Оглавление

Описание задачи

Цель курсовой работы – создание библиотеки, представляющего вспомогательные инструменты для создания приложения суть которого убийство (понижение параметра здоровья до 0) NPC(противников) для выживания на двухмерной плоскости (TopDownShooter). Задача NPC убить (понизить параметр здоровья до 0) игрока. Задача игрока – выжить (иметь параметр здоровья больше 0), убив как можно больше NPC.

Программа предназначена как для конечного пользователя в качестве игры, так и для других разработчиков в качестве игрового движка;
Библиотека предназначена для других разработчиков в качестве игрового движка. Демонстрационное приложение ознакомления функционала библиотеки предназначено для конечного пользователя.

Требование к библиотеке

Требования к интерфейсу:

- Возможность начать игровой процесс заново;
- Возможность поставить игровой процесс на паузу;
- Возможность возобновлять игровой после паузы;
- Возможно выходить из игрового процесса
- Наведение камеры на игрока во время движения;

Требования к игровой механике:

- Появление NPC:
  - Появление NPC с настраиваемым количеством здоровья.
- Поведение NPC:
  - Возможность NPC наносить урон персонажу и двигаться к нему.
- Смерть NPC:
  - Смерть NPC при понижении здоровья до 0, с возможностью оставлять анимацию после смерти.
- Появление персонажа:
  - Возможность задавать координаты появления персонажа с настраиваемым количеством здоровья.
- Движение персонажа:
  - Осуществлять движение кнопками «wasd» с возможностью регулировать скорость движения.
- Стрельба персонажа:
  - Возможность регулировать урон снарядов по NPC, возможность реализовывать перезарядку с регулированной длительностью, которая активируется после регулируемого количества снарядов.

- Смерть персонажа:
    - Смерть персонажа при понижении количества здоровья до 0 с последующей перезагрузкой уровня.
- Возможность управления персонажем:
    - W – движение вперед
    - A – движение влево
    - S – движение назад
    - D – движение направо
    - Курсор мыши – направление оружия героя
    - ПКМ – стрельба персонажа
- Анимация действий
    - Возможность добавления анимации с помощью спрайт листа.

Описание интерфейса библиотеки

В данной библиотеке имеются готовые классы «акторы» для описания логики персонажей, а также классы-обертки над glfw(написанной на языке программирования C) под OpenGL, которые позволяют использовать статический полиморфизм и не переживать за то, что это на C. Работа происходит не с машиной состояний, а с объектами c++.

За счет наследования можно добавить свой объект, который поддерживаться на уровне интерфейсов. Возможность добавления спрайтовой анимации, передавая координаты 1го кадра и кол-ва кадров (по вертикале и горизонтали).

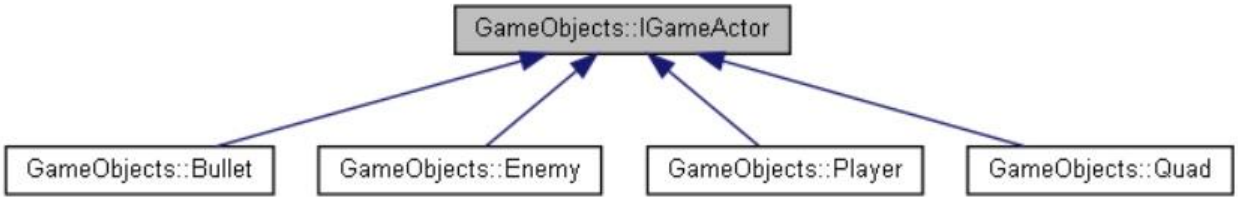


Рисунок 1.1. Архитектура работы библиотеки.

Более подробное описание функций и логики работы библиотеки смотреть: https://mysvn.ru/Whatevenif/anime/yurkin_p_o_coursework/Doxygen/html/files.html

Демонстрационное приложение:

Игровая механика:

- Появление NPC:
  - Появление количества NPC 20 с количеством здоровья 100, при смерти NPC появляется новый.
- Поведение NPC:
  - NPC движется к персонажу со скоростью 10, при близости NPC к персонажу пользователя NPC наносит урон 13 персонажу
- Смерть NPC:
  - NPC умирает при понижении количества здоровья до 0. На месте смерти оставляет «кровавый» след;

Рисунок 2.1. «Кровавый» след после смерти

- Появление персонажа:
  - Персонаж появляется в центре карты с количеством здоровья 100;
- Движение персонажа:
  - Осуществляется кнопками «wasd», персонаж движется со скоростью 0.2 по направлению курсора;
- Стрельба персонажа:
  - Направление стрельбы персонажа зависит от движения курсора мыши. Персонаж делает выстрел в направление курсора мышки 30 урона при нажатии на ПКМ. После 30 выстрелов которые сделал игрок происходит перезарядка которая длится 4сек;
- Смерть персонажа:
  - Персонаж умирает при понижении количества здоровья до 0. После смерти персонажа появляется окно, уведомляющие о смерти, игровой процесс начинается заново через 3(сек);
- Возможность управления персонажем:
  - W – движение вперед
  - A – движение влево
  - S – движение назад
  - D – движение направо
  - Курсор мыши – направление оружия героя
  - ПКМ – стрельба персонажа
- Анимация действий
  - Анимация во время движения персонажа

Рисунок 2.2. Спрайт-лист движения персонажа

- Анимация во время перезарядки персонажа

Рисунок 2.3 Спрайт-лист перезарядки персонажа

- Анимация снаряда

Рисунок 2.4. Спрайт снаряда

- Анимация во время движения NPC

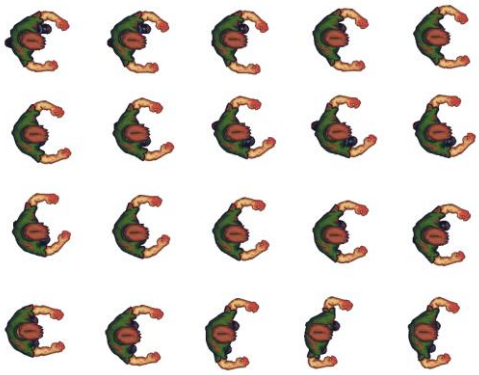


Рисунок 2.5. Спрайт-лист движения NPC

- Анимация NPC во время нанесения урона персонажу

Рисунок 2.6. Спрайт-лист NPC нанесения урона

Язык c++

Среда разработки: CLion 2019

Сторонние библиотеки: GLFW, GLM, JSON for Modern C++

Описание интерфейса

- Запуск программы
  - После запуска программы сразу начинается игровой процесс;
- Окно игрового процесса
  - После запуска программы выводится окно игрового процесса. В центре окна находится персонаж, с которому подходят NCP. Состоит из одной визуальной части;

Рисунок 3.1. Начало игрового процесса

- Пауза игрового процесса
    - При нажатии «p» на клавиатуре игровой процесс останавливается;

Рисунок 3.2. Пауза игрового процесса

- Процесс перезагрузки уровня при смерти персонажа

Рисунок 3.3. Процесс перезагрузки уровня при смерти персонажа

- Игровое поле
    - Состоит из одной визуальной части и текстур кустов, появляется после запуска программы;

Рисунок 3.3. Игровое поле

- Описание игровой механики
    - Было реализовано приложение, удовлетворяющее поставленной задаче. Реализованы стрельба, NPC, пауза, выход;
- Описание игрового процесса
    - Персонаж появляется в центре игрового поля, перемещается и убивает NPC, чтобы не получать урон от NPC;
- Описание NPC

- Один вид NPC, NPC идет напрямую на персонажа, нанося ему урон при достижении героя;

Редактирование демонстрационного приложения

Параметры приложения изменяются в файле проекта config.json, в папке res

- Редактирование параметров объектов:


```
{
  "command": "configure",
  "data": {
    "enemy": {
      "damage": 13.0,
      "enemy_numbers": 20,
      "health": 100.0,
      "velocity": 0.3
    },
    "player": {
      "Player_Position": {
        "x": 0.0,
        "y": 0.0
      },
      "Player_Size": {
        "x": 0.13,
        "y": 0.2
      },
      "ammo": 30,
      "damage": 33.0,
      "health": 100.0,
      "reload_time": 0.8,
      "respawn_time": 3.0,
      "velocity": 0.3
    }
  }
}
```


Рисунок 4.1. Конфигурационный файл демонстрационного приложения.

Для редактирования параметров NPC:


```
"enemy": {
  "damage": 13.0,
  "enemy_numbers": 20,
  "health": 100.0,
  "velocity": 0.3
```


Рисунок 4.2. Часть файла конфигурации для изменения параметров NPC в демонстрационном приложении.

Где:
damage – урон
enemy_numbers – количество

health – здоровье
velocity – скорость

Для редактирования параметров персонажа:

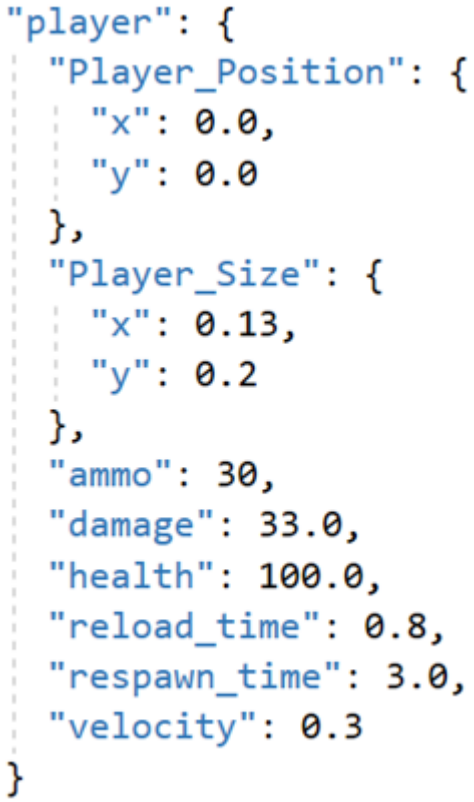


Рисунок 4.3. Часть файла конфигурации для изменения параметров персонажа в демонстрационном приложении.

Где:
ammo – количество снарядов
damage – урон снарядов
health – количество здоровья
reaload_time – время перезарядки
respawn_time – время перезагрузки уровня после смерти персонажа
velocity – скорость

- Редактирование визуальной части:
  В файле test_engine.h в функции “ZombieShooter”

Рисунок 4.4. Часть кода для редактирования визуальной части в файле test_engine.h

Свои изображения спрайтов и спрайт листов необходимо добавить в папку res/chars (в данной папке должны лежать спрайты персонажа и NPC проекта) для спрайтов и res/textures для текстур (в данной папке должны лежать текстуры)

Рисунок 4.5. Часть кода, показывающая путь к спрайту/текстуре

Чтобы заменить спрайт заменить спрайт лист “survivor_sec.png” на свой предварительно добавив свой спрайт лист в папку
Где:
Survivor_Texture – движение персонажа
Survivor_Texture_Reloading – перезарядка персонажа
Zombie_Texture – движение NPC
Zombie_Die_Texture – смерть NPC
Bullet_Texture – текстура снаряда
Tree_Texture – текстура куста на карте
You_Dead_Texture – текстура в процессе перезагрузки игрового процесса
Pause_Texture – текстура в процессе паузы игрового процесса
Map_Texture – текстура карты

Техническое описание

Кроссплатформенная библиотека для создания TopDownShooter игр. Реализованная на языке программирования c++ с помощью графического API OpenGL. Доступна генерация документации Doxygen.

В качестве дополнительных ресурсов были использованы изображения.

Реализованные классы с кратким описанием

Engine	Собирает компоненты игрового процесса
Camera	Обеспечение работы камеры

IndexBuffer	Инкапсулирует работу с индексном буфером на уровне ооп
Shader	Обеспечение работы шейдеров
Sprite	Обеспечение работы спрайтов
SpriteAnimator	Обеспечение анимации спрайтов
Texture2D	Обеспечивает работу текстур таким образом, создавая для каждой текстуры свой буфер, сохраняя его на видеокарте
VertexArray	Упрощает работу с вершинным массивом
VertexBuffer	Инкапсулирует работу с вершинным буфером
Bullet	Объект снаряд
Enemy	Враждебный NPC
IGameActor	Общий интерфейс для всех игровых объектов
Player	Объект персонаж
Quad	Статический спрайт для анимаций, взаимодействующих с пользователем (сцена паузы, картинка после смерти персонажа)

Для более подробного описание смотреть документацию:
https://mysvn.ru/Whatevenif/anime/yurkin_p_o_coursework/Doxygen/html/files.html

Инструкция по сборке демонстрационного приложения:

Для сборки данной программы необходим компилятор, совместимый со стандартом C++17 и CMake версии не ниже 3.15.

Последнюю версию CMake можно скачать по ссылке:
https://cmake.org/download/

Исходный код библиотеки находится:
https://mysvn.ru/Whatevenif/anime/yurkin_p_o_coursework/src/

Для операционной системы Linux выполнить следующие команды:
- sudo apt-get install libgl1-mesa-dev
- sudo apt-get install libxrandr-dev
- sudo apt-get install libxinerama-dev

- sudo apt-get install libxcursor-dev
- sudo apt-get install libxi-dev

Исходный код, файлы проекта и файлы cmake находятся по ссылке: https://drive.google.com/drive/folders/1lSQnBgiAxBDlZvglH8-0aEHc0lhbfSe4?usp=sharing (в каталоге…/TopDownShooter)

Сборка выполняется при помощи кроссплатформенной утилиты CMake. В инструкции описана сборка с использованием Visual Studio 2019 в операционной системе Windows.

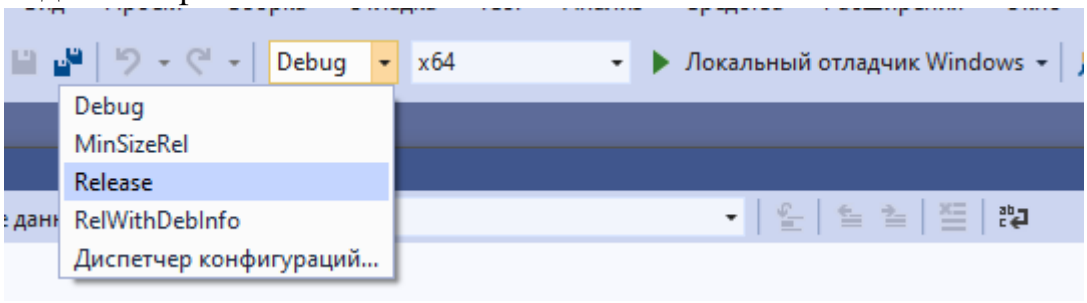
- Запустить CMakeLists.txt в TopDownShooter
- После выполнения CMake появится файл «GameEngine.sln»
- Открыв данный файл в Visual Studio 2019
  - 

Рисунок 3.1 интерфейс Visual Studio 2019

В диспетчере конфигураций выбрать «Realease» и запустить.

- После выполнения компиляции в относительно папки TopDownShooter на пути bin/Release/GameEngine.exe находится исполняемый файл запуска программы

Приложения

Приложение 1

Bullet
Снаряд.

#include <**Bullet.h**>

Bullet()
Конструктор

SetSprite()

Визуальная часть объекта (view)

Render()
Обновляет координаты объекта, находит текущую анимацию объекта

die()
Убирает объект

Update()
Обновляет параметры объекта каждый кадр

diactivate()
Отключает параметры объекта (для паузы)

Activate()
Создает снаряд

Закрытые члены:

SetAnimator()
Дает анимацию объекта

Enemy

NPC.

#include <**Enemy.h**>

Enemy()
Конструктор

SetPlayerTarget()
Дает направление NPC, чтобы NPC шел в персонажу

SetSpawn()
Задает параметры для появления NPC

Render()
Обновляет координаты объекта, находит текущую анимацию объекта

UpdateSprite()

Обновляет значения спрайта для объекта

SetAnimator()
Задает анимацию NPC

Update()
Обновляет значения параметров объекта NPC

Respawn()
Задает параметры для появления NPC после смерти NPC

Stop()
Сохраняет текущее время между кадрами (для паузы)

die()
Убирает объект

Закрытые члены:

SetAction()
Выбирает действие для NPC

IGameActor

Интерфейс объектов.

#include <**IGameActor.h**>

IGameActor()
Конструктор

~IGameActor()
Деструктор

SetAnimator()
Дает анимацию объекта

Render()
Обновляет координаты объекта, находит текущую анимацию объекта

die()
Убирает объект

setVelocity()
Задает скорость объекта

GetCurrentPosition()
Получает координаты позиции объекта

GetCurrentDirection()
Получает вектор направления объекта

GetSize()
Возвращает размер объекта

velocity()
Возвращает скорость объекта

Player

Персонаж.

#include <**Player.h**>

Player()
Конструктор

~Player()
Десктрутор

SetAnimator()
Дает анимацию персонажа

SetBullet()
Стрельба персонажа

SetAnimation()
Дает анимацию в зависимости от состояние объекта (движение, перезарядка)

Render()
Обновляет координаты объекта, находит текущую анимацию объекта

die()
Убирает объект

UpdateSprite()
Обновляет значения спрайта для объекта

getHealth()
Указатель на здоровья игрока, чтобы наносить урон

mouse_controller()
Взаимодействие с мышью

keyboard_controller()
Взаимодействие с клавиатурой

SetAction()
Состояние объекта (движение, перезарядка)

Quad

Статическая сцена (пауза, окно после смерти).

#include <**Quad.h**>

Quad()
Конструктор

~Quad()
Деструктор

SetSprite()
Визуальная часть объекта (view)

UpdateSprite()
Обновляет значения спрайта для объекта

SetPosition()
Координаты позиции объекта

Render()
Обновляет координаты объекта

SetAnimator()
Дает анимацию объекта

die()
Убирает объект

Camera

Вид для пользователя.

#include < **Camera.h**>

Camera()
Конструктор

SetPos()
Регулирует позицию камеры

Render()
Перемещает весь мир, относительно камеры (статическая) мир перемножить на матрицу

Engine

Сборщик компонентов.

#include <**Engine.h**>

Render()
Отображение всех объектов игры

ConfigPlayerAnim()

Добавляет новую анимацию

ConfigPlayerParticles()
Задает пули

SetPlayer()
Добавляет персонажа в игровой процесс

PlayerDeadChecker()
Проверяет мертв ли персонаж

SetDeadSign()
Добавляет сцену после смерти персонажа

ConfigSpritePauseSign()
Задает сцену паузы

CheckPauseMode()
Проверяет есть ли пауза

SetMap()
Добавляет картинку игрового поля

ConfigSpriteMap()
Задает спрайт игрового поля

SetEnemiesSpawns()
Добавляет места появления NPC

SetEnemies()
Добавляет NPC

ConfigEnemies()
Задает NPC

SetSprite()

Визуальная часть объектов (view)

SetSpritePack()
Добавляет спрайт лист

SetTexture()
Добавляет тектуры

SetShader()
Добавляет шейдеры

SetCameraUpdatePosition()
Добавляет обновленное положение камеры

Закрытые члены:

Engine()
Конструктор

IndexBuffer

Индексный буффер.

#include <**IndexBuffer.h**>

IndexBuffer()
Конструктор

~IndexBuffer()
Деструктор

Reset()
Обновляет буффер

bind()
Активирует индексный буффер

unbind()

деактивирует индексный буффер

GetId()
У каждого буффера есть ID, возвращает неизменяемый ID

GetSize()
Размер буффера

Shader

Шейдеры

Открытые члены:

Shader()
Конструктор

~Shader()
Деструктор

use()
Использовать текущую шейдерную программу

glUniform()
Умный полиморфный метод, который перегружает “C” функции glfw которые не используют полиморфизм

Check()
Проверяет скомпилировался ли шейдер и если нет, то сообщает об этом в логах

GetId()
Получить ID шейдера

Sprite

Визуальная часть объекта

#include <**Sprite.h**>

Sprite()
Конструктор

~Sprite()
Деструктор

Init()
Получает в себя значение вершины, ее параметры, индексы вершин в отдельном массиве и к этому индексу получает его параметры, по этим данным генерирует вершинный массив и сам графический объект Отправляет данные на GPU

SetSpriteSheet()
Берет спрайт из спрайт листа

draw()
Отрисовывает все что есть на поле

GetTexture()
Получение текстуры

GetShader()
Получение шейдера

GetCoords()
Получение координат

GetCenter()
Получение координат центра

GetSize()
Получение размера

GetRotation()

Получение угла

GetLayer()
Получение значение в иерархии изображения (ближе, дальше)

SpriteAnimator

Соединяет спрайты в нужном порядке для получении анимации.

#include <**SpriteAnimator.h**>

SpriteAnimator()
Конструктор

~SpriteAnimator()
Деструктор

SetSheet()
Добавляет кусок спрайта в спрайт лист(память)

SetSheetAtlas()
Просит от пользователя левая нижняя правая верхняя граница первого спрайта, задать смещение по x,y координатам, задать сколько спрайтов по вертикали, по горизонтали, дальше цикл вырежет спрайты

AnimationUpdate()
Будет выбирать какой кадр у анимации текущий

ClearAnimPack()
Очищает анимации

Texture2D

Работа текстуры.

#include <**Texture2D.h**>

Texture2D()
Конструктор

~Texture2D()
Деструктор

SetFilter()
Задает параметры для типа фильтра и фильтр того как текстура обрабатывается

SetMode()
Задает как будет обрабатываться незаполненная текстурой часть

FreeTrash()
Освобождает память в оперативной памяти

GenerateMipmap()
Генерирует mipmap (от масштаба будет заменять текстуру) не используется, так как двухмерный игровой процесс

bind()
Активирует индексный буффер

unbind()
деактивирует индексный буффер

GetHeight()
Получить высоту текстуры

GetWidth()
Получить длину текстуры

VertexArray

Массив вершинного буфера, хранит в себе все, что лежит в остальных буферах.

#include <**VertexArray.h**>

VertexArray()
Конструктор

~VertexArray()
Деструктор

bind()
Активирует индексный буффер

unbind()
Деактивирует индексный буффер

EnableAllAttrib()
Включает все атрибуты, которые находятся в VAO (vertex array object)

DisableAllAttrib()
Выключает все атрибуты, которые находятся в VAO (vertex array object)

GetId()
Id VAO в оперативке

GetLastAttrib()
Последнее значение последнего атрибута, чтобы не обратиться в несуществующий атрибут

VertexBuffer

Вершинный буфер.

#include <**VertexBuffer.h**>

VertexBuffer()
Конструктор

~VertexBuffer()
Деструктор

Reset()
Поменяет в оперативной памяти уже существующий буфер, чтобы изменить спрайт для анимации

bind()

Активирует индексный буффер

unbind()
Деактивирует индексный буффер

Более подробное описание:
https://mysvn.ru/Whatevenif/anime/yurkin_p_o_coursework/Doxygen/html/files.html