СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На данный момент уровень развития вычислительной техники позволяет использовать её не только лишь в научной деятельности, где важны быстрые и точные машинные вычисления, но и в области развлечения. Компьютерные игры и всевозможные приложения со временем становятся всё более красочными и реалистичными, вместе с увеличением данного показателя растёт и количество тех, кто любит расслабиться с их помощью. Примеров таких приложений можно привести очень большое количество, однако остановимся на игровом приложении «Bomberman», более известное в русскоязычных кругах как «Бомбермэн».

Впервые на рынок данное игровое приложение вышло двадцать восьмого апреля тысяча девятьсот восемьдесят третьего года в Японии, а затем двадцатого ноября того же года и на европейский рынок.

Сценарий игры изначально отсутствовал, целью игры являлось либо уничтожение всех врагов, либо пройти уровень за определённое отведённое время. Игрок имел возможность ставить бомбы, которые взрываются через несколько секунд и способные разрушить некоторые блоки. За этими блоками могли скрываться различного рода бонусы: увеличение скорости, увеличение количества бомб либо увеличение скорости передвижения персонажа. Из данного функционала складывалось отличное и захватывающее игровое приложение.

Со временем данное игровое приложение обрело очень большое количество интерпретаций, которые пользуются успехом и по сегодняшний день.

Таким образом, данное игровое приложение до сих пор остаётся популярной и актуальной в широком кругу людей. Игровое приложение «Bomberman» может стать темой курсового проекта, целью которого будет создание новой интерпретации данного игрового приложения.

В курсовом проекте будет создан новый вариант данного игрового приложения в классическом стиле в 2D графике с простым и интуитивно понятным интерфейсом и достаточным функционалом, чтобы полностью передать суть данного игрового приложения и обеспечить приятное времяпрепровождение.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

Существует довольно много аналогов данного игрового приложения. В ходе их поиска и изучения наибольшее внимание уделяется программам для операционной системы Windows, онлайн-аналогам и приложениям под игровые консоли.

На рассмотрение представлено игровое приложение «Bomberman», которое было разработано под игровую консоль Nintendo Entertainment System (NES). Вид игрового окна приложения представлен на рисунке 1.1.

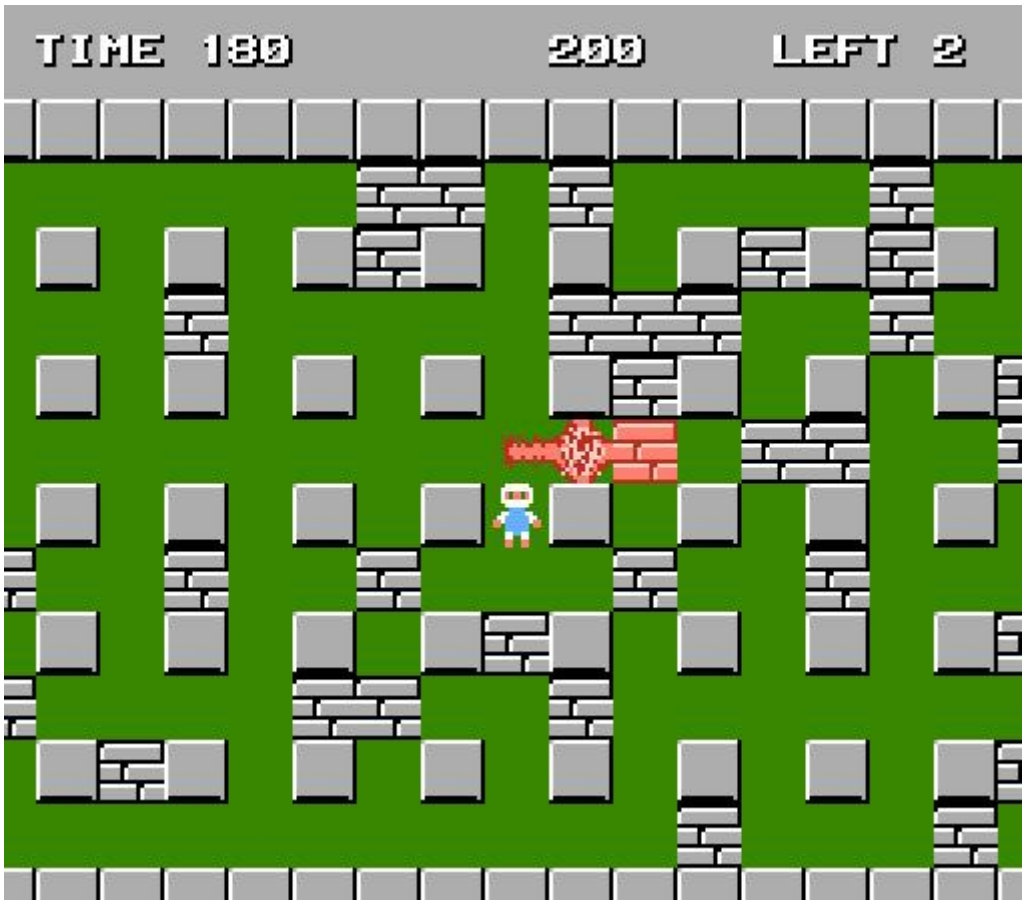

Рисунок 1.1 – Основной экран программы «Bomberman» в NES

Данное приложение обладает примитивным, минималистичным интерфейсом, который не отвлекает от игрового процесса.

Для обеспечения игрового процесса в приложении присутствуют следующие бонусы:

- Огонь: Увеличивает мощность взрыва;
- Бомба: Дополнительная бомба;
- Бомба с фитилём: Возможность взрывать бомбы нажатием кнопки;
- Роликовые коньки: Увеличивает скорость;

– Бомбермэн и стенка: Способность проходить сквозь стены;
– Бомба и стенка: Способность проходить сквозь бомбы;
– Вопросительный знак: Неуязвимость на время.

Главная задача – за отведённое время уничтожить всех врагов на уровне и найти дверь, ведущую на следующий уровень.

Список врагов для обеспечения игрового процесса:

– Шарик: Глупый, движется медленно;
– Луковица: Движется быстро, может преследовать героя;
– Бочка: Глупый, движется быстро;
– Апельсин: Преследует героя, по скорости уступает медведю;
– Амёба: Очень медленный, может проходить сквозь стены;
– Призрак: Движется с нормальной скоростью, может проходить сквозь стены;
– Медведь: Умный, движется быстро, преследует героя;
– Копейка: Умный, движется быстро, может проходить сквозь стены.

Другим аналогом является игровое приложение «Atomic Bomberman». Данное приложение было разработано под платформу Windows для ПК. Вид игрового приложения представлен на рисунке 1.2.

Рисунок 1.2 – Внешний вид окна приложения «Atomic Bomberman» в Windows

Для обеспечения игрового процесса в приложении присутствуют следующие бонусы:

- Дополнительные бомбы;
- Увеличение длины пламени, вплоть до бесконечности;
- Череп: Даёт болезни, например, минимального размера пламя, невозможность ставить бомбы и т.д.;
- Чёрный череп: Даёт до трёх болезней одновременно;
- Ускорение;
- Возможность двигать бомбы;
- Возможность поставить все имеющиеся бомбы в ряд;
- Возможность кидать бомбы;
- Возможность бросаться бомбами;
- Замедление;
- Превращение бомбы в «желе»;
- Установка триггера на бомбу;
- Случайный бонус.

Особенностью данной версии игрового приложения заключается в отсутствии класса врага, так как добавлена возможность многопользовательской игры и реализации искусственного интеллекта для оставшихся объектов бомбермэна, в основе которого лежит цель выжить самому, а не убить врага любой ценой, поэтому противники не столь агрессивны, как в предыдущих аналогах приложения.

Добавлена различного рода анимация, улучшающая восприятие игрового процесса.

Добавлена возможность выбора карты, из списка предложенных, вручную либо случайным образом.

Следующим аналогом является игровое приложение «Bomberman Online», разработанное под игровую консоль Sega Dreamcast. Вид окна игрового приложения представлен на рисунке 1.3.

Рисунок 1.2 – Внешний вид окна приложения «Microsoft Minesweeper»

Для обеспечения игрового процесса в приложении присутствуют следующие бонусы:

- - Дополнительные бомбы;
- Увеличение длины пламени;
- Ускорение;
- Добавление и улучшение брони;
- И т.д.

В связи с введением многопользовательского режима игры были введены следующие стили игры:

- Survival Rule – «Электрические драконы» - это классический стиль, перенесённый из предыдущих аналогов приложения.
- Hyper Bomber Rule – «Красные фениксы» - это стиль, где враги и сам игрок может погибать неограниченное число раз, при этом теряя собранные предметы и призы, целью является сбор бонусов и улучшений;

- Submarine Rule – «Морские принцессы» - это стиль, где карта представляет собой арену, поделенную на 2 части, целью является уничтожение всех противников на арене;
- Ring Match Rule – «Штормовые гиганты» - это стиль, где участники не погибают, а вылетают с арены, целью является достижение максимального числа сброшенных противников;
- Panel Paint Rule – «Железные бульдозеры» - это стиль, где участники с помощью бомб переворачивают плиты на полу арены, целью является перевернуть на свою сторону как можно больше панелей.

Особенностью данного приложения является привлекательный графический интерфейс, хорошая анимация, многопользовательский режим и введение различных стилей игры.

В связи с большим количеством аналогов игрового приложения «Bomberman» были приведены лишь некоторые аналоги, наиболее выделяющиеся и известные пользователям.

1.2 Постановка задачи

В рамках данного курсового проекта должно быть разработано игровое приложение «Bomberman» с графическим представлением в 2D.

В процессе реализации программного средства должны быть разработаны алгоритмы корректного взаимодействия персонажа пользователя с окружающими его объектами.

В результате анализа аналогов данного игрового приложения принято решение выбрать минималитичный, простой интерфейс игрового приложения и создание многопользовательского режима на одном устройстве с использованием необходимого минимума дополнительных функций, с целью не загромождения игрового приложения лишним функционалом, обнаруженным в исследованных аналогах.

В программном средстве планируется реализовать следующие функции:

- Генерация поля бонусов;
- Решение возможных коллизий персонажа с элементами карты;
- Решение возможных коллизий персонажа с объектами бомб;
- Анимированние персонажа и карты;
- -Завершение игры после попадание в огонь от взрыва бомбы;

Для разработки программного средства будет использоваться язык программирования C++ и среда разработки Visual Studio 2015, а также использование сторонней библиотеки SFML для обеспечения графического интерфейса и удобства работы с ним.

2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Интерфейс и графические элементы программного средства

Для создания и обработки элементов графического интерфейса используется графический редактор Adobe Photoshop CC 2015. Далее будет приведён список используемых графических ресурсов.

Для отображения информации о значении полей бонусов персонажа используется изображение, отображающее панель, где будет выводиться информация о персонаже. Данная панель представлена на рисунке 2.1.

Рисунок 2.1 – Изображение панели для отображения информации о персонаже

Для отображения текстуры персонажа и его движения используется набор спрайтов содержащихся в изображении, которое представлено на рисунке 2.2.

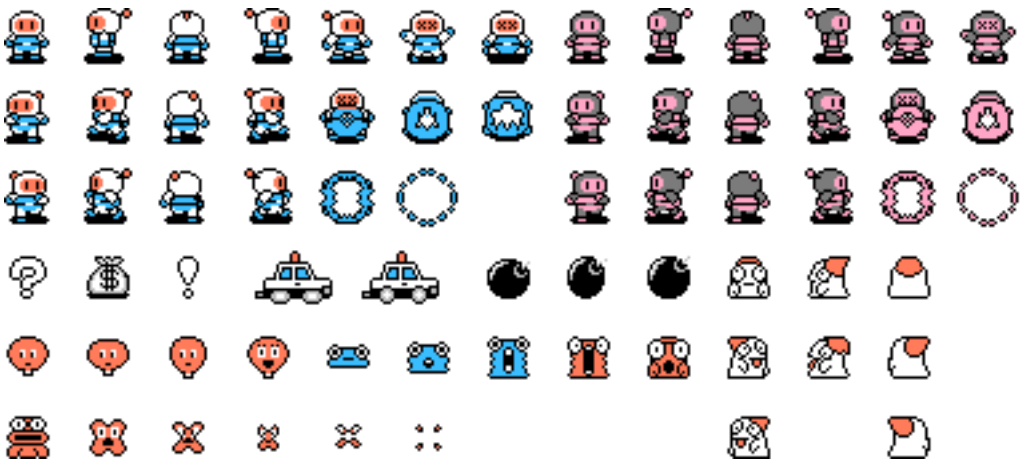

Рисунок 2.2 – Изображение, содержащее текстуры персонажа и его движения

Для отображения текстур карты и бонусов используется набор спрайтов, содержащихся в изображении, которое представлено на рисунке 2.3.

Рисунок 2.3 – Изображение, содержащее текстуры карты и бонусов

Для отображения текстур бомбы и её взрыва используется набор спрайтов, содержащихся в изображении, которое представлено на рисунке 2.4.

Рисунок 2.3 – Изображение, содержащее текстуры карты и бонусов

Данные изображения были заимствованы из внешнего источника http://randomhoohaas.flyingomelette.com/Sprites/spr-bomberman.php. Данный источник содержит бесплатные изображения.

2.2 Представление карты

Для отображения полноценной карты был выбран способ хранения карты в виде массива символов. Вследствие последующего наложения символов карты на символы бонусов и взрывов было решено объявить три массива карты, непосредственно массив карты, массив бонусов и массив взрыва бомбы, которые имеют названия TileMap, TileBonusMap и TileBoomMap соответственно.

Обозначения для TileMap:

- ‘M’ – не разрушаемая часть карты;
- ‘S’ – разрушаемая стена;
- ‘ ’ – покрытие пол;
- ‘1’ – первая стадия разрушения стены;
- ‘2’ – вторая стадия разрушения стены;
- ‘3’ – третья стадия разрушения стены;
- ‘4’ – четвёртая стадия разрушения стены;
- ‘5’ – пятая стадия разрушения стены;

Обозначения для TileBonusMap:

- ‘B’ – бонус на увеличение количества бомб;
- ‘F’ – бонус на увеличение дальности пламени бомб;
- ‘S’ – бонус на увеличение скорости персонажа;

Обозначения для TileBoomMap:

- ‘H’ – горизонтальное направление пламени;
- ‘R’ – крайнее правое направление пламени;
- ‘L’ – крайнее левое направление пламени;
- ‘V’ – вертикальное направление пламени;
- ‘U’ – крайнее верхнее направление пламени;
- ‘D’ – крайнее нижнее направление пламени;

Данные массивы хранятся в заголовочном файле “Map.h”. Помимо самих массивов для отображения карты в данном файле хранятся методы

GetRandomCoordinate(), SetBonusOnMap(), GenerateBonusMap(), SetBoomOnMap(), ResetMaps().

Методы GetRandomCoordinate(), SetBonusOnMap(), GenerateBonusMap() выполняют генерацию поля бонусов случайным образом. В результат работы этих методов можно показать через блок-схему, расположенную ниже:

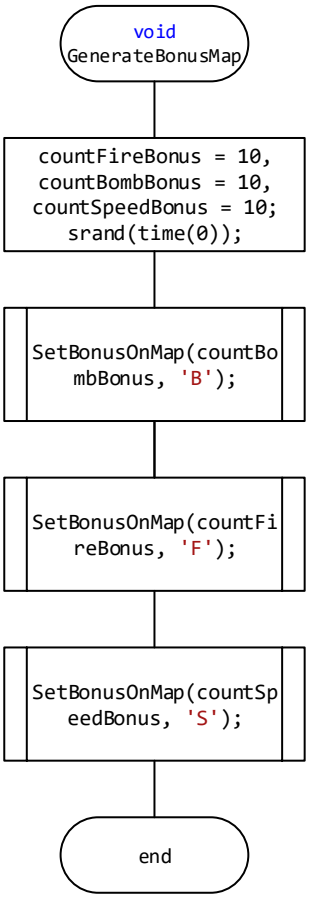

Рисунок 2.4 – Блок-схема метода

Метод ResetMaps() производит переинициализацию массивов карт TileMap, TileBonusMap и TileBoomMap.

2.3 Объявление классов

Для отображения бомбы и возможности взаимодействия персонажа с бомбой объявим класс Bomb:

```
class Bomb
{

private:
	float x, y, width, heigth;
	float nativeTime, boomTime;
public:
	int frame;
	int fire;
	sf::String File;
	sf::Image image;
	sf::Texture texture;
	sf::Sprite sprite;
	sf::Sound soundBoom;
```

```
        bool visible, inPlayer, isBoom, enabled;
        int fireState;

        Bomb();
        {
        }

        void Create(sf::String F, sf::Sound *SoundBoom, float X, float Y, float Width,
float Heigth);
        void Reset();
        void SetPosition(float X, float Y);
        void SetTileDestroyStone(int positionY, int positionX);
        void CheckBombOnFire(float x, float y)
        void Update(float time)
        float GetPositionX()
        float GetPositionY()
        float GetHeigth()
        float GetWidth()
        float GetNativeTime()
};
```

Метод Create() инициализирует все поля класса Bomb, объявленные как для public полей, так и для privet полей. Метод Reset() пере инициализирует поля класса с параметром доступа private.

Методы GetPositionX(), GetPositionY(), GetHeigth(), GetWidth(), GetNativeTime возвращают значения полей x, y, height, width, nativeTime соответственно.

Метод SetPosition() присваивает значения полей x, y.

Метод SetTileDestroyStone() обеспечивает связь между взрывом бомбы и картой. Код данного метода описан ниже:

```
void SetTileDestroyStone(int positionY, int positionX)
{
if ((TileMap[positionY][positionX] == 'S') || (TileMap[positionY][positionX] == '1')

|| (TileMap[positionY][positionX] == '2') || (TileMap[positionY][positionX] == '3')
|| (TileMap[positionY][positionX] == '4') || (TileMap[positionY][positionX] == '5'))
        {
                if (boomTime < 0.2)
                        TileMap[positionY][positionX] = '1';
                else
                        if (boomTime < 0.4)
                                TileMap[positionY][positionX] = '2';
                        else
                                if (boomTime < 0.6)
                                        TileMap[positionY][positionX] = '3';
                                else
                                        if (boomTime < 0.8)
                                                TileMap[positionY][positionX] = '4';
                                        else
                                                if (boomTime < 1)
                                                        TileMap[positionY][positionX] = '5';
                                                else
                                                        TileMap[positionY][positionX] = ' ';
        }
}
```

Метод CheckBombInFire() проверяет, находится ли бомба в позиции, где есть пламя, если бомба окажется в пламени, то значение nativeTime будет равно 3, что в свою очередь вызовет «детонацию» бомбы. Код данного метода описан ниже:

```
void CheckBombOnFire(float x, float y)
```

```
{
	for (int i = (y - 100) / CELL_SIZE; i < (y - 100 + heigth) / CELL_SIZE; i++)
	{
		for (int j = x / CELL_SIZE; j < (x + width) / CELL_SIZE; j++)
		{
			char SymbolInMap = TileBoomMap[i][j];
			if ((SymbolInMap == 'H') || (SymbolInMap == 'R') ||
				(SymbolInMap == 'L') || (SymbolInMap == 'V') ||
				(SymbolInMap == 'D') || (SymbolInMap == 'U') ||
				(SymbolInMap == 'S'))
			{
				nativeTime = 3;
				break;
			}
		}
		if (nativeTime == 3)
			break;
	}
}
```

Блок-схема метода CheckBombOnFire:

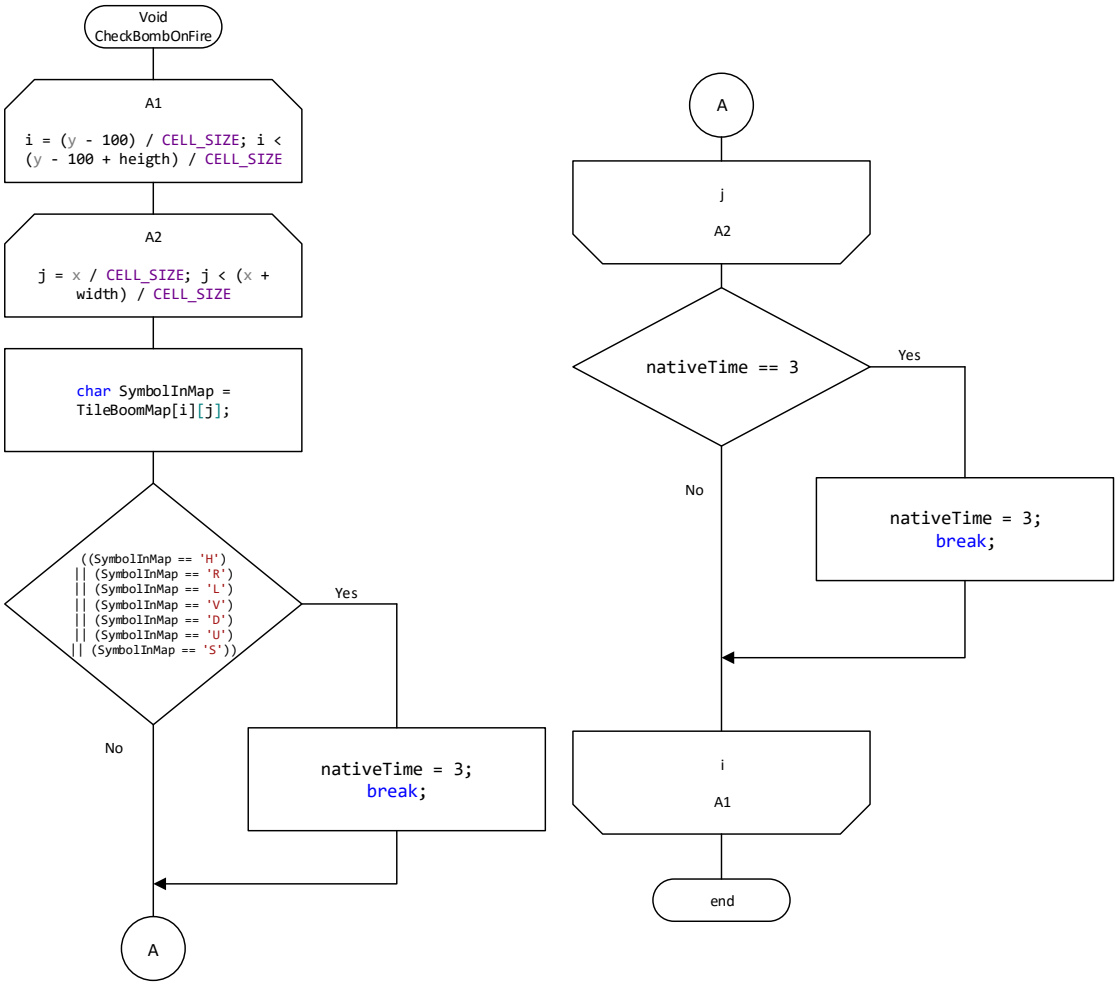

Рисунок 2.5 – Блок-схема метода CheckBombOnFire класса Bomb

Для отображения персонажа и возможности взаимодействия персонажа с бомбой и другими объектами данного проекта объявим класс Player, который будет содержать в себе класс Bomb:

```
class Player
{

	class Bomb
	{
		...;
	};

private:
	float x, y, width, heigth, dx, dy, currentFrame, dieTime;
	int bombState, speedState, fireState, colDIe, shift;
	bool life;
	sf::String name;
public:
	float speed = 0;
	Bomb bomb[5];
	bool isMove, isSelected, isSpace;
	enum { left, rigth, up, down, stay } state;
	sf::String File;
	sf::Texture texture;
	sf::Sprite sprite;
	sf::SoundBuffer bufferBoom, bufferBonus, bufferDie;
	sf::Sound soundBoom, soundBonus, soundDie;
	Player();
	~Player();
	void Create(sf::Image &image, sf::String Name,float X, float Y, float Width, float
Heigth, int Shift);
	void Reset(float X, float Y, int Shift);
	void Update(float time, float realTime);
	void AnimationDie(int Shift, float time);
	void Die(float time);
	void CheckPlayerOnFire(float x, float y);
	bool CheckIntersectionPlayerAndBombX(int number);
	bool CheckIntersectionPlayerAndBombY(int number);
	bool CheckBombInPlayer(int number);
	void CheckCollisionPlayerWithBomb(float Dx, float Dy, float time);
	void CheckCollisionPlayerWithMap(float Dx, float Dy);
	void AnimationWalk(int Shift, float time);
	void Control(float time);
	int GetBombState();
	int GetSpeedState();
	int GetFireState();
	bool GetLife();
	sf::String GetName();
};
```

Метод Create() инициализирует все поля класса Player, объявленные как для public полей, так и для privet полей. Метод Reset() пере инициализирует поля класса с параметром доступа private.

Методы AnimationDie() и AnimationWalk() реализуют циклическое передвижение по спрайту, содержащему текстуры персонажа и переприсваивают спрайтам данного класса новое значение отрисовываемого прямоугольника в зависимости от значения времени. Тем самым реализуется плавную смену текстур персонажа.

Методы GetBombState(), GetSpeedState(), GetFireState(), GetLife(), GetName() возвращают значения полей с параметром доступа private, то есть полей bombState, speedState, fireState, life и name соответственно.

Методы CheckCollisionPlayerWithBomb() и CheckCollisionPlayerWithMap() выполняют проверку на взаимодействие персонажа с бомбой и картой

соответственно, в случае если происходит коллизия с одним из этих объектов происходит переприсваивание позиции персонажа таким образом, чтобы персонаж располагался вплотную к бомбе или карте. Метод CheckCollisionPlayerWithMap(), помимо вышесказанного, выполняет проверку взаимодействия персонажа с бонусами игры, если взаимодействие произошло, то значение бонуса аннулируется, а поле соответствующего значения бонуса персонажа увеличивается на один.

Метод Control() выполняет управление передвижением персонажа. Для обоих персонажей реализована одна и та же модель обработки нажатия клавиши передвижения. Пример кода, отвечающего за передвижение:

```
if (name == "first")
{
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))
	{
		state = left;
		speed = START_SPEED + BONUS_SPEED * speedState;
		AnimationWalk(30, time);
	}
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::D))
	{
		state = rigth;
		speed = START_SPEED + BONUS_SPEED * speedState;
		AnimationWalk(90, time);
	}
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::W))
	{
		state = up;
		speed = START_SPEED + BONUS_SPEED * speedState;
		AnimationWalk(60, time);
	}
	if (sf::Keyboard::isKeyPressed(sf::Keyboard::S))
	{
		state = down;
		speed = START_SPEED + BONUS_SPEED * speedState;
		AnimationWalk(0, time);
	}
}
```

В случае для второго персонажа отличаются лишь поле name и клавиши, отвечающие за движение.

Методы CheckIntersectionPlayerAndBombX() и CheckIntersectionPlayerAndBombY() возвращают результат проверки пересечения персонажа с бомбой по оси X и оси Y. Код данных методов представлен ниже:

```
bool CheckIntersectionPlayerAndBombX(int number)
{
	if ((x + width > bomb[number].GetPositionX()) && (x < bomb[number].GetPositionX()
+ bomb[number].GetWidth()))
		return true;
	else
		return false;
}
bool CheckIntersectionPlayerAndBombY(int number)
{
	if ((y + heigth > bomb[number].GetPositionY()) && (y < bomb[number].GetPositionY()
+ bomb[number].GetHeigth()))
		return true;
	else
		return false;
```

```
}
```

Метод CheckBombInPlayer() делает проверку на совпадение позиции бомбы с персонажем таким образом, что бомба находится внутри персонажа или пересекается с ним. Блок-схема данного метода:

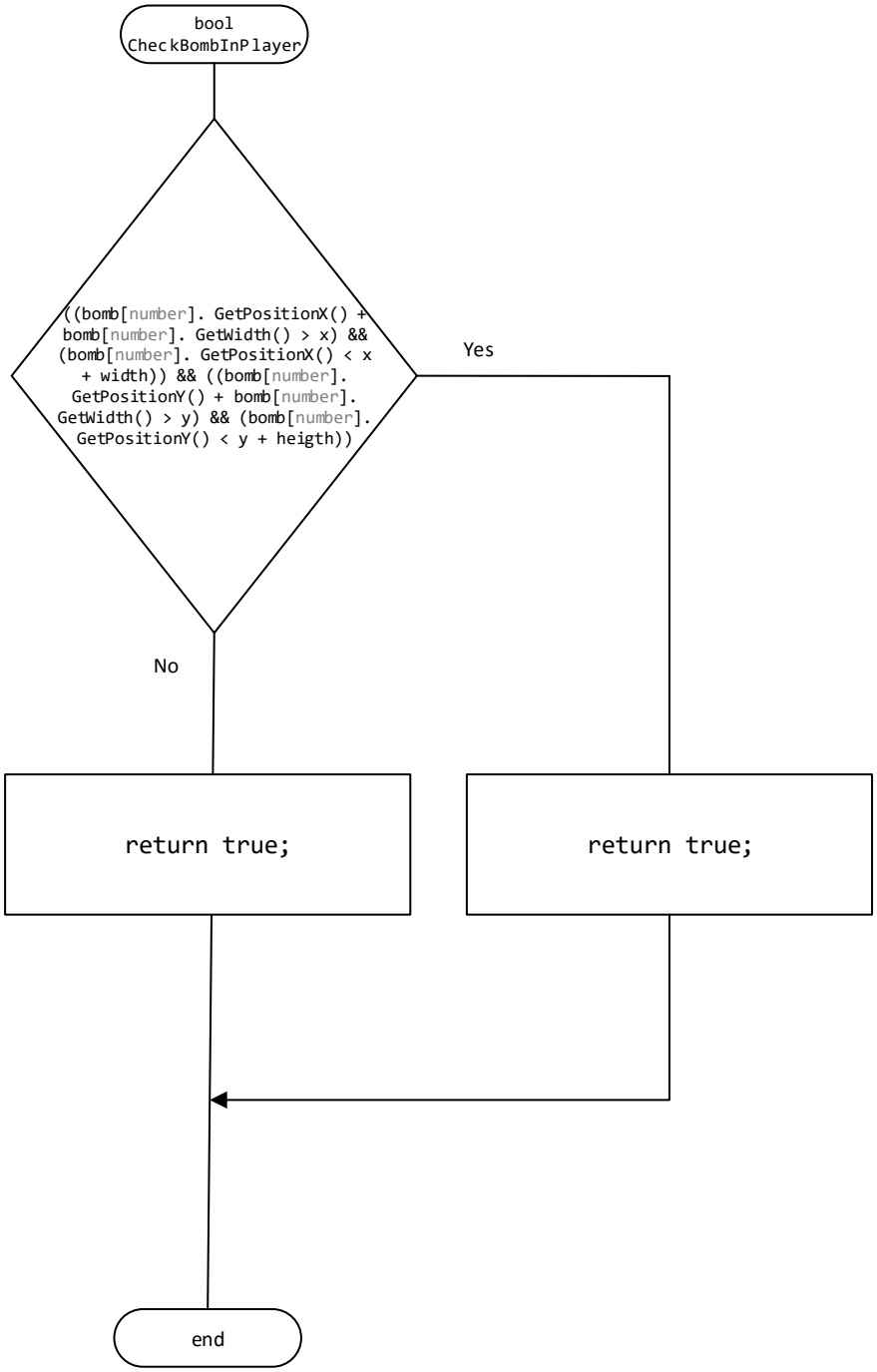

Рисунок 2.6 – Блок-схема метода CheckBombInPlayer класса Player

Для корректного взаимодействия объектов персонажей между собой создадим класс Game, содержащий в себе класс Player:

```
class Game
{
	class Player
	{
```

```
            ...;
        }
public:
        Player bomberman[2];
        sf::Image heroImage;
        Game(sf::String Name)
        void Reset()
```

Конструктор класса Game выполняет формирование объектов персонажей и их инициализацию.

Метод Reset выполняет переинициализацию объектов персонажей, тем самым устанавливает их в начальное положение. Код метода приведён ниже:

```
void Reset()
{
        bomberman[0].Reset(25, 606, 0);
        bomberman[1].Reset(509, 122, 210);
}
```

Блок-схема метода Reset() класса Game:

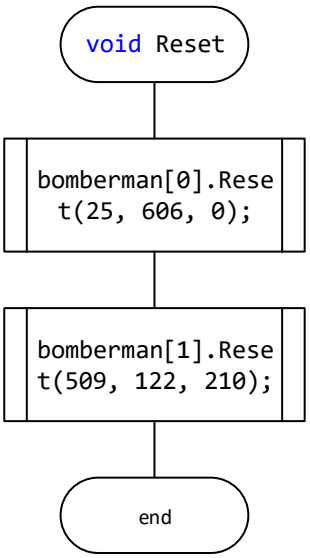

Рисунок 2.7 – Блок-схема метода Reset класса Game

2.4 Отрисовка кадра

Вся плавность смены кадров обуславливается сменой кадров относительно таймера, а не загруженности процессора. Благодаря этому принципу мы достигаем обработки всех событий и всех обновлений кадра на одинаковую величину.

После входа в цикл отрисовки окна происходит вход в цикл обработки событий. В цикле проверяются следующие событи:

- Установка бомбы для обоих персонажей (нажатие клавиш Space, num 0);
- Обработка нажатия клавиши esc;
- Обработка нажатия клавиш F1 и F9

После завершения цикла обработки событий вызывается метод Update класса Player для обоих персонажей, где обновляются значения положения персонажей и взаимодействие их с другими объектами.

После обновления значений объектов происходит отрисовка всего окна, включая информационные поля классов персонажей. Код отрисовки окна игры приведён ниже:

```
windowGame.clear();
windowGame.draw(panelSprite);
for (int i = 0; i < HEIGHT_MAP; i++)
{
	for (int j = 0; j < WIDTH_MAP; j++)
	{
		boomDraw = false;
		if (TileMap[i][j] == 'M')
			mapSprite.setTextureRect(sf::IntRect(28, 0, 16, 16));
		if (TileMap[i][j] == 'S')
			mapSprite.setTextureRect(sf::IntRect(58, 0, 16, 16));
		if (TileMap[i][j] == '1')
			mapSprite.setTextureRect(sf::IntRect(88, 0, 16, 16));
		if (TileMap[i][j] == '2')
			mapSprite.setTextureRect(sf::IntRect(118, 0, 16, 16));
		if (TileMap[i][j] == '3')
			mapSprite.setTextureRect(sf::IntRect(148, 0, 16, 16));
		if (TileMap[i][j] == '4')
			mapSprite.setTextureRect(sf::IntRect(178, 0, 16, 16));
		if (TileMap[i][j] == '5')
			mapSprite.setTextureRect(sf::IntRect(208, 0, 16, 16));
		if (TileMap[i][j] == ' ')
			mapSprite.setTextureRect(sf::IntRect(0, 0, 16, 16));
		if ((TileMap[i][j] == ' ') && (TileBonusMap[i][j] == 'B'))
			mapSprite.setTextureRect(sf::IntRect(328, 0, 16, 16));
		if ((TileMap[i][j] == ' ') && (TileBonusMap[i][j] == 'S'))
			mapSprite.setTextureRect(sf::IntRect(268, 0, 16, 16));
		if ((TileMap[i][j] == ' ') && (TileBonusMap[i][j] == 'F'))
			mapSprite.setTextureRect(sf::IntRect(298, 0, 16, 16));
		if ((TileBoomMap[i][j] == 'S') && (TileMap[i][j] == ' '))
		{
			boomSprite.setTextureRect(sf::IntRect(330, 30, 16, 16));
			boomDraw = true;
		}
		if ((TileBoomMap[i][j] == 'H') && (TileMap[i][j] == ' '))
		{
			boomSprite.setTextureRect(sf::IntRect(360, 60, 16, 16));
			boomDraw = true;
		}
		if ((TileBoomMap[i][j] == 'R') && (TileMap[i][j] == ' '))
		{
			boomSprite.setTextureRect(sf::IntRect(360, 30, 16, 16));
			boomDraw = true;
		}
		if ((TileBoomMap[i][j] == 'V') && (TileMap[i][j] == ' '))
		{
			boomSprite.setTextureRect(sf::IntRect(360, 0, 16, 16));
			boomDraw = true;
		}
		if ((TileBoomMap[i][j] == 'L') && (TileMap[i][j] == ' '))
		{
			boomSprite.setTextureRect(sf::IntRect(300, 30, 16, 16));
			boomDraw = true;
		}
```

```
                if ((TileBoomMap[i][j] == 'U') && (TileMap[i][j] == ' '))
                {
                        boomSprite.setTextureRect(sf::IntRect(330, 0, 16, 16));
                        boomDraw = true;
                }
                if ((TileBoomMap[i][j] == 'D') && (TileMap[i][j] == ' '))
                {
                        boomSprite.setTextureRect(sf::IntRect(330, 60, 16, 16));
                        boomDraw = true;
                }
                mapSprite.setPosition(j * CELL_SIZE, i * CELL_SIZE + 100);
                windowGame.draw(mapSprite);
                if (boomDraw)
                {
                        boomSprite.setPosition(j * CELL_SIZE, i * CELL_SIZE + 100);
                        windowGame.draw(boomSprite);
                }
        }
}
for (int i = 0; i < 5; i++)
{
        if (game.bomberman[0].bomb[i].visible)
                windowGame.draw(game.bomberman[0].bomb[i].sprite);
        if (game.bomberman[1].bomb[i].visible)
                windowGame.draw(game.bomberman[1].bomb[i].sprite);
}
windowGame.draw(game.bomberman[0].sprite);
windowGame.draw(game.bomberman[1].sprite);
if (showFPS)
{
        std::string tempString;
        tempString = std::to_string((int)Framerate);
        text.setString(tempString);
        text.setPosition(1, 1);
        windowGame.draw(text);
}
```

… /* отрисовка информационных полей класса персонажа*/

2.5 Окно помощи

В случае, если в цикле обработки событий будет обработано нажатие клавиши F1 будет вызываться метод открытия окна помощи, причём работа основного окна игры будет остановлено и скрыто до тех пор, пока будет открыто окно помощи.

Окно помощи работает и отрисовывается по тому же принципу, что и основное окно игры. Отличия заключаются в выводимой информации и цикле обработки событий.

В случае закрытия окна помощи, основное окно игры снова будет введено в действие и станет активным.

2.6 Окно вывода победителя

В случае, если во время очередного обновления объекта персонажа он попадёт в пламя, то значение в его поле life будет равняться false, что повлечёт за собой вызов метода открытия окна победителя, где будет выводится информация о том, какой из персонажей победил.

В случае закрытия окна вывода победителя, тогда основное окно игры будет снова введено в действие и станет активным, но для объекта game вызовется метод Reset(), которое повлечёт за собой переинициализацию всех объектов, вложенных в данный. И таким образом игра начнётся заново.

3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

3.1 Начало игры

После запуска приложения открывается основное окно игрового приложения. Вид основного окна игры представлен на рисунке 3.1.

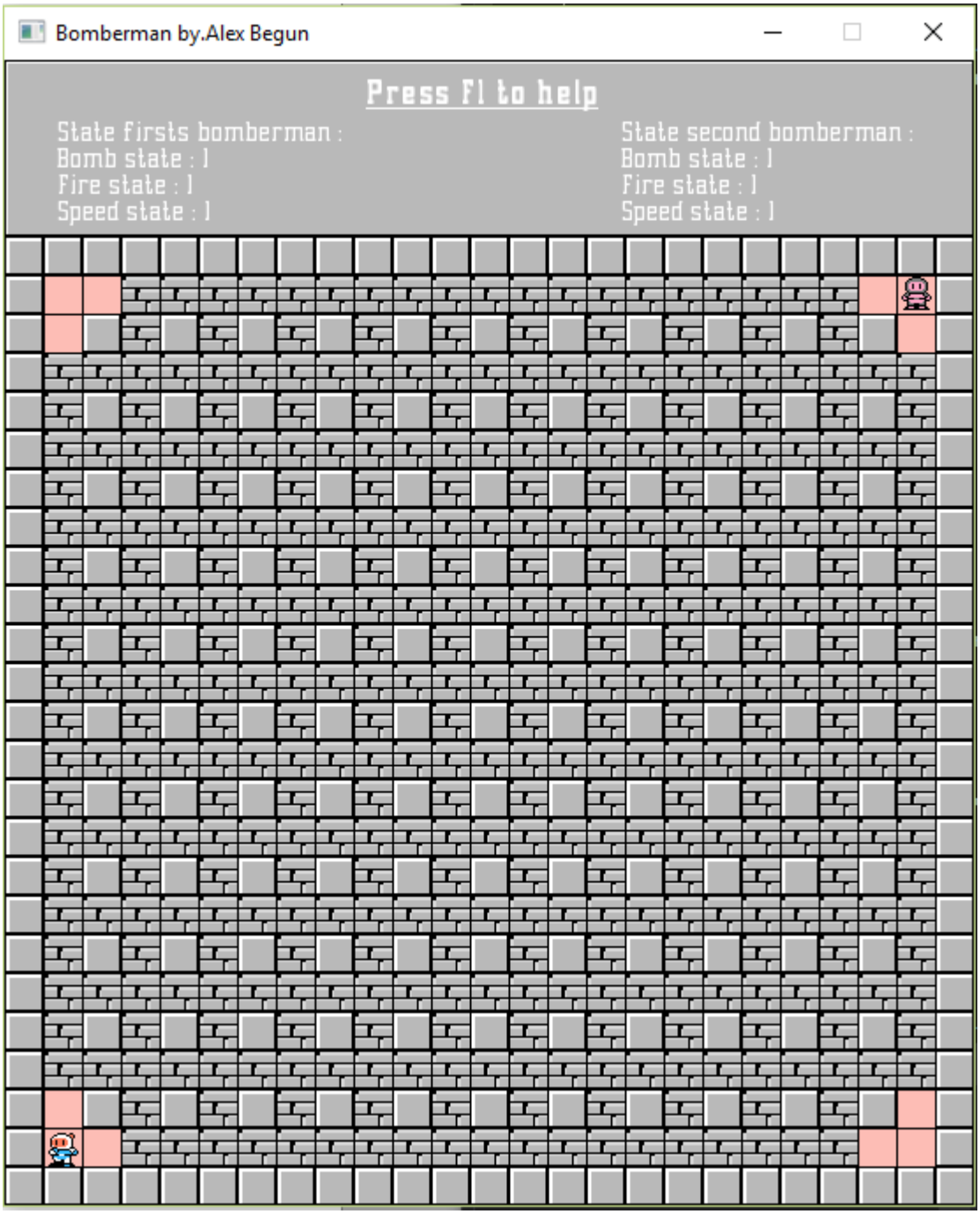

Рисунок 3.1 – Вид основного окна игрового приложения «Bomberman»

На данном рисунке показано, что игровое приложение полностью готово к использованию.

3.2 Вызов помощи

В основном окне игры указаны действия, которые нужно предпринять, чтобы получить помощь. Положение данного указания показано на рисунке 3.2.

Рисунок 3.2 – Положение указателя на помощь

После нажатия клавиши F1 будет открыто окно помощи, содержащее в себе конфигурацию всех клавишей, которые нужны для того, чтобы получилось успешно играть в данное игровое приложение. Вид окна помощи представлен на рисунке 3.3.

Рисунок 3.3 – Вид окна помощи игрового приложения «Bomberman»

3.3 Результат игры

В случае, если один из персонажей попадёт под пламя от бомбы, то он, фактически, «умрёт». После этого последует открытие окна вывода победителя, которое будет уведомлять какой из персонажей остался «жив». Вид окна вывода победителя представлен на рисунке 3.4.

Рисунок 3.4 – Вид окна вывода победителя игрового приложения «Bomberman»

ЗАКЛЮЧЕНИЕ

Было разработано игровое приложение – «Bomberman», с помощью которого, пользователь может удовлетворить свою потребность отвлечься от внешнего мира в компании друга, товарища или любого человека.

Приложение было реализовано с использованием среды для разработки Visual Studio 2015 при помощи языка программирования C++. При разработке активно использовалось программное средство контроля версий – GitHub.

Была освоена сторонняя библиотека SFML, позволяющая использовать графический интерфейс в понятной и доступной форме.

Для повышения качества разрабатываемого продукта использовался ручной метод тестирования.

Игровое приложение обладает минималистичным интерфейсом, что упрощает его использование для рядового пользователя. Со временем данное приложение будет совершенствоваться: будет добавлена функция выбора количества игроков и выбора их по принципу человек/ИИ. Добавление бонусов на временное бесмертие, способность передвигать бомбы. Возможно включение некоторых функций из списка приведённых аналогов.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Голощапов А.Л. «Google Android программирование для мобильных устройств». BHV Санкт-Петербург, 2011
2. Б. Эккель, «Философия Java»;
3. Роберт Мартин «Чистый Код». Питер, 2014;
4. ГОСТ 19.701–90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – Введ. 1992–01–01. – М. : Изд-во стандартов, 1991.
5. http://cyberforum.ru;
6. http://stackoverflow.com/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Main.cpp

```
#ifndef SFML_GRAPHICS
#	define SFML_GRAPHICS
#include <SFML/Graphics.hpp>
#endif // !SFML_GRAPHICS
#include <SFML\Audio.hpp>
#include "Map.h"
#include "Help.h"
#include "Win.h"
#include <iostream>
#include "Constants.h"
#include <windows.h>

class Game
{
	class Player
	{

		class Bomb
		{

		private:
			float x, y, width, heigth;
			float nativeTime, boomTime;

		public:
			int frame;
			int fire;
			sf::String File;
			sf::Image image;
			sf::Texture texture;
			sf::Sprite sprite;
			sf::Sound soundBoom;
			bool visible, inPlayer, isBoom, enabled;
			int fireState;

			Bomb()
			{
			}

			void Create(sf::String F, sf::Sound *SoundBoom, float X, float Y,
float Width, float Heigth)
			{
				x = X; y = Y;
				File = F;
				width = Width; heigth = Heigth;
				frame = 0; nativeTime = 0; boomTime = 0; fireState = 2;
				visible = false; inPlayer = false; isBoom = false; enabled =
false;

				image.loadFromFile("images/" + File);
				texture.loadFromImage(image);
				sprite.setTexture(texture);
				sprite.setTextureRect(sf::IntRect(0, 30, Width, Heigth));
				sprite.setOrigin(width / 2, heigth / 2);
				sprite.setPosition(X + 3 + width / 2, Y + 3 + heigth / 2);
				soundBoom = *SoundBoom;
			}

			void Reset()
```

```
                {
                        frame = 0; nativeTime = 0; boomTime = 0; fireState = 2;
                        visible = false; inPlayer = false; isBoom = false; enabled =
false;
                }

                void SetPosition(float X, float Y)
                {
                        x = X; y = Y;
                }

                void SetTileDestroyStone(int positionY, int positionX)
                {
                        if ((TileMap[positionY][positionX] == 'S') ||
(TileMap[positionY][positionX] == '1')
                                || (TileMap[positionY][positionX] == '2') ||
(TileMap[positionY][positionX] == '3')
                                || (TileMap[positionY][positionX] == '4') ||
(TileMap[positionY][positionX] == '5'))
                                {
                                        if (boomTime < 0.2)
                                                TileMap[positionY][positionX] = '1';
                                        else
                                                if (boomTime < 0.4)
                                                        TileMap[positionY][positionX] = '2';
                                                else
                                                        if (boomTime < 0.6)
                                                                TileMap[positionY][positionX] =
'3';
                                                        else
                                                                if (boomTime < 0.8)

        TileMap[positionY][positionX] = '4';
                                                                else
                                                                        if (boomTime < 1)

        TileMap[positionY][positionX] = '5';
                                                                        else

        TileMap[positionY][positionX] = ' ';
                                }
                }

                void CheckBombOnFire(float x, float y)
                {
                        for (int i = (y - 100) / CELL_SIZE; i < (y - 100 + heigth) /
CELL_SIZE; i++)
                        {
                                for (int j = x / CELL_SIZE; j < (x + width) /
CELL_SIZE; j++)
                                {
                                        char SymbolInMap = TileBoomMap[i][j];
                                        if ((SymbolInMap == 'H') || (SymbolInMap == 'R')
||
                                                (SymbolInMap == 'L') || (SymbolInMap ==
'V') ||
                                                (SymbolInMap == 'D') || (SymbolInMap ==
'U') ||
                                                (SymbolInMap == 'S'))
                                        {
                                                nativeTime = 3;
                                                break;
                                        }
                                }
```

```
                                if (nativeTime == 3)
                                        break;

                        }
                }

                void Update(float time)
                {
                        if (visible == true)
                        {
                                nativeTime += time;
                                CheckBombOnFire(x, y);
                                if (nativeTime < 3)
                                        sprite.setPosition(x + 3 + width / 2, y + 3 +
heigth / 2);
                                else
                                {
                                        soundBoom.play();
                                        visible = false;
                                        isBoom = true;
                                        nativeTime = 0;
                                }
                        }
                        if (isBoom)
                        {
                                int positionX = (int)(x / CELL_SIZE), positionY =
(int)((y - 100) / CELL_SIZE);
                                boomTime += time;
                                if (boomTime < 1)
                                {
                                        SetBoomOnMap(positionX, positionY, 'S');
                                        for (int i = 0; i < fireState; i++)
                                        {
                                                if (TileMap[positionY][positionX + i + 1]
== ' ')
                                                {
                                                        if (i != fireState - 1)
                                                        {
                                                                if (positionX + i + 1 <
HEIGHT_MAP)
        SetBoomOnMap(positionX + i + 1, positionY, 'H');
                                                        }
                                                        else
                                                                if (positionX + i + 1 <
HEIGHT_MAP)
        SetBoomOnMap(positionX + i + 1, positionY, 'R');
                                                }
                                                else
                                                {
                                                        SetTileDestroyStone(positionY,
positionX + i + 1);
                                                        break;
                                                }
                                        }
                                        for (int i = 0; i < fireState; i++)
                                        {
                                                if (TileMap[positionY][positionX - i - 1]
== ' ')
                                                {
                                                        if (i != fireState - 1)
                                                        {
                                                                if (positionX - i - 1 > 0)
```

```
                SetBoomOnMap(positionX - i - 1, positionY, 'H');
                                        }
                                        else
                                                if (positionX - i - 1 > 0)
                                                {

                SetBoomOnMap(positionX - i - 1, positionY, 'L');
                                                }
                                        }
                                        else
                                        {
                                                SetTileDestroyStone(positionY,
positionX - i - 1);
                                                break;
                                        }
                                }
                                for (int i = 0; i < fireState; i++)
                                {
                                        if (TileMap[positionY + i + 1][positionX]
== ' ')
                                        {
                                                if (i != fireState - 1)
                                                {
                                                        if (positionY + i + 1 <
HEIGHT_MAP)

                SetBoomOnMap(positionX, positionY + i + 1, 'V');
                                                }
                                                else
                                                        if (positionY + i + 1 <
HEIGHT_MAP)
                                                        {

                SetBoomOnMap(positionX, positionY + i + 1, 'D');
                                                        }
                                                }
                                                else
                                                {
                                                        SetTileDestroyStone(positionY + i +
1, positionX);
                                                        break;
                                                }
                                        }
                                        for (int i = 0; i < fireState; i++)
                                        {
                                                if (TileMap[positionY - i - 1][positionX]
== ' ')
                                                {
                                                        if (i != fireState - 1)
                                                        {
                                                                if (positionY - i - 1 > 0)

                SetBoomOnMap(positionX, positionY - i - 1, 'V');
                                                        }
                                                        else
                                                                if (positionY - i - 1 <
HEIGHT_MAP)
                                                                {

                SetBoomOnMap(positionX, positionY - i - 1, 'U');
                                                                }
                                                }
                                                else
```

```
                                        {
                                                SetTileDestroyStone(positionY - i -
1, positionX);
                                                break;
                                        }
                                }
                        }
                        if (boomTime > 1)
                        {
                                isBoom = false;
                                enabled = false;
                                SetBoomOnMap(positionX, positionY, ' ');
                                for (int i = 0; i < fireState; i++)
                                {
                                        if (positionX + i + 1 < HEIGHT_MAP)
                                                SetBoomOnMap(positionX + i + 1,
positionY, ' ');
                                        if (positionX - i - 1 > 0)
                                                SetBoomOnMap(positionX - i - 1,
positionY, ' ');
                                        if (positionY + i + 1 < HEIGHT_MAP)
                                                SetBoomOnMap(positionX, positionY +
i + 1, ' ');
                                        if (positionY - i - 1 > 0)
                                                SetBoomOnMap(positionX, positionY -
i - 1, ' ');
                                        if (positionX + i + 1 > 0 && positionX + i
+ 1 < HEIGHT_MAP)
                                                if (TileMap[positionY][positionX +
i + 1] == '5')
                                                        TileMap[positionY][positionX
+ i + 1] = ' ';
                                        if (positionX - i - 1 > 0 && positionX - i
- 1 < HEIGHT_MAP)
                                                if (TileMap[positionY][positionX -
i - 1] == '5')
                                                        TileMap[positionY][positionX
- i - 1] = ' ';
                                        if (positionY + i + 1 > 0 && positionY + i
+ 1 < HEIGHT_MAP)
                                                if (TileMap[positionY + i +
1][positionX] == '5')
                                                        TileMap[positionY + i +
1][positionX] = ' ';
                                        if (positionY - i - 1 > 0 && positionY - i
- 1 < HEIGHT_MAP)
                                                if (TileMap[positionY - i -
1][positionX] == '5')
                                                        TileMap[positionY - i -
1][positionX] = ' ';
                                        }
                                        boomTime = 0;
                                }
                        }
                }

                float GetPositionX()
                {
                        return x;
                }

                float GetPositionY()
                {
                        return y;
```

```
		}

		float GetHeigth()
		{
			return heigth;
		}

		float GetWidth()
		{
			return width;
		}

		float GetNativeTime()
		{
			return nativeTime;
		}
	};

private:
	float x, y, width, heigth, dx, dy, currentFrame, dieTime;
	int bombState, speedState, fireState, colDIe, shift;
	bool life;
	sf::String name;

public:
	float speed = 0;
	Bomb bomb[5];
	bool isMove, isSelected, isSpace;
	enum { left, rigth, up, down, stay } state;
	sf::String File;
	sf::Texture texture;
	sf::Sprite sprite;
	sf::SoundBuffer bufferBoom, bufferBonus, bufferDie;
	sf::Sound soundBoom, soundBonus, soundDie;
	Player()
	{

	}

	void Create(sf::Image &image, sf::String Name,float X, float Y, float
Width, float Heigth, int Shift)
	{
		bufferBoom.loadFromFile("data/Boom.mid");
		soundBoom.setBuffer(bufferBoom);
		bufferBonus.loadFromFile("data/Bonus.mid");
		soundBonus.setBuffer(bufferBonus);
		bufferDie.loadFromFile("data/Die.mid");
		soundDie.setBuffer(bufferDie);
		bombState = 1; speedState = 0; fireState = 2; dieTime = 0;
		life = true;
		name = Name;
		width = Width; heigth = Heigth;
		x = X; y = Y; currentFrame = 0, speed = 0; shift = Shift;
		isMove = false; isSelected = false, isSpace = false;
		state = stay;
		for (int i = 0; i < 5; i++)
		{
			bomb[i].Create("bomberman_bomb_sheet.png", &soundBoom, 0, 0,
16, 16);
		}
		texture.loadFromImage(image);
		sprite.setTexture(texture);
		sprite.setTextureRect(sf::IntRect(0 + shift, 0, width, heigth));
		sprite.setOrigin(width / 2, heigth / 2);
```

```
		sprite.setPosition(X + width / 2, Y + heigth / 2);
	}

	void Reset(float X, float Y, int Shift)
	{
		bombState = 1; speedState = 0; fireState = 2; dieTime = 0;
		life = true;
		x = X; y = Y; currentFrame = 0, speed = 0; shift = Shift;
		isMove = false; isSelected = false, isSpace = false;
		state = stay;
		for (int i = 0; i < 5; i++)
		{
			bomb[i].Reset();
		}
		sprite.setTextureRect(sf::IntRect(0 + shift, 0, width, heigth));
		sprite.setOrigin(width / 2, heigth / 2);
		sprite.setPosition(X + width / 2, Y + heigth / 2);
	}

	void Update(float time, float realTime)
	{
		dx = 0; dy = 0;
		Control(time);
		switch (state)
		{
		case left:
			dx = -speed; dy = 0; break;
		case rigth:
			dx = speed; dy = 0; break;
		case up:
			dx = 0; dy = -speed; break;
		case down:
			dx = 0; dy = speed; break;
		case stay: break;
		}
		x += dx * time;
		CheckCollisionPlayerWithMap(dx, 0);
		CheckCollisionPlayerWithBomb(dx, 0, time);
		//CheckCollisionBombWithBomb(dx, 0, time);                    //
закомментировано, из-за осознания того, что бомбы не двигаются ((((
		y += dy * time;
		CheckCollisionPlayerWithMap(0, dy);
		CheckCollisionPlayerWithBomb(0, dy, time);
		//CheckCollisionBombWithBomb(0, dy, time);                    //
закомментировано, из-за осознания того, что бомбы не двигаются ((((
		speed = 0;
		for (int i = 0; i < 5; i++)
		{
			bomb[i].Update(realTime);
		}
		sprite.setPosition(x + width / 2, y + heigth / 2);
		CheckPlayerOnFire(x, y);
	}

	void AnimationDie(int Shift, float time)
	{
		currentFrame += 0.005*time;
		if (currentFrame > 3)
			currentFrame -= 3;
		sprite.setTextureRect(sf::IntRect(int(currentFrame) * 30, Shift,
PICTURE_BOMBERMAN_WIDTH, PICTURE_BOMBERMAN_HEIGTH));
	}

	void Die(float time)
```

```
		{
			dieTime += time / 3;
			if (dieTime > 3)
				dieTime -= 3;
			AnimationDie((int)(dieTime)* 30, time);
		}

		void CheckPlayerOnFire(float x, float y)
		{
			for (int i = (y - 100) / CELL_SIZE; i < (y - 100 + heigth) /
CELL_SIZE; i++)
			{
				for (int j = x / CELL_SIZE; j < (x + width) / CELL_SIZE; j++)
				{
					char SymbolInMap = TileBoomMap[i][j];
					if ((SymbolInMap == 'H') || (SymbolInMap == 'R') ||
(SymbolInMap == 'L') || (SymbolInMap == 'V') || (SymbolInMap == 'D') || (SymbolInMap ==
'U') || (SymbolInMap == 'S'))
					{
						soundDie.play();
						life = false;
						currentFrame = 0;
						break;
					}
				}
				if (!life)
					break;

			}
		}

		bool CheckIntersectionPlayerAndBombX(int number)
		{
			if ((x + width > bomb[number].GetPositionX()) && (x <
bomb[number].GetPositionX() + bomb[number].GetWidth()))
				return true;
			else
				return false;
		}

		bool CheckIntersectionPlayerAndBombY(int number)
		{
			if ((y + heigth > bomb[number].GetPositionY()) && (y <
bomb[number].GetPositionY() + bomb[number].GetHeigth()))
				return true;
			else
				return false;
		}

		bool CheckBombInPlayer(int number)
		{
			if (((bomb[number].GetPositionX() + bomb[number].GetWidth() > x) &&
(bomb[number].GetPositionX() < x + width)) && ((bomb[number].GetPositionY() +
bomb[number].GetWidth() > y) && (bomb[number].GetPositionY() < y + heigth)))
				return true;
			else
				return false;
		}

		void CheckCollisionPlayerWithBomb(float Dx, float Dy, float time)
		{
			for (int i = 0; i < 5; i++)
			{
				if (bomb[i].inPlayer)
```

```
                        {
                                bomb[i].inPlayer = CheckBombInPlayer(i);
                        }
                        else
                        {
                                if (CheckIntersectionPlayerAndBombY(i) &&
CheckIntersectionPlayerAndBombX(i) && bomb[i].visible)
                                {
                                        if (Dy > 0)
                                        {

      //bomb[i].SetPosition(bomb[i].GetPositionX(), bomb[i].GetPositionY() + Dy * time);
                                                //CheckCollisionBombWithMap(Dx, Dy, i);
                                                y = bomb[i].GetPositionY() - heigth;
                                                break;
                                        }
                                        if (Dy < 0)
                                        {

      //bomb[i].SetPosition(bomb[i].GetPositionX(), bomb[i].GetPositionY() + Dy * time);
                                                //CheckCollisionBombWithMap(Dx, Dy, i);
                                                y = bomb[i].GetPositionY() +
bomb[i].GetHeigth();
                                                break;
                                        }

                                        // закомментировано, из-за осознания того, что бомбы не
двигаются ((((
                                        if (Dx > 0)
                                        {

      //bomb[i].SetPosition(bomb[i].GetPositionX() + Dx * time, bomb[i].GetPositionY());
                                                //CheckCollisionBombWithMap(Dx, Dy, i);
                                                x = bomb[i].GetPositionX() - width;
                                                break;
                                        }
                                        if (Dx < 0)
                                        {

      //bomb[i].SetPosition(bomb[i].GetPositionX() + Dx * time, bomb[i].GetPositionY());
                                                //CheckCollisionBombWithMap(Dx, Dy, i);
                                                x = bomb[i].GetPositionX() +
bomb[i].GetWidth();
                                                break;
                                        }
                                }
                        }
                }
        }



        void CheckCollisionPlayerWithMap(float Dx, float Dy)
        {
                y -= 100;
                for (int i = y / CELL_SIZE; i < (y + heigth) / CELL_SIZE; i++)
                        for (int j = x / CELL_SIZE; j < (x + width) / CELL_SIZE; j++)
                        {
                                if ((TileMap[i][j] == 'M') || (TileMap[i][j] == 'S') ||
(TileMap[i][j] == '1') || (TileMap[i][j] == '2') || (TileMap[i][j] == '3') ||
(TileMap[i][j] == '4') || (TileMap[i][j] == '5'))
                                {
                                        if (Dy > 0)
```

```
                                        if ((y < i * CELL_SIZE) && ((x < (j)*
CELL_SIZE + CELL_SIZE - 3) && (x + width > j * CELL_SIZE + 3)))
                                                y = i * CELL_SIZE - heigth;
                                if (Dy < 0)
                                        if ((y < i * CELL_SIZE + CELL_SIZE - 7) &&
((x < (j)* CELL_SIZE + CELL_SIZE - 3) && (x + width > j * CELL_SIZE + 3)))
                                                y = i * CELL_SIZE + CELL_SIZE - 7;
                                if (Dx > 0)
                                        if ((y < i *CELL_SIZE + CELL_SIZE - 7) &&
(x < j * CELL_SIZE + CELL_SIZE - 3) && (x > j * CELL_SIZE - width + 3))
                                                x = j * CELL_SIZE - width + 3;
                                if (Dx < 0)
                                        if ((y < i * CELL_SIZE + CELL_SIZE - 7) &&
(x < j * CELL_SIZE + CELL_SIZE - 3) && (x > j * CELL_SIZE - width + 3))
                                                x = j * CELL_SIZE + CELL_SIZE - 3;
                                }
                                if ((TileBonusMap[i][j] == 'S') && (TileMap[i][j] !=
'S'))
                                {
                                        soundBonus.play();
                                        if (speedState < 5)
                                                speedState++;
                                        TileBonusMap[i][j] = ' ';
                                }
                                if ((TileBonusMap[i][j] == 'F') && (TileMap[i][j] !=
'S'))
                                {
                                        soundBonus.play();
                                        if (fireState < 5)
                                                fireState++;
                                        TileBonusMap[i][j] = ' ';
                                }
                                if ((TileBonusMap[i][j] == 'B') && (TileMap[i][j] !=
'S'))
                                {
                                        soundBonus.play();
                                        if (bombState < 5)
                                                bombState++;
                                        TileBonusMap[i][j] = ' ';
                                }
                        }

                y += 100;
        }

        void AnimationWalk(int Shift, float time)
        {
                currentFrame += 0.005*time;
                if (currentFrame > 3)
                        currentFrame -= 3;
                sprite.setTextureRect(sf::IntRect(Shift + shift, int(currentFrame) *
30, PICTURE_BOMBERMAN_WIDTH, PICTURE_BOMBERMAN_HEIGTH));
        }

        void Control(float time)
        {
                if (name == "first")
                {
                        if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))
                        {
                                state = left;
                                speed = START_SPEED + BONUS_SPEED * speedState;
                                AnimationWalk(30, time);
                        }
```

```
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::D))
                {
                        state = rigth;
                        speed = START_SPEED + BONUS_SPEED * speedState;
                        AnimationWalk(90, time);
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::W))
                {
                        state = up;
                        speed = START_SPEED + BONUS_SPEED * speedState;
                        AnimationWalk(60, time);
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::S))
                {
                        state = down;
                        speed = START_SPEED + BONUS_SPEED * speedState;
                        AnimationWalk(0, time);
                }
        }
        if (name == "second")
        {
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
                {
                        state = left;
                        speed = START_SPEED + BONUS_SPEED * speedState;
                        AnimationWalk(30, time);
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
                {
                        state = rigth;
                        speed = START_SPEED + BONUS_SPEED * speedState;
                        AnimationWalk(90, time);
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
                {
                        state = up;
                        speed = START_SPEED + BONUS_SPEED * speedState;
                        AnimationWalk(60, time);
                }
                if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
                {
                        state = down;
                        speed = START_SPEED + BONUS_SPEED * speedState;
                        AnimationWalk(0, time);
                }
        }
}

int GetBombState()
{
        return bombState;
}

int GetSpeedState()
{
        return speedState;
}

int GetFireState()
{
        return fireState;
}

bool GetLife()
```

```
		{
			return life;
		}

		sf::String GetName()
		{
			return name;
		}
	};

	public:
		Player bomberman[2];
		sf::Image heroImage;
	Game(sf::String Name)
	{
		heroImage.loadFromFile(Name);
		bomberman[0].Create(heroImage, "first",25, 606, PICTURE_BOMBERMAN_WIDTH,
PICTURE_BOMBERMAN_HEIGTH, 0);
		bomberman[1].Create(heroImage, "second", 509, 122, PICTURE_BOMBERMAN_WIDTH,
PICTURE_BOMBERMAN_HEIGTH, 210);
	}

	void Reset()
	{
		bomberman[0].Reset(25, 606, 0);
		bomberman[1].Reset(509, 122, 210);
	}
};

int main()
{
	//HWND hWnd = GetConsoleWindow();
	//ShowWindow(hWnd, SW_HIDE);
	FreeConsole();
	sf::RenderWindow windowGame(sf::VideoMode(550, 650), "Bomberman by.Alex Begun",
sf::Style::Close | sf::Style::Titlebar);

	sf::Music fonMusic;
	fonMusic.openFromFile("data/Bomber.ogg");
	fonMusic.setLoop(true);
	fonMusic.play();

	sf::Clock clock;
	sf::Time Time;
	float currentFrame = 0;
	bool isSpace = false, isNum = false, inPlayer = false, boomDraw = false, showFPS =
false;
	sf::Font font;

	//font.loadFromFile("data/Inky.ttf");
	font.loadFromFile("data/Brassie.ttf");
	//font.loadFromFile("data/danger.ttf");
	sf::Text text("", font, 20);
	text.setColor(sf::Color::White);
	text.setStyle(sf::Text::Bold | sf::Text::Underlined);

	GenerateBonusMap();

	Game game("images/bomberman2_various_sheet.png");

	sf::Image mapImage, panelImage, boomImage;
	mapImage.loadFromFile("images/bomberman_tiles_sheet_add.png");
	panelImage.loadFromFile("images/panel_state_bomberman.png");
	boomImage.loadFromFile("images/bomberman_bomb_sheet.png");
```

```
	sf::Texture map, panel, boom;
	map.loadFromImage(mapImage);
	panel.loadFromImage(panelImage);
	boom.loadFromImage(boomImage);

	sf::Sprite mapSprite, panelSprite, boomSprite;
	mapSprite.setTexture(map);
	panelSprite.setTexture(panel);
	boomSprite.setTexture(boom);
	panelSprite.setPosition(0, 0);
	mapSprite.setScale(CELL_SIZE / 16, CELL_SIZE / 16); // size picture is 20x20
	boomSprite.setScale(CELL_SIZE / 16, CELL_SIZE / 16); // size picture is 20x20

	float fullTime = 0;
	while (windowGame.isOpen())
	{
		Time = clock.getElapsedTime();
		float time = Time.asMicroseconds();
		float realTime = Time.asSeconds();
		fullTime += realTime;
		float Framerate = 1.f / clock.getElapsedTime().asSeconds();
		//	std::cout << fullTime << std::endl;
		clock.restart();
		time = time / 800;
		sf::Event event;
		while (windowGame.pollEvent(event))
		{
			if ((event.type == sf::Event::Closed) || (event.type ==
event.KeyPressed && event.key.code == sf::Keyboard::Escape))
				windowGame.close();

			if (event.type == event.KeyPressed && event.key.code ==
sf::Keyboard::Space && !isSpace)
			{
				for (int i = 0; i < game.bomberman[0].GetBombState(); i++)
				{
					for (int j = 0; j < 5; j++)
						if ((game.bomberman[0].bomb[j].inPlayer) && (i
!= j))
						{
							inPlayer = true;
							break;
						}
					if (!(inPlayer))
						if (!(game.bomberman[0].bomb[i].enabled) &&
!(isSpace))
						{
							game.bomberman[0].bomb[i].visible = true;
							game.bomberman[0].bomb[i].enabled = true;
							game.bomberman[0].bomb[i].inPlayer = true;
							game.bomberman[0].bomb[i].fireState =
game.bomberman[0].GetFireState();
							float xFloat =
((game.bomberman[0].sprite.getPosition().x - PICTURE_BOMBERMAN_WIDTH / 2) / CELL_SIZE);
							float yFloat =
((game.bomberman[0].sprite.getPosition().y - 100 - PICTURE_BOMBERMAN_HEIGTH / 2) /
CELL_SIZE);

	game.bomberman[0].bomb[i].SetPosition((xFloat - (int)xFloat < 0.5) ? (int)xFloat *
CELL_SIZE : ((int)xFloat + 1) * CELL_SIZE, (yFloat - (int)yFloat < 0.5) ? (int)yFloat *
CELL_SIZE + 100 : ((int)yFloat + 1) * CELL_SIZE + 100);
							isSpace = true;
							break;
```

```
                                        }
                                }
                                inPlayer = false;
                        }
                        if (event.type == event.KeyReleased && event.key.code ==
sf::Keyboard::Space)
                                isSpace = false;

                        if (event.type == event.KeyPressed && event.key.code ==
sf::Keyboard::Numpad0 && !isNum)
                        {
                                for (int i = 0; i < game.bomberman[1].GetBombState(); i++)
                                {
                                        for (int j = 0; j < 5; j++)
                                                if ((game.bomberman[1].bomb[j].inPlayer) && (i
!= j))
                                                {
                                                        inPlayer = true;
                                                        break;
                                                }
                                        if (!(inPlayer))
                                                if (!(game.bomberman[1].bomb[i].enabled) &&
!(isNum))
                                                {
                                                        game.bomberman[1].bomb[i].visible = true;
                                                        game.bomberman[1].bomb[i].enabled = true;
                                                        game.bomberman[1].bomb[i].inPlayer = true;
                                                        game.bomberman[1].bomb[i].fireState =
game.bomberman[1].GetFireState();
                                                        float xFloat =
((game.bomberman[1].sprite.getPosition().x - PICTURE_BOMBERMAN_WIDTH / 2) / CELL_SIZE);
                                                        float yFloat =
((game.bomberman[1].sprite.getPosition().y - 100 - PICTURE_BOMBERMAN_HEIGTH / 2) /
CELL_SIZE);

                game.bomberman[1].bomb[i].SetPosition((xFloat - (int)xFloat < 0.5) ? (int)xFloat *
CELL_SIZE : ((int)xFloat + 1) * CELL_SIZE, (yFloat - (int)yFloat < 0.5) ? (int)yFloat *
CELL_SIZE + 100 : ((int)yFloat + 1) * CELL_SIZE + 100);
                                                        isNum = true;
                                                        break;
                                                }
                                }
                                inPlayer = false;
                        }
                        if (event.type == event.KeyReleased && event.key.code ==
sf::Keyboard::Numpad0)
                                isNum = false;
                        if (event.type == event.KeyPressed && event.key.code ==
sf::Keyboard::F9)
                                if (showFPS)
                                        showFPS = false;
                                else
                                        showFPS = true;
                        if (event.type == event.KeyPressed && event.key.code ==
sf::Keyboard::F1)
                        {
                                windowGame.setVisible(false);
                                ShowHelp();
                                windowGame.setVisible(true);
                                clock.restart();
                        }

                }
                if ((game.bomberman[0].GetLife()) && (game.bomberman[1].GetLife()))
```

```
		{
			game.bomberman[0].Update(time, realTime);
			game.bomberman[1].Update(time, realTime);
		}
		else
		{
			if (!(game.bomberman[0].GetLife()))
				game.bomberman[0].Die(time);
			else
				game.bomberman[1].Die(time);
			fonMusic.stop();
			windowGame.setVisible(false);
			ShowWinner((game.bomberman[0].GetLife() ? game.bomberman[1].GetName()
: game.bomberman[0].GetName()));
			game.Reset();
			ResetMaps();
			GenerateBonusMap();
			clock.restart();
			fonMusic.play();
			windowGame.setVisible(true);
		}
		windowGame.clear();
		windowGame.draw(panelSprite);
		for (int i = 0; i < HEIGHT_MAP; i++)
		{
			for (int j = 0; j < WIDTH_MAP; j++)
			{
				boomDraw = false;
				if (TileMap[i][j] == 'M')
					mapSprite.setTextureRect(sf::IntRect(28, 0, 16, 16));
				if (TileMap[i][j] == 'S')
					mapSprite.setTextureRect(sf::IntRect(58, 0, 16, 16));
				if (TileMap[i][j] == '1')
					mapSprite.setTextureRect(sf::IntRect(88, 0, 16, 16));
				if (TileMap[i][j] == '2')
					mapSprite.setTextureRect(sf::IntRect(118, 0, 16, 16));
				if (TileMap[i][j] == '3')
					mapSprite.setTextureRect(sf::IntRect(148, 0, 16, 16));
				if (TileMap[i][j] == '4')
					mapSprite.setTextureRect(sf::IntRect(178, 0, 16, 16));
				if (TileMap[i][j] == '5')
					mapSprite.setTextureRect(sf::IntRect(208, 0, 16, 16));
				if (TileMap[i][j] == ' ')
					mapSprite.setTextureRect(sf::IntRect(0, 0, 16, 16));
				if ((TileMap[i][j] == ' ') && (TileBonusMap[i][j] == 'B'))
					mapSprite.setTextureRect(sf::IntRect(328, 0, 16, 16));
				if ((TileMap[i][j] == ' ') && (TileBonusMap[i][j] == 'S'))
					mapSprite.setTextureRect(sf::IntRect(268, 0, 16, 16));
				if ((TileMap[i][j] == ' ') && (TileBonusMap[i][j] == 'F'))
					mapSprite.setTextureRect(sf::IntRect(298, 0, 16, 16));
				if ((TileBoomMap[i][j] == 'S') && (TileMap[i][j] == ' '))
				{
					boomSprite.setTextureRect(sf::IntRect(330, 30, 16,
16));
					boomDraw = true;
				}
				if ((TileBoomMap[i][j] == 'H') && (TileMap[i][j] == ' '))
				{
					boomSprite.setTextureRect(sf::IntRect(360, 60, 16,
16));
					boomDraw = true;
				}
				if ((TileBoomMap[i][j] == 'R') && (TileMap[i][j] == ' '))
				{
```

```
                                        boomSprite.setTextureRect(sf::IntRect(360, 30, 16,
16));
                                        boomDraw = true;
                                }
                                if ((TileBoomMap[i][j] == 'V') && (TileMap[i][j] == ' '))
                                {
                                        boomSprite.setTextureRect(sf::IntRect(360, 0, 16, 16));
                                        boomDraw = true;
                                }
                                if ((TileBoomMap[i][j] == 'L') && (TileMap[i][j] == ' '))
                                {
                                        boomSprite.setTextureRect(sf::IntRect(300, 30, 16,
16));
                                        boomDraw = true;
                                }
                                if ((TileBoomMap[i][j] == 'U') && (TileMap[i][j] == ' '))
                                {
                                        boomSprite.setTextureRect(sf::IntRect(330, 0, 16, 16));
                                        boomDraw = true;
                                }
                                if ((TileBoomMap[i][j] == 'D') && (TileMap[i][j] == ' '))
                                {
                                        boomSprite.setTextureRect(sf::IntRect(330, 60, 16,
16));
                                        boomDraw = true;
                                }
                                mapSprite.setPosition(j * CELL_SIZE, i * CELL_SIZE + 100);
                                windowGame.draw(mapSprite);
                                if (boomDraw)
                                {
                                        boomSprite.setPosition(j * CELL_SIZE, i * CELL_SIZE +
100);
                                        windowGame.draw(boomSprite);
                                }
                        }
                }
                for (int i = 0; i < 5; i++)
                {
                        if (game.bomberman[0].bomb[i].visible)
                                windowGame.draw(game.bomberman[0].bomb[i].sprite);
                        if (game.bomberman[1].bomb[i].visible)
                                windowGame.draw(game.bomberman[1].bomb[i].sprite);
                }
                windowGame.draw(game.bomberman[0].sprite);
                windowGame.draw(game.bomberman[1].sprite);
                if (showFPS)
                {
                        std::string tempString;
                        tempString = std::to_string((int)Framerate);
                        text.setString(tempString);
                        text.setPosition(1, 1);
                        windowGame.draw(text);
                }
                text.setString("Press F1 to help");
                text.setPosition(205, 5);
                windowGame.draw(text);


                sf::Text textState("", font, 16);
                std::string tempString;

                textState.setString("State firsts bomberman :");
                textState.setPosition(30, 30);
                windowGame.draw(textState);
```

```
			tempString = std::to_string(game.bomberman[0].GetBombState());
			textState.setString("Bomb state : " + tempString);
			textState.setPosition(30, 45);
			windowGame.draw(textState);

			tempString = std::to_string(game.bomberman[0].GetFireState() - 1);
			textState.setString("Fire state : " + tempString);
			textState.setPosition(30, 60);
			windowGame.draw(textState);

			tempString = std::to_string(game.bomberman[0].GetSpeedState() + 1);
			textState.setString("Speed state : " + tempString);
			textState.setPosition(30, 75);
			windowGame.draw(textState);

			textState.setString("State second bomberman :");
			textState.setPosition(350, 30);
			windowGame.draw(textState);

			tempString = std::to_string(game.bomberman[1].GetBombState());
			textState.setString("Bomb state : " + tempString);
			textState.setPosition(350, 45);
			windowGame.draw(textState);

			tempString = std::to_string(game.bomberman[1].GetFireState() - 1);
			textState.setString("Fire state : " + tempString);
			textState.setPosition(350, 60);
			windowGame.draw(textState);

			tempString = std::to_string(game.bomberman[1].GetSpeedState() + 1);
			textState.setString("Speed state : " + tempString);
			textState.setPosition(350, 75);
			windowGame.draw(textState);

			windowGame.display();
		}
		return 0;
}
```

Win.h

```
#ifndef SFML_GRAPHICS
#	include <SFML\Graphics.hpp>
#endif

void ShowWinner(sf::String name)
{
	sf::RenderWindow windowHelp(sf::VideoMode(550, 180), "You Win!", sf::Style::Close
| sf::Style::Titlebar);
	sf::RectangleShape rectangle(sf::Vector2f(550, 180));
	rectangle.setFillColor(sf::Color(185, 185, 185));
	rectangle.setPosition(0, 0);
	sf::Font font;
	//font.loadFromFile("data/Inky.ttf");
	font.loadFromFile("data/Brassie.ttf");
	//font.loadFromFile("data/danger.ttf");

	sf::Text text("", font, 25);
	text.setColor(sf::Color::White);

	while (windowHelp.isOpen())
	{
		sf::Event event;
```

```
		while (windowHelp.pollEvent(event))
		{
			if ((event.type == sf::Event::Closed) || (event.type ==
event.KeyPressed && ((event.key.code == sf::Keyboard::Return) || (event.key.code ==
sf::Keyboard::Escape))))
				windowHelp.close();
		}
		windowHelp.clear();
		windowHelp.draw(rectangle);

		text.setStyle(sf::Text::Bold | sf::Text::Underlined);
		text.setString("Congratulations!");
		text.setPosition(190, 20);
		windowHelp.draw(text);

		sf::Text winnerText("", font, 30);
		if (name != "first")
			winnerText.setString("The winner was the first player!");
		else
			winnerText.setString("The winner was the second player!");
		winnerText.setStyle(sf::Text::Bold);
		winnerText.setPosition(70, 70);
		windowHelp.draw(winnerText);

		text.setStyle(sf::Text::Bold | sf::Text::Underlined);
		text.setString("To continue, press Escape or Enter");
		text.setPosition(100, 130);
		windowHelp.draw(text);

		windowHelp.display();
	}
}
```

Map.h

```
#ifndef SFML_GRAPHICS
#	define SFML_GRAPHICS
#	include <SFML/Graphics.hpp>
#endif // !SFML_GRAPHICS
#include "Constants.h"
#define HEIGHT_MAP 25
#define WIDTH_MAP 25

sf::String TileBonusMap[HEIGHT_MAP] = {
	"MMMMMMMMMMMMMMMMMMMMMMMMM",
	"M                       M",
	"M M M M M M M M M M M M M",
	"M                       M",
	"M M M M M M M M M M M M M",
	"M                       M",
	"M M M M M M M M M M M M M",
	"M                       M",
	"M M M M M M M M M M M M M",
	"M                       M",
	"M M M M M M M M M M M M M",
	"M                       M",
	"M M M M M M M M M M M M M",
	"M                       M",
	"M M M M M M M M M M M M M",
	"M                       M",
	"M M M M M M M M M M M M M",
	"M                       M",
	"M M M M M M M M M M M M M",
	"M                       M",
	"M M M M M M M M M M M M M",
	"M                       M",
```

```
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "MMMMMMMMMMMMMMMMMMMMMMMMM",
};

const sf::String TileBonusMapConst[HEIGHT_MAP] = {
        "MMMMMMMMMMMMMMMMMMMMMMMMM",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "MMMMMMMMMMMMMMMMMMMMMMMMM",
};

sf::String TileBoomMap[HEIGHT_MAP] = {
        "MMMMMMMMMMMMMMMMMMMMMMMMM",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "MMMMMMMMMMMMMMMMMMMMMMMMM",
};

const sf::String TileBoomMapConst[HEIGHT_MAP] = {
```

```
        "MMMMMMMMMMMMMMMMMMMMMMMMM",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "M M M M M M M M M M M M M",
        "M                       M",
        "MMMMMMMMMMMMMMMMMMMMMMMMM",
};

sf::String TileMap[HEIGHT_MAP] = {
        "MMMMMMMMMMMMMMMMMMMMMMMMM",
        "M  SSSSSSSSSSSSSSSSSSS  M",
        "M MSMSMSMSMSMSMSMSMSMSM M",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "M MSMSMSMSMSMSMSMSMSMSM M",
        "M  SSSSSSSSSSSSSSSSSSS  M",
        "MMMMMMMMMMMMMMMMMMMMMMMMM",
};

const sf::String TileMapConst[HEIGHT_MAP] = {
        "MMMMMMMMMMMMMMMMMMMMMMMMM",
        "M  SSSSSSSSSSSSSSSSSSS  M",
        "M MSMSMSMSMSMSMSMSMSMSM M",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
        "MSMSMSMSMSMSMSMSMSMSMSMSM",
        "MSSSSSSSSSSSSSSSSSSSSSSSM",
```

```
	"MSMSMSMSMSMSMSMSMSMSMSMSM",
	"MSSSSSSSSSSSSSSSSSSSSSSSM",
	"MSMSMSMSMSMSMSMSMSMSMSMSM",
	"MSSSSSSSSSSSSSSSSSSSSSSSM",
	"MSMSMSMSMSMSMSMSMSMSMSMSM",
	"MSSSSSSSSSSSSSSSSSSSSSSSM",
	"MSMSMSMSMSMSMSMSMSMSMSMSM",
	"MSSSSSSSSSSSSSSSSSSSSSSSM",
	"MSMSMSMSMSMSMSMSMSMSMSMSM",
	"MSSSSSSSSSSSSSSSSSSSSSSSM",
	"MSMSMSMSMSMSMSMSMSMSMSMSM",
	"MSSSSSSSSSSSSSSSSSSSSSSSM",
	"MSMSMSMSMSMSMSMSMSMSMSMSM",
	"MSSSSSSSSSSSSSSSSSSSSSSSM",
	"M MSMSMSMSMSMSMSMSMSMSM M",
	"M  SSSSSSSSSSSSSSSSSSS  M",
	"MMMMMMMMMMMMMMMMMMMMMMMMM",
};

int GetRandomCoordinate()
{
	return 1 + rand() % (WIDTH_MAP - 1);
}

void SetBonusOnMap(int countBonus, char symbolBonus)
{
	int x, y;
	while (countBonus > 0)
	{
		x = GetRandomCoordinate();
		y = GetRandomCoordinate();
		if ((TileBonusMap[x][y] != 'M') && (TileMap[x][y] != ' ') &&
(TileBonusMap[x][y] != 'B') && (TileBonusMap[x][y] != 'S') && (TileBonusMap[x][y] !=
'F'))
		{
			TileBonusMap[x][y] = symbolBonus;
			countBonus--;
		}
	}
}

void GenerateBonusMap()
{
	int countFireBonus = 10, countBombBonus = 10, countSpeedBonus = 10;
	srand(time(0));
	SetBonusOnMap(countBombBonus, 'B');
	SetBonusOnMap(countFireBonus, 'F');
	SetBonusOnMap(countSpeedBonus, 'S');
}

void SetBoomOnMap(int j, int i, char Status)
{
	TileBoomMap[i][j] = Status;
}

void ResetMaps()
{
	for (int i = 0; i < HEIGHT_MAP; i++)
		for (int j = 0; j < WIDTH_MAP; j++)
		{
			TileMap[i][j] = TileMapConst[i][j];
			TileBonusMap[i][j] = TileBonusMapConst[i][j];
			TileBoomMap[i][j] = TileBoomMapConst[i][j];
		}
```

```
}
```

Help.h

```
#ifndef SFML_GRAPHICS
#	include <SFML\Graphics.hpp>
#endif

void ShowHelp()
{
	sf::RenderWindow windowHelp(sf::VideoMode(550, 340), "Help", sf::Style::Close |
sf::Style::Titlebar);
	sf::RectangleShape rectangle(sf::Vector2f(550, 340));
	rectangle.setFillColor(sf::Color(185, 185, 185));
	rectangle.setPosition(0, 0);
	sf::Font font;
	//font.loadFromFile("data/Inky.ttf");
	font.loadFromFile("data/Brassie.ttf");
	//font.loadFromFile("data/danger.ttf");

	sf::Text text("", font, 20);
	text.setColor(sf::Color::White);

	while (windowHelp.isOpen())
	{
		sf::Event event;
		while (windowHelp.pollEvent(event))
		{
			if ((event.type == sf::Event::Closed) || (event.type ==
event.KeyPressed && event.key.code == sf::Keyboard::Escape))
				windowHelp.close();
		}
		windowHelp.clear();
		windowHelp.draw(rectangle);
		text.setStyle(sf::Text::Bold | sf::Text::Underlined);
		text.setString("Key config");
		text.setPosition(220, 10);
		windowHelp.draw(text);

		text.setString("First player :");
		text.setPosition(60, 50);
		windowHelp.draw(text);
		text.setString("Second player :");
		text.setPosition(350, 50);
		windowHelp.draw(text);

		text.setStyle(sf::Text::Bold);
		text.setString("Walk Up - W");
		text.setPosition(70, 90);
		windowHelp.draw(text);
		text.setString("Walk Down - S");
		text.setPosition(60, 120);
		windowHelp.draw(text);
		text.setString("Walk Left - A");
		text.setPosition(65, 150);
		windowHelp.draw(text);
		text.setString("Walk Right - D");
		text.setPosition(60, 180);
		windowHelp.draw(text);
		text.setString("Bomb - Space");
		text.setPosition(65, 210);
		windowHelp.draw(text);

		text.setString("Walk Up - Up");
```

```
            text.setPosition(360, 90);
            windowHelp.draw(text);
            text.setString("Walk Down - Down");
            text.setPosition(340, 120);
            windowHelp.draw(text);
            text.setString("Walk Left - Left");
            text.setPosition(345, 150);
            windowHelp.draw(text);
            text.setString("Walk Right - Rigth");
            text.setPosition(340, 180);
            windowHelp.draw(text);
            text.setString("Bomb - NUM 0");
            text.setPosition(355, 210);
            windowHelp.draw(text);

            text.setString("Help - F1");
            text.setPosition(230, 240);
            windowHelp.draw(text);

            text.setString("Show FPS - F9");
            text.setPosition(205, 270);
            windowHelp.draw(text);

            text.setStyle(sf::Text::Bold | sf::Text::Underlined);
            text.setString("Close Window - ESC");
            text.setPosition(185, 300);
            windowHelp.draw(text);

            windowHelp.display();
        }
}
```

Constant.h

```
#define CELL_SIZE 22.0
#define PICTURE_BOMBERMAN_HEIGTH 20.0
#define PICTURE_BOMBERMAN_WIDTH 17.0
#define PICTURE_BOMB_SIZE 16
#define dX 0.1
#define dY 0.1
#define START_SPEED 0.05
#define BONUS_SPEED 0.01
#define LEFT_KEY 0
#define RIGHT_KEY 1
#define UP_KEY 2
#define DOWN_KEY 3
#define COUNT_CELLS 25
```