = nullptr;

NumberWallRoom = nullptr;

RoomCoordinateList = nullptr;

RoomListStructure = nullptr;

RoomNumber = 0;

RoomTypeList = nullptr;

SizeHor = 0;

SizeVer = 0;

StartPos = structPoint(0, 0);

NumberLevel = 1;

return;

}

void classLevel::StartStructureGeneration() {

classLevelStructureGenerator LevelStructureGenerator(9, 1, &SizeVer, &SizeHor, &StartPos); // Создания генератора структуры уровня

LevelStructureGenerator.GenerateStructure(1); // Запуск генерации структуры уровня

RoomNumber = LevelStructureGenerator.getNumberRoom();

LevelMap = LevelStructureGenerator.getLevelMap();

LevelMapId = LevelStructureGenerator.getLevelMapId();

return;

}

void classLevel::StartRoomGeneration() {

/\* Создание массивов хранящих информацию по каждой комнате \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

RoomListStructure = new structWall \* [RoomNumber];

NumberWallRoom = new int[RoomNumber];

RoomCoordinateList = new structPoint[RoomNumber];

RoomTypeList = new int[RoomNumber];

EnemyCounter = new float[RoomNumber];

ObjList = new structObjInformation \* [RoomNumber];

ObjNumber = new int[RoomNumber];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (int i = 0; i < RoomNumber; i++) {

NumberWallRoom[i] = 0;

ObjNumber[i] = 0;

}

for (int i = 0; i < RoomNumber; i++) {

RoomListStructure[i] = new structWall[20]; // 20 - Макс. возможное кол-во стен в комнате

ObjList[i] = new structObjInformation[30];

}

classRoomGenerator RoomGenerator(LevelMap, LevelMapId, &RoomListStructure, &RoomTypeList, &NumberWallRoom,

&EnemyCounter, &ObjList, &ObjNumber, SizeHor, SizeVer, &StartPos, RoomNumber, NumberLevel, &RoomCoordinateList); // Создание генератора комнат

RoomGenerator.Start(); // Запуск генератора

return;

}

int classLevel::getRoomType(int Id) {

return RoomTypeList[Id - 1];

}

structPoint classLevel::getRoomCoordinate(int Id) {

return RoomCoordinateList[Id - 1];

}

float\* classLevel::getRoomEnemyCounter(int Id) {

return &EnemyCounter[Id - 1];

}

structWall\* classLevel::getRoomWall(int Id) {

return RoomListStructure[Id - 1];

}

int classLevel::getRoomNumberWall(int Id) {

return NumberWallRoom[Id - 1];

}

int classLevel::getNumberRoom(structPoint Point) {

return LevelMapId[Point.Y][Point.X];

}

structObjInformation\* classLevel::getObjList(int Id) {

return ObjList[Id - 1];

}

int classLevel::getObjNumber(int Id) {

return ObjNumber[Id - 1];

}

classLevel::~classLevel() {

for (int i = 0; i < SizeVer; i++) {

delete[] LevelMap[i];

delete[] LevelMapId[i];

}

for (int i = 0; i < RoomNumber; i++) {

delete[] RoomListStructure[i];

delete[] ObjList[i];

}

delete[] LevelMap;

delete[] LevelMapId;

delete[] RoomListStructure;

delete[] EnemyCounter;

delete[] NumberWallRoom;

delete[] RoomCoordinateList;

delete[] RoomTypeList;

delete[] ObjNumber;

return;

}

1. Объявление генерации структуры и комнат уровня LevelGeneration.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: LevelGeneration.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление классов classLevelStructureGenerator и \*/

/\* classRoomGenerator. \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"..\\Wall\\Wall.h"

#include"..\\..\\ObjInformation.h"

/\* classLevelStructureGenerator \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classLevelStructureGenerator - класс в котором генерируется \*/

/\* основа уровня (Строение и типы комнат) \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classLevelStructureGenerator {

private:

/\* CheckCanPlacedRoom() - метод класса, который проверяет может ли \*/

/\* быть размещена комната по координатам X и Y. Если комната состоит \*/

/\* из нескольких "квадратов", также проверяются всевозможные \*/

/\* варианты расположения комнаты. \*/

bool CheckCanPlacedRoom(int Room, int\* Pos);

/\* RoomCreate() - метод класса, который создаёт комнату случайного типа \*/

/\*по координатам. \*/

bool RoomCreate();

/\* NextPosition() - метод класса, который определяет следущую позицию, \*/

/\* где будет поставленна комната. Выбор следующего места работает по \*/

/\* принципу ромба, который постепенно расширяется. Центр ромба - \*/

/\* стартовая координата. \*/

void NextPosition();

/\* GenerateTypeRoom() - метод класса, который управляет методами \*/

/\* RoomCreate() и NextPosition(), создает структуру уровня. \*/

void GenerateTypeRoom();

/\*\*\*\*\*\*\*\*\*\* Методы для сжатия карты \*\*\*\*\*\*\*\*\*\*/

/\*Убирают лишние "нулевые" строки и столбцы \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void IndentifySizeVer(); // Вычисляет новый размер по вертикали

void IndentifySizeHor(); // Вычисляет новый размер по горизонтали

void CompressionMap(); // "Сжимает" карту под новые размеры

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* FindStartRoom() - метод класса, который нахождит координаты стартовой \*/

/\* комнаты уровня. \*/

void FindStartRoom();

public:

int\*\* LevelMap; // Карта уровня, отображающая типы комнат

int\*\* LevelMapId; // Карта уровня, отображающая номера комнат

int\* SizeVer; // Размер "сжатой" карты уровня по вертикали

int\* SizeHor; // Размер "сжатой" карты уровня по горизонтали

structPoint\* StartPos; // Координаты стартовой комнаты

int SizeMap; // Размер сжатой комнаты

int LevelNumber; // Номер уровня

int RandomRangeCreateBossRoom; // Диапазон случайности генерации комнаты с боссом

/\*\*\*\*\*\*\* Переменные для создания разных типов комнат \*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int Center; // Переменная для хранения центра карты ( Стартовой комнаты )

int LimitArtifactRoom;

int RoomNumberNeed; // Сколько нужно поставить комнат (Исключение: комната с артефактом, магазин, комната с боссом)

int LimitShopRoom;

int LimitBossRoom;

int NowIdRoom; // Переменная для создания индефикатора комнат

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Переменные для помощи находения позиции следующей комнаты \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

structPoint NextCoordinate; // Координаты возможной следующей комнаты

int Diagonal;

int DirCenterNext; // Переменная определяющая расстояние от центра карты до угла ромба

char Dir; // Переменная отвечающая за направление поиска места для комнаты

bool RoomPlaced; // Определяет поставлена, ли комната в текущем ромбе (Позволяет увеличить ромб)

bool BossRoomCreate;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Переменные для сжатия карты \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int StartVer, StartHor;

int FinishVer, FinishHor;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classLevelGeneralStructure - конструктор принимающий значения: \*/

/\* размер не "сжатой" карты уровня, номер уровня, ссылки на \*/

/\* будущие размеры "сжатой" карты уровня и ссылку на координаты \*/

/\* будущей стартовой комнаты. \*/

classLevelStructureGenerator(int Size, int LvlNumber, int\* \_SizeVer, int\* \_SizeHor, structPoint\* \_StartPos);

/\* GenerateStructure() - метод класса, который начинает генерацию \*/

/\* карты уровня в зависимости от парраметра RandomTypeGeneration \*/

void GenerateStructure(int RandomTypeGeneration);

/\* Методы возвращающие значения \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int getNumberRoom(); // Вовращает кол-во созданных комнат

int\*\* getLevelMap(); // Возвращает карту уровня, где отображены типы комнат

int\*\* getLevelMapId(); // Возвращает карту уровня, ггде отображены номера комнат

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

};

/\* classRoomGenerator \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classRoomGenerator - класс в котором генерируются \*/

/\* комнаты, т.е. генерация стен, генерация "враждебности", \*/

/\* создание основной информации по комнате. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classRoomGenerator {

private:

int\*\* LevelMap; // Карта уровня, отображающая типы комнат

int\*\* LevelMapId; // Карта уровня, отображающая номера комнат

structWall\*\*\* RoomWallList; // Информация по стенам в каждой комнате уровня

int\*\* RoomTypeList; // Массив хранящий

int\*\* NumberWallOfRoom; // Кол-во стен в каждой комнате

float\*\* EnemyCounter; // Показатель "враждебности" в каждой комнате

int SizeHor, SizeVer; // Размеры карты

structPoint\* StartPos; // Координаты стартовой комнаты

int RoomNumber; // Кол-во комнат

int NumberLevel; // Номер уровня

structPoint\*\* RoomCoordinateList; // Координаты каждой комнаты в уровне. Если комната состоит из нескольких "квадратов", то указывает на верхний-левый квадрат.

structObjInformation\*\*\* ObjList;

int\*\* ObjNumber;

/\* CreateWall() - метод класса, который записывает общую информацию о стене, \*/

/\* т.е "создает" стену. \*/

void CreateWall(structPoint Point1, structPoint Point2, structWall\* RoomWallSystem, bool door, bool breakable,

structPoint Point);

/\* RoomStartSettings() - метод класса, который записывыет общую информацию об \*/

/\* комнате. \*/

void RoomStartSettings();

/\* getRoomType() - метод класса, который возвращает тип комнаты по её \*/

/\* номеру. \*/

int getRoomType(int Id);

/\* FindRoomCoordinate() - метод класса, который возвращает координаты \*/

/\* комнаты по ее номеру. \*/

structPoint FindRoomCoordinate(int Id);

/\* GenerateRoomStructure() - метод класса, который создает в каждой \*/

/\* комнате стены. \*/

void GenerateRoomStructure();

public:

/\* classRoomGenerator() - конструктор класса, который принимает \*/

/\* значения: карта уровня по типу и номерам комнат, ссылку на массив, \*/

/\* который хранит информацию про стены в каждой комнате, типы в \*/

/\* каждой комнате, кол-во стен в каждой комнате, \*/

/\* показатель "враждебноси" для каждой комнаты, ссылку на координаты \*/

/\* стартовой комнаты, кол-во комнат в уровне и номер уровня. \*/

classRoomGenerator(int\*\* \_LevelMap, int\*\* \_LevelMapId, structWall\*\*\* \_RoomWallSystem, int\*\* \_RoomTypeList, int\*\* \_NumberWallOfRoom,

float\*\* \_EnemyCounter, structObjInformation\*\*\* \_ObjList, int\*\* \_ObjNumber,int \_SizeHor, int \_SizeVer, structPoint\* \_StartPos, int \_RoomNumber, int \_NumberLevel, structPoint\*\* \_RoomCoordinateList);

/\* Start() - метод класса, который начинает генерацию комнат \*/

void Start();

};

1. Реализация генерации структуры уровня LevelGeneratorStructure.cpp

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: LevelStructure.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classLevelStructure \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cmath>

#include"LevelGeneration.h"

classLevelStructureGenerator::classLevelStructureGenerator(int RoomNumber, int LvlNumber, int\* \_SizeVer, int\* \_SizeHor, structPoint\* \_StartPos) {

SizeVer = \_SizeVer;

SizeHor = \_SizeHor;

StartPos = \_StartPos;

SizeMap = (RoomNumber - 4) \* 2 + 1; // Вычисление размера не "сжатой" карты уровня

Center = (RoomNumber - 4); // Вычисление центра уровня

LevelNumber = LvlNumber;

/\* Установка начальных значений \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

RandomRangeCreateBossRoom = 8;

LimitArtifactRoom = 1;

RoomNumberNeed = 0;

LimitShopRoom = 1;

LimitBossRoom = 1;

NowIdRoom = 1;

NextCoordinate = structPoint(0, 0);

Diagonal = 0;

DirCenterNext = 1;

Dir = 'r';

RoomPlaced = false;

BossRoomCreate = false;

StartVer = -1;

StartHor = -1;

FinishVer = -1;

FinishHor = -1;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Создание карты \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

LevelMap = new int\* [SizeMap];

LevelMapId = new int\* [SizeMap];

for (int i = 0; i < SizeMap; i += 1) {

LevelMap[i] = new int[SizeMap];

LevelMapId[i] = new int[SizeMap];

for (int j = 0; j < SizeMap; j += 1) {

if ((i == 0) || (j == 0) || (i == SizeMap - 1) || (j == SizeMap - 1)) { // Определение краёв массива

LevelMap[i][j] = -1;

LevelMapId[i][j] = -1;

}

else {

LevelMap[i][j] = 0;

LevelMapId[i][j] = 0;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

LevelMap[Center][Center] = 1; // Установка стартовой комнаты в центре уровня

LevelMapId[Center][Center] = NowIdRoom++; // Установление нач. комнате номера "1"

RoomNumberNeed = RoomNumber;

}

bool classLevelStructureGenerator::CheckCanPlacedRoom(int Room, int\* Pos) {

bool res = false;

int CanPosition = 0;

switch (Room) {

case 2:

if (RoomNumberNeed >= 1) {

\*Pos = 1;

res = true;

}

break;

case 3:

if ((RoomNumberNeed == 0) && (abs(NextCoordinate.Y - Center) + abs(NextCoordinate.X - Center) > LevelNumber) && (LimitArtifactRoom > 0)) {

\*Pos = 1;

res = true;

}

break;

case 4:

if ((LimitShopRoom > 0) && (abs(NextCoordinate.Y - Center) + abs(NextCoordinate.X - Center) >= LevelNumber) && (RoomNumberNeed == 0)) {

\*Pos = 1;

res = true;

}

break;

case 5:

if (rand() % RandomRangeCreateBossRoom == 0) {

if ((LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] != 3) && (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] != 3) && (LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] != 3) && (LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] != 3) &&

(LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] != 4) && (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] != 4) && (LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] != 4) && (LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] != 4)) {

int CounterVoid = 0;

if (LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] <= 0) CounterVoid++;

if (LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] <= 0) CounterVoid++;

if (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] <= 0) CounterVoid++;

if (LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] <= 0) CounterVoid++;

if ((RoomNumberNeed + LimitArtifactRoom + LimitShopRoom == 0) && (CounterVoid == 3)) {

\*Pos = 1;

res = true;

}

}

}

else {

RandomRangeCreateBossRoom--;

}

break;

case 6:

if (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] == 0) CanPosition = 1;

if (LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] == 0) CanPosition = 2;

if ((RoomNumberNeed >= 2) && (CanPosition != 0)) {

\*Pos = CanPosition;

res = true;

}

break;

case 7:

if (LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] == 0) CanPosition = 1;

if (LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] == 0) CanPosition = 2;

if ((RoomNumberNeed >= 2) && (CanPosition != 0)) {

\*Pos = CanPosition;

res = true;

}

break;

case 8:

if ((LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] == 0) && (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] == 0)) CanPosition = 1;

if ((LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] == 0) && (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X - 1] == 0)) CanPosition = 2;

if ((LevelMap[NextCoordinate.Y - 1][NextCoordinate.X + 1] == 0) && (LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] == 0)) CanPosition = 3;

if ((RoomNumberNeed >= 3) && (CanPosition != 0)) {

\*Pos = CanPosition;

res = true;

}

break;

case 9:

if ((LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] == 0) && (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X + 1] == 0)) CanPosition = 1;

if ((LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] == 0) && (LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] == 0)) CanPosition = 3;

if ((LevelMap[NextCoordinate.Y - 1][NextCoordinate.X - 1] == 0) && (LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] == 0)) CanPosition = 4;

if ((RoomNumberNeed >= 3) && (CanPosition != 0)) {

\*Pos = CanPosition;

res = true;

}

break;

case 10:

if ((LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] == 0) && (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X + 1] == 0)) CanPosition = 1;

if ((LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] == 0) && (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] == 0)) CanPosition = 2;

if ((LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] == 0) && (LevelMap[NextCoordinate.Y - 1][NextCoordinate.X - 1] == 0)) CanPosition = 4;

if ((RoomNumberNeed >= 3) && (CanPosition != 0)) {

\*Pos = CanPosition;

res = true;

}

break;

case 11:

if ((LevelMap[NextCoordinate.Y + 1][NextCoordinate.X - 1] == 0) && (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] == 0)) CanPosition = 2;

if ((LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] == 0) && (LevelMap[NextCoordinate.Y - 1][NextCoordinate.X + 1] == 0)) CanPosition = 3;

if ((LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] == 0) && (LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] == 0)) CanPosition = 4;

if ((RoomNumberNeed >= 3) && (CanPosition != 0)) {

\*Pos = CanPosition;

res = true;

}

break;

case 12:

if ((LevelMap[NextCoordinate.Y + 1][NextCoordinate.X + 1] == 0) && (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] == 0) && (LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] == 0)) CanPosition = 1;

if ((LevelMap[NextCoordinate.Y + 1][NextCoordinate.X - 1] == 0) && (LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] == 0) && (LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] == 0)) CanPosition = 2;

if ((LevelMap[NextCoordinate.Y - 1][NextCoordinate.X + 1] == 0) && (LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] == 0) && (LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] == 0)) CanPosition = 3;

if ((LevelMap[NextCoordinate.Y - 1][NextCoordinate.X - 1] == 0) && (LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] == 0) && (LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] == 0)) CanPosition = 4;

if ((RoomNumberNeed >= 4) && (CanPosition != 0)) {

\*Pos = CanPosition;

res = true;

}

break;

}

return res;

}

bool classLevelStructureGenerator::RoomCreate() {

bool res = true;

int RoomCanList[15]; // Массив типов комнаты, которые можно поставить

int RoomCanPos[15]; // Массив позиций, как можно поставить эти комнаты

int i = 0;

if (RoomNumberNeed >= 1) {

RoomCanList[0] = 1; // Заполнение части массива "пустышками"

RoomCanList[1] = 1;

RoomCanList[2] = 1;

}

if (RoomNumberNeed >= 1) {

for (i = 3; i < 5; i++) {

RoomCanList[i] = 2; // Заполнение стандартными маленькими комнатами (Для увеличения шанса выбора)

}

}

for (int j = 2; j < 13; j++) {

if (CheckCanPlacedRoom(j, &RoomCanPos[i])) { // Проверка, может ли комната типа j расположиться в этом месте.

RoomCanList[i++] = j; // Добавление номера типа комнаты, в случае успеха

}

}

int Length = i; // Подсчёт длины массива

if (Length != 0) {

int RandNumb = rand() % Length; // Выбор случайной комнаты

if (RoomCanList[RandNumb] != 1) {

RoomPlaced = true;

}

/\* Добавление комнаты на карту уровня \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Если комната состоит из нескольких "квадратов", то \*/

/\* расматриваеться позиция комнаты, дальше происходит \*/

/\* заполнение "квадратов" в которых будет расположена комната \*/

/\* в карте по типу комнат и карте по номерам комнат \*/

/\* соответствующими данными. Затем уменьшается кол-во \*/

/\* комнат, которые нужно поставить. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

switch (RoomCanList[RandNumb]) {

case 1: // Комната "Пустышка"

res = false;

break;

case 2:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

RoomNumberNeed--;

break;

case 3:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

LimitArtifactRoom--;

break;

case 4:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

LimitShopRoom--;

break;

case 5:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

LimitBossRoom--;

break;

case 6:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

if (RoomCanPos[RandNumb] == 1) {

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X] = NowIdRoom;

}

else {

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X] = NowIdRoom;

}

RoomNumberNeed -= 2;

break;

case 7:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

if (RoomCanPos[RandNumb] == 1) {

LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y][NextCoordinate.X + 1] = NowIdRoom;

}

else {

LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y][NextCoordinate.X - 1] = NowIdRoom;

}

RoomNumberNeed -= 2;

break;

case 8:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

if (RoomCanPos[RandNumb] == 1) {

LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y][NextCoordinate.X + 1] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 2) {

LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y][NextCoordinate.X - 1] = NowIdRoom;

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X - 1] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 3) {

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X + 1] = NowIdRoom;

}

RoomNumberNeed -= 3;

break;

case 9:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

if (RoomCanPos[RandNumb] == 1) {

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X + 1] = NowIdRoom;

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 3) {

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y][NextCoordinate.X + 1] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 4) {

LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y][NextCoordinate.X - 1] = NowIdRoom;

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X - 1] = NowIdRoom;

}

RoomNumberNeed -= 3;

break;

case 10:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

if (RoomCanPos[RandNumb] == 1) {

LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y][NextCoordinate.X + 1] = NowIdRoom;

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X + 1] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 2) {

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y][NextCoordinate.X - 1] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 4) {

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X - 1] = NowIdRoom;

}

RoomNumberNeed -= 3;

break;

case 11:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

if (RoomCanPos[RandNumb] == 2) {

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X - 1] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 3) {

LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y][NextCoordinate.X + 1] = NowIdRoom;

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X + 1] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 4) {

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y][NextCoordinate.X - 1] = NowIdRoom;

}

RoomNumberNeed -= 3;

break;

case 12:

LevelMapId[NextCoordinate.Y][NextCoordinate.X] = NowIdRoom;

LevelMap[NextCoordinate.Y][NextCoordinate.X] = RoomCanList[RandNumb];

if (RoomCanPos[RandNumb] == 1) {

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y][NextCoordinate.X + 1] = NowIdRoom;

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X + 1] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 2) {

LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y + 1][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y][NextCoordinate.X - 1] = NowIdRoom;

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y + 1][NextCoordinate.X - 1] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 3) {

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X + 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y][NextCoordinate.X + 1] = NowIdRoom;

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X + 1] = NowIdRoom;

}

if (RoomCanPos[RandNumb] == 4) {

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMap[NextCoordinate.Y - 1][NextCoordinate.X - 1] = RoomCanList[RandNumb];

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X] = NowIdRoom;

LevelMapId[NextCoordinate.Y][NextCoordinate.X - 1] = NowIdRoom;

LevelMapId[NextCoordinate.Y - 1][NextCoordinate.X - 1] = NowIdRoom;

}

RoomNumberNeed -= 4;

break;

}

if (res == true) {

NowIdRoom++;

}

}

else {

res = false;

}

return res;

}

void classLevelStructureGenerator::NextPosition() {

do {

/\* Поиск координат следующей возмможной \*/

/\* комнаты происходит по типу ромба. \*/

/\* Проверяется направление диогонали, \*/

/\* затем если диагональ не дошла до \*/

/\* предельной длины, то координаты \*/

/\* изменяються в соответствии направдению \*/

/\* диагонали. Если нет, то напровление \*/

/\* диагонали изменяется. \*/

switch (Dir) {

case 'u': // Направление диоганали - вверх и вправо

if (Diagonal < DirCenterNext - 1) {

NextCoordinate.X++;

NextCoordinate.Y--;

Diagonal++;

}

else {

/\* Так как это последняя диагональ в \*/

/\* ромбе, то здесь находиться \*/

/\* условие расширения ромба. \*/

if (RoomPlaced || (BossRoomCreate)) { // Если комната на текущем ромбе поставлена

/\* Координаты устанавливаються над комнатой со старта \*/

/\* на новом расстояние. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NextCoordinate.X = Center;

NextCoordinate.Y = Center - DirCenterNext;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DirCenterNext++; // Увеличение размера ромба

RoomPlaced = false;

}

Diagonal = 1;

Dir = 'r';

}

break;

case 'r': // Направление диоганали - вправо и вниз

if (Diagonal < DirCenterNext) {

NextCoordinate.X++;

NextCoordinate.Y++;

Diagonal++;

}

else {

Diagonal = 1;

Dir = 'd';

}

break;

case 'd': // Направление диоганали - вниз и влево

if (Diagonal < DirCenterNext) {

NextCoordinate.X--;

NextCoordinate.Y++;

Diagonal++;

}

else {

Diagonal = 1;

Dir = 'l';

}

break;

case 'l': // Направление диоганали - лево и вверх

if (Diagonal < DirCenterNext) {

NextCoordinate.X--;

NextCoordinate.Y--;

Diagonal++;

}

else {

Diagonal = 1;

Dir = 'u';

}

break;

}

/\* При определённом размере ромба все следующие \*/

/\* позиции комнат будут за границами массива, \*/

/\* поэтому происходит сброс размера ромба. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (DirCenterNext == SizeMap) {

NextCoordinate.X = Center;

NextCoordinate.Y = Center - DirCenterNext;

Diagonal = 0;

DirCenterNext = 1;

Dir = 'r';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

} while ((NextCoordinate.X < 1) || (NextCoordinate.Y < 1)

|| (NextCoordinate.X > SizeMap - 2) || (NextCoordinate.Y > SizeMap - 2)); // Если координаты являються краями (Или выходят за рамки) массива, то производиться поиск новых координат

return;

}

void classLevelStructureGenerator::GenerateTypeRoom() {

NextCoordinate = structPoint(Center, Center - 1); // Определение координат для первой создаваемой комнаты

while (RoomNumberNeed + LimitArtifactRoom + LimitBossRoom + LimitShopRoom > 0) { // Цикл работает, пока нужное кол-во комнат не поставленно

if ((LevelMap[NextCoordinate.Y][NextCoordinate.X] == 0) &&

((LevelMap[NextCoordinate.Y + 1][NextCoordinate.X] > 0) || (LevelMap[NextCoordinate.Y - 1][NextCoordinate.X] > 0)

|| (LevelMap[NextCoordinate.Y][NextCoordinate.X + 1] > 0)

|| (LevelMap[NextCoordinate.Y][NextCoordinate.X - 1] > 0))) { // Комната создаётся, только в том случае, если текущее место не занято и на соседних "квадратах" есть комнаты

RoomCreate();

}

/\* Установка стартовых значений для генерации комнаты с боссом \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if ((RoomNumberNeed + LimitArtifactRoom + LimitShopRoom == 0) && (BossRoomCreate == false)) {

BossRoomCreate = true;

Diagonal = 0;

DirCenterNext = 1;

Dir = 'r';

RoomPlaced = false;

NextCoordinate = structPoint(Center, Center - 1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NextPosition();

}

}

void classLevelStructureGenerator::IndentifySizeVer() {

bool ExistRoom = false; // Проверяет существует ли комната

for (int i = 0; i < SizeMap; i++) {

/\* Проверка существует ли комната в стобце i \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ExistRoom = false;

for (int j = 0; j < SizeMap; j++) {

if (LevelMap[i][j] > 0) {

ExistRoom = true;

break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if ((StartVer == -1) && (ExistRoom)) {

StartVer = i; // Определение первого ненулевого столбца

}

if ((StartVer != -1) && (!ExistRoom)) {

FinishVer = i - 1; // Определение последного ненулевого столбца

break;

}

}

\*SizeVer = FinishVer - StartVer + 1; // Вычисление вертикального размера "сжатой" комнаты

return;

}

void classLevelStructureGenerator::IndentifySizeHor() {

bool ExistRoom = false; // Проверяет существует ли комната

for (int j = 0; j < SizeMap; j++) {

/\* Проверка существует ли комната в строке j \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ExistRoom = false;

for (int i = 0; i < SizeMap; i++) {

if (LevelMap[i][j] > 0) {

ExistRoom = true;

break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if ((StartHor == -1) && (ExistRoom)) {

StartHor = j; // Определение первой ненулевой строки

}

if ((StartHor != -1) && (!ExistRoom)) {

FinishHor = j - 1; // Определение последней ненулевой строки

break;

}

}

\*SizeHor = FinishHor - StartHor + 1; // Вычисление горизантального размера "сжатой" карты

return;

}

void classLevelStructureGenerator::CompressionMap() {

int i, j;

IndentifySizeVer();

IndentifySizeHor();

/\* Добавление границ в размеры карты \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\*(SizeHor) = (\*SizeVer) + 2;

\*(SizeVer) = (\*SizeHor) + 2;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Создание временных карт \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int\*\* ReserveMap = new int\* [\*(SizeVer)];

int\*\* ReserveMapId = new int\* [\*(SizeVer)];

for (i = 0; i < \*(SizeVer); i++) {

ReserveMap[i] = new int[\*(SizeHor)];

ReserveMapId[i] = new int[\*(SizeHor)];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Копирование из "несжатой" карты во временные карты \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (i = 0; i < \*(SizeVer); i++) {

for (j = 0; j < \*(SizeHor); j++) {

if ((i == 0) || (j == 0) || (i == \*(SizeVer)) || (j == \*(SizeHor))) {

ReserveMap[i][j] = 0;

ReserveMapId[i][j] = 0;

}

else {

ReserveMap[i][j] = LevelMap[i + StartVer - 1][j + StartHor - 1];

ReserveMapId[i][j] = LevelMapId[i + StartVer - 1][j + StartHor - 1];

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Удаление "несжатых" карт \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (i = 0; i < SizeMap; i++) {

delete[] LevelMap[i];

delete[] LevelMapId[i];

}

delete[] LevelMap;

delete[] LevelMapId;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Создание "сжатых" карт и копирование в них комнат из \*/

/\* временных карт. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

LevelMap = new int\* [\*SizeVer];

LevelMapId = new int\* [\*SizeVer];

for (i = 0; i < \*SizeVer; i++) {

LevelMap[i] = new int[\*SizeHor];

LevelMapId[i] = new int[\*SizeHor];

}

for (i = 0; i < \*SizeVer; i++) {

for (j = 0; j < \*SizeHor; j++) {

LevelMap[i][j] = ReserveMap[i][j];

LevelMapId[i][j] = ReserveMapId[i][j];

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Удаление временных карт \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (i = 0; i < \*SizeVer; i++) {

delete ReserveMap[i];

delete ReserveMapId[i];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return;

}

void classLevelStructureGenerator::FindStartRoom() {

bool AllBreak = false;

for (int i = 0; i < \*SizeVer; i++) {

for (int j = 0; j < \*SizeHor; j++) {

if (LevelMap[i][j] == 1) { // Стартовая комната найдена

AllBreak = true;

\*StartPos = structPoint(i, j); // Определение координат стартовой комнаты

break;

}

}

if (AllBreak) {

break;

}

}

return;

}

void classLevelStructureGenerator::GenerateStructure(int RandomTypeGeneration) {

/\* Если RandomTypeGeneration = 1, то запускается \*/

/\* генератор случайной карты. \*/

if (RandomTypeGeneration == 1) {

GenerateTypeRoom();

}

CompressionMap();

}

int classLevelStructureGenerator::getNumberRoom() {

return NowIdRoom;

}

int\*\* classLevelStructureGenerator::getLevelMap() {

return LevelMap;

}

int\*\* classLevelStructureGenerator::getLevelMapId() {

return LevelMapId;

}

1. Реализация генерации комнат уровня. RoomGenerator.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: RoomGenerator.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classRoomGenerator \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"LevelGeneration.h"

#include<cmath>

structPoint classRoomGenerator::FindRoomCoordinate(int Id) {

structPoint Coordinate;

bool AllBreak = false;

for (int i = 0; i < SizeVer; i++) {

for (int j = 0; j < SizeHor; j++) {

if (LevelMapId[i][j] - 1 == Id) { // Поиск комнаты по номеру

Coordinate.X = j; // Запись координат комнаты

Coordinate.Y = i;

AllBreak = true;

break;

}

}

if (AllBreak) {

break;

}

}

return Coordinate;

}

classRoomGenerator::classRoomGenerator(int\*\* \_LevelMap, int\*\* \_LevelMapId, structWall\*\*\* \_RoomWallList, int\*\* \_RoomTypeList, int\*\* \_NumberWallOfRoom,

float\*\* \_EnemyCounter, structObjInformation\*\*\* \_ObjList, int\*\* \_ObjNumber,int \_SizeHor, int \_SizeVer, structPoint\* \_StartPos, int \_RoomNumber, int \_NumberLevel, structPoint\*\* \_RoomCoordinateList) {

LevelMap = \_LevelMap;

LevelMapId = \_LevelMapId;

RoomWallList = \_RoomWallList;

RoomTypeList = \_RoomTypeList;

NumberWallOfRoom = \_NumberWallOfRoom;

EnemyCounter = \_EnemyCounter;

SizeHor = \_SizeHor;

SizeVer = \_SizeVer;

StartPos = \_StartPos;

RoomNumber = \_RoomNumber;

NumberLevel = \_NumberLevel;

RoomCoordinateList = \_RoomCoordinateList;

ObjList = \_ObjList;

ObjNumber = \_ObjNumber;

return;

}

void classRoomGenerator::Start() {

RoomStartSettings();

GenerateRoomStructure();

}

void classRoomGenerator::CreateWall(structPoint Point1, structPoint Point2, structWall\* RoomWallSystem, bool door, bool breakable,

structPoint SpritePoint) {

if (!RoomWallSystem) {

throw "classRoomGenerator::CreateWall() : Invalid Adress";

}

(\*RoomWallSystem).Point1 = Point1;

(\*RoomWallSystem).Point2 = Point2;

(\*RoomWallSystem).door = door; // Определение является ли стена проходом

(\*RoomWallSystem).breakable = breakable; // Определение можно ли сломать стену взрывом (Не реализовано)

(\*RoomWallSystem).setSpriteInformation(SpritePoint, structPoint(abs(Point2.X - Point1.X), abs(Point2.Y - Point1.Y)));

return;

}

void classRoomGenerator::GenerateRoomStructure() {

for (int i = 0; i < RoomNumber; i++) { // Генерация стен для каждой комнаты

/\* Чтобы сгенерировать стены, сначала определяется тип комнаты \*/

/\* Затем рассматриваються соседние "квадраты", в которых могут \*/

/\* быть гипотетические соседние комнаты, если соседняя комната \*/

/\* существует, то создаються две стены и проход, если нет \*/

/\* комнаты, то создаеться одна стена. \*/

if ((\*RoomTypeList)[i] != 0) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y][(\*RoomCoordinateList)[i].X - 1] > 0) {

CreateWall(structPoint(0, 0), structPoint(60, 180), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(0, 300), structPoint(60, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 301));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(0, 180), structPoint(60, 300), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(1, 181));

}

else {

CreateWall(structPoint(0, 0), structPoint(60, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 1));

}

(\*NumberWallOfRoom)[i]++;

if (LevelMapId[(\*RoomCoordinateList)[i].Y - 1][(\*RoomCoordinateList)[i].X] > 0) {

CreateWall(structPoint(0, 0), structPoint(260, 60), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(380, 0), structPoint(640, 60), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(381, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(260, 0), structPoint(380, 60), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(261, 1));

}

else {

CreateWall(structPoint(0, 0), structPoint(640, 60), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 1));

}

(\*NumberWallOfRoom)[i]++;

if (((\*RoomTypeList)[i] != 8) && ((\*RoomTypeList)[i] != 12) && ((\*RoomTypeList)[i] != 7) && ((\*RoomTypeList)[i] != 10)) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y][(\*RoomCoordinateList)[i].X + 1] > 0) {

CreateWall(structPoint(580, 0), structPoint(640, 180), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(581, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(580, 300), structPoint(640, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(581, 301));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(580, 180), structPoint(640, 300), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(581, 181));

}

else {

CreateWall(structPoint(580, 0), structPoint(640, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1225, 1));

}

(\*NumberWallOfRoom)[i]++;

}

if (((\*RoomTypeList)[i] <= 5) || ((\*RoomTypeList)[i] == 7) || ((\*RoomTypeList)[i] == 10)) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y + 1][(\*RoomCoordinateList)[i].X] > 0) {

CreateWall(structPoint(0, 420), structPoint(260, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 421));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(380, 420), structPoint(640, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(381, 421));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(260, 420), structPoint(380, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(261, 421));

}

else {

CreateWall(structPoint(0, 420), structPoint(640, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 421));

}

(\*NumberWallOfRoom)[i]++;

}

if (((\*RoomTypeList)[i] >= 6) && ((\*RoomTypeList)[i] != 7) && ((\*RoomTypeList)[i] != 10)) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y + 2][(\*RoomCoordinateList)[i].X] > 0) {

CreateWall(structPoint(0, 900), structPoint(260, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 421));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(380, 900), structPoint(640, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(381, 421));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(260, 900), structPoint(380, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(261, 421));

}

else {

CreateWall(structPoint(0, 900), structPoint(640, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 421));

}

(\*NumberWallOfRoom)[i]++;

}

if (((\*RoomTypeList)[i] >= 6) && ((\*RoomTypeList)[i] != 7) && ((\*RoomTypeList)[i] != 10) && ((\*RoomTypeList)[i] != 11)) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y + 1][(\*RoomCoordinateList)[i].X - 1] > 0) {

CreateWall(structPoint(0, 480), structPoint(60, 660), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(0, 780), structPoint(60, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 301));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(0, 660), structPoint(60, 780), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(1, 181));

}

else {

CreateWall(structPoint(0, 480), structPoint(60, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 1));

}

(\*NumberWallOfRoom)[i]++;

}

if (((\*RoomTypeList)[i] == 6) || ((\*RoomTypeList)[i] == 8) || ((\*RoomTypeList)[i] == 11)) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y + 1][(\*RoomCoordinateList)[i].X + 1] > 0) {

CreateWall(structPoint(580, 480), structPoint(640, 660), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(581, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(580, 780), structPoint(640, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(581, 301));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(580, 660), structPoint(640, 780), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(581, 181));

}

else {

CreateWall(structPoint(580, 480), structPoint(640, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1225, 1));

}

(\*NumberWallOfRoom)[i]++;

}

if (((\*RoomTypeList)[i] >= 7) && ((\*RoomTypeList)[i] != 9) && ((\*RoomTypeList)[i] != 11)) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y - 1][(\*RoomCoordinateList)[i].X + 1] > 0) {

CreateWall(structPoint(640, 0), structPoint(900, 60), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(1020, 0), structPoint(1280, 60), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(381, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(900, 0), structPoint(1020, 60), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(261, 1));

}

else {

CreateWall(structPoint(640, 0), structPoint(1280, 60), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 1));

}

(\*NumberWallOfRoom)[i]++;

}

if (((\*RoomTypeList)[i] >= 7) && ((\*RoomTypeList)[i] != 9) && ((\*RoomTypeList)[i] != 11)) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y][(\*RoomCoordinateList)[i].X + 2] > 0) {

CreateWall(structPoint(1220, 0), structPoint(1280, 180), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(581, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(1220, 300), structPoint(1280, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(581, 301));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(1220, 180), structPoint(1280, 300), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(581, 181));

}

else {

CreateWall(structPoint(1220, 0), structPoint(1280, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1225, 1));

}

(\*NumberWallOfRoom)[i]++;

}

if (((\*RoomTypeList)[i] == 7) || ((\*RoomTypeList)[i] == 8)) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y + 1][(\*RoomCoordinateList)[i].X + 1] > 0) {

CreateWall(structPoint(640, 420), structPoint(900, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 421));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(1020, 420), structPoint(1280, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(381, 421));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(900, 420), structPoint(1020, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(261, 421));

}

else {

CreateWall(structPoint(640, 420), structPoint(1280, 480), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 421));

}

(\*NumberWallOfRoom)[i]++;

}

if (((\*RoomTypeList)[i] == 9) || ((\*RoomTypeList)[i] == 10) || ((\*RoomTypeList)[i] == 12)) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y + 1][(\*RoomCoordinateList)[i].X + 2] > 0) {

CreateWall(structPoint(1220, 480), structPoint(1280, 660), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(581, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(1220, 780), structPoint(1280, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(581, 301));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(1220, 660), structPoint(1280, 780), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(581, 181));

}

else {

CreateWall(structPoint(1220, 480), structPoint(1280, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1225, 1));

}

(\*NumberWallOfRoom)[i]++;

}

if (((\*RoomTypeList)[i] == 9) || ((\*RoomTypeList)[i] == 10) || ((\*RoomTypeList)[i] == 12)) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y + 2][(\*RoomCoordinateList)[i].X + 1] > 0) {

CreateWall(structPoint(640, 900), structPoint(900, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 421));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(1020, 900), structPoint(1280, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(381, 421));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(900, 900), structPoint(1020, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(261, 421));

}

else {

CreateWall(structPoint(640, 900), structPoint(1280, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 421));

}

(\*NumberWallOfRoom)[i]++;

}

if ((\*RoomTypeList)[i] == 9) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y][(\*RoomCoordinateList)[i].X + 1] > 0) {

CreateWall(structPoint(640, 480), structPoint(900, 540), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(1020, 480), structPoint(1280, 540), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(381, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(900, 480), structPoint(1020, 540), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(261, 1));

}

else {

CreateWall(structPoint(640, 480), structPoint(1280, 540), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 1));

}

(\*NumberWallOfRoom)[i]++;

}

if ((\*RoomTypeList)[i] == 10) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y + 1][(\*RoomCoordinateList)[i].X] > 0) {

CreateWall(structPoint(640, 480), structPoint(700, 660), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(640, 780), structPoint(700, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 301));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(640, 660), structPoint(700, 780), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(1, 181));

}

else {

CreateWall(structPoint(640, 480), structPoint(700, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 1));

}

(\*NumberWallOfRoom)[i]++;

}

if ((\*RoomTypeList)[i] == 11) {

if (LevelMapId[(\*RoomCoordinateList)[i].Y][(\*RoomCoordinateList)[i].X - 1] > 0) {

CreateWall(structPoint(-640, 480), structPoint(-380, 540), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(-260, 480), structPoint(0, 540), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(381, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(-380, 480), structPoint(-260, 540), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(261, 1));

}

else {

CreateWall(structPoint(-640, 480), structPoint(0, 540), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 1));

}

(\*NumberWallOfRoom)[i]++;

if (LevelMapId[(\*RoomCoordinateList)[i].Y + 2][(\*RoomCoordinateList)[i].X - 1] > 0) {

CreateWall(structPoint(-640, 900), structPoint(-380, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 421));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(-260, 900), structPoint(0, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(381, 421));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(-380, 900), structPoint(-260, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(261, 421));

}

else {

CreateWall(structPoint(-640, 900), structPoint(0, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 421));

}

(\*NumberWallOfRoom)[i]++;

if (LevelMapId[(\*RoomCoordinateList)[i].Y + 1][(\*RoomCoordinateList)[i].X - 2] > 0) {

CreateWall(structPoint(-640, 480), structPoint(-580, 660), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 1));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(-640, 780), structPoint(-580, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(1, 301));

(\*NumberWallOfRoom)[i]++;

CreateWall(structPoint(-640, 660), structPoint(-580, 780), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], true, false, structPoint(1, 181));

}

else {

CreateWall(structPoint(-640, 480), structPoint(-580, 960), &(\*RoomWallList)[i][(\*NumberWallOfRoom)[i]], false, false, structPoint(645, 1));

}

(\*NumberWallOfRoom)[i]++;

}

}

}

return;

}

int classRoomGenerator::getRoomType(int Id) {

int RoomType = 0;

bool AllBreak = false;

for (int i = 0; i < SizeVer; i++) {

for (int j = 0; j < SizeHor; j++) {

if (LevelMapId[i][j] - 1 == Id) { // Поиск комнаты по номеру

RoomType = LevelMap[i][j]; // Запись типа комнаты

AllBreak = true;

break;

}

}

if (AllBreak) {

break;

}

}

return RoomType;

}

void classRoomGenerator::RoomStartSettings() {

for (int i = 0; i < RoomNumber; i++) {

(\*RoomTypeList)[i] = getRoomType(i);

(\*RoomCoordinateList)[i] = FindRoomCoordinate(i);

if ((\*RoomTypeList)[i] == 5) {

(\*ObjNumber)[i] = 1;

(\*ObjList)[i][0].PointSpawn = structPoint(320,160);

(\*ObjList)[i][0].Type = "PassNextLevel";

}

else {

(\*ObjNumber)[i] = 0;

}

/\* Генерация показателя "враждебности". "Враждебность" зависит \*/

/\* от номера уровня и размера комнаты. (Размер комнаты \*/

/\* определяется её типом) \*/

if (((\*RoomTypeList)[i] > 1) && ((\*RoomTypeList)[i] != 5)) {

if ((\*RoomTypeList)[i] < 5) {

(\*EnemyCounter)[i] = ((float(5) + float(rand() % 6)) \* NumberLevel) + float(rand() % 7);

}

if (((\*RoomTypeList)[i] > 5) && ((\*RoomTypeList)[i] < 8)) {

(\*EnemyCounter)[i] = ((float(10) + float(rand() % 6)) \* NumberLevel) + float(rand() % 12);

}

if (((\*RoomTypeList)[i] > 7) && ((\*RoomTypeList)[i] < 12)) {

(\*EnemyCounter)[i] = ((float(15) + float(rand() % 6)) \* NumberLevel) + float(rand() % 17);

}

if ((\*RoomTypeList)[i] == 12) {

(\*EnemyCounter)[i] = ((float(20) + float(rand() % 6)) \* NumberLevel) + float(rand() % 22);

}

}

else {

(\*EnemyCounter)[i] = 0;

}

}

return;

}

1. Объявление информации о стене. Wall.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Wall.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление структуры structWall \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include<SFML/Graphics.hpp>

#include"..\\..\\..\\System\\Point.h"

/\* structWall \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* structWall - структура, которая является \*/

/\* объектом стена. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct structWall {

structPoint Point1, Point2; // Координаты левого верхнего и нижнего правого угла

bool door = false;

bool breakable = false;

sf::Sprite Sprite;

/\* structWall() - конструктор структуры \*/

structWall();

/\* setSpriteInformation() - Устанавлиает, где на \*/

/\* плоскости игры будет отрисован спрайт. Также \*/

/\* усанавливает область на текстуре, откуда будет \*/

/\* считываться спрайт. \*/

void setSpriteInformation(structPoint Point1,structPoint Point2);

/\* CreateSprite() - Определение файла текстуры \*/

void CreateSprite(sf::Texture\* Texture);

};

1. Реализация информации о стене. Wall.cpp

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Wall.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление структуры structWall \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include<SFML/Graphics.hpp>

#include"..\\..\\..\\System\\Point.h"

/\* structWall \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* structWall - структура, которая является \*/

/\* объектом стена. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct structWall {

structPoint Point1, Point2; // Координаты левого верхнего и нижнего правого угла

bool door = false;

bool breakable = false;

sf::Sprite Sprite;

/\* structWall() - конструктор структуры \*/

structWall();

/\* setSpriteInformation() - Устанавлиает, где на \*/

/\* плоскости игры будет отрисован спрайт. Также \*/

/\* усанавливает область на текстуре, откуда будет \*/

/\* считываться спрайт. \*/

void setSpriteInformation(structPoint Point1,structPoint Point2);

/\* CreateSprite() - Определение файла текстуры \*/

void CreateSprite(sf::Texture\* Texture);

};

1. Объявление камеры. Camera.h

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Camera.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classCamera \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"..\\..\\System\\Point.h"

/\* classCamera \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classCamera - класс который являеться \*/

/\* камерой в игре. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classCamera {

private:

int vspeed, hspeed; // Вертикальная и горизонтальная скорость

void\* vIdList; // Указатель на объект IdList

/\* CameraMove() - метод класса, который \*/

/\* отвечает за движение камеры. \*/

void CameraMove();

public:

/\* classCamera() - конструктор класса, принимает значения: \*/

/\* указатель на объект IdList. \*/

classCamera(void\* IdList);

structPoint CameraCoordinate; // Координаты камеры

void setCoordinate(structPoint \_CameraCoordinate);

/\* CameraSetMove() - метод класса, который \*/

/\* устаавливает горизонтальную и \*/

/\* вертикальнуюскорость камеры. \*/

void CameraSetMove();

};

1. Реализация камеры Camera.cpp

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Camera.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Ревлизация методов класса classCamera \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<cmath>

#include"Camera.h"

#include"..\\..\\System\\IdList.h"

#include"..\\..\\Objects\\GameObjects\\MortalObjects\\Player\\Player.h"

#include"..\\..\\LevelSystem\\Location.h"

classCamera::classCamera(void\* IdList) {

vIdList = IdList;

return;

}

void classCamera::setCoordinate(structPoint \_CameraCoordinate) {

CameraCoordinate = \_CameraCoordinate;

/\* Обнуление скоростей \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

vspeed = 0;

hspeed = 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return;

}

void classCamera::CameraSetMove() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classPlayer\* IdPlayer = static\_cast<classPlayer\*>((\*IdList).getIdPlayer());

if (IdPlayer) { // Игрок существует

hspeed = 0;

vspeed = 0;

/\* Нахождение расстояния от игрока до центра камеры (По горизонтали \*/

/\* и вертикали). \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int DirectX = CameraCoordinate.X + 320 - (\*IdPlayer).Coordinate.X;

int DirectY = CameraCoordinate.Y + 240 - (\*IdPlayer).Coordinate.Y;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if ((abs(DirectX) > 65) || (abs(DirectY) > 65)) { // Если игрок удалился от центра камеры

classLocation\* IdLocation = static\_cast<classLocation\*>((\*IdList).getIdLocation());

int RoomType = (\*IdLocation).getRoomType();

int PlayerHspeed = (\*IdPlayer).getHspeed();

int PlayerVspeed = (\*IdPlayer).getVspeed();

/\* Установка скоростей камеры в зависимости от направления \*/

/\* движения игрока, типы комнаты и расстояния игрока до \*/

/\* центра карты. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if ((DirectX < 0) && (RoomType > 6) && (PlayerHspeed > 0)) {

hspeed = PlayerHspeed;

}

if ((DirectX > 0) && (RoomType > 6) && (PlayerHspeed < 0)) {

hspeed = PlayerHspeed;

}

if ((DirectY < 0) && ((RoomType == 6) || (RoomType > 7)) && (PlayerVspeed > 0)) {

vspeed = PlayerVspeed;

}

if ((DirectY > 0) && ((RoomType == 6) || (RoomType > 7)) && (PlayerVspeed < 0)) {

vspeed = PlayerVspeed;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

CameraMove();

}

return;

}

void classCamera::CameraMove() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classLocation\* IdLocation = static\_cast<classLocation\*>((\*IdList).getIdLocation());

int RoomType = (\*IdLocation).getRoomType();

if (RoomType > 5) {

/\* Проверка на то, что камера не выйдет за \*/

/\* гроницы допстимой площади (Допустимая \*/

/\* площадь зависит от типа комнаты). \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (RoomType != 11) { // Левая граница допустимой площади

if (CameraCoordinate.X + hspeed < 0) {

hspeed = 0;

CameraCoordinate.X = 0;

}

}

else {

if (CameraCoordinate.X + hspeed < -640) {

hspeed = 0;

CameraCoordinate.X = -640;

}

}

if ((RoomType != 6) && (RoomType != 11)) { // Правая граница допустимой площади

if (CameraCoordinate.X + 640 + hspeed > 1280) {

hspeed = 0;

CameraCoordinate.X = 640;

}

}

else {

if (CameraCoordinate.X + 640 + hspeed > 640) {

hspeed = 0;

CameraCoordinate.X = 0;

}

}

if (CameraCoordinate.Y + vspeed < 0) { // Верхняя граница допустимой площади

vspeed = 0;

CameraCoordinate.Y = 0;

}

if (RoomType != 7) { // Нижняя граница допустимой площади

if (CameraCoordinate.Y + 480 + vspeed > 960) {

vspeed = 0;

CameraCoordinate.Y = 480;

}

}

else {

if (CameraCoordinate.Y + 480 + vspeed > 480) {

vspeed = 0;

CameraCoordinate.Y = 0;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

/\* Изменение координат камеры в зависимости от \*/

/\* от скоростей. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CameraCoordinate.X = CameraCoordinate.X + hspeed;

CameraCoordinate.Y = CameraCoordinate.Y + vspeed;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return;

}

1. Объявление главного меню MainMenu.h

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: MainMenu.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classMainMenu \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"..\\..\\System\\SystemMenu.h"

/\* classMainMenu \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classMainMenu - класс который являеться \*/

/\* объектом главного меню. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classMainMenu : public classSystemMenu {

private:

structPoint Coordinate; // Координаты меню

const char\* State[2]{ // Массив возможных состояний

"Start",

"Exit"

};

int SelectState; // Выбранное состояние

int SizeState; // Колличество состояний

void\* vIdList; // Указатель на classIdList

public:

/\* classMainMenu() - конструктор класса который \*/

/\* принимает значения: указатель на classIdList. \*/

classMainMenu(void\* \_IdList);

/\* ChangeState() - метод класса который изменяет \*/

/\* выбранное состояние. \*/

void ChangeState(int Key);

/\* EnterState() - метод класса который запускает \*/

/\* выбранное состояние. \*/

void EnterState();

};

1. Реализация главного меню. MainMenu.cpp

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: MainMenu.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classMainMenu \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"MainMenu.h"

#include"..\\..\\System\\Game.h"

#include"..\\..\\System\\IdList.h"

#define CLNSCR "\x1B[2J"

classMainMenu::classMainMenu(void\* \_IdList) : classSystemMenu(\_IdList,296,87){

SelectState = 0;

SizeState = 2;

vIdList = \_IdList;

Coordinate = structPoint(320 - 296 / 2, 240 - 87 / 2);

DrawSprite(&Coordinate, "images\\MainMenu\\MainMenu.png");

return;

}

void classMainMenu::ChangeState(int Key) {

classSystemMenu::ChangeState(Key, SelectState);

/\* Если выбранное состояние выходит за границы \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (SelectState + 1 > SizeState) {

SelectState = 0;

}

if (SelectState < 0) {

SelectState = SizeState - 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Sprite.setTextureRect(sf::IntRect(2, 1 + SelectState \* 96, 296, 87));

return;

}

void classMainMenu::EnterState() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classGame\* GameId = static\_cast<classGame\*>((\*IdList).getIdGame());

if (!strcmp(State[SelectState], "Start")) {

(\*GameId).ChangeState("Game"); // Изменение состояния игры

}

if (!strcmp(State[SelectState], "Exit")) {

(\*GameId).ChangeState("Exit"); // Изменение состояния игры

}

return;

}

1. Объявление вспомогательного меню. SubMenu.h

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: SubMenu.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classMainMenu \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"..\\..\\System\\SystemMenu.h"

/\* classSubMenu \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classSubMenu - класс который являеться \*/

/\* подменю во время паузы игры. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classSubMenu : protected classSystemMenu {

private:

void\* vIdList; // Указатель на classIdList

const char\* State[2]{ // Массив возможных состояний

"Continue",

"Exit"

};

int SelectState; // Выбранное состояние

int SizeState; // Колличество состояний

structPoint Coordinate; // Координаты меню

public:

/\* classSubMenu() - конструктор класса который \*/

/\* принимает значения: указатель на classIdList. \*/

classSubMenu(void\* \_IdList);

/\* ChangeState() - метод класса который изменяет \*/

/\* выбранное состояние. \*/

void ChangeState(int KeyCode);

/\* EnterState() - метод класса который запускает \*/

/\* выбранное состояние. \*/

void EnterState();

/\* Exit() - метод класса, который запускает \*/

/\* выход игры в главное меню. \*/

void Exit();

};

1. Реализация вспомогательного меню. SubMenu.cpp

/\* figure.cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: SubMenu.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classMainMenu \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"SubMenu.h"

#include"..\\..\\System\\Game.h"

#include"..\\..\\System\\IdList.h"

void classSubMenu::ChangeState(int KeyCode) {

classSystemMenu::ChangeState(KeyCode, SelectState);

/\* Если выбранное состояние выходит за границы \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (SelectState < 0) {

SelectState = SizeState - 1;

}

if (SelectState + 1 > SizeState) {

SelectState = 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Sprite.setTextureRect(sf::IntRect(2, 1 + SelectState \* 166, 117, 159));

return;

}

classSubMenu::classSubMenu(void\* \_IdList) : classSystemMenu(\_IdList,117,159){

vIdList = \_IdList;

SizeState = 2;

SelectState = 0;

Coordinate = structPoint(100, 100);

DrawSprite(&Coordinate, "images\\SubMenu\\SubMenu.png");

return;

}

void classSubMenu::EnterState() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classGame\* IdGame = static\_cast<classGame\*>((\*IdList).getIdGame());

if (!strcmp(State[SelectState],"Continue")) {

(\*IdGame).ChangeState("Game"); // Изменение состояния игры

}

if (!strcmp(State[SelectState], "Exit")) {

(\*IdGame).ChangeState("Start"); // Изменение состояния игры

}

return;

}

void classSubMenu::Exit() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classGame\* IdGame = static\_cast<classGame\*>((\*IdList).getIdGame());

(\*IdGame).ChangeState("Game"); // Изменение состояния игры

}

1. Объявление игрового объекта. Object.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Object.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classObject \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"..\\..\\System\\GraphicSystem.h"

/\* classObject \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classObject - родительский класс, который \*/

/\* отвечает за маску столкновений и отрисовки \*/

/\* спрайта объекта. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classObject {

private:

/\* CreateSprite() - создание спрайта. \*/

void CreateSprite(const char\* \_NameImage, int \_Priority);

protected:

void\* vIdList; // Указатель на структуру хранящую адресса объектов

struct structHitMask { // Прямоугольная структура взаимодействия с другими объектами

int Length;

int Height;

} HitMask;

sf::Texture Texture;

sf::Sprite Sprite;

/\* classObject() - конструктор класса, который принимает значения: \*/

/\* длина,высота объекта, координаты объекта, путь к текстуре объекта \*/

/\* приоритет отрисовки спрайта, длина,высота спрайта у объекта и \*/

/\* указатель на structIdList. \*/

classObject(int \_Length, int \_Height, structPoint \_Coordinate, const char\* \_TextureName, int \_Priority, int \_LengthSprite, int \_HeghtSprite, void\* \_IdList);

/\* PlayerTouchObject() - мето класса, который \*/

/\* проверяет на столкновения игрока с объектом. \*/

bool PlayerTouchObject();

public:

structPoint Coordinate; // Координаты объекта

structPoint CoordinateSprite; // Координаты, где будет отрисовываться спрайт объекта

/\* ~classObject() - деструктор класса. \*/

~classObject();

};

1. Реализация игрового объекта. Object.cpp

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Object.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classObject \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"..\\..\\System\\IdList.h"

#include"Object.h"

#include"MortalObjects\\Player\\Player.h"

classObject::classObject(int \_LengthHitMask, int \_HeightHitMask, structPoint \_Coordinate, const char\* \_NameImage, int \_Priority, int \_LengthSprite, int \_HeightSprite, void\* \_IdList) {

HitMask.Length = \_LengthHitMask;

HitMask.Height = \_HeightHitMask;

Coordinate = \_Coordinate;

/\* Координа отрисовки спрайта переносятся в левый верхний угол \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CoordinateSprite.X = \_Coordinate.X - (\_LengthSprite / 2);

CoordinateSprite.Y = \_Coordinate.Y - (\_HeightSprite / 2);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

vIdList = \_IdList;

CreateSprite(\_NameImage, \_Priority);

return;

}

void classObject::CreateSprite(const char\* \_NameImage, int \_Priority) {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classGraphicSystem\* IdGraphicSystem = static\_cast<classGraphicSystem\*>((\*IdList).getGraphicSystem());

sf::Image Image;

Image.loadFromFile(\_NameImage);

Texture.loadFromImage(Image);

Sprite.setTexture(Texture);

Sprite.setTextureRect(sf::IntRect(1, 1, HitMask.Length, HitMask.Height));

(\*IdGraphicSystem).putSprite(&Sprite, &CoordinateSprite, \_Priority, true);

return;

}

classObject::~classObject() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classGraphicSystem\* IdGraphicSystem = static\_cast<classGraphicSystem\*>((\*IdList).getGraphicSystem());

(\*IdGraphicSystem).deleteSprite(&Sprite);

return;

}

bool classObject::PlayerTouchObject() {

bool res;

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classPlayer\* IdPlayer = static\_cast<classPlayer\*>((\*IdList).getIdPlayer());

res = false;

/\* Проверка на столкновение \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (!(((\*IdPlayer).Coordinate.X - ((\*IdPlayer).HitMask.Length / 2) > Coordinate.X + HitMask.Length / 2) || ((\*IdPlayer).Coordinate.Y - ((\*IdPlayer).HitMask.Height / 2) > Coordinate.Y + HitMask.Height / 2)

|| ((\*IdPlayer).Coordinate.X + ((\*IdPlayer).HitMask.Length / 2) < Coordinate.X - HitMask.Length / 2) || ((\*IdPlayer).Coordinate.Y + ((\*IdPlayer).HitMask.Height / 2) < Coordinate.Y - HitMask.Height / 2))) {

res = true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return res;

}

1. Объявление статического объекта. StaticObject.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: StaticObject.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classStaticObject \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"..\\..\\Object.h"

class classStaticObject : public classObject {

private:

const char\* PhysicState; // Физическое состояние объекта (Не реализовано)

bool CanMove; // Может ли объект двигаться

int vspeed, hspeed; // Вертикальная и горизонтальная скорость

protected:

/\* classStaticObject() - конструктор класса, который принимает \*/

/\* значения: координаты объекта, длина и шириниа объекта, путь \*/

/\* к текстуре объекта, приоритет отрисовки спрайта, длина и \*/

/\* ширина спрайта объекта, указатель на classIdList, \*/

/\* физическое состояние объекта, можетли объект двигаться. \*/

classStaticObject(structPoint \_Coordinate, int \_Length, int \_Height, const char\* \_TextureName, int \_Priority, int \_LengthSprite, int \_HeightSprite, void\* \_IdList, const char\* \_PhysicState, bool \_CanMove);

/\* Move() - метод класса, который передвигает \*/

/\* объект. \*/

void Move();

};

1. Реализация статического объекта. StaticObject.cpp

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: StaticObject.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classStaticObject \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"..\\..\\..\\..\\System\\IdList.h"

#include"..\\..\\MortalObjects\\Player\\Player.h"

#include"StaticObject.h"

classStaticObject::classStaticObject(structPoint \_Coordinate, int \_Length, int \_Height, const char\* \_TextureName, int \_Priority,int \_LengthSprite, int \_HeightSprite, void\* \_IdList, const char\* \_PhysicState, bool \_CanMove) :

classObject(\_Length, \_Height, \_Coordinate, \_TextureName, \_Priority, \_LengthSprite, \_HeightSprite, \_IdList) {

PhysicState = \_PhysicState;

CanMove = \_CanMove;

return;

}

void classStaticObject::Move() {

Coordinate.X = Coordinate.X + hspeed;

Coordinate.Y = Coordinate.Y + vspeed;

return;

}

1. Объявление динамического объекта. ParentDynamicObject.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: classLiveObject.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classParentDynamicObject \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"..\\..\\..\\..\\System\\GraphicSystem.h"

#include"..\\..\\Object.h"

/\* classLiveObject \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classLiveObject - класс который, является \*/

/\* родительским классом для "живых" объктов. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classLiveObject : public classObject {

private:

int HP = 0; // Здоровье объекта

bool Player = false; // Объект является игроком

int Rotation; // Переменная для изменения поворота спрайта

/\* RotationSprite() - метод класса, который \*/

/\* отвечает за поворот спрайта в зависимости \*/

/\* направления движения. \*/

void RotationSprite();

/\* CheckMove() = метод класса, который \*/

/\* проверяет можно ли переместиться с \*/

/\* имеющиеся скоростями. \*/

void CheckMove();

protected:

float vspeed, hspeed; // Вертикальная и горизонтальная скорость

/\* Move() - метод класса, который производит \*/

/\* движение объекта. \*/

void Move();

public:

/\* classParentDynamicObject() - конструктор класса, принимающий значения: \*/

/\* являеться ли игроком, кол-во здоровья, длина HitMash, высота HitMask, \*/

/\* координаты объекта, путь к спрайту, указатель на структуру хранящую \*/

/\* адресса других объектов. \*/

classLiveObject(bool \_Player, int \_HP, int \_Length, int \_Height, structPoint \_Coordinate,

const char\* \_NameImage, int \_Priority,int \_SpriteLength,int \_SpriteHeight, void\* \_IdList);

/\* getVspeed() - метод класса, который возвращает \*/

/\* вертикальную скорость. \*/

int getVspeed();

/\* getHspeed() - метод класса, который возвращает \*/

/\* горионтальную скорость. \*/

int getHspeed();

};

1. Реализация динамического объекта. ParentDynamicObject.cpp

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: LiveObject.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса \*/

/\* classParentDynamicObject \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<cmath>

#include"ParentDynamicObject.h"

#include"..\\..\\..\\..\\LevelSystem\\Location.h"

#include"..\\..\\..\\..\\System\\IdList.h"

void classLiveObject::RotationSprite() {

if (abs(int(vspeed)) >= abs(int(hspeed))) {

if (int(vspeed) >= 0) {

Rotation = 3; // Поворот спрайта вниз

}

else {

Rotation = 1; // Поворот спрайта вверх

}

}

else {

if (int(hspeed) > 0) {

Rotation = 0; // Поворот спрайта направо

}

else {

Rotation = 2; // Поворот спрайта влево

}

}

Sprite.setTextureRect(sf::IntRect(1, 1 + 37 \* Rotation, HitMask.Length, HitMask.Height));

return;

}

void classLiveObject::Move() {

CheckMove();

RotationSprite();

/\* Изменение координат в зависимости от скорости \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Coordinate.X = Coordinate.X + int(hspeed);

Coordinate.Y = Coordinate.Y + int(vspeed);

CoordinateSprite.X = Coordinate.X - (HitMask.Length / 2);

CoordinateSprite.Y = Coordinate.Y - (HitMask.Height / 2);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Sprite.setPosition(float(Coordinate.X), float(Coordinate.Y));

if (Player) { // Только для объекта являющегося игроком

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classLocation\* IdLocation = static\_cast<classLocation\*>((\*IdList).getIdLocation());

(\*IdLocation).UpdatePlayerCoordinate(); // Обновление координаты игрока на карте уровня

}

return;

}

classLiveObject::classLiveObject(bool \_Player, int \_HP, int \_Length, int \_Height, structPoint \_Coordinate, const char\* \_NameImage, int \_Priority , int \_LengthSprite, int \_HeightSprite,void\* \_IdList) :

classObject(\_Length, \_Height, \_Coordinate, \_NameImage,\_Priority, \_LengthSprite, \_HeightSprite, \_IdList) {

HP = \_HP;

HitMask.Length = \_Length;

HitMask.Height = \_Height;

Player = \_Player;

hspeed = 0;

vspeed = 0;

Rotation = 0;

return;

}

void classLiveObject::CheckMove() {

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classLocation\* IdLocation = static\_cast<classLocation\*>((\*IdList).getIdLocation());

/\* Переменные для создания прямоуголника столкновения \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

structPoint ObjPoint1;

structPoint ObjPoint2;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int CanObjectMove = false;

bool Transition = false; // Возможен переход в другую комнату

bool AllBreak = false;

/\* Проверка движения для горизантольной скорости \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while (hspeed != 0) {

/\* Создание углов прямоугольника с помощью координаты центра \*/

/\* и размера маски столкновения. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ObjPoint1.X = Coordinate.X - (HitMask.Length / 2) + ceil(hspeed);

ObjPoint1.Y = Coordinate.Y - (HitMask.Height / 2);

ObjPoint2.X = Coordinate.X + (HitMask.Length / 2) + ceil(hspeed);

ObjPoint2.Y = Coordinate.Y + (HitMask.Height / 2);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CanObjectMove = (\*IdLocation).CanObjectMoveToPoint(ObjPoint1, ObjPoint2); // Проверка на столкновение со стеной

if (CanObjectMove != 2) { // Если объект не стена

if (CanObjectMove == 1) { // Если стена оказываеться открытой дверью

vspeed = 0;

hspeed = 0;

AllBreak = true;

Transition = true;

}

break;

}

hspeed -= (hspeed / abs(hspeed)); // В случае провала, горизонтальная скорость будет постепенно уменьшаться

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (AllBreak == false) { // Если переход возможен, то данную часть можно пропустить

/\* Проверка движения для вертикальной скорости \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while (vspeed != 0) {

/\* Создание углов прямоугольника с помощью координаты центра \*/

/\* и размера маски столкновения. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ObjPoint1.X = Coordinate.X - (HitMask.Length / 2);

ObjPoint1.Y = Coordinate.Y - (HitMask.Height / 2) + ceil(vspeed);

ObjPoint2.X = Coordinate.X + (HitMask.Length / 2);

ObjPoint2.Y = Coordinate.Y + (HitMask.Height / 2) + ceil(vspeed);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

CanObjectMove = (\*IdLocation).CanObjectMoveToPoint(ObjPoint1, ObjPoint2); // Проверка на столкновение со стеной

if (CanObjectMove != 2) { // Если объект не стена

if (CanObjectMove == 1) { // Если стена оказываеться открытой дверью

vspeed = 0;

hspeed = 0;

Transition = true;

}

break;

}

vspeed -= (vspeed / abs(vspeed)); // В случае провала, вертикальная скорость будет постепенно уменьшаться

}

}

/\* Переход игрока в другую конату \*/

/\* Направление куда игрок пошёл \*/

/\* определяеться за счёт проверки \*/

/\* координат игрока и "предельных" \*/

/\* координат комнаты (Которые помогают \*/

/\* одназначно понять в каком направлении \*/

/\* движется игрок). \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if ((Transition) && (Player)) {

switch ((\*IdLocation).getRoomType()) {

case 1:

case 2:

case 3:

case 4:

case 5:

if (Coordinate.Y < 90) {

(\*IdLocation).MovingToAnotherRoom('u');

break;

}

if (Coordinate.Y > 400) {

(\*IdLocation).MovingToAnotherRoom('d');

break;

}

if (Coordinate.X < 80) {

(\*IdLocation).MovingToAnotherRoom('l');

break;

}

if (Coordinate.X > 560) {

(\*IdLocation).MovingToAnotherRoom('r');

break;

}

break;

case 6:

if (Coordinate.Y < 80) {

(\*IdLocation).MovingToAnotherRoom('u');

break;

}

if (Coordinate.Y > 880) {

(\*IdLocation).MovingToAnotherRoom('d');

break;

}

if (Coordinate.X < 80) {

(\*IdLocation).MovingToAnotherRoom('l');

break;

}

if (Coordinate.X > 560) {

(\*IdLocation).MovingToAnotherRoom('r');

break;

}

break;

case 7:

if (Coordinate.Y < 80) {

(\*IdLocation).MovingToAnotherRoom('u');

break;

}

if (Coordinate.Y > 400) {

(\*IdLocation).MovingToAnotherRoom('d');

break;

}

if (Coordinate.X < 80) {

(\*IdLocation).MovingToAnotherRoom('l');

break;

}

if (Coordinate.X > 1200) {

(\*IdLocation).MovingToAnotherRoom('r');

break;

}

break;

case 8:

if (Coordinate.Y < 80) {

(\*IdLocation).MovingToAnotherRoom('u');

break;

}

if (((Coordinate.X < 640) && (Coordinate.Y > 880)) || ((Coordinate.X > 640) && (Coordinate.Y > 400))) {

(\*IdLocation).MovingToAnotherRoom('d');

break;

}

if (Coordinate.X < 80) {

(\*IdLocation).MovingToAnotherRoom('l');

break;

}

if (((Coordinate.Y < 480) && (Coordinate.X > 1200)) || ((Coordinate.Y > 480) && (Coordinate.X > 560))) {

(\*IdLocation).MovingToAnotherRoom('r');

break;

}

break;

case 9:

if (((Coordinate.X < 640) && (Coordinate.Y < 80)) || ((Coordinate.X > 640) && (Coordinate.Y < 680))) {

(\*IdLocation).MovingToAnotherRoom('u');

break;

}

if (Coordinate.Y > 880) {

(\*IdLocation).MovingToAnotherRoom('d');

break;

}

if (Coordinate.X < 80) {

(\*IdLocation).MovingToAnotherRoom('l');

break;

}

if (((Coordinate.Y < 480) && (Coordinate.X > 560)) || ((Coordinate.Y > 480) && (Coordinate.X > 1190))) {

(\*IdLocation).MovingToAnotherRoom('r');

break;

}

break;

case 10:

if (Coordinate.Y < 90) {

(\*IdLocation).MovingToAnotherRoom('u');

break;

}

if (((Coordinate.X < 640) && (Coordinate.Y > 400)) || ((Coordinate.X > 640) && (Coordinate.Y > 870))) {

(\*IdLocation).MovingToAnotherRoom('d');

break;

}

if (((Coordinate.Y < 480) && (Coordinate.X < 80)) || ((Coordinate.Y > 480) && (Coordinate.X < 720))) {

(\*IdLocation).MovingToAnotherRoom('l');

break;

}

if (Coordinate.X > 1200) {

(\*IdLocation).MovingToAnotherRoom('r');

break;

}

break;

case 11:

if (((Coordinate.X > 0) && (Coordinate.Y < 80)) || ((Coordinate.X < 0) && (Coordinate.Y < 570))) {

(\*IdLocation).MovingToAnotherRoom('u');

break;

}

if (Coordinate.Y > 880) {

(\*IdLocation).MovingToAnotherRoom('d');

break;

}

if (((Coordinate.Y < 480) && (Coordinate.X < 80)) || ((Coordinate.Y > 480) && (Coordinate.X < -550))) {

(\*IdLocation).MovingToAnotherRoom('l');

break;

}

if (Coordinate.X > 560) {

(\*IdLocation).MovingToAnotherRoom('r');

break;

}

break;

case 12:

if (Coordinate.Y < 80) {

(\*IdLocation).MovingToAnotherRoom('u');

break;

}

if (Coordinate.Y > 880) {

(\*IdLocation).MovingToAnotherRoom('d');

break;

}

if (Coordinate.X < 80) {

(\*IdLocation).MovingToAnotherRoom('l');

break;

}

if (Coordinate.X > 1200) {

(\*IdLocation).MovingToAnotherRoom('r');

break;

}

break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

return;

}

int classLiveObject::getHspeed() {

return int(hspeed);

}

int classLiveObject::getVspeed() {

return int(vspeed);

}

1. Объявление объекта перехода на следующий уровень. PassNextLevel.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: PassNextLevel.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classPassNextLevel \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"..\\Parent\\StaticObject.h"

/\* classPassNextLevel \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classPassNextLevel - класс, который являеться \*/

/\* объектом перехода на новый уровень. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classPassNextLevel : protected classStaticObject {

public:

/\* classPassNextLevel - конструктор класса, который \*/

/\* принимает значения: указатель на classIdList и \*/

/\* координаты своего нахождения. \*/

classPassNextLevel(void\* \_IdList, structPoint \_Coordinate);

/\* ObjectBehavior() - метод класса, отвечающий \*/

/\* за "поведение" объекта. \*/

virtual void ObjectBehavior();

};

1. Реализация объекта перехода на следующий уровень. PassNextLevel.cpp

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: PassNextLevel.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classPassNextLevel \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"PassNextLevel.h"

#include"..\\..\\..\\..\\System\\IdList.h"

#include"..\\..\\..\\..\\LevelSystem\\LevelSystem.h"

classPassNextLevel::classPassNextLevel(void\* \_IdList, structPoint \_Coordinate) :

classStaticObject(\_Coordinate,30,30,"images\\StaticObjects\\PassNextLevelObj.png",3,40,40,\_IdList,"Static",false) {

return;

}

void classPassNextLevel::ObjectBehavior() {

if (PlayerTouchObject()) { // Если игрок коснулся данного объекта

classIdList\* IdList = static\_cast<classIdList\*>(vIdList);

classLevelSystem\* IdLevelSystem = static\_cast<classLevelSystem\*>((\*IdList).getIdLevelSystem());

(\*IdLevelSystem).NextLevel(); // Переход на следующий уровень

}

return;

}

1. Объявление игрового персонажа. Player.h

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Player.h \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Объявление класса classPlayer \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#pragma once

#include"..\\Parent\\ParentDynamicObject.h"

/\* classPlayer \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* classPlayer - класс, который являеться \*/

/\* игроком. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class classPlayer : public classLiveObject {

private:

/\* SetSpeed() - метод класса, который \*/

/\* отвечает за изменение скорости в \*/

/\* зависимости парраметра Action. \*/

void SetSpeed(int Action, float \*Speed);

public:

/\* classPlayer() - конструктор класса, \*/

/\* который принимает значения: \*/

/\* координаты появления и указатель на \*/

/\* класс хранящий адресса объектов. \*/

classPlayer(structPoint \_Coordinate, void\* \_IdList);

/\* MoveSystem() - метод класса, который \*/

/\* управляет передвижением игрока. \*/

void MoveSystem(int KeyCode = 0);

void Debug();

~classPlayer();

};

1. Реализация игрового персонажа. Player.cpp

/\* figure .cpp \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Filename: Player.cpp \*/

/\* Abstract: This is a simple C++ program \*/

/\* Description: Реализация методов класса classPlayer \*/

/\* Create Date: 2020 / 04 / 13 \*/

/\* Author: Shisharin Roman Alexandrovich \*/

/\* Notes / Platform / Copyright UNIX/Linux, Freeware \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include"Player.h"

#include"..\\..\\..\\..\\System\\KeyCodes.h"

#include"..\\..\\..\\..\\LevelSystem\\Location.h"

#define MaxSpeed 13

classPlayer::classPlayer(structPoint \_Coordinate, void\* \_IdList) :

classLiveObject(true, 3, 30, 30, \_Coordinate, "images\\Player\\Player.png", 2, 30, 30, \_IdList) {

return;

}

void classPlayer::SetSpeed(int Action, float\* Speed) {

switch (Action) {

case 0: // Замедление

if (abs(\*Speed) > 0.5) {

\*Speed -= (\*Speed / abs(\*Speed));

}

break;

case 1: // Ускорение по направлениям вниз или вправо

if (\*Speed < 0) { // Если движение было не в ту сторону

\*Speed += 4;

}

else {

if (\*Speed < MaxSpeed) { // Ограничение скорости

\*Speed += 3;

}

else {

\*Speed = MaxSpeed;

}

}

break;

case 2: // Ускорение по направлениям вверх и влево

if (\*Speed > 0) { // Если движение было не в ту сторону

\*Speed -= 4;

}

else {

if (\*Speed > -MaxSpeed) { // Ограничение скорости

\*Speed -= 2;

}

else {

\*Speed = -MaxSpeed;

}

}

break;

}

return;

}

void classPlayer::MoveSystem(int KeyCode) {

switch (KeyCode) {

case 0: // Остановка движения

SetSpeed(0, &hspeed);

SetSpeed(0, &vspeed);

break;

case VK\_UP: // Движение вверх (Горизонтальная скорость обнуляется)

hspeed = 0;

SetSpeed(2, &vspeed);

break;

case VK\_DOWN: // Движение вниз (Горизонтальная скорость обнуляется)

hspeed = 0;

SetSpeed(1, &vspeed);

break;

case VK\_LEFT: // Движение влево (Вертикальная скорость обнуляется)

SetSpeed(2, &hspeed);

vspeed = 0;

break;

case VK\_RIGHT: // Движение вправо (Вертикальная скорость обнуляеться)

SetSpeed(1, &hspeed);

vspeed = 0;

break;

}

Move(); // Запуск движения

system("cls");

return;

}

classPlayer::~classPlayer() {

return;

}