Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

кафедра Информатики

Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

Создание интернет-сервиса для поиска фильмов и кинотеатров

Студент: гр. 053506 Слуцкий Н.С.

Руководитель: Летохо А.С.

Минск 2022

СОДЕРЖАНИЕ

Введение

На сегодняшний день, например, в нашей стране существует сразу несколько веб-сервисов, которые могут предложить потенциальному пользователю онлайн-каталог фильмов, помочь определиться с выбором, а также выбрать кинотеатр для просмотра. Это относится не только к фильмам, но и к другим видам услуг, товаров и т.д.

Целью данной работы является изучение принципа построения настоящего full-stack restful приложения с использованием выбранных технологий и языков программирования. И основываясь на всём этом создать подобный вышеописанный онлайн-каталог с фильмами, кинотеатрами, возможностью регистрации, авторизации и добавления, например, фильмов в список избранных.

В целом данная курсовая работа является отличной возможностью более детально изучить веб-программирование со многих сторон, а также создать рабочий проект для будущего портфолио. Автор не может сказать, что на данном этапе это приложение может быть реально полезным для настоящих пользователей в жизни, однако оно являет собой очень большой фундамент, который можно дополнять всякими микро-сервисами без особых усилий. Это значит, что функционал, например по созданию кастомной панели администратора или по возможности писать отзывы на фильм могут добавиться достаточно легко.

Анализ предметной области

REST API

REST API — это способ взаимодействия сайтов и веб-приложений с сервером. Его также называют RESTful.

У RESTful есть 7 принципов написания кода интерфейсов.

Отделения клиента от сервера (Client-Server). Клиент — это пользовательский интерфейс сайта или приложения, например, поисковая строка видеохостинга. В REST API код запросов остается на стороне клиента, а код для доступа к данным — на стороне сервера. Это упрощает организацию API, позволяет легко переносить пользовательский интерфейс на другую платформу и дает возможность лучше масштабировать серверное хранение данных.

Отсутствие записи состояния клиента (Stateless). Сервер не должен хранить информацию о состоянии (проведенных операций) клиента. Каждый запрос от клиента должен содержать только ту информацию, которая нужна для получения данных от сервера.

Кэшируемость (Casheable). В данных запроса должно быть указано, нужно ли кэшировать данные (сохранять в специальном буфере для частых запросов). Если такое указание есть, клиент получит право обращаться к этому буферу при необходимости.

Единство интерфейса (Uniform Interface). Все данные должны запрашиваться через один URL-адрес стандартными протоколами, например, HTTP. Это упрощает архитектуру сайта или приложения и делает взаимодействие с сервером понятнее.

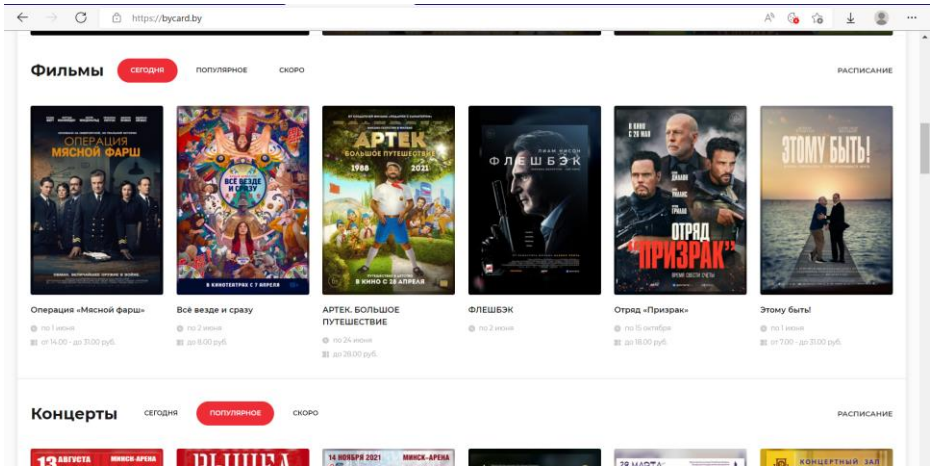
Многоуровневость системы (Layered System). В RESTful сервера могут располагаться на разных уровнях, при этом каждый сервер взаимодействует только с ближайшими уровнями и не связан запросами с другими.

Предоставление кода по запросу (Code on Demand). Серверы могут отправлять клиенту код (например, скрипт для запуска видео). Так общий код приложения или сайта становится сложнее только при необходимости.

Начало от нуля (Starting with the Null Style). Клиент знает только одну точку входа на сервер. Дальнейшие возможности по взаимодействию обеспечиваются сервером.

Анализ существующих аналогов

Основной аналог, откуда бралась идея, - это сайт bycard.by, самый популярный в нашей стране сайт для таких целей. Он представляет из себя огромную платформу, частичку которой автор и попытался реализовать в данном курсовом проекте за IV семестр.



Постановка задачи

Основной задачей ставится проведение настоящей параллельной разработки серверной и клиентской части fullstack-приложения с учётом всех нюансов разработки как сервера, так и клиента, так и используемых технологий.
Необходимо уделить достаточное внимание следующим каким-то базовым критериям:

- Разработка приятного для пользования пользовательского интерфейса
- Прорабатывание всей логики клиентской части
  - Навигация
  - Верификация форм
  - Кэширование уже полученных данных с сервера
  - Максимальная декомпозиция компонентов
- Разработка нужных моделей в базе данных
- Создание грамотного набора путей для запросов на сервер
- Вопросы авторизации и аутентификации
- Логирование

- Тестирование серверной части
- Др.

Разработка приложения

Среда разработки (используемое ПО)

В качестве сред разработки были выбраны продукты от JetBrains: JetBrains WebStorm и JetBrains PyCharm с бесплатной студенческой лицензией. Также была использована вспомогательная программа Postman для тестирования запросов к серверу и программа PgAdmin для начальной настройки базы данных PostgreSQL. Ну и, конечно же, веб-браузер. В моём случае – Microsoft Edge.

Требования к программе

Программа должна иметь интуитивно понятный интерфейс, корректно уведомлять пользователя о всех успехах/неудачах. Например, при регистрации, входе в систему или попытке найти несуществующий фильм или кинотеатр.
Программный продукт должен обеспечивать должную безопасность сохранности данных пользователя при авторизации, аутентификации.

Используемые технологии, языки программирования и обоснование того или иного выбора

Для начала клиентская часть.
Используемый стек – TypeScript, ReactJS, MobX.
TypeScript был выбран вместо обычного JavaScript ввиду наличия более-менее строгой типизации и удобства разработки. Уже на этапе написания кода можно отловить 90% всех потенциальных ошибок с типами, которые могут возникнуть во время выполнения программы. Это очень удобно и при наличии навыков разработки на TypeScript очень ускоряет процесс разработки.
ReactJS был выбран из-за наличия у автора опыта разработки с использованием этой библиотеки.
MobX был выбран из-за крайней простоты в использовании. Помогает не писать дополнительный и часто в дальнейшем непонятный код и, следовательно, сэкономить время. Во многих настоящих проектах в последние годы в качестве state-менеджера используется именно MobX, а не Redux. Также сыграл свою роль факт, что MobX написан с нуля на

TypeScript и, следовательно, там “встроенная” типизация, а ещё интегрируется с React более легко. В Redux же иногда необходимо дополнительно прописывать типы и т.д.
Теперь о серверной части.
Используемый стек: Python, Django, Django Rest Framework.
Изначально использовался язык программирования C++ и микрофреймворк Crow. Однако в данном случае оказалась важна непосредственно скорость разработки, поэтому спустя некоторое время функционал был переписан на Python + Django. Опыта работы с этим фреймворком Django и его “расширением” Django Rest Framework у автора до этого проекта не было — поэтому пришлось с нуля освоить основы работы с ними. Однако суммарное время всё равно по итогу, по мнению автора, меньше, чем если бы он писал на C++, Crow и каких-нибудь дополнительных библиотеках, которые пришлось бы непременно устанавливать.

1.1. Генерация и соединение “комнат”

Теперь мы создаем комнату случайного размера в каждом листе дерева. Конечно, комната должна находиться в соответствующем подземелье. Благодаря дереву у нас не может быть двух перекрывающихся комнат.

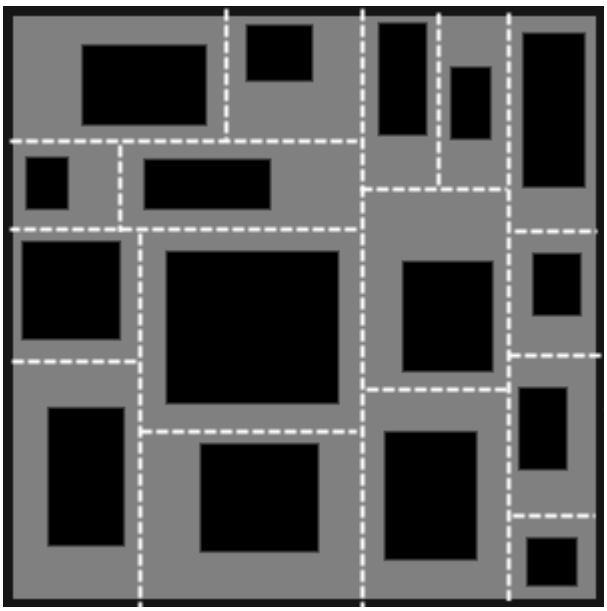


Рисунок 2.4 - Комнаты в листьях дерева

Чтобы построить коридоры, мы перебираем все листья дерева, соединяя каждый лист с его сестрой.

Рисунок 2.5 - Соединение подземелий 4 уровня

Теперь мы поднимаемся на один уровень в дереве и повторяем процесс для родительских подобластей.

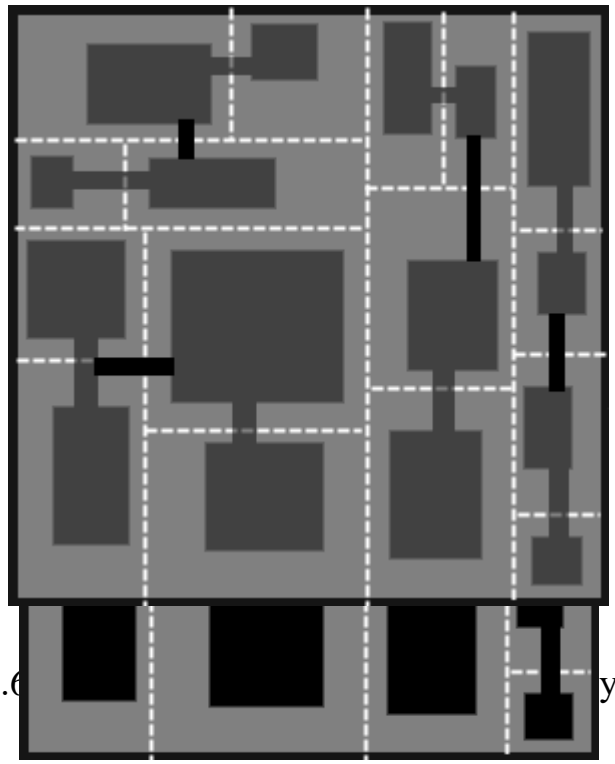


Рисунок 2.6 уровня

Мы повторяем процесс до тех пор, пока не подключим первые два подподземелья A и B:

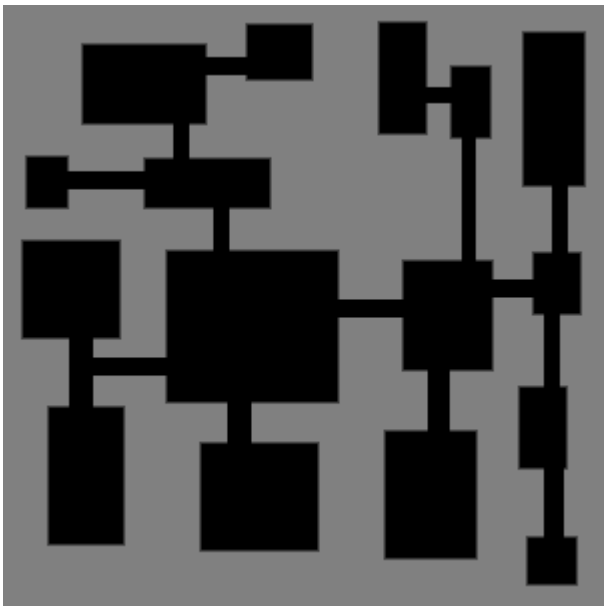


Рисунок 2.7 - готовое подземелье

1.2. Объекты-участники. Размещение на QGraphicsScene

Всего в программе используется 8 типов объектов(названия по классам):
1. **HeroSlime** - главный персонаж от лица которого игрок будет проходить игру.
2. **Altar** - ключевой объект в прохождении. На уровень создаётся всего 3 экземпляра.
3. **Exit** - объект для успешного окончания прохождения (1 экземпляр).
4. **Ghost** - враждебный объект со свойством передвижения (6 экземпляров).
5. **Web** - объект-ловушка со свойством замедления (20 экземпляров).
6. **Spears** - объект-ловушка (20 экземпляров).
7. **Speedboost** - объект-баф со свойством ускорения (3 экземпляра).
8. **Lava** - объект-ловушка (12 экземпляров).

Для правильного размещения на QGraphicScene необходимо соблюдать определённый порядок добавления на сцену. Сначала идут самые крупные объекты, располагаясь на самом дальнем слое. Однако, перед тем как добавить второстепенные объекты добавляются объекты типа altar и exit. Объект типа heroslime добавляется в последнюю очередь из-за необходимости расположения на самом верхнем слое. Для размещения была разработана специальная функция colliding, а также необходимые функции addExit, addAltars, addWebs, addSpears, addLavas, addSpeedies, addGhosts.
Все объекты хранятся в списках типа QList. Помещаются в него с помощью add-функций.

1.3. Перемещение и взаимодействие объектов

Только два типа объектов обладают свойством перемещения:
Ghost
HeroSlime

Перемещение по сцене осуществляется с помощью QKeyEvent, который считывает Объектом типа Ghost невозможно управлять с помощью

клавиатуры. Его скорость и направления определяются рандомно для создания эффекта резкости, непредсказуемости.

Для ограничения перемещения в объектах Ghost и HeroSlime есть объект класса QImage, который представляет собой копию карты подземелья, что создавалась с помощью дерева. Программа считывает данные о местоположении объектов на сцене и сравнивает их с картой. Если цвет пикселя местоположения объекта не совпадает с нужным, то объект перемещается на своё предыдущее размещение.нажатие клавиш и с помощью структуры switch управление происходит с помощью клавиш “W”, “A”, “S”, “D” (Русская раскладка не поддерживается).
При нажатии на клавиши с помощью встроенного таймера у HeroSlime меняются коэффициенты перемещения. Каждую 1000/60 секунды коэффициенты сбрасываются и считывание происходит сначала. Также для определения скорости у HeroSlime есть специальные показатели скорости speed, xspeed, yspeed.

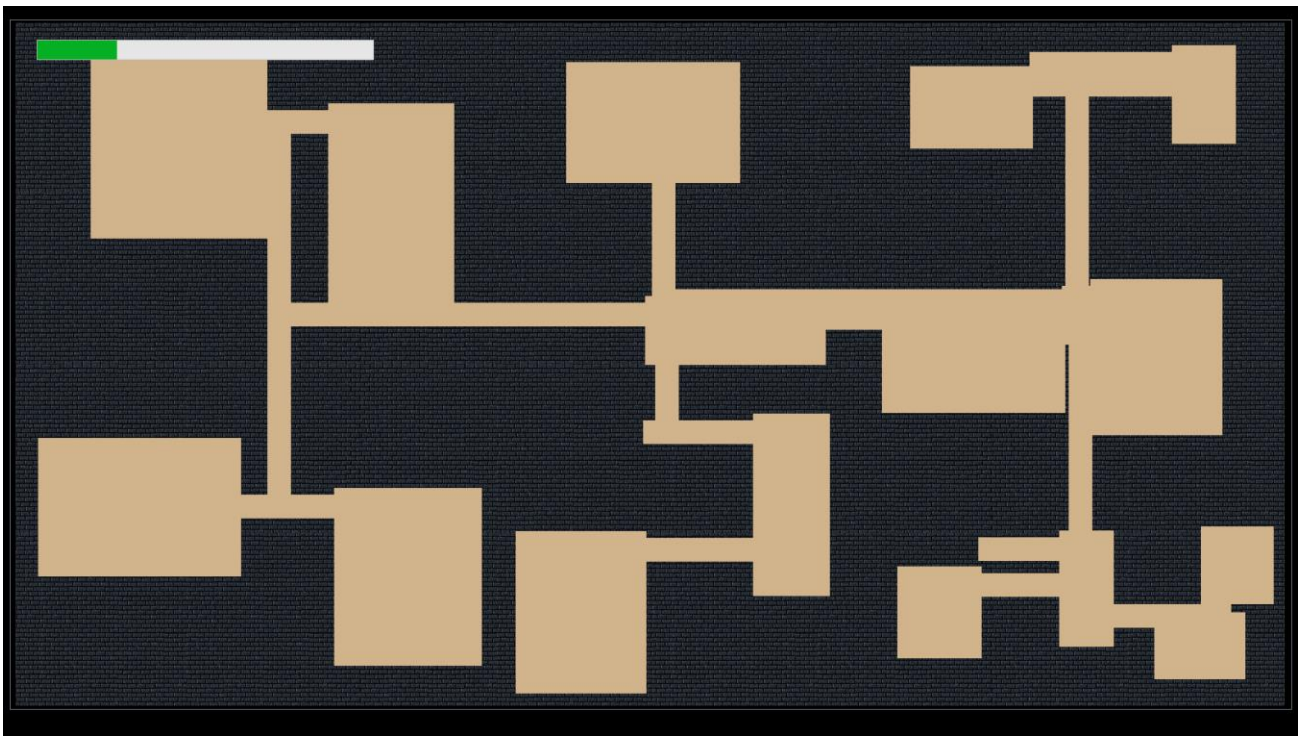


Рисунок 2.8 - пример карты подземелья и ограничивающей области перемещения

Для создания взаимодействия используется QTimerEvent, который на каждый счёт таймера вызывает определённые методы внутри основного класса. В случае с HeroSlime он проверяет, находится ли объект в зоне действия остальных. Если да, то взаимодействие несёт определённый эффект. (Web - уменьшает скорость, Lava - урон, Ghost - урон, Spear - урон, SpeedBoost- увеличение скорости, Exit - завершение игры).

1.4. Отрисовка объектов в движении

Для отрисовки объектов используется особенность Qt - сигналы и слоты. Соединив слот интервала таймера и виртуальный метод advance мы получим вызываемый каждый интервал метод отрисовки.
Для половины объектов существуют разные состояния:

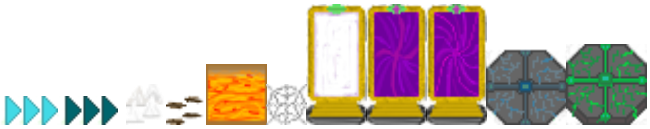


Рисунок 2.9 - пример состояния объектов.

Также для объектов типа Ghost и HeroSlime существуют состояния движения в зависимости от xspeed и yspeed:
HeroSlime:

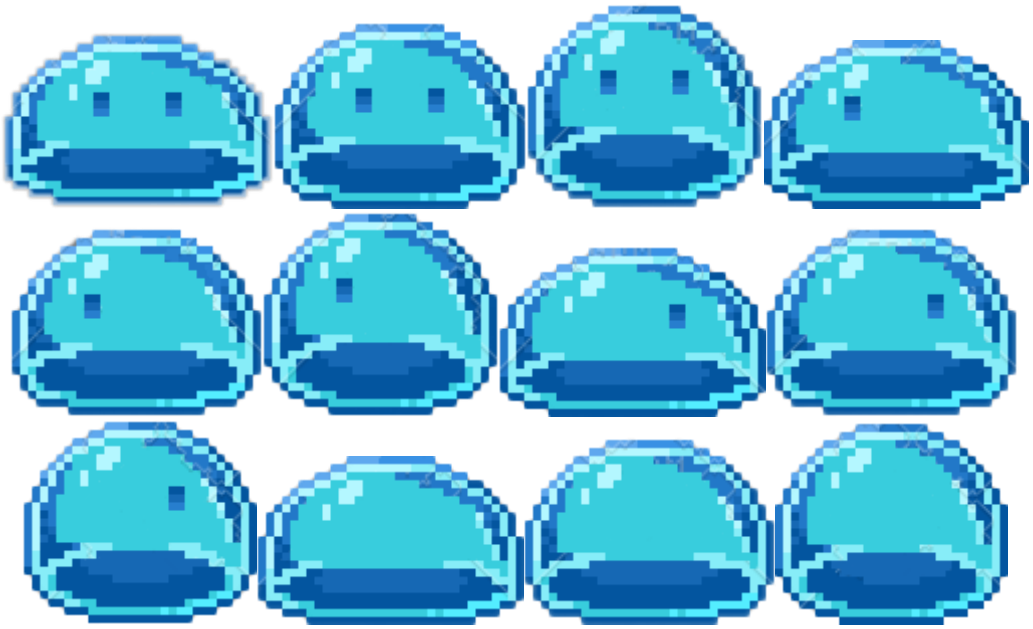


Рисунок 2.10 - все состояния HeroSlime

Ghost:

Рисунок 2.11 - все состояния Ghost

Прототипы и удалённые объекты:

Рисунок 2.12 - прототип Ghost

Рисунок 2.13 - HolySlime

2. Методика работы с полученной программой

При запуске программы сначала появляется стартовое окно с двумя кнопками без фона(“Start”, “Exit”). Start осуществляет начало работы алгоритма и генерацию подземелья, Exit - безопасный выход.

Рисунок 3.1 - стартовый экран

После нажатия кнопки происходит перенос в другое окно - gameProc.
В нём отображается игровой персонаж с эффектом scale(масштабирование) и наложенным градиентом для создания эффекта ограничения видимости.

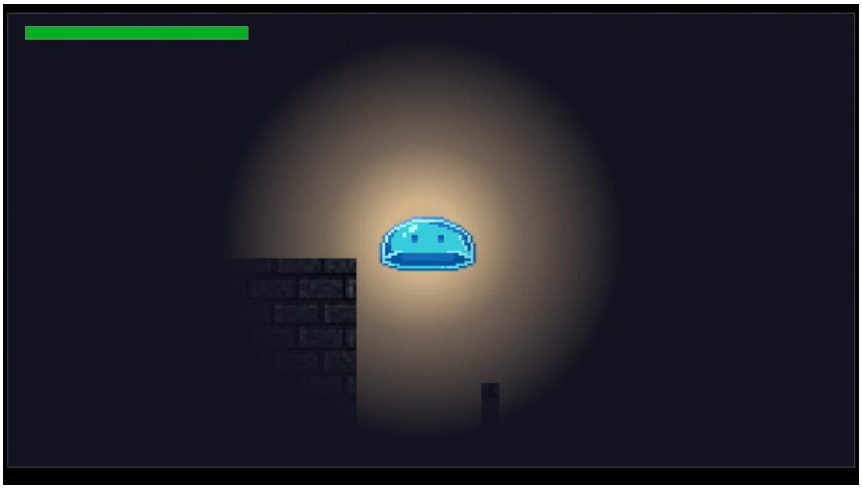


Рисунок 3.2 - Игровой процесс

При получении урона зелёная полоса уменьшается. Если она станет полностью белой, то игра завершиться. Или же игрок может активировать все объекты типа Altar для успешного окончания игры.
Следовательно, существует две концовки:

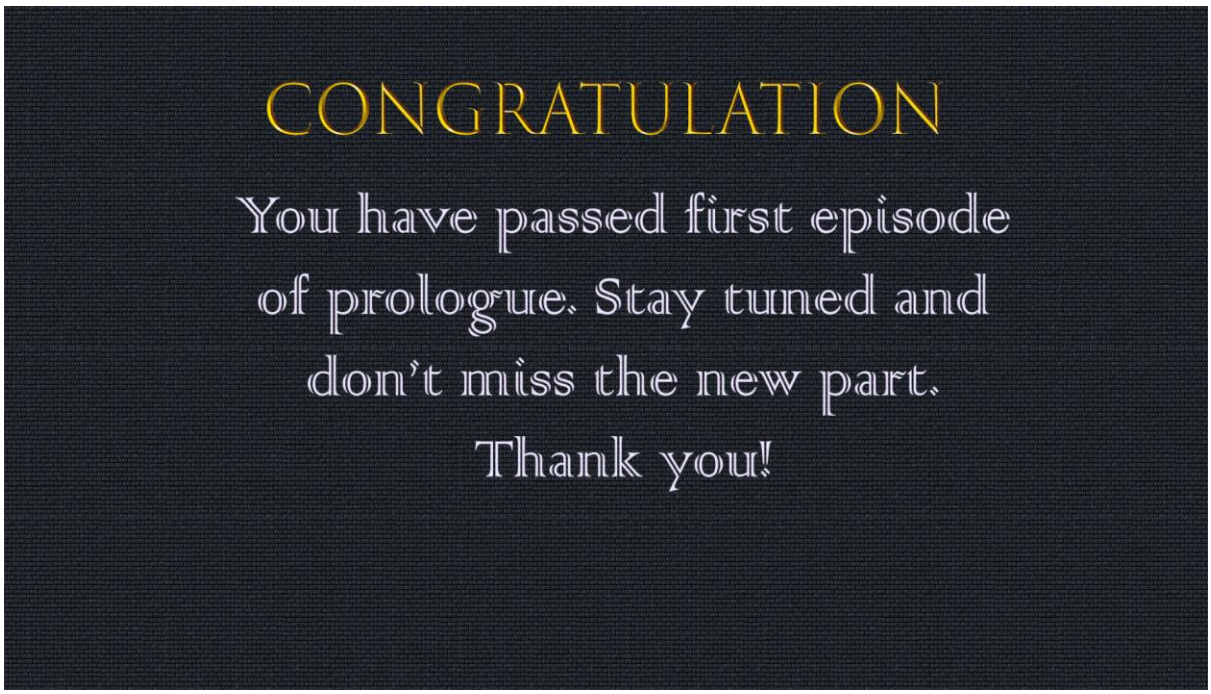


Рисунок 3.3 - хорошая концовка

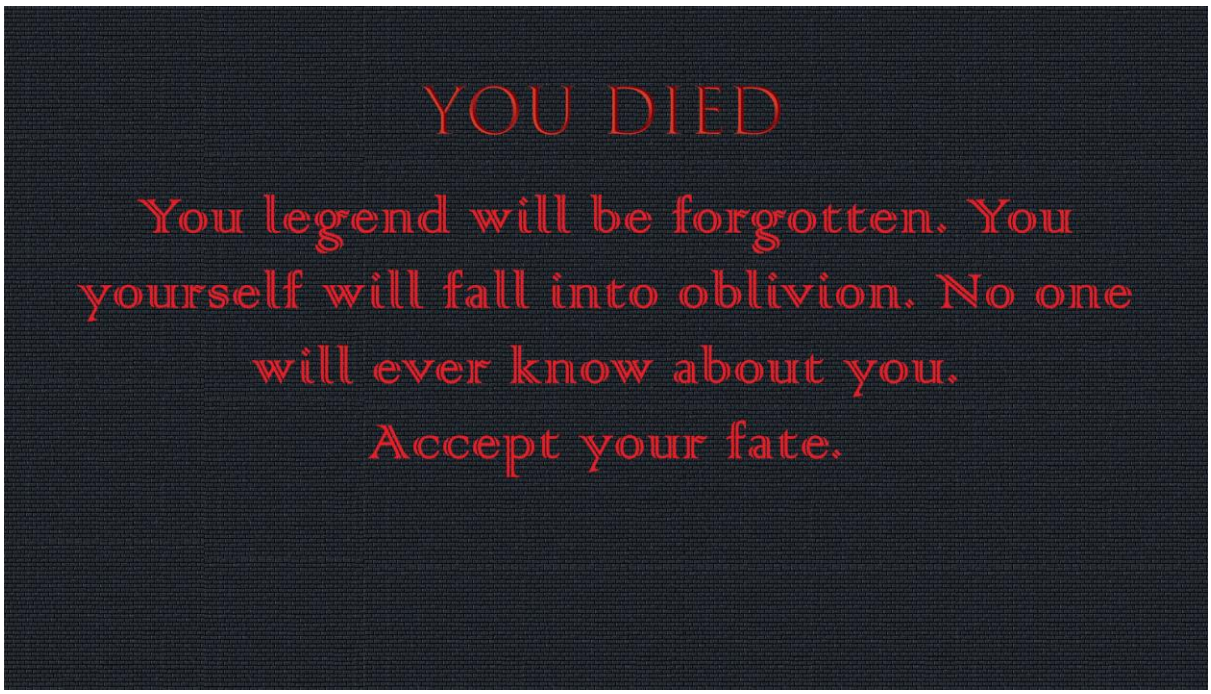


Рисунок 3.4 - плохая концовка

Также в игре присутствуют титры, которые появляются в самом конце:

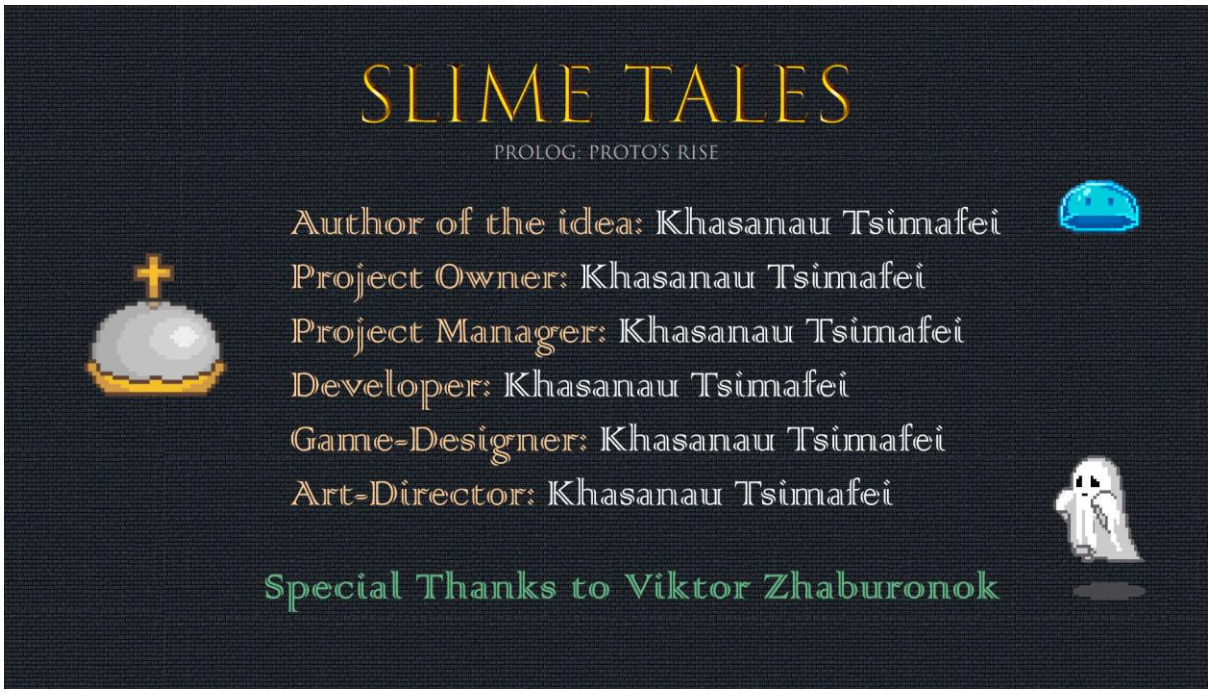


Рисунок 3.5 - Титры

Далее игрок попадает снова на стартовый экран, где может выйти из приложения.

Заключение

В результате работы на языке программирования C++ в среде Qt было разработано Roguelike-приложение с динамической генерацией уровня и случайным расположением объектов. Созданный алгоритм можно будет применять в следующих работах, совершенствуя его. Также была создана основа для 2D-игр с перемещением и взаимодействием объектов, сменой их состояний.

Литература

[1] Qt-documentation [Электронный ресурс]: Документация по всем классам . – Электронные данные. – Режим доступа: https://doc.qt.io

[2] Roguebasin [Электронный ресурс]: Основы разработки Roguelike приложения на языках C++, Java. – Электронные данные. – Режим доступа: http://roguebasin.roguelikedevelopment.org

Приложение 1. Исходный код

gameProc.h

```
#ifndef GAMEPROC_H
#define GAMEPROC_H

#include "altar.h"
#include <QMainWindow>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QImage>
#include "heroslime.h"
#include "spear.h"
#include "lava.h"
#include "web.h"
#include"exit.h"
#include "aftergame.h"
#include "speedboost.h"
#include <QTimer>
#include <QKeyEvent>
#include <QGraphicsItemAnimation>
#include <QGraphicsView>
#include "dividetree.h"
#include <QMediaPlayer>
#include <QMediaPlaylist>
#include "ghost.h"

#include<QWidget>
#include <QPainter>
#include <QStyleOptionGraphicsItem>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
  Q_OBJECT

public:
  MainWindow(QWidget *parent = nullptr);
  ~MainWindow();
  int timerId, poc = 0;
  void spawnSlime(DivideTree *param);
  void addAltars();
  void addSpears();
```

```
    void addLava();
    void addWebs();
    void addSpeeds();
    void addExit();
    void addGhosts();
    void addGrad();
    bool colliding(double x, double y, double n);
    Room* randRoom();
    Room *leftFind();
private:
    Ui::MainWindow *ui;
    QGraphicsScene *scene;
    QGraphicsView *view;
    HeroSlime *slime;
    QTimer *animatic, *damage;
    QTimerEvent * ar;
    QMediaPlayer *sound;
    QMediaPlaylist *playlist;
    DivideTree *param;
    QGraphicsPixmapItem *grad;
    QRandomGenerator *ht;
    class afterGame* past;
    QList<Altar*> altars;
    QList<Spear*> spears;
    QList<Lava*> lavas;
    QList<SpeedBoost*> speedies;
    QList<Web*> webs;
    QList<Ghost*> ghosts;
    bool ex  =false;
    Altar *altar;
    Ghost *ghost;
    Spear *rSpear;
    Lava *lava;
    Web *web;
    Exit *exit;
    SpeedBoost *speed;
    QWidget *widget;
    QStyleOptionGraphicsItem *option;
    QPainter *painter;
    protected:
        void timerEvent(QTimerEvent *event);
};


#endif // GAMEPROC_H
```

gameProc.cpp

```
#include "gameProc.h"
#include "ui_gameProc.h"



MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
   , ui(new Ui::MainWindow)
{
   ui->setupUi(this);
   QPalette p = palette();
   p.setColor(QPalette::Highlight,QColor(89, 128,200));
   p.setColor(QPalette::Base,QColor(89, 128,200, 0));
   ui->progressBar->setPalette(p);
   view = ui->graphicsView;
   scene = new QGraphicsScene(0,0, 1881, 1010);
   view->setScene(scene);
   scene->setStickyFocus(true);
   this->scene->setSceneRect(0,0, 1881, 1010);
   ht = QRandomGenerator::global();
   playlist = new QMediaPlaylist;
   for( int i =0; i < 3; i++)
   {
      playlist->addMedia(QUrl("C:/Users/briti/Pictures/Project/Terraria Music - Underground
Corruption (320  kbps).mp3"));
      playlist->addMedia(QUrl("C:/Users/briti/Pictures/Project/Terraria Music - Dungeon (320
kbps).mp3"));
   }

   playlist->setCurrentIndex(1);
   sound = new QMediaPlayer();
   sound->setPlaylist(playlist);
   sound->setVolume(15);
   sound->play();

   animatic = new QTimer(this);
   connect(animatic,SIGNAL(timeout()),scene,SLOT(advance()));
   animatic->start(1000/60);

   param = new DivideTree(1881, 1010);
   param->createTree(param->root, 4);
   param->print(param->root);
   param->connect(param->root);
   grad = new QGraphicsPixmapItem(QPixmap("C:/Users/briti/Pictures/Project/grad.png"));
```

```
    scene->addPixmap(*( param->layer1));

     addAltars();
     addExit();
     addSpears();
     addLava();
     addWebs();
     addSpeeds();
     addGhosts();
    spawnSlime(param);
    view->setViewportUpdateMode(QGraphicsView::FullViewportUpdate);
     scene->addItem(grad);
    timerId = startTimer(0);
    this->showFullScreen();
     view->scale(8,8);
}

MainWindow::
~MainWindow()
{
    param->deleteTree(param->root);
    spears.clear();
    webs.clear();
    altars.clear();
    lavas.clear();
    speedies.clear();
    ghosts.clear();
    delete web;
    delete rSpear;
    delete param;
    delete sound;
    delete scene;
    delete animatic;
    delete playlist;
    delete ui;
}

void MainWindow::spawnSlime(DivideTree *param)
{
    Room *proto = param->root;
    while(proto->pLeft != nullptr)
    {
        proto = proto->pLeft;
    }
    slime = new HeroSlime(proto->roomX + proto->roomW/2, proto->roomY + proto-
>roomL/2);
    slime->map = param->layer1->toImage();
```

```
    scene->addItem(slime);
    slime->setPos(proto->roomX + proto->roomW/2 - 12, proto->roomY + proto->roomL/2 -
10);
    slime->xpos = proto->roomX + proto->roomW/2 - 12;
    slime->ypos =proto->roomY + proto->roomL/2 -10;
}

void MainWindow::addAltars()
{
    Room *prev, *proto = leftFind();
    for(int i = 0; i < 3; i++)
    {
        prev = randRoom();
        if( !prev->parent->pLeft->altar && !prev->parent->pRight->altar && prev != proto)
        {
            prev->altar =  true;
                altar = new Altar(prev->roomX+prev->roomW/2 - 20, prev->roomY+prev->roomL/2
- 20);
                altars.append(altar);
                scene->addItem(altar);
                altar-> setPos(prev->roomX+prev->roomW/2 - 20, prev->roomY+prev->roomL/2 -
20);
        }
        else
        {
            i--;
        }
    }
}

void MainWindow::addSpears()
{
    int  rX, rY;
    Room *prev, *proto = leftFind();
    for(int i =0; i < 20; i++)
    {
        prev = randRoom();
        rX = prev->roomX +ht->bounded(20, prev->roomW) - 20;
        rY = prev->roomY +ht->bounded(20, prev->roomL) - 20;
        if(colliding(rX, rY, 20) && prev->amSpear <3 && proto != prev)
        {
            prev->amSpear++;
            rSpear = new Spear(rX, rY);
            spears.append(rSpear);
            scene->addItem(rSpear);
            rSpear->setPos(rX,rY);
        }
        else
```

```
        {
            i--;
        }
    }
}

void MainWindow::addLava()
{
    int  rX, rY;
    Room *prev, *proto = leftFind();
    for(int i =0; i < 12; i++)
    {
       prev = randRoom();
        rX = prev->roomX +ht->bounded(30, prev->roomW) - 30;
        rY = prev->roomY +ht->bounded(30, prev->roomL) - 30;
        if(colliding(rX, rY, 30) && prev->amLava < 1 && proto != prev)
        {
            prev->amLava++;
            lava = new Lava(rX, rY);
            lavas.append(lava);
            scene->addItem(lava);
            lava->setPos(rX,rY);
        }
        else
        {
            i--;
        }
    }
}

void MainWindow::addWebs()
{
    int  rX, rY;
    Room *prev, *proto = leftFind();
    for(int i =0; i < 20; i++)
    {
       prev = randRoom();
        rX = prev->roomX +ht->bounded(20, prev->roomW) - 20;
        rY = prev->roomY +ht->bounded(20, prev->roomL) - 20;
        if(colliding(rX, rY, 20) && prev->amWeb < 3 && prev != proto)
        {
            prev->amWeb++;
            web = new Web(rX, rY);
            webs.append(web);
            scene->addItem(web);
            web->setPos(rX,rY);
        }
        else
```

```
        {
            i--;
        }
    }
}

void MainWindow::addSpeeds()
{
    int  rX, rY;
    Room *prev, *proto = leftFind();
    for(int i =0; i < 3; i++)
    {
        prev = randRoom();
        rX = prev->roomX +ht->bounded(30, prev->roomW) - 30;
        rY = prev->roomY +ht->bounded(30, prev->roomL) - 30;
        if(colliding(rX, rY, 30) && !prev->speedboost && proto != prev)
        {
            prev->speedboost = true;
            speed = new SpeedBoost(rX, rY);
            speed->active = true;
            speedies.append(speed);
            scene->addItem(speed);
            speed->setPos(rX,rY);
        }
        else
        {
            i--;
        }
    }
}

void MainWindow::addExit()
{
    Room *prev = leftFind();

    bool run = true;
    while(run)
    {
        prev = randRoom();
        if( !prev->parent->pLeft->altar && !prev->parent->pRight->altar )
        {
                exit = new Exit(prev->roomX+prev->roomW/2 - 20, prev->roomY+prev->roomL/2 -
40);
                scene->addItem(exit);
                exit-> setPos(prev->roomX+prev->roomW/2 - 20, prev->roomY+prev->roomL/2 -
40);
                run = false;
        }
```

```
    }
}

void MainWindow::addGhosts()
{
    int  rX, rY;
    Room *prev, *proto = leftFind();
    for(int i =0; i < 6; i++)
    {
        prev = randRoom();
        rX = prev->roomX +ht->bounded(30, prev->roomW) - 30;
        rY = prev->roomY +ht->bounded(30, prev->roomL) - 30;
        if( !prev->ghost && !prev->parent->pLeft->ghost && !prev->parent->pLeft->ghost &&
(prev->altar || proto != prev) )
        {
            prev->ghost = true;
            ghost = new Ghost(rX, rY);
            ghost->map = param->layer1->toImage();
            ghosts.append(ghost);
            scene->addItem(ghost);
            ghost->setPos(rX,rY);
        }
        else
        {
            i--;
        }
    }
}

bool MainWindow::colliding(double x, double y, double n)
{
    for(int i = 0; i < spears.size(); i++)
    {
        if((x < spears[i]->x + 20 && x > spears[i]->x && y < spears[i]->y + 20 && y > spears[i]->y)
                ||(x + n< spears[i]->x + 20 && x + n> spears[i]->x && y +n< spears[i]->y + 20 && y +
n> spears[i]->y)
                ||(x< spears[i]->x + 20 && x> spears[i]->x && y +n< spears[i]->y + 20 && y + n>
spears[i]->y)
                ||(x + n< spears[i]->x + 20 && x + n> spears[i]->x && y< spears[i]->y + 20 && y>
spears[i]->y))
        {
            return false;
        }
    }

    for(int i = 0; i < altars.size(); i++)
    {
        if((x < altars[i]->x + 40 && x > altars[i]->x && y < altars[i]->y + 40 && y > altars[i]->y)
```

```
            ||(x + n< altars[i]->x + 40 && x + n> altars[i]->x && y + n< altars[i]->y + 40 && y +
n> altars[i]->y)
            ||(x < altars[i]->x + 40 && x > altars[i]->x && y + n< altars[i]->y + 40 && y + n>
altars[i]->y)
            ||(x + n< altars[i]->x + 40 && x + n> altars[i]->x && y < altars[i]->y + 40 && y >
altars[i]->y))
      {
         return false;
      }
   }

   for(int i = 0; i < lavas.size(); i++)
   {
      if((x < lavas[i]->x + 30 && x > lavas[i]->x && y < lavas[i]->y + 30 && y > lavas[i]->y)
            ||(x + n< lavas[i]->x + 30 && x + n> lavas[i]->x && y + n< lavas[i]->y + 30 && y + n>
lavas[i]->y)
            ||(x < lavas[i]->x + 30 && x > lavas[i]->x && y + n< lavas[i]->y + 30 && y + n>
lavas[i]->y)
            ||(x + n< lavas[i]->x + 30 && x + n> lavas[i]->x && y < lavas[i]->y + 30 && y >
lavas[i]->y))
      {
         return false;
      }

   }

   for(int i = 0; i < speedies.size(); i++)
   {
      if((x < speedies[i]->x + 30 && x > speedies[i]->x && y < speedies[i]->y + 30 && y >
speedies[i]->y)
            ||(x + n< speedies[i]->x + 30 && x + n> speedies[i]->x && y +n< speedies[i]->y + 30
&& y + n> speedies[i]->y)
            ||(x < speedies[i]->x + 30 && x > speedies[i]->x && y +n< speedies[i]->y + 30 && y
+ n> speedies[i]->y)
            ||(x + n< speedies[i]->x + 30 && x + n> speedies[i]->x && y < speedies[i]->y + 30
&& y > speedies[i]->y))
      {
         return false;
      }
   }

   for(int i = 0; i < webs.size(); i++)
   {
      if((x < webs[i]->x + 20 && x > webs[i]->x && y < webs[i]->y + 20 && y > webs[i]->y)
            ||(x + n< webs[i]->x + 20 && x + n> webs[i]->x && y +n< webs[i]->y + 20 && y + n>
webs[i]->y)
            ||(x < webs[i]->x + 20 && x > webs[i]->x && y +n< webs[i]->y + 20 && y + n>
webs[i]->y)
```

```
            ||(x + n< webs[i]->x + 20 && x + n> webs[i]->x && y< webs[i]->y + 20 && y >
webs[i]->y))
        {
            return false;
        }
    }

    if((x < exit->x + 30 && x > exit->x && y < exit->y + 60 && y > exit->y)
            ||(x + n< exit->x + 30 && x + n> exit->x && y +n< exit->y + 60 && y + n> exit->y)
            ||(x < exit->x + 30 && x > exit->x && y +n< exit->y + 60 && y + n> exit->y)
            ||(x + n< exit->x + 30 && x + n> exit->x && y< exit->y + 60 && y > exit->y))
    {
        return false;
    }
    return true;
}

Room *MainWindow::randRoom()
{
    int choice;
    //ht = QRandomGenerator::global();
    Room *proto, *prev;
        proto = param->root;
        while(proto != nullptr)
        {
            prev = proto;
            choice =  ht->bounded(0,2);
            if(choice == 1)
            {
                proto = proto->pRight;
            }
            else
            {
                 proto = proto->pLeft;
            }
        }
        return prev;
}

Room *MainWindow::leftFind()
{
    Room *proto = param->root;
    while(proto->pLeft != nullptr)
    {
        proto = proto->pLeft;
    }
    return proto;
}
```

```
void MainWindow::timerEvent(QTimerEvent *event)
{
  view->centerOn(slime->xpos+15, slime->ypos +7);
  grad->setPos(slime->xpos - 238+15, slime->ypos - 128+7);
  for(int i =0; i < altars.size(); i++)
  {
     if(slime->xpos +15< altars[i]->x +40 && slime->xpos + 15> altars[i]->x
           &&  slime->ypos +7 > altars[i]->y   && slime->ypos +7< altars[i]->y + 40)
     {
        altars[i]->active = false;
     }
  }

  if(!altars[0]->active && !altars[1]->active && !altars[2]->active)
  {
     exit->active = true;
  }
  if(poc == 10000)
  {
     for(int i =0; i < spears.size(); i++)
     {
        if(slime->xpos +15< spears[i]->x +20 && slime->xpos + 15> spears[i]->x
              &&  slime->ypos +7 > spears[i]->y   && slime->ypos +7< spears[i]->y + 20)
        {
           spears[i]->active = true;
           slime->health -= 3;
        }
     }

     for(int i =0; i < speedies.size(); i++)
     {
        if(slime->xpos +15< speedies[i]->x +30 && slime->xpos + 15> speedies[i]->x
              &&  slime->ypos +7 > speedies[i]->y   && slime->ypos +7< speedies[i]->y + 14
&& speedies[i]->active)
        {
           speedies[i]->active = false;
           slime->speed += 0.5;
        }
     }

     for(int i =0; i < webs.size(); i++)
     {
        if(slime->xpos +15< webs[i]->x +20 && slime->xpos + 15> webs[i]->x
              &&  slime->ypos +7 > webs[i]->y   && slime->ypos +7< webs[i]->y + 20)
        {
           scene->removeItem(webs[i]);
           webs.removeAt(i);
```

```
            if(slime->speed >0.1)
            {
                slime->speed -= 0.3;
            }
        }
    }

    for(int i =0; i < lavas.size(); i++)
    {
        if(slime->xpos +15< lavas[i]->x +30 && slime->xpos + 15> lavas[i]->x
                &&  slime->ypos +7 > lavas[i]->y   && slime->ypos +7< lavas[i]->y + 30)
        {
            slime->health -= 5;
        }
    }

    for(int i =0; i < ghosts.size(); i++)
    {
        if(slime->xpos +15< ghosts[i]->xpos +30 && slime->xpos + 15> ghosts[i]->xpos
                &&  slime->ypos +7 > ghosts[i]->ypos   && slime->ypos +7< ghosts[i]->ypos + 30)
        {
            slime->health -= 10;
        }
    }
    poc = 0;
}
else
{
    poc++;
}

ui->progressBar->setValue(slime->health);
if(slime->health<= 0 && !ex )
{
    ex = true;
    past = new afterGame(nullptr, false);
    past->showFullScreen();
    sound->stop();
      this->close();
    this->~MainWindow();
}

if(slime->xpos +15< exit->x +30 && slime->xpos + 15> exit->x
        &&  slime->ypos +7 > exit->y   && slime->ypos +7< exit->y + 40 && exit->active &&
!ex)
{
    ex = true;
    past = new afterGame();
```

```
        past->showFullScreen();
         sound->stop();
        this->close();
        this->~MainWindow();
    }
}
```

heroslime.h

```
#ifndef HEROSLIME_H
#define HEROSLIME_H

#include <QGraphicsObject>
#include <QPainter>
#include <QKeyEvent>
#include <QWidget>
#include <QStyleOptionGraphicsItem>
#include <QImage>


class HeroSlime: public QGraphicsObject
{
public:
    int health = 100;
    double speed = 2;
    double xspeed = 0, yspeed =0;
    double xpos = 0, ypos = 0;
    double prevx =0,prevy =0;
    double w, a, s, d;
    QImage map;
    QWidget *widget;
    QStyleOptionGraphicsItem *option;
    QPainter *painter;
    HeroSlime(double xpos, double ypos);

    // QGraphicsItem interface
public:
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget);


    // QGraphicsItem interface
public:
    QRectF boundingRect() const;

    // QGraphicsItem interface
protected:
    void keyPressEvent(QKeyEvent *event);
    void keyReleaseEvent(QKeyEvent *event);
```

```
    // QGraphicsItem interface
public:
    void advance(int phase);
};

#endif // HEROSLIME_H
```

heroslime.cpp

```
#include "heroslime.h"



HeroSlime::HeroSlime(double xpos, double  ypos)
{
    this->xpos = xpos;
    this->ypos = ypos;
    w = a = s = d = 0;
    painter = new QPainter;
    widget = new QWidget;
    option = new QStyleOptionGraphicsItem;
    setFlag(QGraphicsObject::ItemIsFocusable);
    setFlag(QGraphicsItem::ItemIsMovable);
    setFocus();
}
void HeroSlime::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget)
{
    if( w == 0 && s == 0 && a == 0 && d == 0 )
    {
        painter->drawPixmap(0, 0, 28,
16,QPixmap("C:/Users/briti/Pictures/Project/Slime/Back1.png"));
    }
    else if(w != 0)
    {
        if(w >= 1 && w <=5)
        {
            painter->drawPixmap(0, 0, 28,
16,QPixmap("C:/Users/briti/Pictures/Project/Slime/Up1.png"));
        }
        else if(w >= 6 && w <= 10)
        {
             painter->drawPixmap(5, 0, 22,
20,QPixmap("C:/Users/briti/Pictures/Project/Slime/Up2.png"));
        }
        else if(w >= 11 && w <= 15)
```

```
    {
        painter->drawPixmap(7, 0, 18,
22,QPixmap("C:/Users/briti/Pictures/Project/Slime/Up3.png"));
    }
  }
  else if(s != 0)
  {
    if(s >= 1 && s <= 5)
    {
        painter->drawPixmap(0, 0, 28,
16,QPixmap("C:/Users/briti/Pictures/Project/Slime/Back1.png"));
    }
    else if(s >= 6 && s <=10 )
    {
        painter->drawPixmap(5, 0, 22,
20,QPixmap("C:/Users/briti/Pictures/Project/Slime/Back2.png"));
    }
    else if(s >= 11 && s <= 15)
    {
        painter->drawPixmap(7, 0, 18,
22,QPixmap("C:/Users/briti/Pictures/Project/Slime/Back3.png"));
    }
  }
  else if(d != 0)
  {
    if( d >= 1 && d <=5)
    {
        painter->drawPixmap(0, 0, 28,
16,QPixmap("C:/Users/briti/Pictures/Project/Slime/Right1.png"));
    }
    else if( d >= 6 && d <=10)
    {
        painter->drawPixmap(0, 0, 22,
20,QPixmap("C:/Users/briti/Pictures/Project/Slime/Right2.png"));
    }
    else if( d >= 11 && d <= 15)
    {
        painter->drawPixmap(0, 0, 18,
22,QPixmap("C:/Users/briti/Pictures/Project/Slime/Right3.png"));
    }
  }
  else if(a != 0)
  {

    if( a >= 1 && a <=5)
    {
        painter->drawPixmap(0, 0, 28,
16,QPixmap("C:/Users/briti/Pictures/Project/Slime/Left1.png"));
```

```
        }
        else if( a >= 6 && a <=10)
        {
            painter->drawPixmap(0, 0, 22,
20,QPixmap("C:/Users/briti/Pictures/Project/Slime/Left2.png"));
        }
        else if( a >= 11 && a <= 15)
        {
            painter->drawPixmap(0, 0, 18,
22,QPixmap("C:/Users/briti/Pictures/Project/Slime/Left3.png"));
        }
    }
}

QRectF HeroSlime::boundingRect() const{}

void HeroSlime::keyPressEvent(QKeyEvent *event)
{
    switch (event->key()) {
    case Qt::Key_W:
        w++;
        a = s = d =0;
        if(w > 15)
        {
            w = 1;
        }
        if(xspeed != 0)
        {
            xspeed = 0.5*xspeed;
        }
        //xspeed = 0;
        yspeed = -speed;
        break;
    case Qt::Key_A:
        a++;
        w =s =d =0;
        if(a > 15)
        {
            a = 1;
        }
        if(yspeed != 0)
        {
            yspeed = 0.5*yspeed;
        }
         // yspeed = 0;
        xspeed = -speed;
        break;
    case Qt::Key_S:
```

```
        s++;
        a =w =d =0;
        if(s > 15)
        {
            s = 1;
        }
        if(xspeed != 0)
        {
            xspeed = 0.5*xspeed;
        }
          //xspeed = 0;
        yspeed  = speed;
        break;
    case Qt::Key_D:
        d++;
        w = a = s =0;
        if(d > 15)
        {
            d = 1;
        }
        if(yspeed != 0)
        {
            yspeed = 0.5*yspeed;
        }
         // yspeed = 0;
        xspeed = speed;
        break;
    default:
        break;
    }
}

void HeroSlime::keyReleaseEvent(QKeyEvent *event)
{
    switch (event->key()) {
    case Qt::Key_W:
        yspeed = 0;
        break;
    case Qt::Key_A:
        xspeed = 0;
        break;
    case Qt::Key_S:
        yspeed  = 0;
        break;
    case Qt::Key_D:
        xspeed = 0;
        break;
    default:
```

```
        break;
    }
}

void HeroSlime::advance(int phase)
{
    paint(painter, option, widget);
    if(phase)
    {
        if(map.pixelColor(xpos+12 , ypos +15) == QColor(210, 180, 140) &&
map.pixelColor(xpos+12 , ypos ) == QColor(210, 180, 140)
                && map.pixelColor(xpos , ypos +7) == QColor(210, 180, 140) &&
map.pixelColor(xpos+28 , ypos  +7) == QColor(210, 180, 140))
        {
            prevx = xpos;
            prevy = ypos ;
             moveBy(xspeed, yspeed);
             xpos+=xspeed;
             ypos+=yspeed;
        }
        else
        {
            w = s = d = a = 0;
            this->setPos(prevx, prevy);
            xpos = prevx;
            ypos = prevy;
            setFocus();
        }
        this->update();
    }
}
```

altar.h

```
#ifndef ALTAR_H
#define ALTAR_H

#include <QGraphicsObject>


#include <QPainter>
#include <QStyleOptionGraphicsItem>

class Altar: public QGraphicsItem
{
public:
    double x = 0, y = 0;
```

```
    bool active = true;
   Altar(double x, double y);

   // QGraphicsItem interface
public:
   QRectF boundingRect() const;
   void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget);
};



#endif // ALTAR_H
```

altar.cpp

```
#include "altar.h"


Altar::Altar(double x, double y)
{
   this->x = x;
   this->y = y;
}

QRectF Altar::boundingRect() const
{
}

void Altar::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget)
{
   if(active)
   {
       painter->drawPixmap(0, 0, 40,
40,QPixmap("C:/Users/briti/Pictures/Project/altar/actAltar.png"));
   }
   else
   {
       painter->drawPixmap(0, 0, 40,
40,QPixmap("C:/Users/briti/Pictures/Project/altar/disAltar.png"));
   }
}
```

dividetree.h

```
#ifndef DIVIDETREE_H
#define DIVIDETREE_H
#include <room.h>
```

```
#include <QPixmap>
#include <QPainter>
#include <QTime>
#include <QRandomGenerator>


class DivideTree
{
public:
   DivideTree();
    DivideTree(int weight, int length);
      void add(int x, int y, int weight, int length, Room *parent, int left, int roomX = 0 , int roomY
= 0, int roomW = 0, int roomL = 0);
      void createTree(Room *root, int coeff);
      void print(Room *root);
      void connect(Room *root);
      void deleteTree(Room *root);
      Room  *root = nullptr;
      QPixmap  *layer1 = new QPixmap("C:/Users/briti/Pictures/Project/newwalll.png");
     QPixmap lay = layer1-
>scaled(QSize(1881,1010),Qt::IgnoreAspectRatio,Qt::SmoothTransformation);
      QPainter *m_Painter = new QPainter(&lay);
       QRandomGenerator *ht;
};

#endif // DIVIDETREE_H
```

dividetree.cpp

```
#include "dividetree.h"

DivideTree::DivideTree()
{
}

DivideTree::DivideTree( int weight, int length)
{
   root = new Room(0,0,weight,length);
   layer1 = &lay;
   m_Painter->setPen(QPen(QColor(210, 180, 140),1,Qt::NoPen));
   m_Painter->setBrush(QBrush(QColor(210, 180, 140), Qt::SolidPattern));
}

void DivideTree::add(int x, int y, int weight, int length, Room *parent, int left, int roomX , int
roomY, int roomW, int roomL)
{
    Room *proto = new Room(x, y, weight, length,parent, roomX, roomY, roomW, roomL,
nullptr, nullptr);
```

```
    if(left == 1)
    {
        parent->pLeft = proto;
    }
    else
    {
        parent->pRight = proto;
    }
}

void DivideTree::createTree(Room *root, int coeff)
{
   ht = QRandomGenerator::global();
   coeff--;

   int  x1, y1, x2, y2, w1, w2, l1, l2;
   int rx, ry, rw, rh;
   if( coeff >= 0)
   {
      x1 = root->x;
      y1 = root->y;
      if(root->weight  < root->height) //horizontal
      {
         x2 = x1;
         w1 = w2 = root->weight;
         y2 = root->height*(40+ht->bounded(0,30))/100+(y1);
         l1 = abs(y2 - y1);
         l2 = root->height - l1;
      }
      else //vertical
      {
         x2 = root->weight*(40+ht->bounded(0,30))/100+(x1);
         w1 = abs(x2 - x1);
          y2 = y1;
         w2 = root->weight - w1;
         l1 = l2 = root->height;
      }
      if(coeff == 0)
      {
         rx = x1+w1*ht->bounded(5,50)/100;
         ry = y1+l1*ht->bounded(5,50)/100;
         rw = (w1 - (rx - x1))*ht->bounded(50,95)/100;
         rh = (l1 - (ry - y1))*ht->bounded(50,95)/100;
         this->add(x1,y1,w1,l1,root, 1, rx, ry, rw, rh);

         rx = x2+w2*ht->bounded(5,50)/100;
         ry = y2+l2*ht->bounded(5,50)/100;
         rw = (w2 - (rx - x2))*ht->bounded(50,95)/100;
```

```
            rh = (l2 - (ry - y2))*ht->bounded(50,95)/100;
            this->add(x2,y2,w2,l2,root,2,rx,ry,rw,rh);
        }
        else
        {
            this->add(x1, y1, w1, l1, root, 1);
            this->add(x2, y2 ,w2, l2, root, 2);
            createTree(root->pLeft, coeff);
            createTree(root->pRight, coeff);

        }
    }
}

void DivideTree::print(Room *root)
{
    if(root!= nullptr )
    {
        if(root->pLeft == nullptr && root->pRight == nullptr)
        {
            m_Painter->drawRect(root->roomX-5, root->roomY-5, root->roomW+10, root-
>roomL+10);
        }
        else
        {
            print(root->pRight);
            print(root->pLeft);
        }
    }
}

void DivideTree::connect(Room *root)
{
    int maxX, maxY, maxW, maxH;
    if(root!= nullptr && root->pLeft != nullptr && root->pRight!=nullptr )
    {
        connect(root->pRight);
        connect(root->pLeft);

            if(root->weight  < root->height )
            {
                if(root->pLeft->roomX > root->pRight->roomX)
                {
                    m_Painter->drawRect( root->pLeft->roomX +5 , root->pLeft->roomY + root-
>pLeft->roomL, 35,
                                         root->pRight->roomY - (root->pLeft->roomY + root->pLeft-
>roomL));
                    maxX =  root->pLeft->roomX +5;
```

```
                maxY = root->pLeft->roomY + root->pLeft->roomL;
                maxW = 35;
                maxH = root->pRight->roomY - (root->pLeft->roomY + root->pLeft->roomL);
            }
            else
            {
                m_Painter->drawRect(root->pRight->roomX  +5 , root->pLeft->roomY + root->pLeft->roomL, 35,
                                    root->pRight->roomY - (root->pLeft->roomY + root->pLeft->roomL));
                maxX =  root->pRight->roomX  +5;
                maxY = root->pLeft->roomY + root->pLeft->roomL;
                maxW = 35;
                maxH =  root->pRight->roomY - (root->pLeft->roomY + root->pLeft->roomL);
            }

        }
        else
        {
            if(root->pLeft->roomY < root->pRight->roomY)
            {
                m_Painter->drawRect(root->pLeft->roomX + root->pLeft->roomW, root->pRight->roomY + 5,
                                    root->pRight->roomX - (root->pLeft->roomX + root->pLeft->roomW), 35);
                maxX =  root->pLeft->roomX + root->pLeft->roomW;
                maxY = root->pRight->roomY + 5;
                maxW = root->pRight->roomX - (root->pLeft->roomX + root->pLeft->roomW);
                maxH =  35;
            }
            else
            {
                m_Painter->drawRect(root->pLeft->roomX + root->pLeft->roomW, root->pLeft->roomY + 5,
                                    root->pRight->roomX - (root->pLeft->roomX + root->pLeft->roomW), 35);
                maxX =  root->pLeft->roomX + root->pLeft->roomW;
                maxY = root->pRight->roomY + 5;
                maxW = root->pRight->roomX - (root->pLeft->roomX + root->pLeft->roomW);
                maxH =  35;
            }
        }
        root->roomX =  maxX;
        root->roomY =  maxY;
        root->roomW =  maxW;
        root->roomL =  maxH;
     m_Painter->drawRect(maxX, maxY, maxW, maxH);
   }
}
```

```
void DivideTree::deleteTree(Room *root)
{
   if(root != nullptr)
   {
       deleteTree(root->pLeft);
       deleteTree(root->pRight);
       delete root;
   }
}
```

altar.h

```
#ifndef ALTAR_H
#define ALTAR_H

#include <QGraphicsObject>


#include <QPainter>
#include <QStyleOptionGraphicsItem>

class Altar: public QGraphicsItem
{
public:
    double x = 0, y = 0;
    bool active = true;
   Altar(double x, double y);

   // QGraphicsItem interface
public:
   QRectF boundingRect() const;
   void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget);
};



#endif // ALTAR_H
```

altar.cpp

```
#include "altar.h"


Altar::Altar(double x, double y)
{
   this->x = x;
   this->y = y;
```

```
}

QRectF Altar::boundingRect() const
{
}

void Altar::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget)
{
  if(active)
  {
     painter->drawPixmap(0, 0, 40,
40,QPixmap("C:/Users/briti/Pictures/Project/altar/actAltar.png"));
  }
  else
  {
     painter->drawPixmap(0, 0, 40,
40,QPixmap("C:/Users/briti/Pictures/Project/altar/disAltar.png"));
  }
}
```

trap.h

```
#ifndef TRAP_H
#define TRAP_H

#include <QGraphicsItem>



class Trap: public QGraphicsItem
{
public:
  double x = 0, y = 0;
  bool active = false;
  Trap(double x, double y);

  QRectF boundingRect() const;
  void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget);
};


#endif // TRAP_H
```

trap.cpp

```
#include "trap.h"
```

```
Trap::Trap(double x, double y)
{
  this->x = x;
  this->y = y;
}

QRectF Trap::boundingRect() const
{
}

void Trap::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget)
{
}
```

room.h

```
#ifndef ROOM_H
#define ROOM_H


class Room
{
public:
  Room();
  Room(int x = 0, int y = 0, int w = 0, int l = 0,Room *parent =nullptr,  int rx = 0, int ry = 0, int
rW = 0, int rL = 0, Room *pLeft = nullptr, Room *pRight = nullptr);
  int x, y,  weight, height;
  int roomX, roomY, roomW, roomL;
  bool altar = false, ghost = false, speedboost = false;
  int amSpear, amLava, amWeb;
  Room *pLeft, *pRight, *parent;
};

#endif // ROOM_H
```

room.cpp

```
#include "room.h"

Room::Room(){}

Room::Room(int x, int y, int w, int l, Room *parent, int rx, int ry, int rW, int rL, Room *pLeft,
Room *pRight)
{
  this->x = x;
```

```
    this->y = y;
    weight = w;
    height = l;
    roomX = rx;
    roomY = ry;
    roomW = rW;
    roomL = rL;
    this->pLeft = pLeft;
    this->pRight = pRight;
    this->parent = parent;
}
```

ghost.h

```
#ifndef GHOST_H
#define GHOST_H

#include <QGraphicsObject>
#include <QPainter>
#include <QWidget>
#include <QStyleOptionGraphicsItem>
#include <QImage>
#include <QRandomGenerator>


class Ghost: public QGraphicsObject
{
public:
    Ghost(  double xpos, double ypos);
      QRandomGenerator *ht = QRandomGenerator::system();
    double xspeed = ht->bounded(6.0) - 3.0, yspeed = ht->bounded(6.0) - 3.0;
    double xpos = 0, ypos = 0;
    double prevx =0,prevy =0;
    int w = 0, a = 0, s = 0, d = 0;

    QImage map;
    QWidget *widget = new QWidget;
    QStyleOptionGraphicsItem *option = new QStyleOptionGraphicsItem;;
    QPainter *painter = new QPainter;


    // QGraphicsItem interface
public:
    void advance(int phase);
    QRectF boundingRect() const;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget);
};
```

```
#endif // GHOST_H
```

ghost.cpp

```
#include "ghost.h"

Ghost::Ghost(  double xpos , double ypos )
{
   this->xpos = xpos;
   this->ypos = ypos;
}

void Ghost::advance(int phase)
{
   paint(painter, option, widget);
   if(phase)
   {
      if(map.pixelColor(xpos+30 , ypos +15) == QColor(210, 180, 140) &&
map.pixelColor(xpos+15 , ypos ) == QColor(210, 180, 140)
            && map.pixelColor(xpos , ypos +15) == QColor(210, 180, 140) &&
map.pixelColor(xpos+15 , ypos  +30) == QColor(210, 180, 140))
      {
         prevx = xpos;
         prevy = ypos ;
         moveBy(xspeed, yspeed);
          xpos +=xspeed;
          ypos +=yspeed;

          if(yspeed > 0)
          {
             s++;
             a = w = d =0;
          }
           if(yspeed < 0)
          {
             w++;
             a = s = d =0;
          }
           if(xspeed < 0)
          {
             d++;
             a = s = w =0;
          }
           if(xspeed > 0)
          {
             a++;
```

```
            w = s = d =0;
        }

    }
    else
    {
        this->setPos(prevx, prevy);
        xpos = prevx;
        ypos = prevy;
        xspeed = ht->bounded(6.0) - 3.0;
        yspeed = ht->bounded(6.0) - 3.0;
    }
  }
}

QRectF Ghost::boundingRect() const
{

}

void Ghost::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget)
{
  if(s > 0 )
  {
     painter->drawPixmap(0, 0, 30,
30,QPixmap("C:/Users/briti/Pictures/Project/ghost/back1.png"));
  }
  else if(w > 0)
  {
     painter->drawPixmap(0, 0, 30,
30,QPixmap("C:/Users/briti/Pictures/Project/ghost/up1.png"));
  }
  else if(d > 0)
  {
     painter->drawPixmap(0, 0, 30,
30,QPixmap("C:/Users/briti/Pictures/Project/ghost/left1.png"));
  }
  else if(a > 0)
  {
     painter->drawPixmap(0, 0, 30,
30,QPixmap("C:/Users/briti/Pictures/Project/ghost/right1.png"));
  }

}
```