министерство НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
Российской федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Тюменский государственный университет»

институт математики и компьютерных наук

Кафедра программного обеспечения

Курсовая работа по направлению

«Математическое обеспечение и администрирование

информационных систем»

на тему «Разработка приложения для расчета характеристик и визуализации сложных физических систем в волновой оптике»

Выполнил: студент 2 курса

22 МОАИС 184-2 группы

Юсуф М.А.

Научный руководитель:

доцент кафедры ПО

Гаврилова Н.М.

Тюмень 2020

Оглавление

[ГЛАВА 1. Разработка идеи 4](#_Toc116915672)

[ГЛАВА 2. Реализация движка 5](#_Toc116915673)

[ГЛАВА 3 Реализация и описание игры. 8](#_Toc116915674)

[Заключение 10](#_Toc116915675)

[Список литературы 11](#_Toc116915676)

[Приложение 1 11](#_Toc116915677)

**ВВЕДЕНИЕ**

В качестве курсовой работы за 2 курс по дисциплине ЯП высокого уровня было получено задание – написать игру с использованием ООП.

Цель работы: разработать игру с использованием ООП.

Для достижения цели необходимо выполнить следующие задачи:

1. Разработать идею игры.
2. Написать движок для её реализации.
3. Реализовать игру на написанном движке

# ГЛАВА 1. Разработка идеи

В среднем требуемое время на разработку трёхмерной игры гораздо больше, чем для разработки двумерной. Поэтому было принято решение написать 2D-игру.

Основная идея – игра-головоломка, в которой требуется довести объект от одной точки до другой. Основная сложность заключается в управлении. Объект будет представлять трёхмерную коробку размером 1х1х2, которая при нажатии кнопок перемещения переворачивается на нужные стороны. При достижении игроком определённого поля будет загружаться требуемый уровень.

Поле состоит из клеток, по которым игрок передвигается. Если объект лежит длинной стороной на поле, то он занимает 2 клетки, если меньшей – 1 клетку. Для упрощения написания полей реализован конструктор карт, который принимает на вход файл формата .txt. Расположение карт - \Maps . В файле хранится само поле в виде массива, где 0 – пустая клетка, все остальное клетки, по которым может перемещаться игрок. Первая строка файла пропускается программой, она служит для комментария. После массива хранится начальные координаты игрока и скорость передвижения. (Пример файла - приложение 1).

Для каждой карты обязателен документ с описанием событий. На данный момент реализовано одно событие – OnPosition. Это событие активируется при нахождении модельки игрока на определённой позиции. Формат документа .txt , название – такое же, как и у карты. Первая строка в файле используется для комментария. Структура описания события:

Первая строка – тип (OnPosition).

Вторая – название события.

Третья – координаты, на которых событие срабатывает (только для OnPosition)

Четвёртая и последующие – команды, выполняемые для данного события (выход, загрузка карты, перемещение на позицию)

Последняя строка события - EVENT\_END

Последняя строка документа – EVENTS\_END

Каждая строка, кроме последней события и документа, заканчивается символом «;». Пример файла событий приведён в (приложение 2).

# ГЛАВА 2. Реализация движка

Для скорости и производительности игры язык был выбран C++. Для отрисовки объектов используется набор библиотек Simple and Fast Multimedia Library (SFML).

Для воспроизведения анимаций написан компонент отдельный (AnimationComponent). В нём для определённого анимационного листа изменяется текущий спрайт, в соответствии с заданной скоростью и размерами спрайта. Анимация может быть прерываемой или - нет. Компонент хранится в двух файлах: Components\AnimationComponent.h и Components\AnimationComponent.cpp . Дополнительно компонент использует класс Animation. В нём реализован метод получения нового спрайта, в соответсвии с анимационным листом и скоростью – play . Метод принимает на вход const float& dt – ссылка на разницу времени, и возвращает const bool& (true – если последняя текстура спрайта в анимационном листе, false - иначе). Метод isDone возвращает результат выполнения анимации (true – если анимация не воспроизводится, иначе - false). Метод reset начинает воспроизведение анимации заново.

Основные методы и переменные анимационной компоненты: (таблица 1)

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Входные и выходные данные |
| play | Воспроизводит анимацию, изменяя входной спрайт на следующий по анимационному листу. | На вход передаётся ключ воспроизводимой анимации, разница во времени, приоритет воспроизводимой анимации.  На выходе логическое выражение, определяющее закончено воспроизведение или нет. |
| isDone | Возвращает false, если анимация воспроизводится | На вход – ключ анимации, на выходе логическое значение. |
| addAnimation | Добавляет в список анимаций новую с заданными параметрами | На вход – ключ анимации, анимационный лист, скорость воспроизведения анимации и начальные, конечные положения спрайта и его размеры. |

Таблица 1 – Основные методы анимационной компоненты

Для описания объектов и игрока реализован класс Entity. Он включает в себя область анимационной компоненты. Дополнительно класс хранит в себе физическую модель карты, это используется в методе перемещения для определения на какое поле можно переместиться. Основные переменные и методы описаны в таблице 2:

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Входные/выходные данные |
| Sprite | Переменная. Отображаемый спрайт entity |  |
| AnimationComponent | Переменная. Анимационный компонент для объекта. |  |
| update | Виртуальный метод обновления объекта, вызывается на каждой итерации игры. | На вход – разница времени. |
| render | Виртуальный метод отрисовки объекта. Рисует спрайт на экране | На вход – ссылка на рисуемый объект. |
| move | Виртуальный метод перемещения объекта в пространстве. | На вход разница времени и координаты, но которые нужно переместиться. |

Таблица 2 – класс Entity

Класс игрока – Player. Является дочерним классом для Entity и переопределяет методы задания позиции игрока(setPosition), передвижения(move) и обновления(update). В нём инициализируются все компоненты и переменные.

Класс игровых состояний – State. Использует область Player. Инициализирует используемые нажатия клавиатуры, хранит положение мыши, окно приложения. Дополнительно хранит в себе ссылку на словарь всех игровых состояний. Основные методы – update и render. Update обновляет итерацию игры, render – отрисовывает все объекты.

Класс Button – описывает объект кнопки. Отрисовывается методом render, обновляется – update. Если курсор мыши в области обновления кнопки, то её состояние изменяется на BTN\_HOVER, если курсор не в области – BTN\_IDLE, если кнопка мыши нажата и курсор в области – BTN\_ACTIVE. При состоянии BTN\_ACTIVE. В классах состояния игры при обновлении итерации проверяется нажатие левой кнопки мыши, если пользователь нажал – вызывается у всех кнопок метод isPressed() и возвращает true, если состояние BTN\_ACTIVE.

Класс MainMenuState – описывает главное меню игры. Наследуется от State. Использует Button и GameState. В обновлении обрабатывает нажатие кнопок мыши и переводит на другие состояния игры.

Класс GameState – игровой класс. Наследуется от State. Создаёт объект игрока и карты. В нём обрабатывается ввод с клавиатуры и мыши.

В классе Game запускается состояние главного меню и в бесконечном цикле обновляется приложение, вызывая метод update класса State.

# ГЛАВА 3 Реализация и описание игры.

В главном меню используется заранее отрисованный задний фон, располагается в \ResourceFiles\Images\Backgrounds. При старте игры помимо фона отрисовываются 4 кнопки – NewGame, settings, test и quit (рисунок 1). При нажатии NewGame проигрывается анимация перехода к следующему меню и запускается стартовый уровень игры. Кнопки settings и test неактивны, используются для тестирования. Нажав на quit приложение завершится.

(Рисунок 1. Главное меню)

Нажав на NewGame запускается следующее состояние – GameState и загружается стартовая карта Map0. Состояние главного меню убирается. Создаётся и отрисовывается игрок. Он может перемещаться по карте и загружать нужные ему уровни. (рисунок 2)

(Рисунок 2. Стартовая карта)

При достижении игроком портала выполняется событие, соответствующее данной позиции. Во время выполнения события проверяется требуемое действие: выход, перемещение на координаты или загрузка следующей карты. К примеру: достигнув портала 2 игрок переместится на следующую карту (рисунок 3):

(Рисунок 3. Следующая карта)

# Заключение

В ходе курсовой работы была реализована двумерная игра, использующая ООП. Были выполнены все поставленные задачи:

1. Разработана идею игры.
2. Написан движок для её реализации.
3. Реализована игру.

# Список литературы

# Приложение 1