Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

КЛИЕНТ-СЕРВЕРНОЕ ИГРОВОЕ ПРИЛОЖЕНИЕ “BOMBERMAN”

БГУИР КП 1-40 01 01 08 ПЗ

Студент: гр. 551006 Жизневский В.С.

Руководитель: acc. Дубровский Р. А.

Минск 2017

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc484211531)

[1 СРАВНЕНИЕ АНАЛОГОВ 8](#_Toc484211532)

[1.1 Bomberman для NES 8](#_Toc484211533)

[1.2 Super Bomberman R 9](#_Toc484211534)

[2 ПОСТАНОВКА ЗАДАЧИ 12](#_Toc484211535)

[3 ПРОЕКТИРОВАНИЕ. РАЗРАБОТКА 13](#_Toc484211536)

[3.1 Описание ресурсов 13](#_Toc484211537)

[3.1.1C# 13](#_Toc484211538)

[3.1.2Microsoft Visual Studio 13](#_Toc484211539)

[3.1.3Windows Forms 14](#_Toc484211540)

[3.2 Структура приложения 14](#_Toc484211541)

[3.3 Основной игровой процесс 16](#_Toc484211542)

[3.3.1 Основное игровое ядро 17](#_Toc484211543)

[3.3.2 Классы игровых объектов 19](#_Toc484211544)

[3.3.3 Игровые сценарии 25](#_Toc484211545)

[3.4 Серверная часть 27](#_Toc484211546)

[3.4.1 Прослушивание входящих соединений. Создание соединения клиента с сервером. Отсылка состояния карты 28](#_Toc484211547)

[3.4.2 Анализ сообщений 29](#_Toc484211548)

[3.5 Клиентская часть 30](#_Toc484211549)

[4 ПРОВЕРКА КОРРЕКТНОСТИ РАБОТЫ 31](#_Toc484211550)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 33](#_Toc484211551)

[5.1 Системные требования 33](#_Toc484211552)

[5.2 Запуск программы. Начало игры 33](#_Toc484211553)

[5.3 Правила игры 33](#_Toc484211554)

[5.4 Управление 33](#_Toc484211555)

[ЗАКЛЮЧЕНИЕ 34](#_Toc484211556)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 35](#_Toc484211557)

[ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ 36](#_Toc484211558)

# **ВВЕДЕНИЕ**

Индустрия развлечений оказалась одной из первых, занявших свою крепкую позицию в период быстрого развития компьютерных технологий наряду с военной и медицинской. Люди поняли, что намного удобнее хранить и воспроизводить какую-либо игру (а может, даже несколько) на одном устройстве, не выходя из дома. Так и появились компьютерные игры.

Ни для кого не секрет, что компьютерные игры прочно заняли свою позицию в современной индустрии развлечений. Существуют попытки выделить компьютерные игры как отдельную область искусства, наряду с театром, кино и т.п. Разработка игр может оказаться не только увлекательным, но и прибыльным делом, примеров этому предостаточно в истории.

Первые игры отличались простотой интерфейса и логики, но со временем они становились все сложнее и сложнее, над их созданием уже работал не один человек, а целая команда разработчиков. Первые примитивные компьютерные и видео игры были разработаны в 1950-х и 1960-х годах. Они работали на таких платформах, как осциллографы, университетские мейнфреймы и компьютеры EDSAC. Самой первой компьютерной игрой стал симулятор ракеты, созданный в 1942 году Томасом Голдсмитом Младшим (англ. Thomas T. Goldsmith Jr.) и Истл Рей Менном (англ. Estle Ray Mann). Позже, в 1952 году, появилась программа "OXO", имитирующая игру "крестики-нолики", созданная А.С. Дугласом как часть его докторской диссертации в Кембриджском Университете. Игра работала на большом университетском компьютере, известном как EDSAC (Electronic Delay Storage Automatic Calculator). В настоящее время, разработка игры - это многомиллионный процесс, в котором задействована целая команда разработчиков, сложные современные технологии и даже маркетинговые ходы.

Первая игра серии Bomberman была выпущена в 1983 году для ряда домашних компьютеров и игровой консоли NES. В Европе игра распространялась под названием Eric and the Floaters. Игра получила огромную популярность и стала началом одной из крупнейших серий видеоигр, в настоящее время включающей более 60. Игры серии разрабатывались и издавались как самой Hudson Soft, так и рядом других компаний и выходили на большинстве существующих игровых систем.

Большинство игр серии выполнены в жанре лабиринтной аркады. Игрок управляет персонажем, находящимся в лабиринте, состоящем из разрушаемых и неразрушаемых стен. Он может оставлять бомбу, взрывающуюся через небольшое время и разрушающую стены рядом с ней. Специальные бонусы могут увеличить количество одновременно оставляемых бомб, дальность их взрыва, скорость перемещения героя, дать возможность взрыва бомб по нажатию кнопки, невосприимчивость от взрыва бомб, прохождение сквозь разрушаемые стены. На уровне присутствуют противники. В некоторых играх серии целью игры является нахождение скрытой за одной из разрушаемых стен двери, ведущей в следующий уровень с предварительным уничтожением врагов. Другие игры рассчитаны на многопользовательскую игру на одном экране, целью в них является победа над всеми противниками.

В связи с популярностью и легендарностью данной игры целью данной курсовой работы было выбрано создание многопользовательского ремейка оригинальной версии игры “Bomberman” для NES 1985 года выпуска.

# **СРАВНЕНИЕ АНАЛОГОВ**

Как уже было упомянуто ранее, серия игр Bomberman насчитывает около 60 игр. Bomberman побывал на большом количестве платформ (Gameboy, Sega, Sony PS 1, 2, 3, мобильные телефоны и т.д). На данный момент имеется много игр, интерпретирующих и расширяющих основные игровых правила оригинальной игры. Далее будут рассмотрены несколько из них.

## **Bomberman для NES**

Bomberman NES (рисунок 1.1) — первая из игр серии, выпущенная для консоли NES. Разработчиком является Hudson Soft. В Европе известна по названием Eric and the Floaters.

Главный персонаж игры — Бомбермен (представленный как робот, которому нужно сбежать с завода по производству бомб). Задача героя — расставлять по уровню бомбы, взрывать стены, мешающие проходу и врагов, которые убивают прикосновением.

Главная задача Бомбермена — за отведенное на уровне время уничтожить врагов и найти среди стен дверь, ведущую на другой уровень. Если время заканчивается, это не карается смертью, как в других играх, однако игрок не получает бонусные очки за завершение уровня, и в дополнение ко всему, неожиданно появляется множество врагов-«копеек», избежать столкновения с которыми практически невозможно.

Через каждые 5 уровней игрок попадает в пустой бонус-уровень, где бесконечно появляются враги, задача героя быстро их всех уничтожить, зарабатывая на этом очки.

Всего в игре 50 уровней.

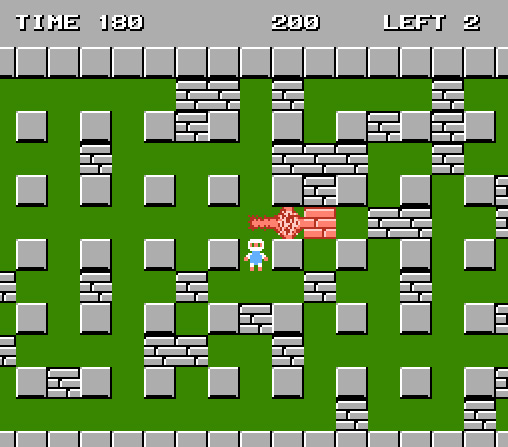


Рисунок 1.1 – Bomberman NES

Несмотря на то, что игра уже вышла давно и графически устарела, она всё равно остается ещё в какой-то мере популярной и востребованной.

## **Super Bomberman R**

Super Bomberman R (рисунок 1.2) была выпущена 3 марта 2017 года компанией Konami для консоли Nintendo Switch.

В **Super Bomberman R** простое правило игры остается тем же самым: управлять главным героем (Bomberman), метать бомбы и сражаться с врагами и соперниками.

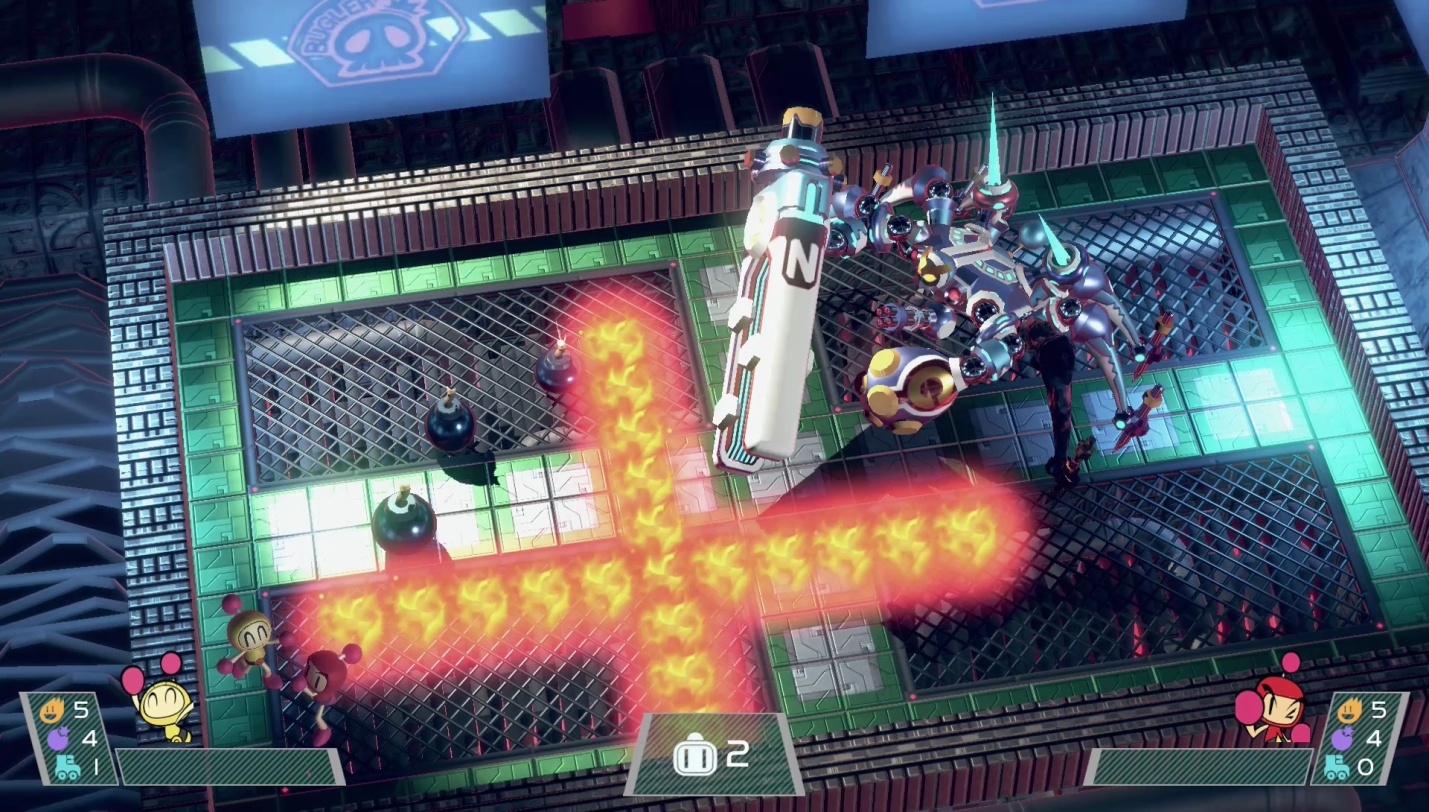


Рисунок 1.2 – Super Bomberman R

Присутствует соревновательный режим битвы “Battle” (рисунок 1.3) а сюжетную кампанию в режиме одиночной игры “Story” также возможно будет проходить совместно: каждый из 50 уровней можно зачищать один за другим вдвоём. Кроме того, появилось новое правило, когда условие прохождения уровня заключается не только в уничтожении всех врагов.



Рисунок 1.3 – Super Bomberman R режим “Battle”

Особенности игры:

* bomberman возвращается с той же игровой механикой и управлением, но со свежим идеями, обогатившими классический выпуск;
* фотореалистичная графика;
* братья и сёстры Bomberman и хорошо известные враги возвращаются с яркими красками индивидуальности;
* трёхмерные карты с динамическими разрушаемыми объектами окружения;
* режим битвы для максимум 8 игроков, локальные сражения и сетевые баталии;
* богатая сюжетная история со звёздным составом актёров озвучки.

Следует отметить, что на данный момент это последняя игра серии, выпущенная компанией Konami, которая на данный момент владеет правами на данную франшизу.

На основе анализа аналогов, а также на основе тенденций последних лет популярности мультиплеерных игр было решено реализовать ремейк Bomberman NES для сетевой игры друг против друга.

# **ПОСТАНОВКА ЗАДАЧИ**

Разработать клиент-серверное игровое приложение “Bomberman”, реализующее основные механики одноименной игры, необходимые для игры друг против друга. Приложение будет состоять из клиента и сервера.

Сервер игрового приложения “Bomberman” должен реализовывать следующий функционал:

* основной игровой процесс;
* генерация карты;
* просчёт состояиня карты;
* прослушивание входящих подключений;
* идентификация клиента на сервере;
* обработка сообщений о действиях клиентов;
* рассылку текущего сострояния карты с целью дальнейшей отрисовки его клиентом.

Клиент игрового приложения “Bomberman” должен реализовывать следующий функционал:

* идентификация пользователя;
* подключение к серверу;
* получение состояния карты от сервера;
* отрисовка состояния карты;
* управление и отправка серверу сообщений о действиях пользователя.

# **ПРОЕКТИРОВАНИЕ. РАЗРАБОТКА**

## **Описание ресурсов**

Разработка данного приложения будет происходить в IDE Microsoft Visual Studio 2013 на языке C# при использовании стандартных библиотек языка C#, встроенных в Visual Studio. Приложение будет разрабатываться в Windows Forms

# C#

**C#** — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, и, в особенности, Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

# Microsoft Visual Studio

Microsoft Visual Studio — продукт компании Microsoft, включающий интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данный продукт позволяет разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

# Windows Forms

Windows Forms — интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, С++, так и на VB.Net, J# и др.

Внутри .NET Framework, Windows Forms реализуется в рамках [пространства имён](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%B8%D0%BC%D1%91%D0%BD_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) System.Windows.Forms.

## **Структура приложения**

Данное приложение будет иметь клиет-серверную архитектуру (рисунок 3.1). Сервер (рисунок 3.2) будет состоять непосредственно из серверной части, отвечающей за прослушивание входящих подключений, обработку сообщений о действиях клиентов, рассылку текущего состояния карты клиентам, и части, реализующий игровой процесс (просчёт физики, движения игроков, взырвов и т.д.).

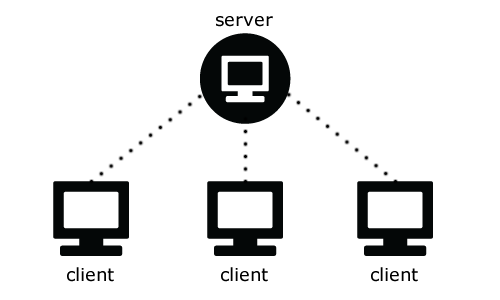


Рисунок 3.1 – Обобщённая схема клиент-серверной архитектуры

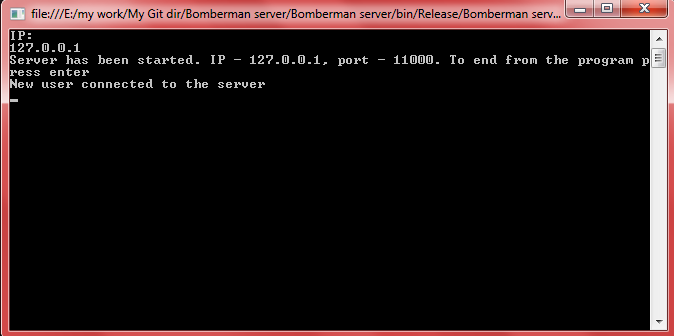


Рисунок 3.2 – Сервер

Клиент будет состоять непосредственно из клиентской части,

отвечающей за подключение к серверу, посылку сообщений о действиях пользователя, получение состояния карты, и части, которая отрисосвывает текущее состояние карты. Приложение клиента будет состоять из окна подключения пользователя к серверу (рисунок 3.3) и непосредственно основного окна приложения, где будет происходить отрисовка (рисунок 3.4).

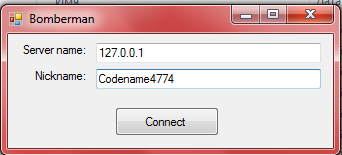
****

Рисунок 3.3 – Окно подключения

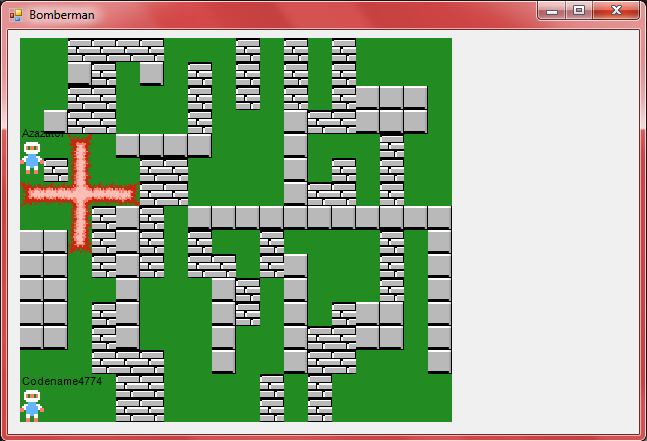


Рисунок 3.4 – Основное окно

## **Основной игровой процесс**

Для начала надо определиться с правилами игры.

Правила игры будут схожи с правилами оригинальной игры 1985 года к некоторыми отличиями: В игре будет присутствовать только мультиплеерный режим, в котором игроки соревнуются друг с другом. Задача каждого игрока убить как можно больше других игроков. В отличие от оригинальной игры на поле будут отсутвовать другие враги кроме остальных игроков. В остальном механики те же: игрок может расставлять бомбы, которые взрываются через определённое количество времени. На карте присутсутвуют разрушаемые и неразрушаемые стены. Убить другого игрока можно взрывом бомбы, но следует быть осторожным, т.к. взрывная волна может убить и самого игрока.

Весь игровой процесс будет просчитываться на сервере (состояние всех объектов, сценарии взрывов, физика движения игроков, физика взрыва стен и т.д.), после чего текущее состояние будет оправляться всем клиентам для дальнейшей отрисовки. Отрисовка будет происходить в окне, созданном средствами Windows Forms. Отрисовка будет происходить при помощи классов платформы .Net, находящихся в пространстве имён System.Drawing, System.Windows.Forms.

### Основное игровое ядро

Как приложение клиент, так и приложение сервер содержат в себе что-то наподобие игрового ядра, которе отвечает за реализацию самого игрового процесса. На стороне клиента за это отвечает класс GameCoreClient, на стороне сервера класс GameCoreServer.

Интерфейс класса GameCoreClient:

…

private Client client; //Содержит ссылку на объект типа Client. Используется для отпраки сообщений

public ObjectsLists objectsList; //Содержит в себе список всех игровых объектов

…

private void DrawPlayers(object state); //Отрисовка игроков

private void DrawStaticWalls(object state); //Отрисовка статических(неразрушаемых) стен

private void DrawDynamicWalls(object state); //Отрисовка динамических(разрушаемых) стен

private void DrawExplosions(object state);

//Отрисовка взрывов

private void DrawBombs(object state);

//Отрисовка бобм

private void DrawPlayerNames(object state); //Отрисовка ников игроков

public void CalcBuff(); //Отрисовка всех объектов. Содержит в себе вызовы выше перечисленных методов

public void ChangeBuffer(); //Смена буфера. В проекте реализована двойная буферизация с целью устранения мерцания изображения

public void RedrawFunc(); //Функция перерисовки окна. Содержит в себе рендер текущего состояния, очистка и смена текущего буфера

public void TimerEvent(object sender, EventArgs e);//Основное событие, происходящее по прерыванию таймера. Содержит в себе вызов метода RedrawFunc

public void KeyPressEvent(object sender, KeyPressEventArgs e); //Обработчик события нажатия клавишы пользователем.

void GetDirection(); //Просчёт направления движения игрока в замисимости от нажатых клавиш

public void KeyUpEvent(object sender, KeyEventArgs e); //Обработчик события отжатия клавишы пользователем

public void Dispose(); //Очистка ресурсов

private void LoadImages(string resDir); //Загрузка всех спрайтов

public void startCore(); //Запуск ядра

public GameCoreClient( int width, int height, Graphics graphicControl, Client client, int id, string dirResources); //Конструктор класса

Интерфейс класса GameCoreServer:

public delegate void SendFunc();

private SendFunc sendFunc;

// Содержит ссылка на метод отправки текущего состояния всех игровых объектов

public PhysicalMap map; //Содержит в себе текущую физическую модель карты

public ObjectsLists objectsList; //Содержит в себе список всех текущих игровых объектов

…

public void ChangePhysicalState(); //Просчитывание физического состояния карты и игроков

public void TimerEvent(object sender, EventArgs e);

); //Основное событие, происходящее по прерыванию таймера. Происходит просчёт движения игроков, просчёт физического состояния карты(происходит вызов метода ChangePhysicalState) и происходит отправка текущего состояния карты

public void OnDeathPlayer(PhysicalObject player); //Обработка события сметри игрока. Запускается сценарий смерти игрока

public void DeletePlayerFromField(object player); //Обработка конца сценария смерти игрока. Удаления игрока с поля

public void ExplosionBomb(PhysicalObject bomb); //Обработка события начала детонации бомбы. Запуск сценария детонации бомбы

public void DeleteBombFromField(object bomb); //Обработка конца сценария детонации бомбы. Запуск сценария взрыва бомбы

public void DeleteExplosionFromField(object explosion); //Обработка конца сценария взрыва

public void StartDestroingDynamicWall(DynamicWall wall); //Обработка события уничтожения бомбы. Запуск сценария уничтожения стены

public void DeleteDynamicWallFromField(object wall); //Обработка конца сценария уничтожения стены

private void GenerateWalls(); //Генерация карты

public void StartCore(); //Запуск ядра

public GameCoreServer(int width, int height, Size playerSize, Size playerOnDeathSize, Size bombSize, Size explosionSize, Size wallSize, string dirResources, SendFunc sendFunc); //Конструктор класса

### Классы игровых объектов

Всего в игре присутствует 4 вида игровых объектов: игрок (рисунок 3.5), бомба (рисунок 3.6), статическая стена (рисунок 3.7), динамическая стена (рисунок 3.8), взрыв (рисунок 3.9). Соответствующие классы: Player, Bomb, PhysicalObject, DynamicWall.

E:\my work\My Git dir\Bomberman client\bomberman.png

Рисунок 3.5 – Игрок

E:\my work\My Git dir\Bomberman client\Bomberman client\bin\Client\Resources\Bomb\bomb.png

Рисунок 3.6 – Бомба

E:\my work\My Git dir\Bomberman client\Bomberman client\bin\Client\Resources\Walls\StaticWall.png

Рисунок 3.7 – Статическая стена

E:\my work\My Git dir\Bomberman client\Bomberman client\bin\Client\Resources\Walls\DynamicWall.png

Рисунок 3.8 – Динамическая стена

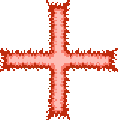


Рисунок 3.9 – Взрыв

Классовая иерархия изображена ниже (рисунок 3.10).

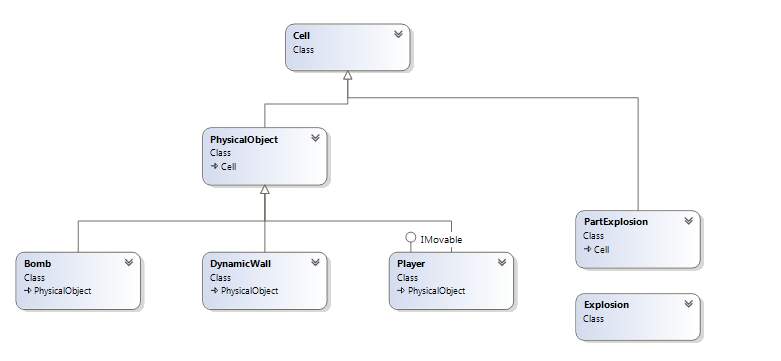


Рисунок 3.10 – Иерархия игровых объектов

Интерфейс класса Cell:

private Point location; //Позиция ячейки

public int X { set {} get {} }

public int Y { set {} get {} }

// Свойства, через которые осуществляется досуп к location

public Cell(Point location); //Коструктор класса

Интерфейс класса PhysicalObject:

public readonly Size size; //Размер объекта

public int currSpriteOffset; // Текущее смещение в спрайте

public virtual void ChangeMapMatrix(PhysicalMap PhysicalMap); //Нанесение объекта на физическую карту

public delegate void DeleteObjectFunc(PhysicalObject obj);

protected DeleteObjectFunc deleteObjectFunc; //Сссылка на функцию удаления объекта из списка

public PhysicalObject(Point location);

public PhysicalObject(Point location, Size size)

: base(location);

public PhysicalObject(Point location, Size spriteSize, DeleteObjectFunc deleteObjectFunc )

: base(location); //Кострукторы класса

Интерфейс класса Bomb:

private int areaOfExplosion; //Радиус взрыва

public Player owner; //Владелец бобмы

public bool isBlowedUp; //Бобма детонирует?

public void TimerEvent(object sender, EventArgs e); //Вызов метода удаления бомбы по окончанию детонации

public override void ChangeMapMatrix(PhysicalMap PhysicalMap); //Нанесение объекта на физическую карту

public Bomb(Point location, int areaOfExplosion, DeleteObjectFunc deleteBombFunc, Player owner)

: base(location); //Конструктор класса

Интерфейс класса DynamicWall:

public bool isBlowedUpNow; //Уничтожается ли сейчас стена?

public bool isWallBlowedUp(PhysicalMap map); //Проверка, взорвана ли стена

public DynamicWall(Point location, Size size);

: base(location, size) //Конструктор класса

Интерфейс класса Player:

public enum Direction { UP, DOWN, LEFT, RIGHT };

public Direction direction { get; set; } //Направление движения игрока

public Direction prevDirection { get; set; } //Предыдущее направление движения игрока

public enum AnimState { TURNUP, TURNUP1, TURNUP2, TURNDOWN, TURNDOWN1, TURNDOWN2, TURNLEFT, TURNLEFT1, TURNLEFT2, TURNRIGHT, TURNRIGHT1, TURNRIGHT2 };

public AnimState currAnimState;

//Текущее состояние анимации движения

public enum BombLevel { low = 2, medium = 3, high = 4 };

public BombLevel bombLevel; //Уровень бобм игрока (радиус бомб, бросаемых данным игроком)

public int prevLocation; //Предыдущее местоположение

private int step = 4; //Длина шага (в пикселях)

public readonly int maxCountBombs; //Максимальное количество бомб данного игрока, которое может одновременно находится на поле

private int currCountBombs;

public int CurrCountBombs { get { } set { } }

//Текущее количество бобм на поле

public delegate void SpawnPlayerFunc();

private bool isDead;

public bool IsDead { get { } set { } }

//Мёртв ли игрок?

private bool isDying;

public bool IsDying { get { } set { } }

//Умирает ли сейчас игрок? (Запущен ли сейча сценарий смерти игрока)

public BombFactory bombFactory; //Данный класс “Производит” бомбы, принадлежащие данному пользователю

public bool isMoved; //Двигается ли сейча игрок?

private Point newLocation = new Point(); //Используется в просчётах физики движения

public readonly int id; //ID пользователся на сервере

public bool isObjectOnWay(PhysicalMap map); //Имеется ли объект на пути движения игрока?

public override void ChangeMapMatrix(PhysicalMap PhysicalMap); //Нанесение объекта на физическую карту

public void OnMove(PhysicalMap map); //Просчёт новых координат

public bool isPlayerBlowedUp(PhysicalMap map); //Проверка, нашёл ли игрок на взрыв

private string thisName;

public string PlayerName { get { } set { } }

//Имя игрока

public Bitmap GetAnimState(Bitmap texture); //Взятие спрайта игрока в соответсвии с текущим состоянием движения

public Player(Point location, Size spriteSize, string name, DeleteObjectFunc deletePlayerFunc, Size bombSize, DeleteObjectFunc deleteBombFunc, int id)

: base(location, spriteSize, deletePlayerFunc); //Конструктор класса

Интерфейс класса PartExplosion:

private int countStates; //Количество состояний

private int currState; //текущее состояние

public int currSpriteOffset; //Текущее смещение в спрайте

public Size size; //Размер

private Explosion.OnEndAllExplosionFunc onEndFunc; //Метод, вызываемый по окончанию сценария части взрыва

public enum KindExplosionTexture { explosionTextureHorizontalMiddle, explosionTextureLeftEdge, explosionTextureRightEdge, explosionTextureVerticalMiddle, explosionTextureUpEdge, explosionTextureBottomEdge }

public Explosion.KindExplosionTexture kindExplosion; //Тип части взрыва

public void ChangeState(); //Изменение состояния части взрыва

public PartExplosion(Explosion.KindExplosionTexture kindExplosion, Size size, Point location, int countStates, Explosion.OnEndAllExplosionFunc onEndFunc) : base(location); //Коструктор класса

Интерфейс класса Explosion:

private int radius; //Радиус взрыва

public List<PartExplosion> partsExplosionLeft;

public List<PartExplosion> partsExplosionRight;

public List<PartExplosion> partsExplosionBottom;

public List<PartExplosion> partsExplosionUp;

public PartExplosion partExplosionCenter;

//Части взрыва

private Point explosionLocation; //Положение центра

private Point newLocation; //Используется в просчётах координат частей взрыва

private int countParts; //Количество частей

public delegate void OnEndAllExplosionFunc(object sender);

private OnEndAllExplosionFunc onEndAllExplosionFunc; //Метод, вызывемый по окочании взрыва

Size size; //размер части взрыва

public enum KindExplosionTexture { explosionTextureHorizontalMiddle, explosionTextureLeftEdge, explosionTextureRightEdge, explosionTextureVerticalMiddle, explosionTextureUpEdge, explosionTextureBottomEdge, explosionTextureCenter };

//Типы спрайтов частей взрыва

private enum ExplosionDirection { UP, DOWN, LEFT, RIGHT }; //Направление взрыва

public void ChangeState(); //Изменение состояния взрыва

public void onEndExplosionFunc(object sender); //Метод, вызываемые при окончании части взрыва

private bool canWePlaceExplosion(int X, int Y, PhysicalMap map, ExplosionDirection direction); //Провека, можно ли расположить в данной точке часть взрыва

private void InitExplosion(Size size, Point location, int radius,PhysicalMap map); //Инициализация взрыва

public void ChangePhysicalMap(PhysicalMap map); //Нанесение объекта на физическую карту

public void DrawPart(BufferedGraphics currBuffer, List<Bitmap> images, PartExplosion partExplosion); //Отрисовка части взрыва

public void DrawExplosion(BufferedGraphics currBuffer, List<Bitmap> images); //Отрисовка взрыва

public Explosion(Size size, Point location, int radius, PhysicalMap map, OnEndAllExplosionFunc funcEnd); //Конструктор класса

### Игровые сценарии

Всего в игре присутствует следующие игровые сценарии: сценарий детонации бобмы (рисунок 3.11), уничтожения динамической стены (рисунок 3.12), сценарий взрыва (рисунок 3.13), сценарий смерти игрока (рисунок 3.14). Сценарии в рамках данного проекта представляют собой запланированное изменение состояния игровых объектов. Количество состояний фиксированное, изменение происходит через определенные интервалы времени.

Сценарии детонации бобмы, уничтожения динамической стены, сценарий смерти игрока схожи по своей сути: в течении всего сценария объект изменяет своё отображеине на карте, по окончанию сценария объект удаляется с игровой сцены. Сценарий взрыва несколько сложнее, т.к. помимо выше перечисленного нужно просчитать положение его частей на карте и их количество.

E:\my work\My Git dir\Bomberman client\Bomberman client\bin\Client\Resources\Bomb\bombExplosion.png

Рисунок 3.11 – Состояния бомбы при детонации

E:\my work\My Git dir\Bomberman client\Bomberman client\bin\Client\Resources\Walls\DynamicWallDestroy.png

Рисунок 3.12 – Состояния динамической стены при уничтожении

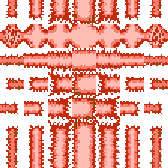


Рисунок 3.13 – Состояния частей взрыва

E:\my work\My Git dir\Bomberman client\bomberman_death.png

Рисунок 3.14 – Состояния игрока во время смерти

В проекте сценарии реализуют классы ScriptEngine, SimpleScript, ExplosionScript. Класс ScriptEngine является главным, который инициализирует все сценарии (скрипты).

Интерфейс класса ScriptEngine:

public delegate void OnEndFunc(object sender); //Сигнатура метода, вызываемого по окончанию сценария

public delegate void OnChangeFunc(); //Сигнатура метода, вызываемого по изсенению состояния

public delegate void TimerEvent(object sender, EventArgs e);//Сигнатура метода, вызываемого по страбатыванию таймера

public void StartSimpleScript(PhysicalObject obj, OnEndFunc onEndFunc, int delay, int countStates);

public void StartSimpleScript(PhysicalObject obj, Size onScriptSize, OnEndFunc onEndFunc, int delay, int countStates); //Запуск простого скритпа, т.е. всех, кроме скрипта взрыва

public void StartExplosion(Explosion explosion, OnEndFunc onEndFunc, int delay, int countStates); //Запуск скрипта взрыва

Класс SimpleScript отвечает за просчёт простых сценариев (детонация бомбы, уничтожение стены, смерть игрока).

Интерфейс класса SimpleScript:

private int countStates; //Количество состояний в сценарии

private int currState; //Текущее состояние

private dynamic obj; //Изменяемый объект

private ScriptEngine.OnEndFunc onEnd; //Метод, вызываемый по окончанию сценария

…

public void OnTimerEvent(object sender, EventArgs e); //Событие изменение состояния

private void SetOffset(); //Установка смещения в спрайте

public SimpleScript(PhysicalObject obj, ScriptEngine.OnEndFunc onEndFunc, int countStates, int delay);

public SimpleScript(PhysicalObject obj, Size onScriptSize, ScriptEngine.OnEndFunc onEndFunc, int countStates, int delay); //Конструкторы класса

Интерфейс класса Explosion схож с интерфейсов класса SimpleScript.

## **Серверная часть**

Схему работы сервера в данном проекте можно описать следующей схемой (рисунок 3.1). После инициализации и генерации карты, сервер начинает прослушивать входящие соединения. При подключении нового игрока сервер создаёт новое соединение с данным клиентом, генерирует уникальный id для данного пользователя в рамках данного сервера, после чего сервер начинает отсылать текущее состояние карты пользователю. Также сервер анализирует приходящие сообщения от клиентов и вносит изменения в состояние карты.

В рамках данного проекта будет использоваться стек протоколов TCP/IP, а также классы платформы .Net пространства имён System.Net, System.Net.Sockets.

1. Прослушивание входящих соединений. Создание соединения клиента с сервером. Отсылка состояния карты

За серверную часть отвечает класс ServerCore.

Интерфейс класса ServerCore:

public readonly IPAddress ipAdress; //Информация о IP адресе сервера

public readonly int portControl; //Порт входящих соединений

public readonly int maxLengthQueue; //Максимальная длина очереди входящих соединений

private int bufferSize;

public readonly IPEndPoint ipEndPointControl; //Конечная точка

public readonly Socket socketListener; //Сокет, прослушивающий входящие соединения

public readonly List<Socket> socketsList; //Список сокетов пользователей, на которые идёт отправка информации

public GameCoreServer gameCoreServer; //Ссылка на игровое ядро сервера

public BinaryFormatter serializer; //Отвечает за сериализацию состояния карты

public MessageAnalyzerServer messageAnalyzer; //Анализирует сообщения от пользователей

private int idCounter; //Счётчик id. Предназначен дял генерации уникального id для нового пользователя

public ServerCore(string host, int portControl, int maxLengthQueue, int sendFrequency); //Конструктор класса

public void SendData(); //Посылка текущего состояния карты

private void ReceiveCallback(object sender, SocketAsyncEventArgs e); //Вызывается при получении данных от пользователя. Происходит анализ сообщения

private void SendId(Socket client); //Посылка id пользователю

private void AddPlayerToList(string playerName); //Добавление игрока в список игроков

public void StartListen(object state); //Старт прослушки сходящих сообщений

Схема подключения была описана ранее (раздел 4.3). Следует отметить лишь то, что отсылка данных идёт по таймеру.

1. Анализ сообщений

За анализ вообщений отвечает класс MessageAnalyzer. Описание типов сообщений содержит класс KindMessages.

Интерфейс класса MessageAnalyzer:

GameCoreServer gameCoreServer; //Ссылка на игровое ядро сервера. Через неё происходит изменение состояния карты

public int GetNextValue(byte[] message, ref int i); //Взять следующее значение из сообщения

private Player FindPlayer(int id); //Найти игрока в списке, который отослал сообщение

public void AnalyzeMessage(byte[] message); //Анализ всего сообщения

public MessageAnalyzerServer(GameCoreServer gameCoreServer); //Конструктор класса

Интерфейс класса KindMessages:

public enum KindMessage { Player = 0 }; //Тип сообщения

public enum KindPlayerMessages { NewDirection = 0, Spawn = 1, Death = 2, PlaceBomb = 3, Connect = 4, Disconnect = 5, Location = 6, StopWalking = 7 }; //Информация сообщения игрока: игрок выбрал новое направление движения, игрок нажал клавишу возрождения, игрок умер, игрок нажал клавишу установки бобмы, игрок присоединился к серверу, игрок отсоединился от сервера, позиция игрока на карте, игрок остановился

public enum Direction { UP, DOWN, LEFT, RIGHT }; //Направление движения игрока

Разбор сообщения представляет собой последовательное чтение значение и их последовательный анализ. Разбор сообщения продолжается в зависимости от прочитанного значения.

## **Клиентская часть**

Клиентская часть данного проекта представлена классами Client и KindMessages.

Схема работы клиента: клиент, после подключения к серверу начинает получать состояние карты и отрисовывать его, а также он в зависимости от действий пользователя генерирует соответствующее сообщение и посылает его серверу. Интерфейс класса KindMessages описан в разделе 3.4.2.

Интерфейс класса Client:

…

private byte[] receivedData; //буфер для получаемых данных

public GameClasses.GameCoreClient gameCore; //Ссылка на игровое ядро клента. Через неё идёт передача данных для отрисовки

private BinaryFormatter serializer; //Десериализиует состояние карты

public readonly int id; //id клиента

public Client(string host, int portControl, string playerName); //Конструктор класса

private void GetBufferFromServer(object sender, SocketAsyncEventArgs e); //получение состояния карты от сервера

public void SendMessageToServer(params int[] data); //посылка сообщения клиенту

public void Dispose(); //очистка ресурсов

public void StartRecieving(object state); //Запуск получения данных от сервера

# **ПРОВЕРКА КОРРЕКТНОСТИ РАБОТЫ ПРОГРАММНОГО СРЕДСТВА**

Ввиду небольшого размера проекта и небольшого функционала и количетсва рабочих окон данного проекта, было решено описать проверку корректности работы данного средства.

При незаполнении IP-адреса сервера пользователю будет выведено соответствущее сообщение о том, что поле долно быть заполнено (рисунок 4.1).

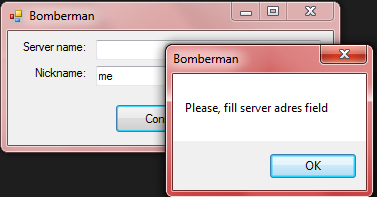


Рисунок 4.1 – Ошибка ”поле адрес сервера не заполнено”

При незаполнении никнейма пользователю будет выведено соответствущее сообщение о том, что поле долно быть заполнено (рисунок 4.2).

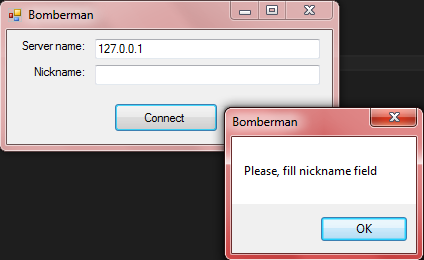


Рисунок 4.2 – Ошибка ”поле никнейм не заполнено”

Если клиенту не удалось подключиться к серверу, то будет выведено сообщение (рисунок 4.3, 4.4).

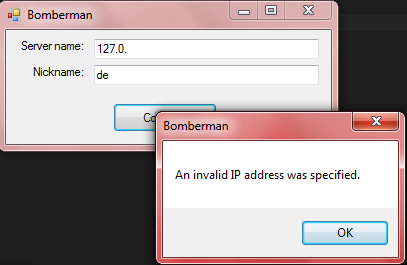


Рисунок 4.3 – Ошибка ”некорректный адрес”

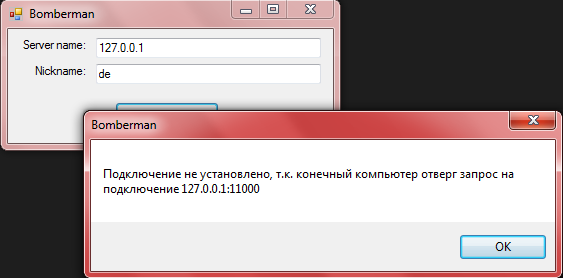


Рисунок 4.4 – Ошибка ”не удалось подключиться к серверу”

# **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

1. **Системные требования**

Для успешного запуска игры нужно наличие поддержки видеокартой

DX9:

* процессор с частотой 1,2 ГГц или выше;
* 400 МБ оперативной памяти.

Также потребуется подключение к интернету или локальной сети (в зависимости от того, где расположен сервер).

1. **Запуск программы. Начало игры**

Для запуска клиента необходимо открыть исполняемый файл Bomberman client.exe. Откроется начальное окно программы (рисунок 3.3). После удачного подключения начнётся игра (рисунок 3.4).

Для запуска сервера необходимо открыть исполняемый файл Bomberman server.exe. Откроется начальное окно программы (рисунок 3.4), где следует ввести IP – адрес сервера, после чего сервер начнёт свою работу.

1. **Правила игры**

Игрок может расставлять бомбы, которые взрываются через определённое количество времени. На карте присутсутвуют разрушаемые и неразрушаемые стены. Убить другого игрока можно взрывом бомбы, но следует быть осторожным, т.к. взрывная волна может убить и самого игрока. Цель: убивать других игроков в соревновательных целях.

1. **Управление**

Для управления игроком во время игрового процесса предоставлены следующие клавиши:

* w, a, s, d (управление игроком. Вверх, влево, вниз, вправо соответственно);
* space (установка бомбы);
* enter (возрождение).

**ЗАКЛЮЧЕНИЕ**

Безусловно, игра “Bomberman” является классикой в области компьютерных игр. Несмотря на возраст данной игры, на сегодняшний день

франшиза данной игры насчитывает уже около 60 игр и в серии продолжают появлятся новые проекты, о чём свидетельствует игра Super Bomberman R, которая вышла 3 марта 2017 года, что говорит об актуальности данной игры и в наше время.

В рамках данного курсового проекта было разработано клент-серверное игровое приложение “Bomberman”. В данной программе реализованы основные механики оригинальной игры, а также клиентская и серверная часть приложения, обеспечивающие сетевую игру.

При разработке программы в качестве основы использовалась классы, распологающиеся в пространтсвах имён System.Windows.Forms, System.Net, System.Net.Sockets платформы .Net.

Поставленная цель выполнена. Программа реализует все поставленные задачи.

В будущем планируется программу усовершенствовать, а именно:

* Добавить звуковое сопровождение;
* усовершенствовать графическую составляющую. Добавить остальные игровые механики из оригинальной игры;
* добавить регистрацию, ведение статистики;
* добавить статистику игрока на сервере;
* создать конструктор карт;
* оптимизация игры;
* создание новых режимов и т.д.

Список использованных источников

[1]  [Васильев, Алексей C#. Объектно-ориентированное программирование / Алексей Васильев. - М.: Питер, 2012. - 320 c.](http://www.piter.com/collection/yazyk-programmirovaniya-ss-s/product/clr-via-c-programmirovanie-na-platforme-microsoft-net-framework-45-na-yazyke-c-4-e-izd-2)

[2] Каталог API и справочных материалов [Электронный ресурс]. –https://msdn.microsoft.com/library

[3] Фленов, Михаил Библия C# (+ CD-ROM) / Михаил Фленов. - М.: БХВ-Петербург, 2011. - 560 c

[4] Танненбаум Э. Компьютерные сети / Э. Танненбаум –СПб: Питер, 2002. -356 с.

[5] Конструирование программ и языки программирования: Учено-методическое пособие для студ. спец. 40 01 01 “Программное обеспечение информационных технологий” дневной формы обуч. В 2 ч./ В.В.Бахтизин [и др.] – Минск: БГУИР 2006.

[6] Шупляк В.И.: С++. Практический курс: учеб. пособие / Шупляк В.И – Минск: Новое знание, 2008. – 576 с.

[7] Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы. 3-е изд.: учебник для вузов / В.Г. Олифер, Н.А.Олифер. –СПб: Питер, 2006. -958с.

**ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Threading;

namespace Bomberman\_client

{

public partial class FormConnect : Form

{

public FormConnect()

{

InitializeComponent();

}

private void ButtonConnect\_Click(object sender, EventArgs e)

{

if (TextBoxServerName.Text.Length <= 0)

{

MessageBox.Show("Please, fill server adres field", "Bomberman", MessageBoxButtons.OK);

}

else

{

if (TextBoxNickname.Text.Length <= 0)

{

MessageBox.Show("Please, fill nickname field", "Bomberman", MessageBoxButtons.OK);

}

else

{

try

{

Client client = new Client(TextBoxServerName.Text, 11000, TextBoxNickname.Text);

FormGame formGame = new FormGame(client);

formGame.Show();

this.Hide();

ThreadPool.QueueUserWorkItem(formGame.client.StartRecieving);

}

catch (Exception exeption)

{

MessageBox.Show(exeption.Message, "Bomberman", MessageBoxButtons.OK);

}

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Threading;

namespace Bomberman\_client

{

public partial class FormGame : Form

{

GameClasses.GameCoreClient gameCore;

public Client client;

public FormGame(Client client)

{

InitializeComponent();

this.client = client;

Graphics graphicsControl = MainField.CreateGraphics();

gameCore = new GameClasses.GameCoreClient

(

MainField.Width, MainField.Height, graphicsControl, client, client.id, (Environment.CurrentDirectory + "\\Resources\\")

);

this.KeyUp += gameCore.KeyUpEvent;

this.KeyPress += gameCore.KeyPressEvent;

client.gameCore = gameCore;

}

private void FormGame\_Load(object sender, EventArgs e)

{

//ThreadPool.QueueUserWorkItem(client.StartRecieving);

gameCore.StartCore();

}

private void FormGame\_FormClosing(object sender, FormClosingEventArgs e)

{

gameCore.Dispose();

client.Dispose();

}

private void FormGame\_FormClosed(object sender, FormClosedEventArgs e)

{

Application.Exit();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Drawing;

using Bomberman\_server;

using System.Threading;

using System.Timers;

namespace Bomberman\_client.GameClasses

{

public class GameCoreClient

{

private Graphics graphicControl;

private BufferedGraphicsContext currentContext = new BufferedGraphicsContext();

public BufferedGraphics buffer1, buffer2, currBuffer;

private Color buffColor = new Color();

private Brush brush;

private Font font;

private Client client;

public ObjectsLists objectsList;

public ObjectsLists bufferObjectsList;

public int delay;

private System.Timers.Timer timer;

const char KEY\_UP = (char)Keys.W;

const char KEY\_DOWN = (char)Keys.S;

const char KEY\_LEFT = (char)Keys.A;

const char KEY\_RIGHT = (char)Keys.D;

const char KEY\_PLACE\_BOMB = (char)Keys.Space;

const char KEY\_SPAWN = (char)Keys.Enter;

private bool isUpPressed = false;

private bool isDownPressed = false;

private bool isLeftPressed = false;

private bool isRightPressed = false;

public Bitmap bombTexture;

public Bitmap bombExplosionTexture;

public Bitmap playerTexture;

public Bitmap playerDieTexture;

public Bitmap staticWallTexture;

public Bitmap dynamicWallTexture;

public Bitmap dynamicWallDestroyTexture;

public List<Bitmap> explosionsTexture;

public int id;

private void DrawPlayers(object state)

{

lock (currBuffer.Graphics)

{

foreach (Player player in objectsList.players)

{

if (!player.IsDead)

{

if (!player.IsDying)

{

currBuffer.Graphics.DrawImage(player.GetAnimState(playerTexture), player.X, player.Y);

}

else

{

lock (playerDieTexture)

{

currBuffer.Graphics.DrawImage(playerDieTexture, player.X, player.Y, new Rectangle(new Point(player.currSpriteOffset, 0), player.size), GraphicsUnit.Pixel);

}

}

}

}

}

}

private void DrawStaticWalls(object state)

{

lock (currBuffer.Graphics)

{

for (int i = 0; i < objectsList.staticWalls.Count; i++)

{

lock (staticWallTexture)

{

currBuffer.Graphics.DrawImage(staticWallTexture, objectsList.staticWalls[i].X, objectsList.staticWalls[i].Y);

}

}

}

}

private void DrawDynamicWalls(object state)

{

lock (currBuffer.Graphics)

{

for (int i = 0; i < objectsList.dynamicWalls.Count; i++)

{

lock (objectsList.dynamicWalls[i])

{

if (!objectsList.dynamicWalls[i].isBlowedUpNow)

{

lock (dynamicWallTexture)

{

currBuffer.Graphics.DrawImage(dynamicWallTexture, objectsList.dynamicWalls[i].X, objectsList.dynamicWalls[i].Y);

}

}

else

{

lock (dynamicWallDestroyTexture)

{

currBuffer.Graphics.DrawImage(dynamicWallDestroyTexture, objectsList.dynamicWalls[i].X, objectsList.dynamicWalls[i].Y, new Rectangle(new Point(objectsList.dynamicWalls[i].currSpriteOffset, 0), objectsList.dynamicWalls[i].size), GraphicsUnit.Pixel);

}

}

}

}

}

}

private void DrawExplosions(object state)

{

lock (currBuffer.Graphics)

{

for (int i = 0; i < objectsList.explosions.Count; i++)

{

lock (explosionsTexture)

{

objectsList.explosions[i].DrawExplosion(currBuffer, explosionsTexture);

}

}

}

}

private void DrawBombs(object state)

{

lock (currBuffer.Graphics)

{

for (int i = 0; i < objectsList.bombs.Count; i++)

{

lock (bombTexture)

{

if (objectsList.bombs[i].isBlowedUp)

{

currBuffer.Graphics.DrawImage(bombExplosionTexture, objectsList.bombs[i].X, objectsList.bombs[i].Y, new Rectangle(new Point(objectsList.bombs[i].currSpriteOffset, 0), objectsList.bombs[i].size), GraphicsUnit.Pixel);

}

else

{

currBuffer.Graphics.DrawImage(bombTexture, objectsList.bombs[i].X, objectsList.bombs[i].Y);

}

}

}

}

}

private void DrawPlayerNames(object state)

{

lock (currBuffer.Graphics)

{

foreach (Player player in objectsList.players)

{

if (!player.IsDead)

{

if (!player.IsDying)

{

currBuffer.Graphics.DrawString(player.PlayerName, font, brush, player.X, player.Y - (font.Size \* 2));

}

}

}

}

}

public void CalcBuff()

{

ThreadPool.QueueUserWorkItem(DrawPlayers);

ThreadPool.QueueUserWorkItem(DrawBombs);

ThreadPool.QueueUserWorkItem(DrawExplosions);

ThreadPool.QueueUserWorkItem(DrawDynamicWalls);

ThreadPool.QueueUserWorkItem(DrawStaticWalls);

ThreadPool.QueueUserWorkItem(DrawPlayerNames);

}

public void RedrawFunc()

{

lock (currBuffer)

{

currBuffer.Render();

}

ChangeBuffer();

currBuffer.Graphics.Clear(buffColor);

CalcBuff();

}

public void ChangeBuffer()

{

if (currBuffer == buffer1)

{

currBuffer = buffer2;

}

else

{

currBuffer = buffer1;

}

objectsList = bufferObjectsList;

}

public void TimerEvent(object sender, EventArgs e)

{

RedrawFunc();

}

public void KeyPressEvent(object sender, KeyPressEventArgs e)

{

switch ( Char.ToUpper(e.KeyChar) )

{

case KEY\_UP:

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.NewDirection, (int)KindMessages.Direction.UP);

isUpPressed = true;

}

break;

case KEY\_DOWN:

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.NewDirection, (int)KindMessages.Direction.DOWN);

isDownPressed = true;

}

break;

case KEY\_LEFT:

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.NewDirection, (int)KindMessages.Direction.LEFT);

isLeftPressed = true;

}

break;

case KEY\_RIGHT:

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.NewDirection, (int)KindMessages.Direction.RIGHT);

isRightPressed = true;

}

break;

case KEY\_PLACE\_BOMB:

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.PlaceBomb);

}

break;

case KEY\_SPAWN:

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.Spawn);

}

break;

}

}

void GetDirection()

{

if (isUpPressed)

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.NewDirection, (int)KindMessages.Direction.UP);

}

else

if (isDownPressed)

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.NewDirection, (int)KindMessages.Direction.DOWN);

}

else

if (isLeftPressed)

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.NewDirection, (int)KindMessages.Direction.LEFT);

}

else

if (isRightPressed)

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.NewDirection, (int)KindMessages.Direction.RIGHT);

}

else

{

client.SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.StopWalking);

}

}

public void KeyUpEvent(object sender, KeyEventArgs e)

{

switch (e.KeyValue)

{

case KEY\_UP:

{

isUpPressed = false;

GetDirection();

}

break;

case KEY\_DOWN:

{

isDownPressed = false;

GetDirection();

}

break;

case KEY\_LEFT:

{

isLeftPressed = false;

GetDirection();

}

break;

case KEY\_RIGHT:

{

isRightPressed = false;

GetDirection();

}

break;

}

}

public void Dispose()

{

timer.Stop();

timer.Dispose();

bombTexture.Dispose();

bombExplosionTexture.Dispose();

playerTexture.Dispose();

playerDieTexture.Dispose();

explosionsTexture.Clear();

staticWallTexture.Dispose();

dynamicWallTexture.Dispose();

dynamicWallDestroyTexture.Dispose();

currBuffer.Dispose();

buffer1.Dispose();

buffer2.Dispose();

}

private void LoadImages(string resDir)

{

this.bombTexture = new Bitmap(resDir + "Bomb\\bomb.png");

this.bombExplosionTexture = new Bitmap(resDir + "Bomb\\BombExplosion.png");

this.playerTexture = new Bitmap(resDir + "Player\\bomberman\_new.png");

this.playerDieTexture = new Bitmap(resDir + "Player\\bomberman\_death.png");

this.explosionsTexture.Add(new Bitmap(resDir + "Explosion\\ExplosionHorizontalMiddle.png"));

this.explosionsTexture.Add(new Bitmap(resDir + "Explosion\\ExplosionLeftEdge.png"));

this.explosionsTexture.Add(new Bitmap(resDir + "Explosion\\ExplosionRightEdge.png"));

this.explosionsTexture.Add(new Bitmap(resDir + "Explosion\\ExplosionVerticalMiddle.png"));

this.explosionsTexture.Add(new Bitmap(resDir + "Explosion\\ExplosionUpEdge.png"));

this.explosionsTexture.Add(new Bitmap(resDir + "Explosion\\ExplosionBottomEdge.png"));

this.explosionsTexture.Add(new Bitmap(resDir + "Explosion\\ExplosionCenter.png"));

this.staticWallTexture = new Bitmap(resDir + "Walls\\StaticWall.png");

this.dynamicWallTexture = new Bitmap(resDir + "Walls\\DynamicWall.png");

this.dynamicWallDestroyTexture = new Bitmap(resDir + "Walls\\DynamicWallDestroy.png");

}

public void StartCore()

{

timer.Start();

}

public GameCoreClient

(

int width, int height, Graphics graphicControl, Client client, int id, string dirResources

)

{

this.graphicControl = graphicControl;

timer = new System.Timers.Timer();

delay = 60;

timer.Interval = delay;

timer.Elapsed += TimerEvent;

this.id = id;

this.client = client;

this.objectsList = new ObjectsLists();

this.bufferObjectsList = new ObjectsLists();

this.explosionsTexture = new List<Bitmap>();

buffer1 = currentContext.Allocate(graphicControl, new Rectangle(0, 0, width, height));

buffer2 = currentContext.Allocate(graphicControl, new Rectangle(0, 0, width, height));

currBuffer = buffer1;

buffColor = Color.ForestGreen;

brush = new SolidBrush(Color.Black);

font = new Font("Arial", 8);

LoadImages(dirResources);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

using System.Net.Sockets;

using System.Net;

using System.Windows.Forms;

using System.Drawing;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

using Bomberman\_server;

namespace Bomberman\_client

{

public class Client

{

public readonly IPAddress ipAdress;

public readonly int portControl;

private int bufferSize;

public readonly IPEndPoint ipEndPointControl;

public readonly Socket socketControl;

private byte[] receivedData;

public GameClasses.GameCoreClient gameCore;

private BinaryFormatter serializer;

public readonly int id;

private bool isEnded = false;

public Client(string host, int portControl, string playerName)

{

//this.ipHost = Dns.GetHostEntry(host);

this.portControl = portControl;

this.ipAdress = IPAddress.Parse(host);

this.bufferSize = 1024 \* 1024;

this.ipEndPointControl = new IPEndPoint(ipAdress, portControl);

this.socketControl = new Socket(ipAdress.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

this.serializer = new BinaryFormatter();

receivedData = new byte[bufferSize];

try

{

socketControl.Connect(ipEndPointControl);

MemoryStream playerNameBuffer = new MemoryStream();

serializer.Serialize(playerNameBuffer, playerName);

socketControl.Send(playerNameBuffer.ToArray());

int countReceivedData = socketControl.Receive(receivedData);

id = BitConverter.ToInt32(receivedData, 0);

}

catch(Exception e)

{

throw e;

}

}

private void GetBufferFromServer(object sender, SocketAsyncEventArgs e)

{

MemoryStream stream = new MemoryStream(e.Buffer, 0, e.BytesTransferred);

if (stream.Capacity != 0)

{

try

{

ObjectsLists buffer = (ObjectsLists)serializer.Deserialize(stream);

lock (gameCore.bufferObjectsList)

{

gameCore.bufferObjectsList = buffer;

}

}

catch (Exception exception)

{

MessageBox.Show(exception.ToString(), "Bomberman", MessageBoxButtons.OK);

}

finally

{

if (!isEnded)

{

SocketAsyncEventArgs eventArgs = new SocketAsyncEventArgs();

eventArgs.Completed += GetBufferFromServer;

byte[] temp = new byte[1024 \* 1024];

eventArgs.SetBuffer(temp, 0, 1024 \* 1024);

(sender as Socket).ReceiveAsync(eventArgs);

}

}

}

}

public void SendMessageToServer(params int[] data)

{

List<byte> message = new List<byte>();

for (int i = 0; i < data.Length; i++ )

{

message.AddRange(BitConverter.GetBytes(data[i]));

}

if (!isEnded)

{

SocketAsyncEventArgs sendArgs = new SocketAsyncEventArgs();

sendArgs.SetBuffer(message.ToArray(), 0, message.Count);

socketControl.SendAsync(sendArgs);

}

}

public void Dispose()

{

SendMessageToServer(id, (int)KindMessages.KindMessage.Player, (int)KindMessages.KindPlayerMessages.Disconnect);

socketControl.Shutdown(SocketShutdown.Both);

socketControl.Disconnect(false);

socketControl.Close();

isEnded = true;

}

public void StartRecieving(object state)

{

SocketAsyncEventArgs eventArgs = new SocketAsyncEventArgs();

eventArgs.Completed += GetBufferFromServer;

byte[] temp = new byte[1024 \* 1024];

eventArgs.SetBuffer(temp, 0, 1024 \* 1024);

socketControl.ReceiveAsync(eventArgs);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Timers;

using Bomberman\_server;

namespace Bomberman\_client.GameClasses

{

[Serializable]

public class GameCoreServer

{

public delegate void SendFunc();

[NonSerialized]

private SendFunc sendFunc;

[NonSerialized]

public PhysicalMap map;

public ObjectsLists objectsList;

[NonSerialized]

public int delay;

[NonSerialized]

private Timer timer;

[NonSerialized]

private ScriptEngine scriptEngine;

[NonSerialized]

public Size playerSize;

[NonSerialized]

public Size playerOnDeathSize;

[NonSerialized]

public Size bombSize;

[NonSerialized]

public Size explosionSize;

[NonSerialized]

public Size wallSize;

[NonSerialized]

public List<Point> spawnPoints;

[NonSerialized]

public Random randomGen;

public void ChangePhysicalState()

{

lock (objectsList)

{

foreach (Player player in objectsList.players)

{

if (!player.IsDead)

{

player.ChangeMapMatrix(map);

}

}

for (int i = 0; i < objectsList.bombs.Count; i++)

{

objectsList.bombs[i].ChangeMapMatrix(map);

}

for (int i = 0; i < objectsList.explosions.Count; i++)

{

objectsList.explosions[i].ChangePhysicalMap(map);

}

for (int i = 0; i < objectsList.dynamicWalls.Count; i++)

{

if ((objectsList.dynamicWalls[i].isWallBlowedUp(map)) && (!objectsList.dynamicWalls[i].isBlowedUpNow))

{

objectsList.dynamicWalls[i].isBlowedUpNow = true;

StartDestroingDynamicWall(objectsList.dynamicWalls[i]);

}

objectsList.dynamicWalls[i].ChangeMapMatrix(map);

}

for (int i = 0; i < objectsList.staticWalls.Count; i++)

{

objectsList.staticWalls[i].ChangeMapMatrix(map);

}

foreach (Player player in objectsList.players)

{

if (player.isPlayerBlowedUp(map))

{

OnDeathPlayer(player);

}

}

}

}

public void TimerEvent(object sender, EventArgs e)

{

foreach (Player player in objectsList.players)

{

player.OnMove(map);

}

sendFunc();

map.ClearCurrMatrix();

map.SwitchMatrix();

ChangePhysicalState();

}

public void OnDeathPlayer(PhysicalObject player)

{

var temp = player as Player;

scriptEngine.StartSimpleScript(temp, playerOnDeathSize, DeletePlayerFromField, 200, 6);

temp.IsDying = true;

temp.isMoved = false;

}

public void DeletePlayerFromField(object player)

{

var temp = player as Player;

temp.IsDead = true;

temp.currSpriteOffset = 0;

}

public void ExplosionBomb(PhysicalObject bomb)

{

var temp = bomb as Bomb;

temp.isBlowedUp = true;

scriptEngine.StartSimpleScript(temp, DeleteBombFromField, 100, 3);

}

public void DeleteBombFromField(object bomb)

{

var temp = bomb as Bomb;

if (objectsList.bombs.IndexOf(temp) >= 0)

{

temp.owner.CurrCountBombs--;

objectsList.bombs.Remove(temp);

Explosion tempExpl = new Explosion(explosionSize, new Point(temp.X, temp.Y), (int)temp.owner.bombLevel, map, DeleteExplosionFromField);

objectsList.explosions.Add(tempExpl);

scriptEngine.StartExplosion(tempExpl, DeleteExplosionFromField, 100, 7);

}

}

public void DeleteExplosionFromField(object explosion)

{

var temp = explosion as Explosion;

objectsList.explosions.Remove(temp);

}

public void StartDestroingDynamicWall(DynamicWall wall)

{

scriptEngine.StartSimpleScript(wall, DeleteDynamicWallFromField, 200, 6);

}

public void DeleteDynamicWallFromField(object wall)

{

var temp = wall as DynamicWall;

if (objectsList.dynamicWalls.IndexOf(temp) >= 0)

{

objectsList.dynamicWalls.Remove(temp);

}

}

private void GenerateWalls()

{

const int sizeCell = 24;

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(2 \* sizeCell, 0 \* sizeCell), 4, MapGenerator.LineDirection.Horizontal, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(3 \* sizeCell, 1 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(1 \* sizeCell, 2 \* sizeCell), 1, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(2 \* sizeCell, 2 \* sizeCell), 2, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(0 \* sizeCell, 3 \* sizeCell), 1, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(1 \* sizeCell, 5 \* sizeCell), 1, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(3 \* sizeCell, 7 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(3 \* sizeCell, 11 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(4 \* sizeCell, 13 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(5 \* sizeCell, 13 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(5 \* sizeCell, 5 \* sizeCell), 5, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(6 \* sizeCell, 5 \* sizeCell), 2, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(7 \* sizeCell, 1 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(7 \* sizeCell, 8 \* sizeCell), 2, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(8 \* sizeCell, 9 \* sizeCell), 1, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(9 \* sizeCell, 0 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(9 \* sizeCell, 10 \* sizeCell), 2, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(10 \* sizeCell, 8 \* sizeCell), 2, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(10 \* sizeCell, 14 \* sizeCell), 2, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(11 \* sizeCell, 0 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(12 \* sizeCell, 3 \* sizeCell), 1, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(12 \* sizeCell, 6 \* sizeCell), 1, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(12 \* sizeCell, 12 \* sizeCell), 4, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(13 \* sizeCell, 0 \* sizeCell), 4, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(13 \* sizeCell, 5 \* sizeCell), 2, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(13 \* sizeCell, 11 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(15 \* sizeCell, 4 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.dynamicWalls.AddRange(MapGenerator.GenerateLineDynamicWall(new Point(15 \* sizeCell, 8 \* sizeCell), 3, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(1 \* sizeCell, 3 \* sizeCell), 1, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(0 \* sizeCell, 8 \* sizeCell), 5, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(1 \* sizeCell, 8 \* sizeCell), 5, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(2 \* sizeCell, 1 \* sizeCell), 1, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(4 \* sizeCell, 7 \* sizeCell), 6, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(4 \* sizeCell, 4 \* sizeCell), 4, MapGenerator.LineDirection.Horizontal, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(5 \* sizeCell, 1 \* sizeCell), 1, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(7 \* sizeCell, 7 \* sizeCell), 11, MapGenerator.LineDirection.Horizontal, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(8 \* sizeCell, 10 \* sizeCell), 4, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(11 \* sizeCell, 3 \* sizeCell), 4, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(11 \* sizeCell, 9 \* sizeCell), 5, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(17 \* sizeCell, 8 \* sizeCell), 6, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(14 \* sizeCell, 11 \* sizeCell), 2, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(15 \* sizeCell, 11 \* sizeCell), 2, MapGenerator.LineDirection.Vertical, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(14 \* sizeCell, 2 \* sizeCell), 3, MapGenerator.LineDirection.Horizontal, wallSize));

objectsList.staticWalls.AddRange(MapGenerator.GenerateLineStaticWall(new Point(14 \* sizeCell, 3 \* sizeCell), 3, MapGenerator.LineDirection.Horizontal, wallSize));

spawnPoints.Add(new Point(0 \* sizeCell, 0 \* sizeCell));

spawnPoints.Add(new Point(0 \* sizeCell, 14 \* sizeCell));

spawnPoints.Add(new Point(17 \* sizeCell, 0 \* sizeCell));

spawnPoints.Add(new Point(14 \* sizeCell, 13 \* sizeCell));

}

public void StartCore()

{

timer.Start();

}

public GameCoreServer

(

int width, int height, Size playerSize, Size playerOnDeathSize,

Size bombSize, Size explosionSize, Size wallSize, string dirResources, SendFunc sendFunc

)

{

this.map = new PhysicalMap(width, height);

objectsList = new ObjectsLists();

spawnPoints = new List<Point>();

this.sendFunc = sendFunc;

timer = new Timer();

delay = 60;

timer.Interval = delay;

timer.Elapsed += TimerEvent;

randomGen = new Random();

scriptEngine = new ScriptEngine();

this.playerSize = playerSize;

this.bombSize = bombSize;

this.explosionSize = explosionSize;

this.wallSize = wallSize;

this.playerOnDeathSize = playerOnDeathSize;

GenerateWalls();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Bomberman\_client.GameClasses;

using System.Drawing;

namespace Bomberman\_server

{

public class MapGenerator

{

public enum LineDirection { Horizontal, Vertical }

public static List<PhysicalObject> GenerateLineStaticWall(Point location, int count, LineDirection lineDirection, Size wallSize)

{

List<PhysicalObject> result = new List<PhysicalObject>();

switch (lineDirection)

{

case LineDirection.Horizontal:

{

for (int i = 0; i < count; i++)

{

result.Add(new PhysicalObject(new Point(location.X + wallSize.Width \* i, location.Y), wallSize));

}

}

break;

case LineDirection.Vertical:

{

for (int i = 0; i < count; i++)

{

result.Add(new PhysicalObject(new Point(location.X, location.Y + wallSize.Height \* i), wallSize));

}

}

break;

}

return result;

}

public static List<DynamicWall> GenerateLineDynamicWall(Point location, int count, LineDirection lineDirection, Size wallSize)

{

List<DynamicWall> result = new List<DynamicWall>();

switch (lineDirection)

{

case LineDirection.Horizontal:

{

for (int i = 0; i < count; i++)

{

result.Add(new DynamicWall(new Point(location.X + wallSize.Width \* i, location.Y), wallSize));

}

}

break;

case LineDirection.Vertical:

{

for (int i = 0; i < count; i++)

{

result.Add(new DynamicWall(new Point(location.X, location.Y + wallSize.Height \* i), wallSize));

}

}

break;

}

return result;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Bomberman\_client.GameClasses;

using System.Drawing;

using Bomberman\_client;

public class MessageAnalyzerServer

{

GameCoreServer gameCoreServer;

public int GetNextValue(byte[] message, ref int i)

{

int result = BitConverter.ToInt32(message, i);

i += sizeof(int);

return (result);

}

private Player FindPlayer(int id)

{

foreach (Player searchedPlayer in gameCoreServer.objectsList.players)

{

if (searchedPlayer.id == id)

{

return searchedPlayer;

}

}

return null;

}

public void AnalyzeMessage(byte[] message, ref bool isDisconnected)

{

int i = 0;

int idClient = GetNextValue(message, ref i);

int kindMessage;

kindMessage = GetNextValue(message, ref i);

switch (kindMessage)

{

case (int)KindMessages.KindMessage.Player:

{

int kindPlayerAction = GetNextValue(message, ref i);

switch (kindPlayerAction)

{

case (int)KindMessages.KindPlayerMessages.NewDirection:

{

Player searchedPlayer = FindPlayer(idClient);

searchedPlayer.direction = (Player.Direction)GetNextValue(message, ref i);

searchedPlayer.isMoved = true;

}

break;

case (int)KindMessages.KindPlayerMessages.Spawn:

{

Player searchedPlayer = FindPlayer(idClient);

if (searchedPlayer.IsDead)

{

int nextPos = gameCoreServer.randomGen.Next(gameCoreServer.spawnPoints.Count - 1);

while (searchedPlayer.prevLocation == (nextPos = gameCoreServer.randomGen.Next(gameCoreServer.spawnPoints.Count - 1)))

{

}

searchedPlayer.prevLocation = nextPos;

Point newLocation = gameCoreServer.spawnPoints[nextPos];

searchedPlayer.X = newLocation.X;

searchedPlayer.Y = newLocation.Y;

searchedPlayer.IsDying = false;

searchedPlayer.IsDead = false;

}

}

break;

case (int)KindMessages.KindPlayerMessages.Death:

{

}

break;

case (int)KindMessages.KindPlayerMessages.PlaceBomb:

{

Player searchedPlayer = FindPlayer(idClient);

if (searchedPlayer.CurrCountBombs != searchedPlayer.maxCountBombs)

{

gameCoreServer.objectsList.bombs.Add(searchedPlayer.bombFactory.GetBomb(searchedPlayer.bombLevel, new Point(searchedPlayer.X, searchedPlayer.Y)));

searchedPlayer.CurrCountBombs++;

}

}

break;

case (int)KindMessages.KindPlayerMessages.Connect:

{

//gameCoreServer.objectsList.players.Add(new Player(new Point(20, 20), gameCoreServer.playerSize, "", gameCoreServer.DeletePlayerFromField, gameCoreServer.bombTexture, gameCoreServer.bombSize, gameCoreServer.ExplosionBomb, idClient));

}

break;

case (int)KindMessages.KindPlayerMessages.Disconnect:

{

gameCoreServer.objectsList.players.Remove(FindPlayer(idClient));

isDisconnected = true;

}

break;

case (int)KindMessages.KindPlayerMessages.StopWalking:

{

Player searchedPlayer = FindPlayer(idClient);

searchedPlayer.isMoved = false;

}

break;

}

}

break;

case (int)KindMessages.KindMessage.Wall:

{

}

break;

}

}

public MessageAnalyzerServer(GameCoreServer gameCoreServer)

{

this.gameCoreServer = gameCoreServer;

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

using Bomberman\_client.GameClasses;

using System.Drawing;

namespace Bomberman\_server

{

class Program

{

static void Main(string[] args)

{

int width = 432;

int height = 384;

try

{

Console.WriteLine("IP:");

string IPAddress = Console.ReadLine();

ServerCore serverCore = new ServerCore(IPAddress, 11000, 5, 40);

serverCore.gameCoreServer = new GameCoreServer(width, height, new Size(24, 32), new Size(24, 24), new Size(24, 24), new Size(24, 24), new Size(24, 24), (Environment.CurrentDirectory + "\\Resources\\"), serverCore.SendData);

serverCore.messageAnalyzer = new MessageAnalyzerServer(serverCore.gameCoreServer);

Console.WriteLine("Server has been started. IP - {0}, port - {1}. To end from the program press enter", serverCore.ipAdress.ToString(), serverCore.portControl);

serverCore.gameCoreServer.StartCore();

ThreadPool.QueueUserWorkItem(serverCore.StartListen);

while (Console.ReadKey().Key != ConsoleKey.Enter)

{

}

}

catch (Exception e)

{

Console.WriteLine(e.ToString());

while (Console.ReadKey().Key != ConsoleKey.Enter)

{

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

using System.Threading;

using System.Timers;

using Bomberman\_client.GameClasses;

using System.Drawing;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

using Bomberman\_client;

namespace Bomberman\_server

{

public class ServerCore

{

public readonly IPAddress ipAdress;

public readonly int portControl;

public readonly int maxLengthQueue;

private int bufferSize;

public readonly IPEndPoint ipEndPointControl;

public readonly Socket socketListener;

public readonly List<Socket> socketsList;

public GameCoreServer gameCoreServer;

public BinaryFormatter serializer;

public MessageAnalyzerServer messageAnalyzer;

private int idCounter;

public ServerCore(string host, int portControl, int maxLengthQueue, int sendFrequency)

{

this.portControl = portControl;

this.maxLengthQueue = maxLengthQueue;

this.ipAdress = IPAddress.Parse(host);

this.bufferSize = 1024 \* 1024;

this.idCounter = 0;

this.ipEndPointControl = new IPEndPoint(ipAdress, portControl);

this.socketListener = new Socket(ipAdress.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

this.socketsList = new List<Socket>();

this.serializer = new BinaryFormatter();

try

{

socketListener.Bind(ipEndPointControl);

}

catch(Exception e)

{

throw e;

}

}

public void SendData()

{

lock (gameCoreServer.objectsList)

{

MemoryStream data = new MemoryStream();

ObjectsLists temp = gameCoreServer.objectsList;

serializer.Serialize(data, temp);

data.Flush();

foreach (Socket client in socketsList)

{

SocketAsyncEventArgs sendArgs = new SocketAsyncEventArgs();

sendArgs.UserToken = temp;

sendArgs.SetBuffer(data.ToArray(), 0, data.ToArray().Length);

client.SendAsync(sendArgs);

}

}

}

private void ReceiveCallback(object sender, SocketAsyncEventArgs e)

{

bool isDisconnected = false;

messageAnalyzer.AnalyzeMessage(e.Buffer, ref isDisconnected);

if (isDisconnected)

{

(sender as Socket).Shutdown(SocketShutdown.Both);

(sender as Socket).Disconnect(true);

socketsList.Remove((sender as Socket));

(sender as Socket).Close();

Console.WriteLine("user has been disconnected");

}

else

{

byte[] buffer = new byte[bufferSize];

SocketAsyncEventArgs eventArgs = new SocketAsyncEventArgs();

eventArgs.Completed += ReceiveCallback;

eventArgs.SetBuffer(buffer, 0, bufferSize);

(sender as Socket).ReceiveAsync(eventArgs);

}

}

private void SendId(Socket client)

{

SocketAsyncEventArgs sendArgs = new SocketAsyncEventArgs();

sendArgs.SetBuffer(BitConverter.GetBytes(idCounter), 0, sizeof(int));

idCounter++;

client.SendAsync(sendArgs);

}

private void AddPlayerToList(string playerName)

{

Point newLocation = gameCoreServer.spawnPoints[gameCoreServer.randomGen.Next(gameCoreServer.spawnPoints.Count - 1)];

gameCoreServer.objectsList.players.Add(new Player(newLocation, gameCoreServer.playerSize, playerName, gameCoreServer.DeletePlayerFromField, gameCoreServer.bombSize, gameCoreServer.DeleteBombFromField, idCounter - 1));

}

public void StartListen(object state)

{

socketListener.Listen(maxLengthQueue);

while (true)

{

Socket newClient = socketListener.Accept();

Console.WriteLine("New user connected to the server");

socketsList.Add(newClient);

byte[] buffer = new byte[bufferSize];

newClient.Receive(buffer);

MemoryStream playerNameStream = new MemoryStream(buffer);

string playerName = (string)serializer.Deserialize(playerNameStream);

SendId(newClient);

AddPlayerToList(playerName);

SocketAsyncEventArgs eventArgs = new SocketAsyncEventArgs();

eventArgs.Completed += ReceiveCallback;

eventArgs.SetBuffer(buffer, 0, bufferSize);

newClient.ReceiveAsync(eventArgs);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Bomberman\_client.GameClasses;

using System.Drawing;

using System.Timers;

namespace Bomberman\_client.GameClasses

{

[Serializable]

public class Bomb : PhysicalObject

{

private int areaOfExplosion;

public Player owner;

public bool isBlowedUp;

public void TimerEvent(object sender, EventArgs e)

{

(sender as Timer).Enabled = false;

(sender as Timer).Dispose();

deleteObjectFunc(this);

}

public override void ChangeMapMatrix(PhysicalMap PhysicalMap)

{

for (int i = Y; i < Y + size.Height; i++)

{

for (int j = X; j < X + size.Width; j++)

{

PhysicalMap.MapMatrix[i][j] = (int)PhysicalMap.KindOfArea.BOMB;

}

}

}

public Bomb(Point location, int areaOfExplosion, DeleteObjectFunc deleteBombFunc, Player owner)

: base(location)

{

Timer timerExplosion = new Timer();

timerExplosion.Interval = 3000;

this.deleteObjectFunc = deleteBombFunc;

timerExplosion.Elapsed += TimerEvent;

this.areaOfExplosion = areaOfExplosion;

timerExplosion.Enabled = true;

this.isBlowedUp = false;

this.owner = owner;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Bomberman\_client.GameClasses

{

[Serializable]

public class BombFactory

{

//private Image bombSprite;

private Size bombSize;

public PhysicalObject.DeleteObjectFunc deleteBombFunc;

private Player owner;

public Bomb GetBomb(Player.BombLevel bombLevel, Point location)

{

return new Bomb(location, (int)bombLevel, deleteBombFunc, owner );

}

public BombFactory(Size bombSize, PhysicalObject.DeleteObjectFunc deleteFunc, Player owner)

{

//this.bombSprite = bombSprite;

this.bombSize = bombSize;

deleteBombFunc = deleteFunc;

this.owner = owner;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

namespace Bomberman\_client.GameClasses

{

[Serializable]

public class Cell

{

private Point location;

public int X

{

set

{

location.X = value;

}

get

{

return location.X;

}

}

public int Y

{

set

{

location.Y = value;

}

get

{

return location.Y;

}

}

public Cell(Point location)

{

this.location = location;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Bomberman\_client.GameClasses

{

[Serializable]

public class DynamicWall : PhysicalObject

{

public bool isBlowedUpNow = false;

public bool isWallBlowedUp(PhysicalMap map)

{

for (int i = Y; i < Y + size.Height; i++)

{

for (int j = X; j < X + size.Width; j++)

{

if (map.MapMatrix[i][j] == (int)PhysicalMap.KindOfArea.EXPLOSION)

{

return true;

}

}

}

return false;

}

public DynamicWall(Point location, Size size)

: base(location, size)

{

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

namespace Bomberman\_client.GameClasses

{

[Serializable]

public class Explosion

{

private int radius;

public List<PartExplosion> partsExplosionLeft;

public List<PartExplosion> partsExplosionRight;

public List<PartExplosion> partsExplosionBottom;

public List<PartExplosion> partsExplosionUp;

public PartExplosion partExplosionCenter;

private Point explosionLocation;

private Point newLocation;

private int countParts;

public delegate void OnEndAllExplosionFunc(object sender);

private OnEndAllExplosionFunc onEndAllExplosionFunc;

Size size;

public enum KindExplosionTexture { explosionTextureHorizontalMiddle, explosionTextureLeftEdge, explosionTextureRightEdge, explosionTextureVerticalMiddle, explosionTextureUpEdge, explosionTextureBottomEdge, explosionTextureCenter };

private enum ExplosionDirection { UP, DOWN, LEFT, RIGHT };

public void ChangeState()

{

for (int i = 0; i < partsExplosionBottom.Count; i++)

{

partsExplosionBottom[i].ChangeState();

}

for (int i = 0; i < partsExplosionUp.Count; i++)

{

partsExplosionUp[i].ChangeState();

}

for (int i = 0; i < partsExplosionLeft.Count; i++)

{

partsExplosionLeft[i].ChangeState();

}

for (int i = 0; i < partsExplosionRight.Count; i++)

{

partsExplosionRight[i].ChangeState();

}

partExplosionCenter.ChangeState();

}

public void onEndExplosionFunc(object sender)

{

countParts--;

if (countParts == 0)

{

onEndAllExplosionFunc(this);

}

}

private bool canWePlaceExplosion(int X, int Y, PhysicalMap map, ExplosionDirection direction)

{

bool result = false;

lock (map.MapMatrix)

{

switch (direction)

{

case ExplosionDirection.UP:

{

for (int j = X; j < X + size.Width; j++)

{

for (int k = Y; (k < Y + size.Height) && (k < map.Height); k++)

{

if (map.MapMatrix[k][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

int i = Y;

while (map.MapMatrix[i][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

i++;

}

newLocation.X = X;

newLocation.Y = i - 1;

return true;

}

}

}

}

break;

case ExplosionDirection.DOWN:

{

for (int j = X; j < X + size.Width; j++)

{

for (int k = Y + size.Height; (k > Y) && (k > 0); k--)

{

if (map.MapMatrix[k][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

int i = Y + size.Height;

while (map.MapMatrix[i][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

i--;

}

newLocation.X = X;

newLocation.Y = i - size.Height + 2;

return true;

}

}

}

}

break;

case ExplosionDirection.LEFT:

{

for (int i = Y; i < Y + size.Height; i++)

{

for (int k = X; (k < X + size.Width) && (k < map.Width); k++)

{

if (map.MapMatrix[i][k] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

int j = X;

while (map.MapMatrix[i][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

j++;

}

newLocation.X = j;

newLocation.Y = Y;

return true;

}

}

}

}

break;

case ExplosionDirection.RIGHT:

{

for (int i = Y; i < Y + size.Height; i++)

{

for (int k = X + size.Width; (k > X) && (k > 0); k--)

{

if (map.MapMatrix[i][k] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

int j = X + size.Width;

while (map.MapMatrix[i][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

j--;

}

newLocation.X = j - size.Width + 2;

newLocation.Y = Y;

return true;

}

}

}

}

break;

}

return result;

}

}

private void InitExplosion

(

Size size, Point location, int radius,

PhysicalMap map

)

{

const int countStates = 7;

partsExplosionBottom = new List<PartExplosion>();

partsExplosionLeft = new List<PartExplosion>();

partsExplosionRight = new List<PartExplosion>();

partsExplosionUp = new List<PartExplosion>();

countParts = 0;

for (int i = 0; i < radius; i++)

{

if ( (location.X - size.Width \* (i + 1)) > 0)

{

if (!canWePlaceExplosion(location.X - size.Width \* (i), location.Y, map, ExplosionDirection.LEFT))

{

if (i < radius - 1)

{

partsExplosionLeft.Add(new PartExplosion(KindExplosionTexture.explosionTextureHorizontalMiddle, size, new Point((location.X - size.Width \* (i + 1)), location.Y), countStates, onEndExplosionFunc));

}

else

{

partsExplosionLeft.Add(new PartExplosion(KindExplosionTexture.explosionTextureLeftEdge, size, new Point((location.X - size.Width \* (i + 1)), location.Y), countStates, onEndExplosionFunc));

}

countParts++;

}

else

{

partsExplosionLeft.Add(new PartExplosion(KindExplosionTexture.explosionTextureHorizontalMiddle, size, new Point(newLocation.X, newLocation.Y), countStates, onEndExplosionFunc));

break;

}

}

else

{

if (i < radius - 1)

{

partsExplosionLeft.Add(new PartExplosion(KindExplosionTexture.explosionTextureHorizontalMiddle, size, new Point(0, location.Y), countStates, onEndExplosionFunc));

}

else

{

partsExplosionLeft.Add(new PartExplosion(KindExplosionTexture.explosionTextureLeftEdge, size, new Point(0, location.Y), countStates, onEndExplosionFunc));

}

countParts++;

break;

}

}

for (int i = 0; i < radius; i++)

{

if ((location.X + size.Width \* (i + 1) ) < map.Width)

{

if (!canWePlaceExplosion(location.X + size.Width \* (i), location.Y, map, ExplosionDirection.RIGHT))

{

if (i < radius - 1)

{

partsExplosionRight.Add(new PartExplosion(KindExplosionTexture.explosionTextureHorizontalMiddle, size, new Point((location.X + size.Width \* (i + 1)), location.Y), countStates, onEndExplosionFunc));

}

else

{

partsExplosionRight.Add(new PartExplosion(KindExplosionTexture.explosionTextureRightEdge, size, new Point((location.X + size.Width \* (i + 1)), location.Y), countStates, onEndExplosionFunc));

}

countParts++;

}

else

{

partsExplosionLeft.Add(new PartExplosion(KindExplosionTexture.explosionTextureHorizontalMiddle, size, new Point(newLocation.X, newLocation.Y), countStates, onEndExplosionFunc));

break;

}

}

else

{

if (i < radius - 1)

{

partsExplosionRight.Add(new PartExplosion(KindExplosionTexture.explosionTextureHorizontalMiddle, size, new Point((map.Width - size.Width), location.Y), countStates, onEndExplosionFunc));

}

else

{

partsExplosionRight.Add(new PartExplosion(KindExplosionTexture.explosionTextureRightEdge, size, new Point(map.Width - size.Width, location.Y), countStates, onEndExplosionFunc));

}

countParts++;

break;

}

}

for (int i = 0; i < radius; i++)

{

if ((location.Y - size.Height \* (i + 1)) > 0)

{

if (!canWePlaceExplosion(location.X, location.Y - size.Height \* (i), map, ExplosionDirection.UP))

{

if (i < radius - 1)

{

partsExplosionUp.Add(new PartExplosion(KindExplosionTexture.explosionTextureVerticalMiddle, size, new Point(location.X, location.Y - size.Height \* (i + 1)), countStates, onEndExplosionFunc));

}

else

{

partsExplosionUp.Add(new PartExplosion(KindExplosionTexture.explosionTextureUpEdge, size, new Point(location.X, location.Y - size.Height \* (i + 1)), countStates, onEndExplosionFunc));

}

countParts++;

}

else

{

partsExplosionUp.Add(new PartExplosion(KindExplosionTexture.explosionTextureVerticalMiddle, size, new Point(newLocation.X, newLocation.Y), countStates, onEndExplosionFunc));

break;

}

}

else

{

if (i < radius - 1)

{

partsExplosionUp.Add(new PartExplosion(KindExplosionTexture.explosionTextureVerticalMiddle, size, new Point(location.X, 0), countStates, onEndExplosionFunc));

}

else

{

partsExplosionUp.Add(new PartExplosion(KindExplosionTexture.explosionTextureUpEdge, size, new Point(location.X, 0), countStates, onEndExplosionFunc));

}

countParts++;

break;

}

}

for (int i = 0; i < radius; i++)

{

if ((location.Y + size.Height \* (i + 1)) < map.Height)

{

if (!canWePlaceExplosion(location.X, location.Y + size.Height \* (i), map, ExplosionDirection.DOWN))

{

if (i < radius - 1)

{

partsExplosionBottom.Add(new PartExplosion(KindExplosionTexture.explosionTextureVerticalMiddle, size, new Point(location.X, location.Y + size.Height \* (i + 1)), countStates, onEndExplosionFunc));

}

else

{

partsExplosionBottom.Add(new PartExplosion(KindExplosionTexture.explosionTextureBottomEdge, size, new Point(location.X, location.Y + size.Height \* (i + 1)), countStates, onEndExplosionFunc));

}

countParts++;

}

else

{

partsExplosionUp.Add(new PartExplosion(KindExplosionTexture.explosionTextureVerticalMiddle, size, new Point(newLocation.X, newLocation.Y), countStates, onEndExplosionFunc));

break;

}

}

else

{

if (i < radius - 1)

{

partsExplosionBottom.Add(new PartExplosion(KindExplosionTexture.explosionTextureVerticalMiddle, size, new Point(location.X, map.Height - size.Height), countStates, onEndExplosionFunc));

}

else

{

partsExplosionBottom.Add(new PartExplosion(KindExplosionTexture.explosionTextureBottomEdge, size, new Point(location.X, map.Height - size.Height), countStates, onEndExplosionFunc));

}

countParts++;

break;

}

}

partExplosionCenter = new PartExplosion(KindExplosionTexture.explosionTextureCenter, size, location, countStates, onEndExplosionFunc);

countParts++;

}

public void ChangePhysicalMap(PhysicalMap map)

{

for (int i = 0; i < partsExplosionBottom.Count; i++)

{

partsExplosionBottom[i].ChangePhysicalMap(map);

}

for (int i = 0; i < partsExplosionUp.Count; i++)

{

partsExplosionUp[i].ChangePhysicalMap(map);

}

for (int i = 0; i < partsExplosionLeft.Count; i++)

{

partsExplosionLeft[i].ChangePhysicalMap(map);

}

for (int i = 0; i < partsExplosionRight.Count; i++)

{

partsExplosionRight[i].ChangePhysicalMap(map);

}

partExplosionCenter.ChangePhysicalMap(map);

}

public void DrawPart(BufferedGraphics currBuffer, List<Bitmap> images, PartExplosion partExplosion)

{

lock (currBuffer)

{

lock (images)

{

lock (images[(int)partExplosion.kindExplosion])

{

currBuffer.Graphics.DrawImage(images[(int)partExplosion.kindExplosion], partExplosion.X, partExplosion.Y, new Rectangle(new Point(partExplosion.currSpriteOffset, 0), partExplosion.size), GraphicsUnit.Pixel);

}

}

}

}

public void DrawExplosion(BufferedGraphics currBuffer, List<Bitmap> images)

{

for (int j = 0; j < partsExplosionBottom.Count; j++)

{

DrawPart(currBuffer, images, partsExplosionBottom[j]);

}

for (int j = 0; j < partsExplosionUp.Count; j++)

{

DrawPart(currBuffer, images, partsExplosionUp[j]);

}

for (int j = 0; j < partsExplosionLeft.Count; j++)

{

DrawPart(currBuffer, images, partsExplosionLeft[j]);

}

for (int j = 0; j < partsExplosionRight.Count; j++)

{

DrawPart(currBuffer, images, partsExplosionRight[j]);

}

DrawPart(currBuffer, images, partExplosionCenter);

}

public Explosion

(

Size size, Point location, int radius,

PhysicalMap map, OnEndAllExplosionFunc funcEnd

)

{

onEndAllExplosionFunc = funcEnd;

this.size = size;

this.explosionLocation = location;

this.radius = radius;

InitExplosion(size, location, radius, map);

ChangeState();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Bomberman\_client.GameClasses

{

[Serializable]

public class PartExplosion : Cell

{

private int countStates;

private int currState;

public int currSpriteOffset;

public Size size;

private Explosion.OnEndAllExplosionFunc onEndFunc;

public Explosion.KindExplosionTexture kindExplosion;

public enum KindExplosionTexture { explosionTextureHorizontalMiddle, explosionTextureLeftEdge, explosionTextureRightEdge, explosionTextureVerticalMiddle, explosionTextureUpEdge, explosionTextureBottomEdge }

public void ChangeState()

{

currSpriteOffset += size.Width;

currState++;

}

public void ChangePhysicalMap(PhysicalMap map)

{

int firstBorder;

if (Y + size.Height > map.Height)

{

firstBorder = map.Height;

}

else

{

firstBorder = Y + size.Height;

}

int secondBorder;

if (X + size.Width > map.Width)

{

secondBorder = map.Width;

}

else

{

secondBorder = X + size.Width;

}

for (int i = Y; i < firstBorder; i++)

{

for (int j = X; j < secondBorder; j++)

{

map.MapMatrix[i][j] = (int)PhysicalMap.KindOfArea.EXPLOSION;

}

}

}

public PartExplosion(Explosion.KindExplosionTexture kindExplosion, Size size, Point location, int countStates, Explosion.OnEndAllExplosionFunc onEndFunc) : base(location)

{

this.kindExplosion = kindExplosion;

this.size = size;

this.countStates = countStates;

this.onEndFunc = onEndFunc;

currState = 0;

currSpriteOffset = 0;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Bomberman\_client.GameClasses

{

[Serializable]

public class PhysicalObject : Cell

{

public readonly Size size;

public int currSpriteOffset;

public virtual void ChangeMapMatrix(PhysicalMap PhysicalMap)

{

for (int i = Y; i < Y + size.Height; i++)

{

for (int j = X; j < X + size.Width; j++)

{

PhysicalMap.MapMatrix[i][j] = (int)PhysicalMap.KindOfArea.PHYSICACOBJECT;

}

}

}

public delegate void DeleteObjectFunc(PhysicalObject obj);

protected DeleteObjectFunc deleteObjectFunc;

public PhysicalObject(Point location)

: base(location)

{

// size = texture.Size;

currSpriteOffset = 0;

}

public PhysicalObject(Point location, Size size)

: base(location)

{

this.size = size;

currSpriteOffset = 0;

}

public PhysicalObject(Point location, Size spriteSize, DeleteObjectFunc deleteObjectFunc )

: base(location)

{

this.size = spriteSize;

this.deleteObjectFunc = deleteObjectFunc;

currSpriteOffset = 0;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

namespace Bomberman\_client.GameClasses

{

[Serializable]

public class Player : PhysicalObject, GameInterfaces.IMovable

{

public enum Direction { UP, DOWN, LEFT, RIGHT };

public Direction direction { get; set; }

public Direction prevDirection { get; set; }

public enum AnimState { TURNUP, TURNUP1, TURNUP2, TURNDOWN, TURNDOWN1, TURNDOWN2, TURNLEFT, TURNLEFT1, TURNLEFT2, TURNRIGHT, TURNRIGHT1, TURNRIGHT2 };

public AnimState currAnimState;

public enum BombLevel { low = 2, medium = 3, high = 4 };

public BombLevel bombLevel = BombLevel.low;

[NonSerialized]

public int prevLocation;

private int step = 4;

public readonly int maxCountBombs;

private int currCountBombs;

public int CurrCountBombs

{

get

{

return currCountBombs;

}

set

{

currCountBombs = value;

}

}

public delegate void SpawnPlayerFunc();

private bool isDead;

public bool IsDead

{

set

{

isDead = value;

}

get

{

return isDead;

}

}

private bool isDying;

public bool IsDying

{

set

{

isDying = value;

}

get

{

return isDying;

}

}

public BombFactory bombFactory;

public bool isMoved;

private Point newLocation = new Point();

public readonly int id;

/\*

private bool isObjectOnDirection(byte point)

{

return( (point == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT) || (point == (int)PhysicalMap.KindOfArea.PLAYER) || (point == (int)PhysicalMap.KindOfArea.BOMB) );

}

\*/

public bool isObjectOnWay(PhysicalMap map)

{

lock (map.MapMatrix)

{

bool result = false;

switch (direction)

{

case Direction.UP:

{

for (int j = X; j < X + size.Width; j++)

{

if (map.MapMatrix[Y - 1][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT )

{

int i = Y;

while ( (map.MapMatrix[i][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT))

{

i++;

}

newLocation.X = X;

newLocation.Y = i;

return true;

}

}

}

break;

case Direction.DOWN:

{

for (int j = X; j < X + size.Width; j++)

{

if (map.MapMatrix[Y + size.Height][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

int i = Y + size.Height;

while ( (map.MapMatrix[i][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT))

{

i--;

}

newLocation.X = X;

newLocation.Y = i - size.Height + 1;

return true;

}

}

}

break;

case Direction.LEFT:

{

for (int i = Y; i < Y + size.Height; i++)

{

if (map.MapMatrix[i][X - 1] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

int j = X;

while ((map.MapMatrix[i][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT))

{

j++;

}

newLocation.X = j;

newLocation.Y = Y;

return true;

}

}

}

break;

case Direction.RIGHT:

{

for (int i = Y; i < Y + size.Height; i++)

{

if (map.MapMatrix[i][X + size.Width] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT)

{

int j = X + size.Width;

while ((map.MapMatrix[i][j] == (int)PhysicalMap.KindOfArea.PHYSICACOBJECT))

{

j--;

}

newLocation.X = j - size.Width + 1;

newLocation.Y = Y;

return true;

}

}

}

break;

}

return result;

}

}

public override void ChangeMapMatrix(PhysicalMap PhysicalMap)

{

for (int i = Y; i < Y + size.Height; i++)

{

for (int j = X; j < X + size.Width; j++)

{

PhysicalMap.MapMatrix[i][j] = (int)PhysicalMap.KindOfArea.PLAYER;

}

}

}

public void OnMove(PhysicalMap map)

{

if (!isDying)

{

if (isMoved)

{

switch (direction)

{

case Player.Direction.UP:

{

if (Y - step > 0)

{

if (!isObjectOnWay(map))

{

Y -= step;

if (currAnimState == AnimState.TURNUP)

{

currAnimState = AnimState.TURNUP1;

}

else

{

if (currAnimState == AnimState.TURNUP1)

{

currAnimState = AnimState.TURNUP2;

}

else

{

currAnimState = AnimState.TURNUP1;

}

}

}

else

{

X = newLocation.X;

Y = newLocation.Y;

}

}

else

{

Y = 0;

}

}

break;

case Player.Direction.DOWN:

{

if ((map.Height > (Y + size.Height + step)))

{

if (!isObjectOnWay(map))

{

Y += step;

if (currAnimState == AnimState.TURNDOWN)

{

currAnimState = AnimState.TURNDOWN1;

}

else

{

if (currAnimState == AnimState.TURNDOWN1)

{

currAnimState = AnimState.TURNDOWN2;

}

else

{

currAnimState = AnimState.TURNDOWN1;

}

}

}

else

{

X = newLocation.X;

Y = newLocation.Y;

}

}

else

{

Y = map.Height - size.Height;

}

}

break;

case Player.Direction.LEFT:

{

if (X - step > 0)

{

if (!isObjectOnWay(map))

{

X -= step;

if (currAnimState == AnimState.TURNLEFT)

{

currAnimState = AnimState.TURNLEFT1;

}

else

{

if (currAnimState == AnimState.TURNLEFT1)

{

currAnimState = AnimState.TURNLEFT2;

}

else

{

currAnimState = AnimState.TURNLEFT1;

}

}

}

else

{

X = newLocation.X;

Y = newLocation.Y;

}

}

else

{

X = 0;

}

}

break;

case Player.Direction.RIGHT:

{

if ((map.Width > (X + size.Width + step)))

{

if (!isObjectOnWay(map))

{

X += step;

if (currAnimState == AnimState.TURNRIGHT)

{

currAnimState = AnimState.TURNRIGHT1;

}

else

{

if (currAnimState == AnimState.TURNRIGHT1)

{

currAnimState = AnimState.TURNRIGHT2;

}

else

{

currAnimState = AnimState.TURNRIGHT1;

}

}

}

else

{

X = newLocation.X;

Y = newLocation.Y;

}

}

else

{

X = map.Width - size.Width;

}

}

break;

}

}

else

{

switch (direction)

{

case Direction.UP:

{

currAnimState = AnimState.TURNUP;

}

break;

case Direction.DOWN:

{

currAnimState = AnimState.TURNDOWN;

}

break;

case Direction.LEFT:

{

currAnimState = AnimState.TURNLEFT;

}

break;

case Direction.RIGHT:

{

currAnimState = AnimState.TURNRIGHT;

}

break;

}

}

}

}

public bool isPlayerBlowedUp(PhysicalMap map)

{

for (int i = Y; i < Y + size.Height; i++)

{

for (int j = X; j < X + size.Width; j++)

{

if (map.MapMatrix[i][j] == (int)PhysicalMap.KindOfArea.EXPLOSION)

{

return true;

}

}

}

return false;

}

private string thisName;

public string PlayerName

{

set

{

thisName = value;

}

get

{

return thisName;

}

}

public Bitmap GetAnimState(Bitmap texture)

{

lock (texture)

{

Bitmap result = null;

Bitmap currBitmap = texture as Bitmap;

switch (currAnimState)

{

case AnimState.TURNUP:

{

result = currBitmap.Clone(new Rectangle(new Point(0, size.Height), size), texture.PixelFormat);

}

break;

case AnimState.TURNUP1:

{

result = currBitmap.Clone(new Rectangle(new Point(size.Width, size.Height), size), texture.PixelFormat);

}

break;

case AnimState.TURNUP2:

{

result = currBitmap.Clone(new Rectangle(new Point(size.Width \* 2, size.Height), size), texture.PixelFormat);

}

break;

case AnimState.TURNDOWN:

{

result = currBitmap.Clone(new Rectangle(new Point(0, 0), size), texture.PixelFormat);

}

break;

case AnimState.TURNDOWN1:

{

result = currBitmap.Clone(new Rectangle(new Point(size.Width, 0), size), texture.PixelFormat);

}

break;

case AnimState.TURNDOWN2:

{

result = currBitmap.Clone(new Rectangle(new Point(size.Width \* 2, 0), size), texture.PixelFormat);

}

break;

case AnimState.TURNLEFT:

{

result = currBitmap.Clone(new Rectangle(new Point(size.Width \* 3, size.Height), size), texture.PixelFormat);

}

break;

case AnimState.TURNLEFT1:

{

result = currBitmap.Clone(new Rectangle(new Point(size.Width \* 4, size.Height), size), texture.PixelFormat);

}

break;

case AnimState.TURNLEFT2:

{

result = currBitmap.Clone(new Rectangle(new Point(size.Width \* 5, size.Height), size), texture.PixelFormat);

}

break;

case AnimState.TURNRIGHT:

{

result = currBitmap.Clone(new Rectangle(new Point(size.Width \* 3, 0), size), texture.PixelFormat);

}

break;

case AnimState.TURNRIGHT1:

{

result = currBitmap.Clone(new Rectangle(new Point(size.Width \* 4, 0), size), texture.PixelFormat);

}

break;

case AnimState.TURNRIGHT2:

{

result = currBitmap.Clone(new Rectangle(new Point(size.Width \* 5, 0), size), texture.PixelFormat);

}

break;

}

return result;

}

}

public Player(Point location, Size spriteSize, string name, DeleteObjectFunc deletePlayerFunc, Size bombSize, DeleteObjectFunc deleteBombFunc, int id)

: base(location, spriteSize, deletePlayerFunc)

{

thisName = name;

isMoved = false;

isDead = false;

isDying = false;

maxCountBombs = 2;

currCountBombs = 0;

this.id = id;

this.currSpriteOffset = 0;

this.prevLocation = 0;

bombFactory = new BombFactory(bombSize, deleteBombFunc, this);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Timers;

namespace Bomberman\_client.GameClasses

{

public class ExplosionScript

{

private int countStates;

private int currState;

private Explosion obj;

private ScriptEngine.OnEndFunc onEnd;

public Timer currTimer;

public void OnTimerEvent(object sender, EventArgs e)

{

currState--;

if (currState == 0)

{

currTimer.Enabled = false;

currTimer.Dispose();

currTimer = null;

onEnd(obj);

}

else

{

obj.ChangeState();

}

}

public ExplosionScript(Explosion explosion, ScriptEngine.OnEndFunc onEndFunc, int countStates, int delay)

{

currTimer = new Timer();

currTimer.Interval = delay;

this.onEnd = onEndFunc;

this.countStates = countStates;

this.currState = this.countStates;

this.obj = explosion;

currTimer.Elapsed += OnTimerEvent;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Bomberman\_client.GameClasses

{

public class PhysicalMap

{

public enum KindOfArea { NONE = 0, PHYSICACOBJECT = 1, EXPLOSION = 2, PLAYER = 3, BOMB = 4 };

private byte[][] mapMatrix1;

private byte[][] mapMatrix2;

public enum NumbOfMapMatrix { FIRST, SECOND };

public NumbOfMapMatrix currMatrix;

public byte[][] MapMatrix

{

get

{

switch(currMatrix)

{

case NumbOfMapMatrix.FIRST:

{

return mapMatrix1;

}

break;

case NumbOfMapMatrix.SECOND:

{

return mapMatrix2;

}

break;

default:

{

return mapMatrix1;

}

break;

}

}

private set

{

}

}

private int width;

public int Width

{

get

{

return width;

}

private set

{

}

}

private int height;

public int Height

{

get

{

return height;

}

private set

{

}

}

public PhysicalMap(byte[][] mapMatrix)

{

this.mapMatrix1 = mapMatrix;

}

public void ClearCurrMatrix()

{

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

MapMatrix[i][j] = (int)KindOfArea.NONE;

}

}

}

public void SwitchMatrix()

{

if (currMatrix == NumbOfMapMatrix.FIRST)

{

currMatrix = NumbOfMapMatrix.SECOND;

}

else

{

currMatrix = NumbOfMapMatrix.FIRST;

}

}

public PhysicalMap(int width, int height)

{

mapMatrix1 = new byte[height][];

mapMatrix2 = new byte[height][];

for (int i = 0; i < height; i++)

{

mapMatrix1[i] = new byte[width];

}

for (int i = 0; i < height; i++)

{

mapMatrix2[i] = new byte[width];

}

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

mapMatrix1[i][j] = (int)KindOfArea.NONE;

}

}

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

mapMatrix1[i][j] = (int)KindOfArea.NONE;

}

}

currMatrix = NumbOfMapMatrix.FIRST;

this.width = width;

this.height = height;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Bomberman\_client.GameClasses;

namespace Bomberman\_server

{

[Serializable]

public class ObjectsLists

{

public List<Bomb> bombs;

public List<Explosion> explosions;

public List<PhysicalObject> staticWalls;

public List<DynamicWall> dynamicWalls;

public List<Player> players;

public ObjectsLists()

{

bombs = new List<Bomb>();

explosions = new List<Explosion>();

staticWalls = new List<PhysicalObject>();

dynamicWalls = new List<DynamicWall>();

players = new List<Player>();

}

public ObjectsLists(List<Bomb> bombs, List<Explosion> explosions, List<PhysicalObject> staticWalls, List<DynamicWall> dynamicWalls, List<Player> players)

{

this.bombs = bombs;

this.explosions = explosions;

this.staticWalls = staticWalls;

this.dynamicWalls = dynamicWalls;

this.players = players;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Timers;

namespace Bomberman\_client.GameClasses

{

public class ScriptEngine

{

public delegate void OnEndFunc(object sender);

public delegate void OnChangeFunc();

public delegate void TimerEvent(object sender, EventArgs e);

public void StartSimpleScript(PhysicalObject obj, OnEndFunc onEndFunc, int delay, int countStates)

{

SimpleScript states = new SimpleScript(obj, onEndFunc, countStates, delay);

states.currTimer.Enabled = true;

}

public void StartSimpleScript(PhysicalObject obj, Size onScriptSize, OnEndFunc onEndFunc, int delay, int countStates)

{

SimpleScript states = new SimpleScript(obj, onScriptSize, onEndFunc, countStates, delay);

states.currTimer.Enabled = true;

}

public void StartExplosion(Explosion explosion, OnEndFunc onEndFunc, int delay, int countStates)

{

ExplosionScript script = new ExplosionScript(explosion, onEndFunc, countStates, delay);

script.currTimer.Start();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Timers;

namespace Bomberman\_client.GameClasses

{

class SimpleScript

{

private int countStates;

private int currState;

private dynamic obj;

private ScriptEngine.OnEndFunc onEnd;

//private Bitmap sprite;

public Timer currTimer;

private Bitmap temp;

Size onScriptSize;

public void OnTimerEvent(object sender, EventArgs e)

{

currState--;

if (currState == 0)

{

currTimer.Stop();

currTimer.Dispose();

currTimer = null;

//temp.Dispose();

//sprite.Dispose();

onEnd(obj);

}

else

{

SetOffset();

}

}

private void SetOffset()

{

//temp = new Bitmap(sprite.Clone(new Rectangle(new Point(((countStates - 1) - (currState - 1)) \* onScriptSize.Width, 0), onScriptSize), sprite.PixelFormat));

obj.currSpriteOffset = ((countStates - 1) - (currState - 1)) \* onScriptSize.Width;

}

public SimpleScript(PhysicalObject obj, ScriptEngine.OnEndFunc onEndFunc, int countStates, int delay )

{

currTimer = new Timer();

currTimer.Interval = delay;

this.obj = obj;

this.onEnd = onEndFunc;

this.countStates = countStates;

this.currState = this.countStates;

//this.sprite = new Bitmap (sprite as Bitmap);

this.onScriptSize = obj.size;

currTimer.Elapsed += OnTimerEvent;

}

public SimpleScript(PhysicalObject obj, Size onScriptSize, ScriptEngine.OnEndFunc onEndFunc, int countStates, int delay)

{

currTimer = new Timer();

currTimer.Interval = delay;

this.obj = obj;

this.onEnd = onEndFunc;

this.countStates = countStates;

this.currState = this.countStates;

//this.sprite = new Bitmap(sprite as Bitmap);

this.onScriptSize = onScriptSize;

OnTimerEvent(currTimer, new EventArgs());

currTimer.Elapsed += OnTimerEvent;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Bomberman\_client.GameClasses;

namespace Bomberman\_client.GameInterfaces

{

public interface IMovable

{

void OnMove(PhysicalMap PhysicalMap);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Bomberman\_client

{

public class KindMessages

{

public enum KindMessage { Player = 0, Wall = 1 };

public enum KindPlayerMessages { NewDirection = 0, Spawn = 1, Death = 2, PlaceBomb = 3, Connect = 4, Disconnect = 5, Location = 6, StopWalking = 7 };

public enum Direction { UP, DOWN, LEFT, RIGHT };

}

}