Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

ДИСЦИПЛИНА ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНЫХ СИСТЕМ

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:
Пошаговая 2D РПГ Игра «Dungeons&Monsters»

Студент ИУ6-52 _____ **А.С. Бурлаков**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ **А.А. Шайхутдинов**
(Подпись, дата) (И.О.Фамилия)

2018 г.

РЕФЕРАТ

Расчетно-пояснительная записка __ с., _ рис., _ источника(ов).

Объектом разработки является клиент для игровой программы «Пошаговая 2D РПГ Игра «Dungeons&Monsters»» (в дальнейшем – игра) и средства для расчета взаимодействий между игровыми объектами.

Цель работы – изучение разработки игрового движка и принципов игрового проектирования, а также изучение принципов работы простейшего искусственного интеллекта.

Поставленная цель достигается за счет средств, предоставляемых языком программирования C++ стандарта C++11 и библиотеки Qt. В качестве среды для разработки используется Qt creator 5.7. Разрабатываемая игра поддерживает функцию сохранения и загрузки игрового процесса, а также процедурной генерации подземелья.

Ключевые слова – C/C++, Qt creator, игра, пошаговая РПГ, 2D.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время компьютерные игры очень популярны. Чаще всего для создания компьютерных игр используют специализированные средства и платформы, такие как unreal engine, unity и т.д., однако опыт в написании движка к игре на «чистом» языке очень полезен. В данной работе производится проектирование компьютерной игры в жанре РПГ на языке C++ стандарта C++11 с использованием библиотек Qt.

Разработка такой игры является актуальной, так как игры подобного жанра весьма популярны и потребность в новых представителях не исчезает до сих пор. Возможность исследовать игровой лабиринт с монстрами с хорошим искусственным интеллектом позволит увеличить популярность данного продукта.

Для данной игры существует аналог – мобильная игра pixel dungeon, однако так как она предназначена для мобильного телефона, то это не очень удобно с точки зрения размера экрана и возможности модификации. Разработанная игра поддерживает возможность модификации, и используется на компьютере, что повышает удобность её использования.

ОСНОВНАЯ ЧАСТЬ

1 Анализ требований и уточнение спецификаций

1.1 Исследование предметной области

Компьютерная игра – это компьютерная программа, служащая для организации игрового процесса (геймплея), связи с партнёрами по игре, или сама выступающая в качестве партнёра[1]. Чаще всего в играх присутствуют главное меню, внутриигровое меню, сама игра, функции сохранения и загрузки и сам игровой процесс.

Игровой процесс – компонент игры, который отвечает за интерактивное взаимодействие игры и игрока. Он описывает, как игрок взаимодействует с игровым миром, как он реагирует на действия игрока и как определяется набор действий, которые в дальнейшем предлагает игроку игра.

Жанр РПГ (или ролевая игра) – жанр компьютерных игр, основанный на элементах игрового процесса традиционных настольных игровых игр. В ролевой игре игрок управляет одним или несколькими персонажами, которые обладают определенными наборами характеристик, например, здоровье, урон от атаки, защита, шанс уклонения, шанс критического удара и т.д..

2D игра – игра, действия которой располагаются в двухмерной плоскости координат x, y с видом на игровой мир “сверху”.

2D Игры жанра РПГ сегодня достаточно популярны и способны «захватить» человека на долгое время. В дальнейшем название курсового проекта будет обозначаться понятием “проект”.

1.2 Выбор технологии, языка и среды разработки

Технологией программирования называется совокупность методов и средств, которые используются в процессе разработки программного обеспечения. Для разработки проекта принято решение об использовании

спиральной схемы разработки, так как она обладает следующими преимуществами:

- возможность поэтапной реализации запланированных функций программы;
- возможность уточнения ТЗ в процессе разработки;
- возможность изменения программного кода в любой итерации путём удаления ненужных функций программы или добавления новых.

Для написания программы необходимо выбрать язык программирования и среду разработки. Для разработки программы принято решение использовать объектно-ориентированный подход, так как его использование позволяет уменьшить связность кода и его количество. Кроме того, выбор данного подхода обусловлен тем, что используемая в разработке библиотека использует тот же подход.

В качестве языка выбран язык C++. Данный выбор обусловлен следующими причинами:

- C++ - достаточно мощный язык и имеет средства для создания эффективных программ любого назначения;
- Язык поддерживает выбранный подход программирования;
- Язык обеспечивает раздельную компиляцию модулей, прост в использовании.

Для создания данного программного продукта выбрана среда Qt, так как он обладает достаточно дружественным и удобным интерфейсом, в нём предусмотрены средства для отладки программы. Кроме того удобным является то, что Qt обладает уникальной системой сигналов и слотов, которые впоследствии будут использоваться в проекте.

1.3 Построение диаграммы вариантов использования

Согласно техническому заданию игра является однопользовательской, следовательно, единственным пользователем данного программного обеспечения является игрок.

Основными аспектами взаимодействия пользователя с игрой являются:

1) Выбор действий из главного меню;
2) Изменение состояний игры путём нажатия клавиш;
3) Выбор действий из внутриигрового меню.

Помимо этого следует учитывать, что внутриировое меню вызывается непосредственно из запущенной игры. Исходя из этих рассуждений, получена диаграмма вариантов использования (рисунок 1.1).

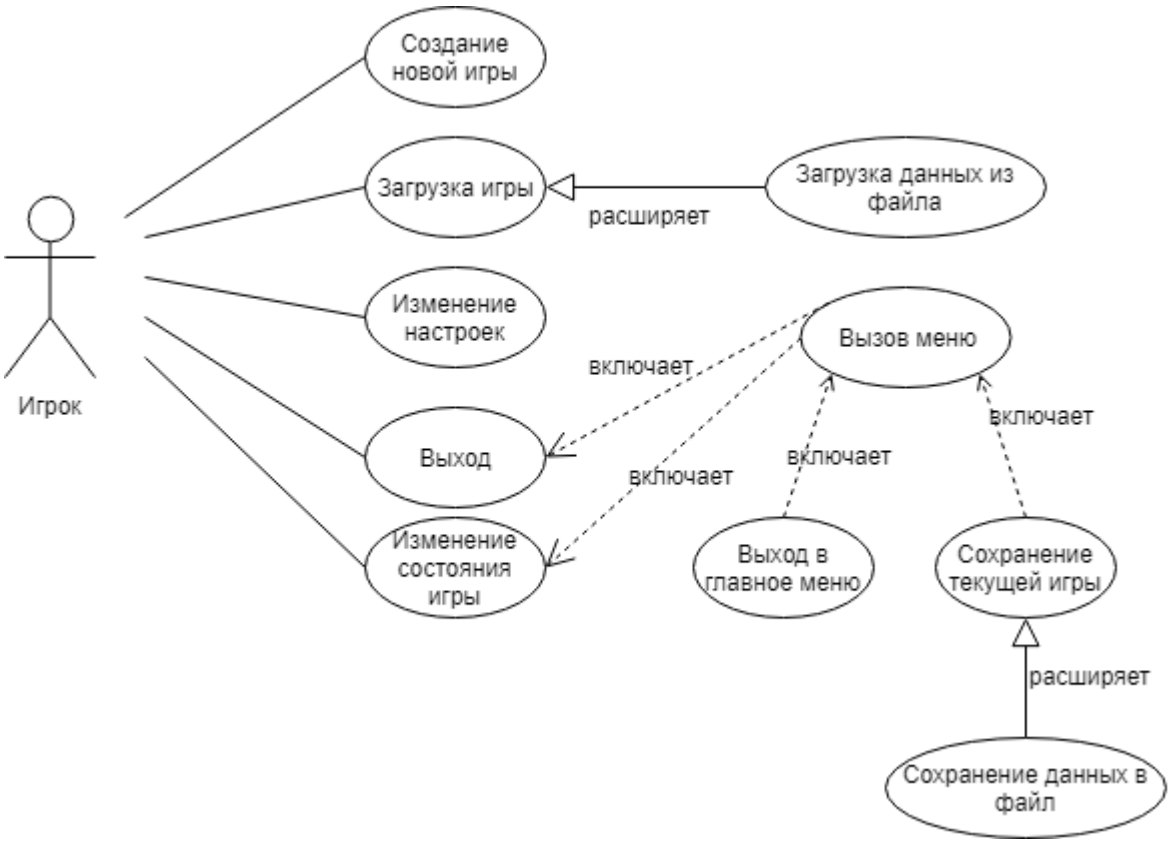

Рисунок 1.1 – Диаграмма вариантов использования

1.4 Разработка концептуальной модели предметной области

В концептуальную модель заданной предметной области входят: игрок, компьютер, игра, уровень, игровое поле, игровая ячейка, игровой объект, герой, шаблонизированный монстр. Концептуальная модель предметной области представлена на рисунке 1.2.

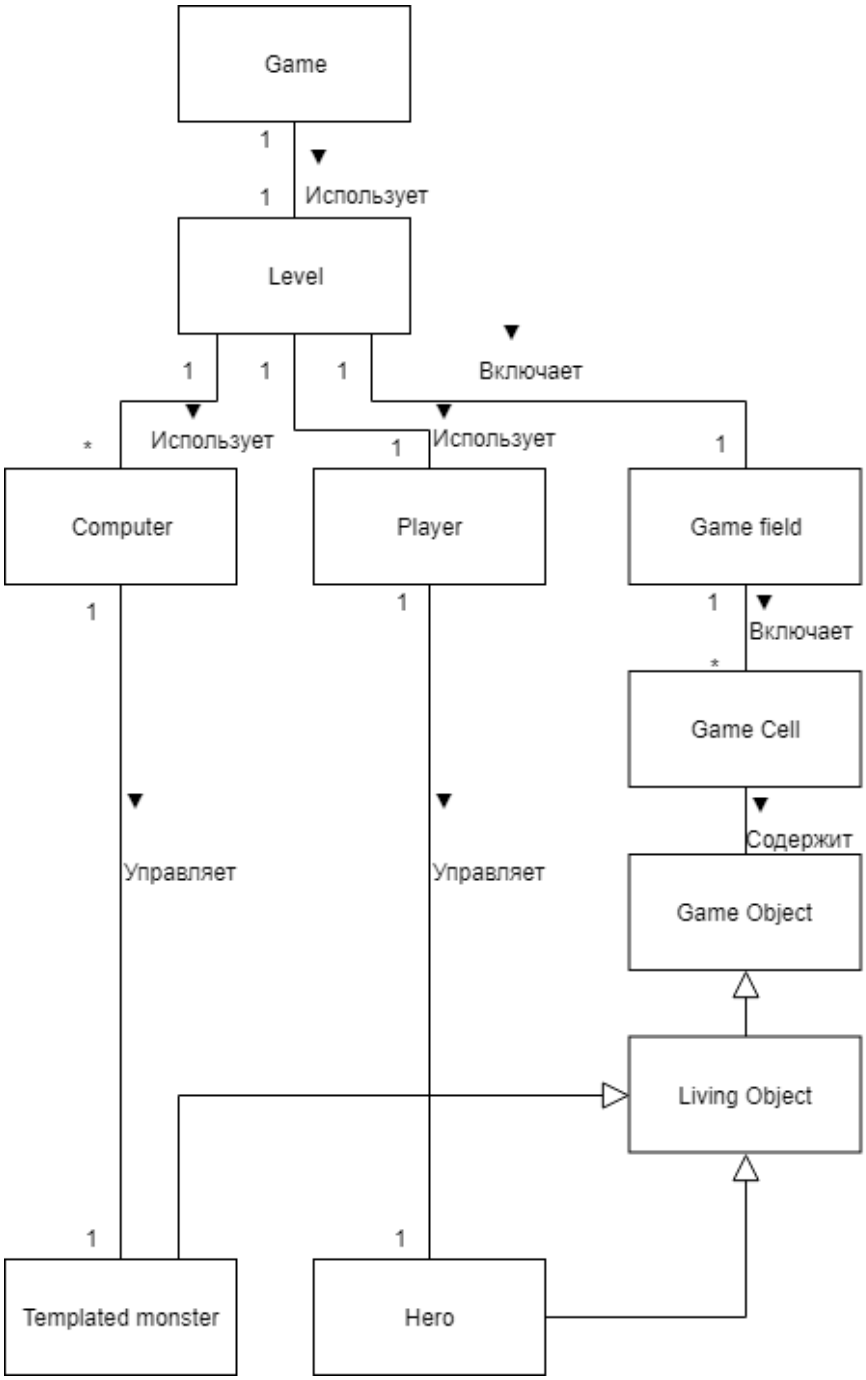

Рисунок 1.2 – Концептуальная модель предметной области

Основная задумка игры заключается в том, что любой игровой объект привязан к конкретной клетке игрового поля, в данном проекте называемой ячейкой. Так, допустим, перемещение игрового объекта будет выполняться следующим образом: информация о наличии в текущей ячейке будет удалена для ячейки, и добавлена в ту ячейку, в которую игровой объект переходит.

Главным классом предметной области является класс Game, который будет создавать новые уровни, загружать и сохранять их при необходимости, помимо этого, отрисовывать главное меню.

Класс Level включает в себя класс игрового поля Game field, в котором содержится информация обо всех ячейках. Помимо этого данный класс будет обрабатывать действия игрока и компьютера, поэтому показа связь использования им классов Player и Computer, которые, в свою очередь, управляют привязанным к ним героем и монстром, соответственно.

Класс Game field, в свою очередь, будет включать в себя матрицу ячеек и методы их создания и изменения их состояний.

Класс Game cell содержит информацию обо всех игровых объектах, находящихся в данной ячейке.

Класс Game Object – основной класс объектов, от него впоследствии будут наследоваться все виды объектов и сами объекты. Так, например, в данном проекте от этого класса наследуется класс LivingObject, который содержит поля и методы, необходимые любому живому объекту – здоровье, урон, модель.

2 Проектирование структуры и компонентов программного продукта

2.1 Разработка интерфейса

Интерфейс разрабатываемой игры предназначен для удобства включения, загрузки, сохранения, постановки на паузы игры и выхода из неё. Исходя из этого выбран процедурно-ориентированный тип пользовательского интерфейса, а именно меню.

Для игры принято решение создать главное и внутриигровое меню. Главное меню предоставляет пользователю следующие варианты выбора: начать новую игру, загрузить игру, изменить настройки окна и выйти из игры. Внутриигровое меню предоставляет следующие опции: продолжить игру, сохранить игру, выйти в главное меню, выйти из игры.

Помимо меню в качестве интерфейса можно выделить следующие окна: сама игра, окно загрузки и окно сохранения.

На основании предпроектных исследований и изучении работы программы разработан интерфейс, соответствующий графу состояний интерфейса, представленному на рисунке 2.1.

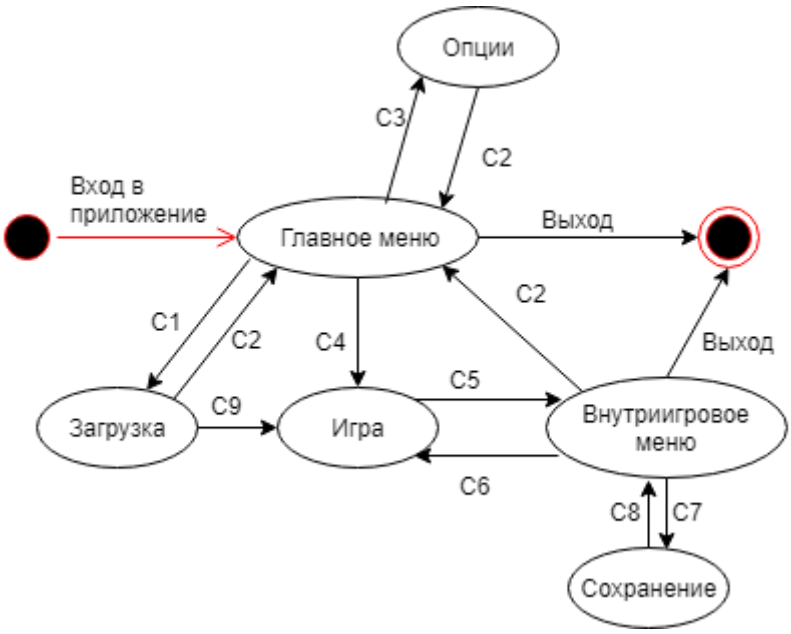

Рисунок 2.1 – граф состояний интерфейса

Обозначение на графе состояний интерфейса:

С1 – Нажатие на кнопку «Загрузка»;

С2 – Нажатие на кнопку «Главное меню»;

С3 – Нажатие на кнопку «Опции»;

С4 – Нажатие на кнопку «Новая игра»;

С5 – Нажатие на кнопку клавиатуры «ESC»;

С6 – Нажатие на кнопку «Продолжить игру»;

С7 – Нажатие на кнопку «Сохранить»;

С8 – Выбор слота сохранения;

С9 – Выбор слота загрузки.

2.2 Реализация предметной области

В ходе разработки программного продукта модель предметной области претерпела определенные изменения – детализированы некоторые классы, а также добавлено несколько новых классов. Полученная в результате иерархия классов представлена на рисунке 2.2.

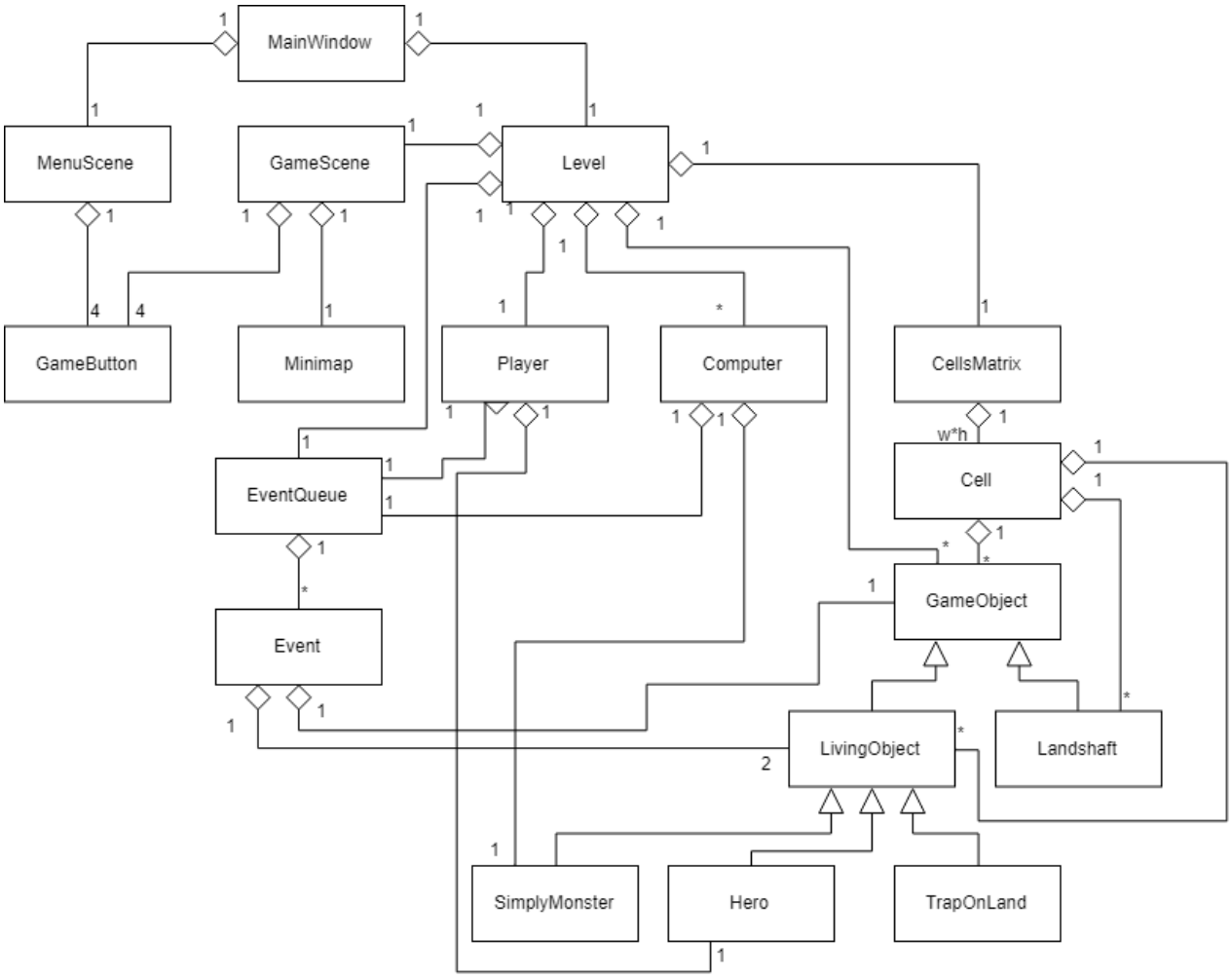

Рисунок 2.2 – Упрощенная диаграмма классов для расчета взаимодействия игровых объектов уровня реализации

Рассмотрим подробно основные спроектированные классы.

1. Класс MainWindow

Данный класс является главным в игре. Объект данного класса вызывается из функции main исходного файла – main.cpp.

В данном классе содержатся формы, в которых реализуется вся отрисовка графики, в том числе и интерфейс.

Кроме этого, этот класс содержит следующие поля:

- level класса Level, которое является членом игрового класса и которое вызывает функцию отрисовки игры
- menu класса MenuScene, которое создает главное меню и отрисовывает его, помимо этого данный объект отдает в главное меню сигнал StartGame, который, в свою очередь, запускает слот start, генерирующий новую игру.
- timer класса QTimer, который каждые 20 мс запускает слот onTimer, который, в свою очередь, запускает обновление состояния игры.

И класс содержит метод newGame, который создает новый объект level и подключает все необходимые связи для отрисовки, связывает сигналы и слоты.

Помимо этого, класс содержит 4 слота: start(), который запускает таймер и вызывает метод newGame, closeGame(), который осуществляет выход из игры, OpenMenu(), который останавливает работу таймера и отрисовывает главное меню и onTimer(), выполняющий вызов функции обновления игры.

2. Класс Level

Класс Level – главный игровой класс, в нём обрабатываются все игровые события, взаимодействия, нажатия на клавиши игрока и работа искуственного интеллекта.

Данный класс содержит следующие поля:

- field класса CellsMatrix, содержащий информацию об игровом поле и всех объектах на нём;

- graphics класса GameScene, использующийся для отрисовки измененного состояния игры;
- player класса Player, который содержит информацию об управляемым игроком герое;
- enemy класса Computer, который содержит информацию о врагах и функции обновления состояния. Как это происходит, будет рассказано непосредственно в классе Computer;
- events класса EventQueue – очередь событий, в которую попадают все события, которые будут обрабатываться методами класса activateEvent(event).

Основные методы данного класса:

- generateField – данный метод вызывает методы класса CellsMatrix, которые создают и процедурно заполняют игровое поле field;
- updateLevel – метод вызывает функции обновления состояния каждого управляемого игрового объекта (методы полей enemy и player), запускает выполнение событий events и вызывает метод отрисовки игры render.
- CheckMoving – данный метод обрабатывает нажатие клавиш и мышки, получая данные о точке нажатия от поля graphics. Обработка заключается в создании игрового события перемещения или атаки героя. Кроме этого метод запускает цепную реакцию обновления состояний игрового процесса.
- render – этот метод отрисовывает текущее игровое состояние игры, передавая в метод render поля graphics информацию о всех игровых объектах.

3. Класс EventQueue

Данный класс является реализацией одного из паттернов игрового проектирования, а именно синглтоном. Синглтон – порождающий шаблон проектирования, который гарантирует, что в однопроцессорном приложении

будет единственный экземпляр некоторого класса. Класс используется как собиратель игровых событий от всех объектов, которые их создают.

4. Класс CellsMatrix

Данный класс является классом игрового поля и объединяет все игровые ячейки, в которых находятся все игровые объекты – от стен и ландшафта до героя.

Данный класс имеет 3 поля: поле matrix класса QVector<QVector<Cell*>>, в котором и содержится вся информация о ячейках и информационные поля wSize и hSize, содержащие информацию о ширине и высоте игрового поля соответственно.

Основные методы класса перечислены ниже:

- dfs – вспомогательный метод, вызываемый методом placeWalls. Метод возвращает ответ на вопрос, можно ли поставить в данном месте стену так, чтобы в результате граф переходов не разбился на 2 графа, или, чтобы из любой точки поля, в которую можно пойти, можно было пройти в любую другую точку;
- placeWalls – метод возвращает двухмерный массив типа bool, в котором единицы – поставленные классом стены, а нули – свободные от стен места. В результате вызова данной функции формируется карта расположения стен и свободного от них пространства;
- generateMatrix – метод инициализирует матрицу matrix и задает значения полям wSize и hSize в соответствии с переданными ему аргументами;
- fillMatrix – метод создает объекты стен, ландшафта, ловушек и расставляет их по игровому полю. Для расстановки стен метод получает данные от метода placeWalls.

5. Графический класс GameScene

Данный класс отвечает за отрисовку игры. Основными полями данного класса являются:

- Поле lastClickPos, которое впоследствии передается в метод CheckMoving класса level;
- Поля leftX и leftY, которые указывают на координаты левого верхнего угла, что позволяет отрисовывать лишь необходимые игровые объекты;
- Информационные поля hasClicked и menuOpened, первое отвечает на вопрос – было ли нажатие на клавиши или мышь, а второе за режим отрисовки игры – если menuOpened – false, отрисовывается игра, если menuOpened – true, поверх игры прорисовывается полупрозрачное внутриигровое меню.

6. Класс Computer

Данный класс является классом, который управляет логикой ИИ монстров.

Класс имеет следующие поля:

– Поле monster класса SimplyMonster – монстр, управляемый данным классом;

– Поля x, y, dx, dy класса int, которые отображают текущую позицию монстра и изменения координат x и y в следующем шаге обновления;

– Поле matrix класса CellsMatrix, которое показывает состояние игрового поля на момент определения действия монстра, используемый для принятия решения о том, что монстр будет делать;

– Поле eQueue класса EventQueue, используемое для добавления новых событий изменения состояния монстра в очередь событий.

Для изменения состояния монстра и установки значений полей в классе присутствуют следующие методы:

– setField(CellsMatrix* matr), устанавливающее значение поля matrix и добавляющая на новое игровое поля объект монстра;

– attackAction(), проверяющее в зоне видимости монстра наличие врага(игрока) и, в случае, если игрок есть в зоне видимости,

создающий событие следования или атаки игрока в зависимости от дальности до него;

- moveAction(), задающее новую позицию движения игрока, или следующую согласно старой точки назначения. Маршрут к этой точке прокладывается алгоритмом поиска пути, заключающемся в нумерации клеток, где номер клетки – минимальное расстояние до клетки, из которой начинается поиск;
- update(), вызывающий методы attackAction() и moveAction();
- isUpdateDone(), проверяющий выполнение обновления состояния монстра в текущем цикле обновлений и, в случае, если состояние не изменилось, вызывающая новое изменение состояния.

Все остальные классы не являются настолько важными, поэтому приведу их краткое описание в таблице 2.1:

Таблица 2.1 – Описание классов

Название	Назначение
MenuScene	Графический класс, в котором отображается главное меню.
GameButton	Графический объект - кнопка, которая используется в классах GameScene и MenuScene.
GameEvent	Класс внутриигрового события, который содержит данные о типе события и всех параметров указанного типа события.
Cell	Класс-ячейка, массив которых является полем класса CellsMatrix. Данный класс владеет информацией обо всех объектах, которые находятся в данной ячейке, её координаты и свойства её проходимости.
GameObject	Класс игрового объекта – предок любого последующего класса объекта игры. В нём содержатся информация о координатах объекта, типе объекта, его идентификаторе и размере.
Landshaft	Класс ландшафта – такой объект прорисовывается абсолютно всегда и содержится как отдельное поле в классе Cell.
LivingObject	Класс живого объекта, содержит основную информацию о живых объектах в игре и владеет информацией об их уроне, текущем здоровье, максимальном здоровье и методами для изменения и получения информации об этом.

TrapOnLand	Класс объекта-ловушки, наследуемый от класса LivingObject. В классе Level создается атакующее со стороны ловушки событие, когда на неё кто-либо наступает. Ловушка, в свою очередь также получает урон, вследствие чего она не является долговечной и после некоторого количества использований исчезает.
Player	Класс, управляющий изменением состояния героя и управляемый игроком. Из-за того, что все команды, вводимые игроком вводятся в классе GameScene, а обрабатываются в классе Level, класс имеет ограниченный функционал.
Hero	Данный класс содержит функцию отрисовки игрового объекта героя, а также поля уровня и опыта.
SimplyMonster	Данный класс содержит функцию отрисовки игрового объекта простейшего монстра

2.3 Реализация основных алгоритмов

В ходе проектирования было реализованы следующие алгоритмы:

- Алгоритм проверки возможности установки стены в данном месте (см. рис. 2.3)
- Алгоритм компьютерного зрения искусственного интеллекта
- Алгоритм поиска пути – данный алгоритм является алгоритмом разметки расстояний на графе, по которым впоследствии восстанавливается маршрут.

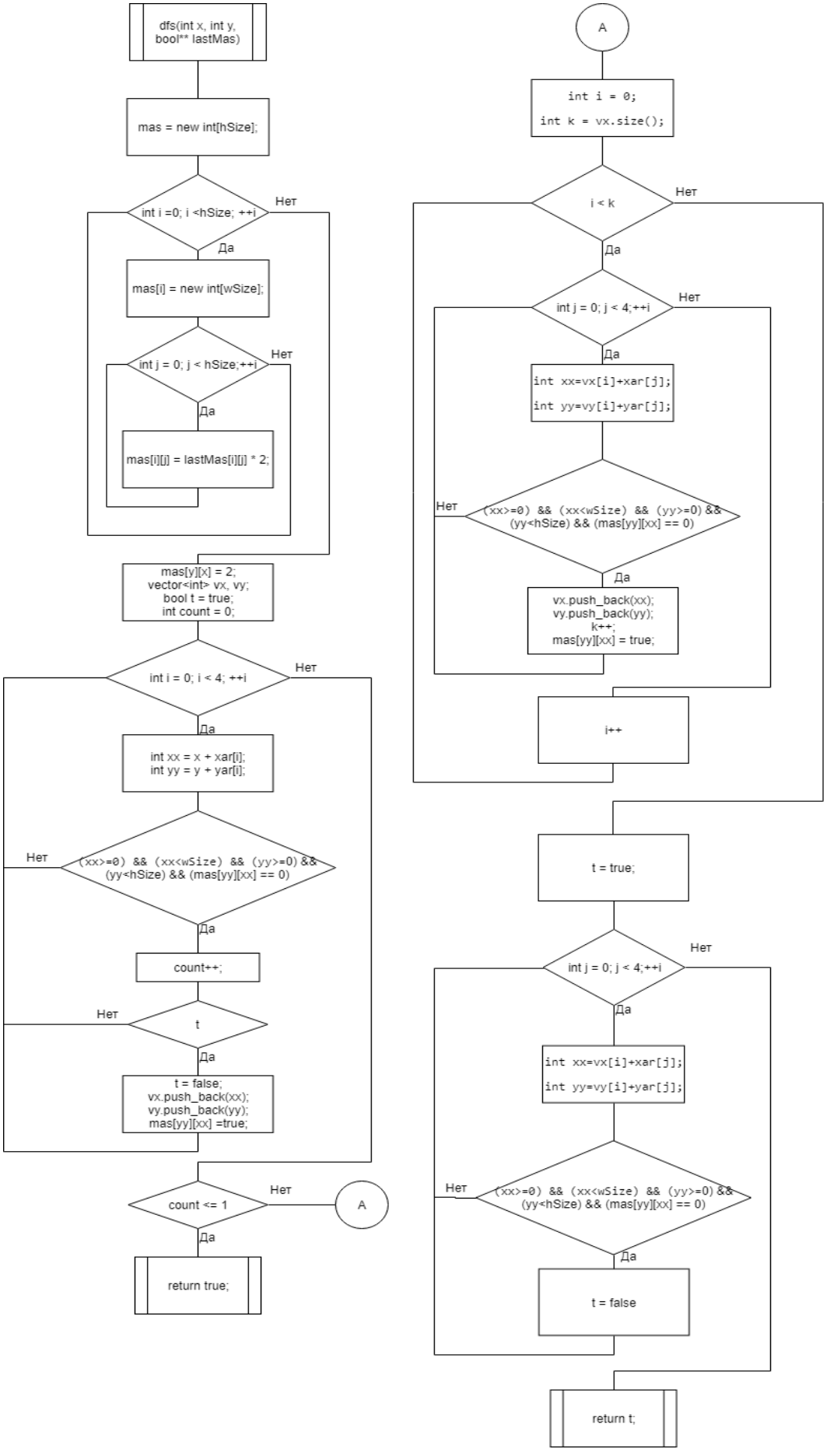

Рисунок 2.3 – схема алгоритма проверки возможности установки стены

2.4 Декомпозиция программы на модули

Для каждого класса среда Qt Creator создает 2 файла – файл .h, который содержит объявление класса и все прототипы методов и файл .cpp, в котором содержится реализация этих методов. Так как в проекте присутствует 17 классов, принято решение объединить эти 2 файла на диаграмме компоновки в один файл, названный так же, как и имя класса. Получившаяся в результате диаграмма компоновки представлена на рисунке 2.3.

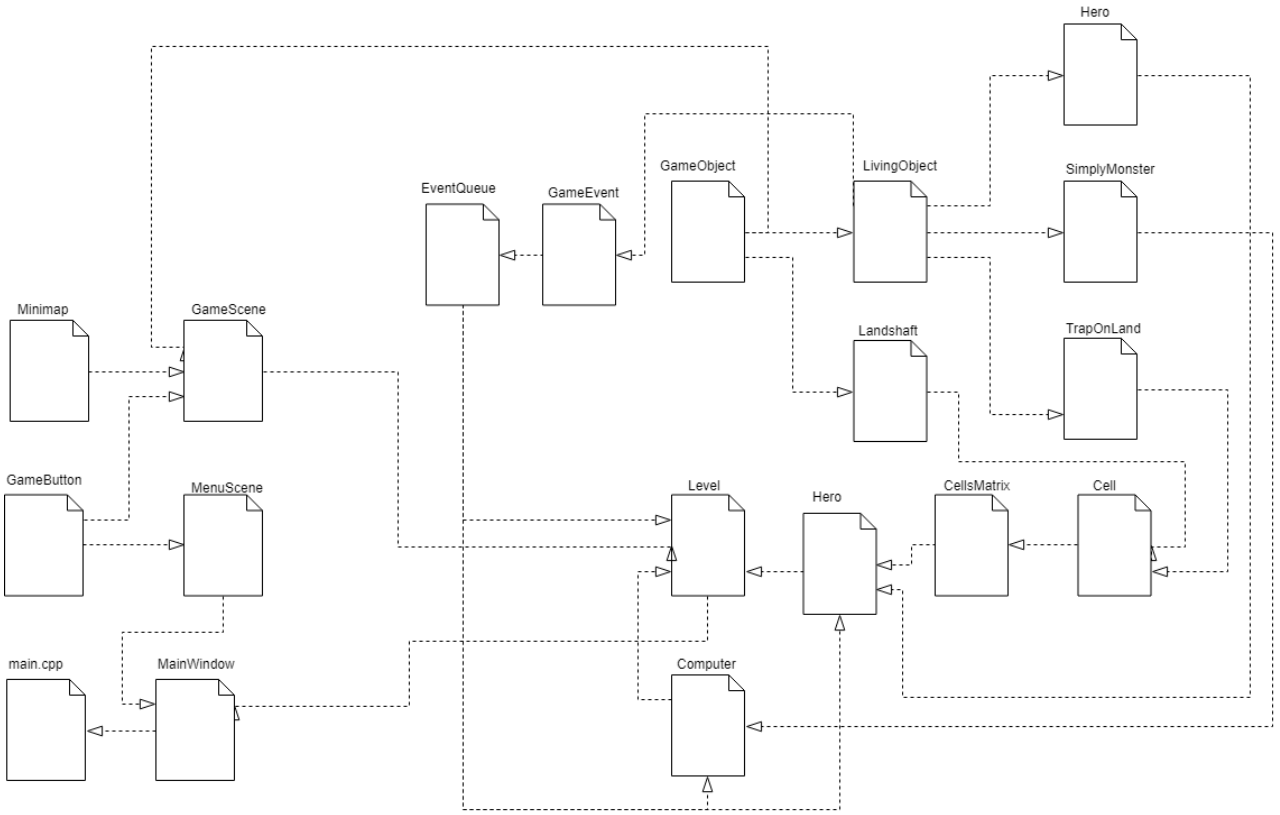

Рисунок 2.4 – Диаграмма компоновки программы

3 Выбор стратегии тестирования и разработка тестов

Существуют три стратегии тестирования программного продукта: ручное тестирование, структурное тестирование, выполняемое по методу «белого ящика» и функциональное тестирование, которое выполняется по методу «черного ящика».

Структурное тестирование основывается на знании внутренней структуры программного обеспечения, и, исходя из внутренней структуры, строятся тестовые наборы для проверки правильности реализации логики программы.

Функциональное тестирование основывается на том, что структура программного обеспечения неизвестно. В таком случае тесты строятся, опираясь на функции, которые выполняются программой.

Для тестирования программного продукта выбрано функциональное тестирование, так как пользователю данного проекта предоставляется только интерфейс с определенным набором выполняемых функций. Работу этих функций будет необходимо проверить на различных наборах входных данных.

3.1 Функциональное тестирование

При функциональном тестировании различают несколько методов формирования тестовых наборов: эквивалентное разбиение, анализ граничных значений, анализ причинно-следственных связей, предположение об ошибке.

Методом эквивалентных разбиений выделены следующие классы эквивалентности:

Ограничение на значение параметра	Правильные классы эквивалентности	Неправильные классы эквивалентности
Кнопки меню	Нажатие на кнопку	Нажатие на свободное пространство
Слоты загрузки	Нажатие на слот, в котором присутствует запись об игре	Нажатие на пустой слот
Передвижения игрока	Передвижение по ячейкам, в которые можно переходить	Передвижение по ячейкам, в которые переходить нельзя, к примеру, стены

Атака игроком	При наличии здоровья <0 монстр исчезает	После потери всего здоровья монстр не пропадает

Методом анализа граничных значений найдено 2 набора тестов – проверка первого и последнего слотов сохранения и загрузки и проверка возможности выйти за границы игрового поля.

В результате рассуждений приложение протестировано на следующих наборах тестов:

№	Условие теста	Ожидаемый результат	Реакция программы
1	Нажатие на кнопку «New game» в главном меню	Создание новой игры и её отрисовка на экране	Появилось окно с новой игрой
2	Нажатие на кнопку «Exit»	Прекращение работы приложения	Приложение закрылось
3	Нажатие не на кнопку	Отсутствие реакции	-
4	Переход игрока на свободную ячейку	Изменение позиции героя	Позиция героя изменилась
5	Герой перешел на поле с ловушкой	Здоровье героя изменилось	Здоровье героя уменьшилось
6	Здоровье героя опустилось ниже нуля	Переход в главное меню	Произошел переход в главное меню
7	Попытка перехода за пределы игрового поля	Отсутствие реакции	-
8	Попытка перехода в стену	Отсутствие реакции	-
9	Нажатие на кнопку клавиатуры ESC	Остановка игры, открытие внутриигрового меню	Работа игры остановилась, открылось меню
10	Попытка нажатия на кнопки изменения позиции героя при открытом меню	Отсутствие реакции	-
11	Нажатие на кнопку «Continue»	Закрытие меню, продолжение работы игры	Меню исчезло
12	Нажатие на кнопку «Exit to main menu»	Открытие главного меню	Открылось главное меню
13	Нажатие на кнопку «Exit from game»	Закрытие приложения	Приложение закрылось

3.2 Комплексное тестирование

Для тестирования программного продукта используем минимальное комплексное тестирование, которое предполагает:

- тщательную проверку руководства;
- тестирование минимальных конфигураций технических средств;
- тестирование возможности редактирования команд и повторения их в любой последовательности;
- тестирование устойчивости к ошибкам пользователя.

Команды программного продукта можно выполнять в любой последовательности, так как в каждом из окон игроку предоставляется лишь ограниченный набор выполняемых команд, которые протестированы на этапе функционального тестирования. Учитывая это, можно сказать, что программный продукт устойчив к ошибкам пользователя.

3.3 Оценочное тестирование

После завершения комплексного тестирования приступим к оценочному тестированию, целью которого является тестирование программы на соответствие основным требованиям.

Рассмотрим следующие критерии тестирования системы:

1) Тестирование удобства использования – последовательная проверка соответствия программного продукта и документации на него основным положениям технического задания;
2) Тестирование удобства эксплуатации – анализ психологических факторов, возникающих при работе с программным обеспечением, позволяющий определить, удобен ли интерфейс;
3) Тестирование требований к памяти – определение реальных потребностей в оперативной и внешней памяти.

Для проведения тестирования удобства эксплуатации программный продукт был предоставлен нескольким студентам 3 курса кафедры ИУ6 МГТУ им. Н.Э.Баумана. Ими сделаны выводы о том, что интерфейс достаточно удобен и достаточно понятен, чтобы в нём можно было разобраться без дополнительного обучения.

Запущенная игра с размером игрового поля 20x20 тратит 14.7Мб оперативной памяти и практически не загружает процессор.

Тестирование показало, что система соответствует поставленным требованиям.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта создана игровая программа, содержащая все необходимые игровые меню и базовые объекты. Помимо этого получен опыт разработки движка игры с нуля, применения различных шаблонов проектирования и работы с искусственным интеллектом и процедурной генерацией мира.

Во время разработки игровой программы использовалась спиральная схема разработки программного продукта, которая показала свою эффективность на практике. Кроме того, использовался объектный подход к проектированию, который позволил упростить процесс написания кода и уменьшить связность модулей программы.

Программный продукт протестирован с использованием следующих видов тестирования:

- функциональное тестирование, позволившее проверить ошибки в логике работы программы, которые можно получить, используя лишь графический интерфейс;
- комплексное тестирование, позволившее проверить соответствие программы документации, ресурсоёмкость программного продукта и возможность его модификации;
- оценочное тестирование, проводимое с помощью других людей, позволившее оценить уровень удобства эксплуатации программного продукта, соответствие его документации и количеству используемой памяти.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Иванова Г.С. Технология программирования: Учебник для вузов М. : Кнорус, 2013. 333 с.
2. Конспекты лекций по курсу «Технология разработки программных систем»
3. Иванова Г.С., Ничушкина Т.Н. Оформление текстовых документов. Методические указания по оформлению расчетно-пояснительных записок дипломных и квалификационных работ. М.: Издательство МГТУ им. Н.Э.Баумана, 2004. 10 с.
4. Иванова Г.С. Основы программирования: Учебник для втузов. 4-е изд. М.: Издательство МГТУ им. Н.Э.Баумана, 2007.

ПРИЛОЖЕНИЕ А

ПРИЛОЖЕНИЕ Б