Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему:

«РЕАЛИЗАЦИЯ ИГРЫ «ЗМЕЙКА»»

БГУИР КП 1-40 01 01 521 ПЗ

Студент: гр. 051005 Станкевич Е. Ю.

Руководитель: Болтак С.В.

Минск 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Тема данной работы заключается в создании игры. В современном обществе популярным видом досуга является игровая деятельность. Это занятие не только детей и школьников, играют люди разных возрастов и профессий. Люди могут играть для развлечения и для отдыха, в компании и в одиночку. Игры покупают или просто скачивают в интернете. Устраивают турниры и соревнования, победы на которых приносят заработок.

Развивается игровая индустрия стремительно, появляются новые подходы к созданию игр. Уровень графики значительно вырос с момента их первого появления. Существуют разные категории игровых приложений: от шутеров и стратегий до головоломок и словесных игр. Создаются они с помощью дополнительных средств разработки: специальных библиотек, движков и др. Используются разные языки программирования. Существующие сегодня средства позволяют изображать сложные 3D-объекты и физические процессы с реалистичностью жизни. Тем не менее так же популярными остаются и простые 2D-платформеры.

Платформер — жанр компьютерных игр, в которых основу игрового процесса составляют прыжки по платформам, лазанье по лестницам, сбор предметов, необходимых для победы над врагами или завершения уровня. Многие игры подобного жанра характеризуются нереалистичностью, рисованной мультяшной графикой. Для определения столкновений между игровыми объектами могут использоваться как крупные плитки, так и пиксели.

В этой работе я отражу этапы разработки 2D-платформера. Я создам привлекательный интерфейс, вызывающий интерес, а также немного трудностей в прохождении. Я проведу анализ литературных источников с выявлением необходимых черт программы, сформирую постановку задачи, покажу разработку алгоритма программы, объясню его выбор, обосную отдельные технические моменты. Также покажу корректность работы отдельных моментов программы, необходимую при взаимодействии с пользователем. Достижения отражу в заключении, где выявлю сильные и слабые сторону полученного программного средства.

1.АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ И СУЩЕСТВУЮЩИХ АНАЛОГОВ

1.1 Анализ литературных источников.

В ходе анализа выявлено три основных типа 2D-платформеров.

Первый тип – основанный на плитках. Карта мира состоит из плиток, которые являются или не являются препятствием для передвижения. Движение персонажа может быть ограничено плитками, это самый простой способ реализации данного способа. Могут использоваться специальные анимации для плавного передвижения игрока. Вычисляется клетка, в которую прибыл игрок, затем проверяется, является ли она препятствием и персонаж перемещается, если это можно сделать, а в это время проигрывается плавная анимация. В таком случае персонаж не может находиться между двумя плитками, он всё время находится в какой-то одной(или большем количестве, это зависит от размера фигурки). Такой способ создаёт трудности в реализации движения по диагонали и прыжках по дуге, потому они часто не используются. Однако, используя карту из плиток можно дать персонажу возможность перемещаться свободно. Пересечения с препятствиями определяются по прямоугольнику, в который вписывается фигурка персонажа. Также можно усложнять работу с препятствием: например, при движении по горизонтали что-то является препятствием, а в прыжке его можно преодолеть (так называемые односторонние платформы). Известная «Super Mario Bros.» сделана по этому принципу. Популярен он и сегодня.

Второй способ использует пиксели для определения столкновений. Это значительно улучшает детализацию, но реализовать это труднее. Такой способ отражает игра «Worms».

Третий способ завязан на векторах. Он по качеству графики схож с предыдущим, но требует меньше памяти. Сегодня он популярен благодаря существованию движков для создания игр.

1.2 Обзор аналогов.

Я рассмотрю два примера, которые используют первый способ реализации из пункта 1 этого раздела. Учитывая качественный подход создателей игр к их делу, их достоинства и недостатки определяются из индивидуальных предпочтений в игровом процессе. Я выявлю общие, которые могут привлекать внимание пользователей.

а) «Lode Runner»

«Lode Runner» — двухмерная компьютерная игра в жанре платформера-головоломки, впервые выпущенная в 1983 году. Она

состоит из большого количества уровней. Игроку предоставляется управление человечком без возможности прыгать. Задача игрока состоит в сборе нескольких одинаковых предметов, расположенных в разных частях уровня. Когда все предметы будут собраны, вверху уровня появляется лестница, по которой игрок переходит на следующий уровень. У персонажа есть враги, двигающиеся в его направлении. При соприкосновении с врагом уровень проигрывается и начинается заново. В случае исчерпания количества попыток игра заканчивается. Соприкосновение не учитывается когда персонаж проходит по обездвиженному в ловушке врагу, а также когда он падает на них сверху.

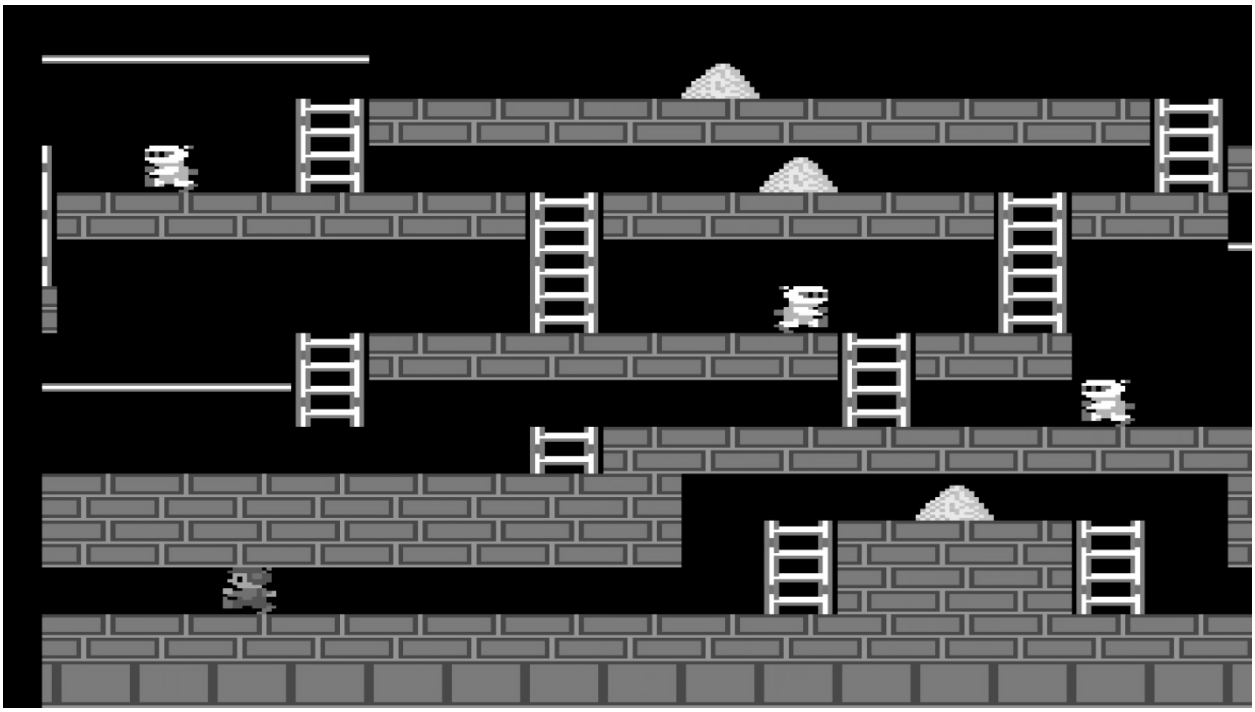

Рис. 1.2.1 – «Lord Runner»

Достоинства:

-большое количество уровней;

-наличие лестниц;

-удобное управление;

-интересное взаимодействие с врагами (возможность загнать их в ловушку).

Недостатки:

-быстрая смерть (от одного прикосновения к врагу);

-невозможность прыгать.

б) «Super Mario Bros.»

Марио является одним из самых известных игровых персонажей в мире. Он является главным героем игры вместе со своим братом Луиджи

(для второго игрока). Цель игры – пройти через Грибное Королевство и спасти заточённую принцессу. По пути Марио встречаются злые враги(черепахи, дикобразы и др), взаимодействие с которыми различается. По пути встречается большое количество различных бонусов и секретных хранилищ с сокровищами. Игрок может изменяться в размерах и получать сверхспособности, если съест хороший бонус. Игра состоит из разных миров и стиль окружающей обстановки меняется на протяжении игрового процесса. Имеются секретные переходы. После прохождения игры можно начать её снова в более сложном режиме.

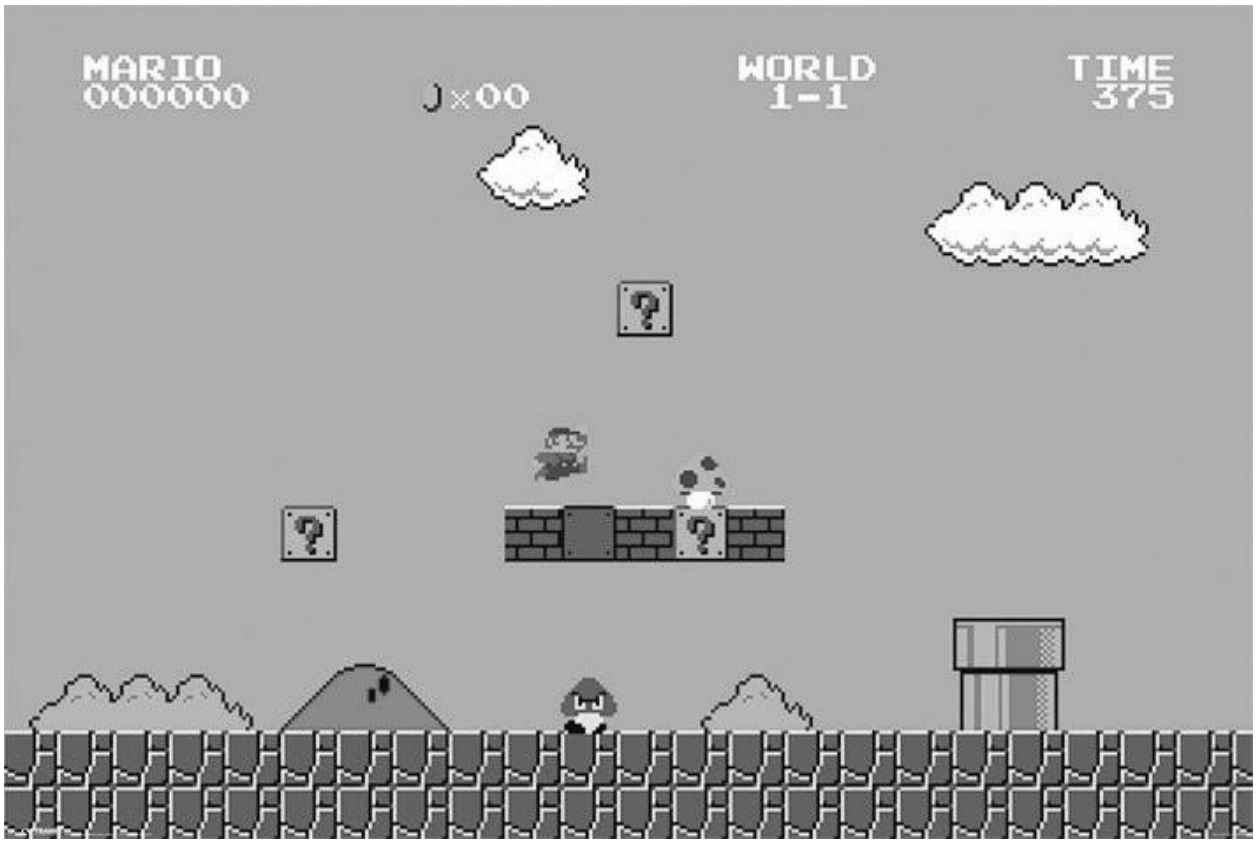

Рис. 1.2.2 – «Super Mario Bros.»

Достоинства:
-разнообразные миры;
-привлекательная цель игры;
-наличие различных бонусов;
-забавный персонаж;
-удобное управление.

Недостатки обнаружены не были, игра является хорошим примером 2D-платформера.

Эти примеры показывают, как схожи игры такого плана. В интернете можно найти множество подобных им. Но их продолжают создавать и дальше, так как такие детали, как другая карта, персонаж, другая цель,

особый приз в конце или изменённый набор бонусов, влияют на интерес к игре. Важным так же является наличие звука. Это заметно улучшает игровой процесс и дополняет общую картину приложения. Фоновая картинка, особые плитки, умеющие стрелять враги и многое другое может быть реализовано для повышения интереса и привлечения пользователей.

1.3 Обзор библиотек.

Для разработки выбран язык C++. Он предоставляет несколько библиотек, которые упрощают создание программного средства. Считаются простыми для подобного приложения – SFML(Simple and fast multimedia library) и MFC(Microsoft foundation classes). MFC упрощает разработку графических приложений для Microsoft Windows, но SFML даёт лучшую возможность легко создавать любую 2D-графику: начиная от простейших одиночных геометрических фигур (типа треугольника) и заканчивая полноценными играми-платформерами, как в этом проекте.

2.ПОСТАНОВКА ЗАДАЧИ

Целью данной работы является разработка игрового средства. Полученная игра должна быть удобна в использовании, понятна, привлекательна. Существование большого количества аналогов затрудняет создание уникальной по своей идее игры, так как изучить все существующие не представляется возможным. Однако возможно установить общие идеи, которые могут повысить качество игровой программы.

Повысить интерес к игре может:

- наличие необычной цели(хотя выбор классической в виде спасения кого-то(как в «Super Mario Bros.») так же имеет место быть);

- выбор из большого числа персонажей;

-изменение внешнего вида персонажа (формы, цвета, одежды, размера и другое);

- наличие разнообразных уровней;

- наличие разных миров в игре;

- смена стиля карты в зависимости от уровня;

-наличие необычных элементов карты (движущихся платформ, плиток нестандартной формы);

- альтернативные концовки (в зависимости от действий персонажа в игре, от сбора определённого количества предметов и другое);

-разнообразное звуковое сопровождение (звуки ударов, падений, прыжков, фоновая музыка);

-и др.

В плане написания программы важно учесть, что многие объекты могут обладать схожими свойствами, и их можно не реализовывать для всех объектов в отдельности.

Полезным является сохранение игрового прогресса для того, чтобы при повторном входе в игру пользователь мог продолжить с того уровня, на котором остановился. Хорошим дополнением является масштабируемость окна программы.

Таким образом, можно сформулировать требования, которым должно соответствовать программное средство. Оно должно решить следующие задачи:

а) удовлетворить требования к внешнему виду:

-интуитивно понятный интерфейс;

-наличие пояснений возможных действий;

-визуально приятная графика;

б) реализовать физические процессы;

в) удовлетворить требования к эффективной организации процедур и функций:

-общая реализация процедур и функций, присущих разным объектам.

г) организовать звуковое сопровождение;

д) предоставить возможность сохранения прогресса (уровня, здоровья, очков) в файл.

Для разработки выбрана ИСР Visual Studio, язык C++, библиотека SFML.

3. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1. Создание истории и интерфейса игры.

Необычной и интересной показалась идея игры, в которой главный герой – это змейка, ползущая по земле и собирающая мышек (так как змеи едят мышей). Вся история отображена на рисунке 3.3.1.а).

Управление стандартное – с помощью стрелочек на клавиатуре.

В большинстве игр предполагается вознаграждение за прохождение. Это может быть какая-то медаль, письменное поздравление, открытие особых режимов игры и другое. Я выбрала поздравление с наступающим новым годом, так как это во время разработки актуально (рис. 3.3.1.а).). А целью данной игры является получение этого вознаграждения. Для этого надо набрать достаточное количество мышек (и шариков).

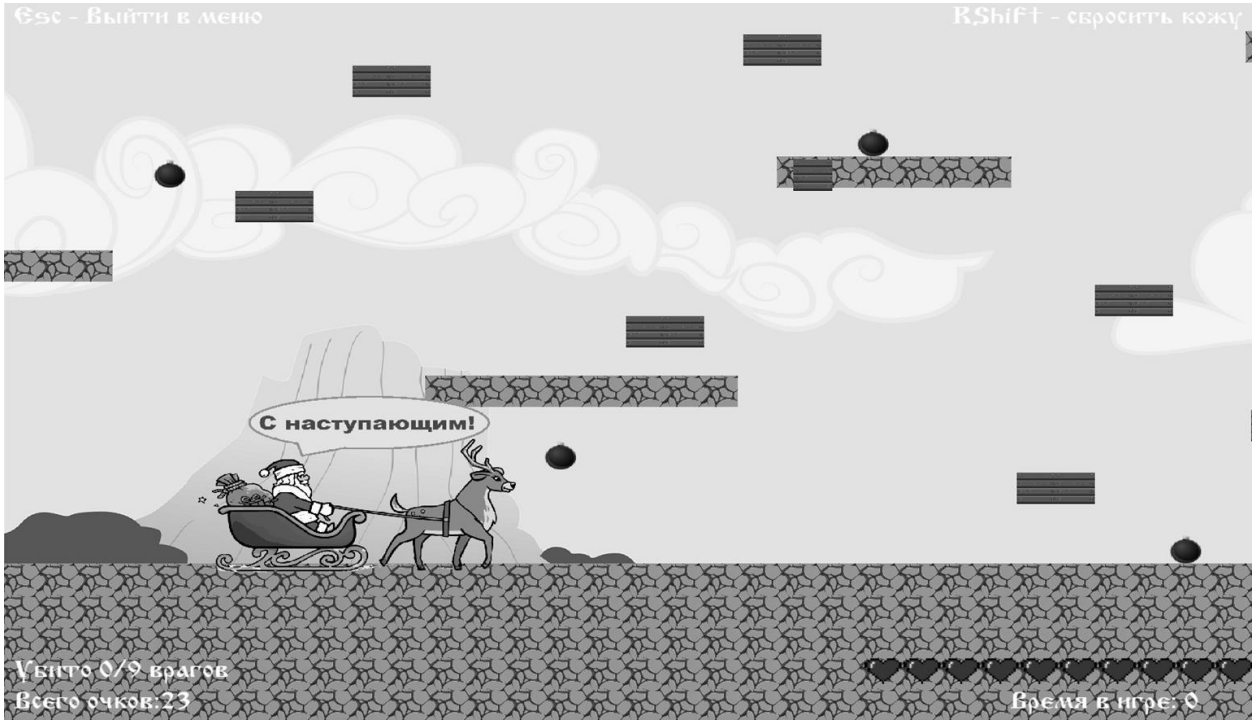

рис. 3.1.а). Поздравление в конце игры

Если такое количество не наберётся, будет выведена презрительная надпись (рис. 3.1.б).).

Подробнее об интерфейсе в руководстве пользователя (раздел 5).

рис 3.1.б). Презрительная надпись.

3.2. Определение структур данных.

Важной частью разработки программного средства – выбор подходящих структур для хранения данных, используемых при взаимодействии с программой.

3.2.1. Работа с классами.

В объектно-ориентированном программировании типы данных могут содержать не только данные, но и функции, которые будут работать с этими данными. Для определения такого типа данных в языке C++ используется ключевое слово class. Использование ключевого слова class определяет новый пользовательский тип данных — класс. В языке C++ переменная класса называется экземпляром (или «объектом») класса. Помимо хранения данных, классы могут содержать и функции. Функции, определенные внутри класса, называются методами.

Классы посчитались удобной структурой для хранения информации об игроке, врагах и других схожих объектах. Для этого используется родительский класс Creature(рис 3.2.1. а).) и дочерние классы, которые будут описаны ниже.

В родительском классе создаются переменные, характеризующие свойства, являющиеся общими для объектов разных типов:

- координаты;
- скорость;

- состояние жизни (жив / мёртв);
- текстура;
-спрайт;
- имя;
-и др.

```
class Creature{
public:
    float x, y;
    float w, h, dx, dy, speed, movetime, ch = 0;
    enum { left, right, jump, up, down, stay } state;
    bool shoots;
    int shootingtime;
    bool life, moves, onground;
    int health;
    Texture texture;
    Sprite sprite;
    String name;
    Creature(Image &image, float X, float Y, float W, float H, String Name){ ... }

    FloatRect getRect(){ ... }

    virtual void update(float time) = 0;
};
```

рис. 3.2.1. а). Класс Creature

Процедура update используется для обновления информации об объекте, она переопределяется в дочерних классах.

Подробнее расскажу о классе игрока. Он наследуется от Creature. В нём добавляются две дополнительные процедуры:

1) control, где обрабатываются нажатия клавиш, то есть реализуется управление. Используются клавиши «влево», «вправо», «вверх». Происходит изменение скоростей по направлениям(dx, dy), смена рисующихся картинок персонажа. Также обрабатывается нажатие клавиши «F», которое изменяет цвет змейки во время движения (изменяется картинка, которая используется для спрайта змейки).

2) interactWithMap, где обрабатывается взаимодействие с плитками карты уровня. Проверяется столкновение не со всеми плитками, а только с соседними, что оптимизирует проверку.

Внимание можно уделить особенности игрока – мигании при ранении или в состоянии неуязвимости. Для этих состояний используются

дополнительные переменные типа bool, а также переменные, которые считают время. В зависимости от того, сколько прошло времени с перехода в какое-либо из вышеуказанных состояний, выбирается цвет змейки. Помимо этого в движении змейка может менять свой цвет (по нажатию клавиши F). Схема реализации функции update класса Player показана на рис. 3.2.1.б). В ней происходит вызов двух перечисленных выше процедур, изменение координат, установка камеры (она следит за игроком), а также обработка особых состояний игрока.

Дочерние классы от Creature также создавались для других объектов:

а) Enemy для врагов. В нём так же определяется процедура взаимодействия с картой. Он отталкивается от стенок и падает вниз, если стенки нет на пути и он доходит до края места, на котором стоит.

б) MovingPlatform для движущихся платформ. Они так же имеют скорость, направление, двигаются по времени и взаимодействуют с игроком, потому для них создан свой класс.

в) Bullet для пуль (огненных шаров).

рис. 3.2.1.б). Схема процедуры update класса Player.

3.2.2. Работа с массивом карты.

Для карты выбрана простая структура хранения – массив строк. Обозначения:

- b – стреляющий враг (коричневый);
- f – ядовитый цветок (кактус);
- h – сердечко (дополнительная жизнь);
- l – маленькая движущаяся платформа;
- m – мышка (которую можно есть);
- n – новогодний шарик (не бомба);
- p – широкая движущаяся платформа;
- z – зелье неуязвимости;
- ‘ ‘(пробел) – пустота (отображается задний фон);
- 0 – стенка.

Загрузка строк для массива происходит из текстового файла (пример на рис. 3.2.2.а).).

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000g0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000     h0000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000     000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000  000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000          000000000000
0000000000000000000000000000000000000000000000000000000000000000000000   0 b  0000000000000000
0000000000000000000000000000000000000000000000000000000000000000000      00000000000000000000
0000000000000000000000000000000000000000000000000           0000000000000000000000000
0000000000000000000000000000000000000000000         00000000000000000000000000000
0                                  b                                        b     0
0                          0000000000000000000000                                0
0             h                                          h                      0
0                                      f                                         0
0          m         p                                                         0
0                                                                 00000000000    0
0g           00000000000                     l                                 0
0  p                          m              p                                 0
0               m                e f                                          0
0  s  m  0                0000000000000000                      m                0
0   m    0                                  0                                m   0
0     b m0    ffff 0     e    0           h0      fffffffffffffzfffffffffff     d0
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

рис. 3.2.2.а). Файл с уровнем.

3.2.3. Работа со списками.

Для хранения объектов врагов, пуль и платформ, которые появляются и уничтожаются, удобно использовать списки. Я использую список из библиотеки <list>. Это двусвязный список, каждый элемент которого

содержит 2 указателя: один указывает на следующий элемент списка, а другой — на предыдущий элемент списка. List предоставляет доступ только к началу и к концу списка — произвольный доступ запрещен. Для прохода по элементам списка используются итераторы.

3.3. Определение пунктов меню.

При входе в игру на экране появляется меню (рис. 3.3.а).). Нажатием левой кнопки мыши можно перейти к определённому его пункту. Пункты меню стандартные. Имеется четыре кнопки, они описаны ниже.

3.3.1. История. При первом входе в игру предполагается прочитать историю, чтобы понять, в чём дело (рис. 3.3.1. а).).

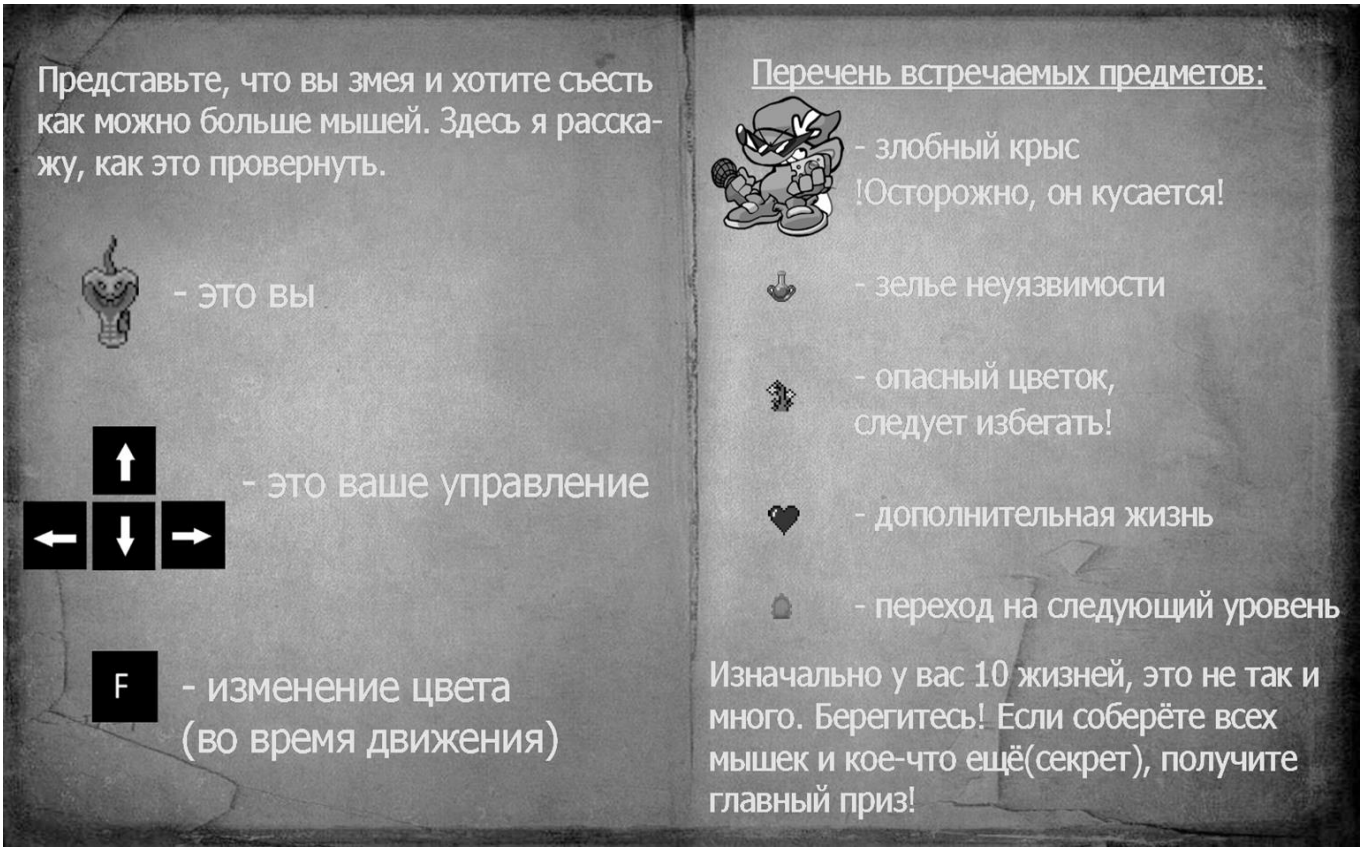

рис. 3.3.1.а). История игры.

По нажатию клавиши esc можно выйти обратно в меню.

3.3.2. Новая игра.

При нажатии на кнопку новой игры происходит установление по умолчанию значений переменных счёта, здоровья игрока, номера уровня.

Игра начинается сначала, с первого уровня, без возможности вернуть сохранённый прогресс.

3.3.3. Продолжить.

При нажатии на «Продолжить» происходит продолжение игры с того уровня, на котором остановился пользователь, с теми характеристиками, которые были в момент конца предыдущего уровня (или с теми, что загружены из файла). Если игра уже была закончена, то на экран выведется презрительная надпись или покажется поздравление (в зависимости от того, как она была пройдена перед последним сохранением).

3.3.4. Выход.

Обработка нажатия на «Выход» происходит в две стадии:

а) Сохранение значений. Значения переменных (здоровье игрока, счёт, уровень, переменные для цвета змейки) сохраняются в текстовый файл score.txt(рис.3.3.4.а).).

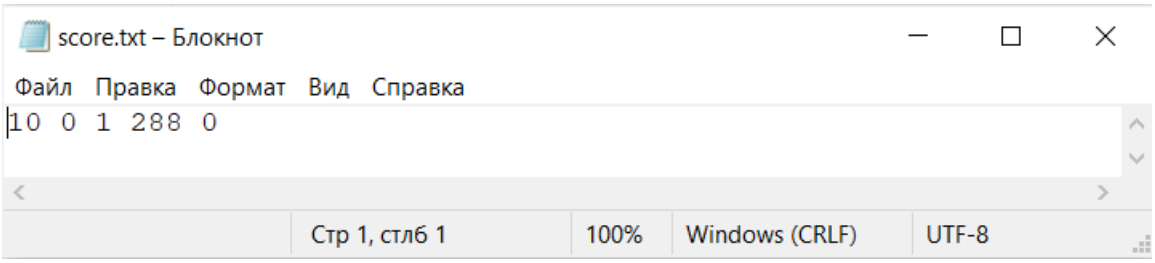

рис.3.3.4.а). Файл со значениями.

б) Закрытие окна. Закрывается окно программы.

3.4. Игровой процесс.

Игровой процесс содержит так называемый главный цикл программы, в котором происходят основные действия. Его схема на рис. 3.4.а). В нём происходит обновление объектов, а также отображение их на экране в определённом порядке (сначала карта, затем остальные объекты, чтобы их было видно). Помимо основных элементов игры отображаются подсказки по управлению – по выходу в меню и рестарту уровня. Помимо «пассивных» действий происходит обработка события закрытия окна, это входит в условие основного цикла.

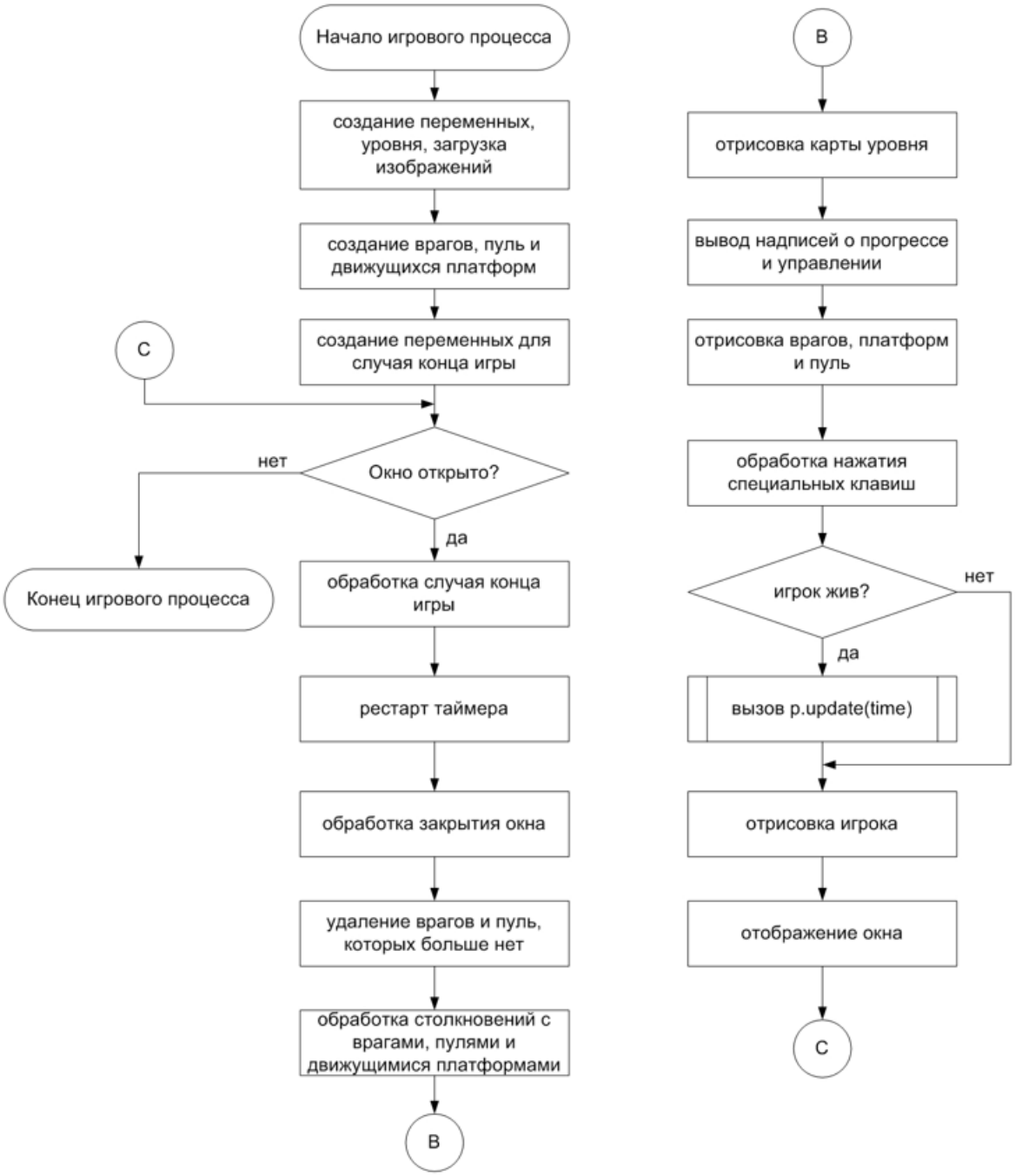

рис. 3.4.а). Схема игрового процесса.

3.5. Работа со звуком.

Для того чтобы придать больше презентабельности игровому приложению, используется звуковое сопровождение. Для работы со звуком используется класс Music для основной темы и победной музыки в случае конца игры как удачного, так и не очень, и класс Sound для коротких звуков. Добавлены звуки:

- удара врага;
- получения урона;
- сбора мышек, подарков, сердечек;

- открывания двери в новый уровень;
- смерти игрока;
- огненного шара.

Музыка начинает играть при входе в программу, главная тема зациклена. Когда игра заканчивается, она становится тише и играет победная песня. Затем снова играет главная тема.

рис. 3.3.а). Меню программы.

3.6. Структура программы.

В первую очередь происходит загрузка данных из файла и присваивание взятых оттуда значений переменным. Затем отображается меню и программа «ожидает» реакции пользователя. То есть цикл длится, пока не будет закрыто окно и обрабатывается нажатие на кнопки меню (и крестика в правом верхнем углу). В соответствии с пунктом меню происходит какое-то действие, затем выход обратно в меню. Подробнее показано на схеме на рис.3.6.а).

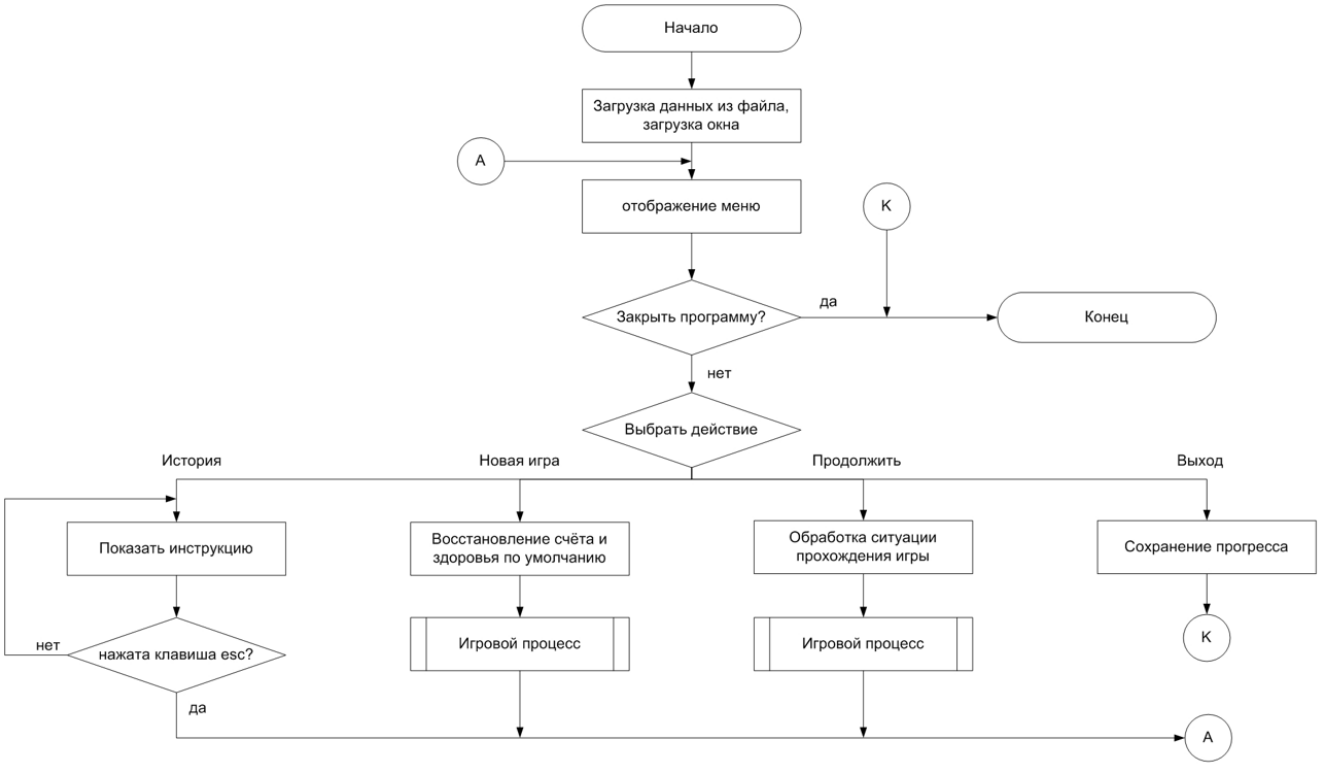

рис.3.6.а). Схема программы.

4. ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

4.1 Тестирование отображено в табл. 4.1.а).

табл. 4.1.а). Тестирование программы.

№ Теста	Цель теста	Способ проверки	Ожидаемый результат	Вердикт
1.	Проверить отображение истории игры	Нажатие кнопк «История» в меню	Отображение истории	Пройден
2.	Проверить загрузку уровня из файла	Нажатие кнопки «Новая игра»	Загрузка первого уровня	Пройден
3.	Проверить корректность работы управления	Нажатие трёх клавиш управления	Движение в сторону стрелки нажатой клавиши	Пройден
4.	Проверить корректность работы прыжков	Попытки запрыгнуть на «ступеньки» повыше и спрыгнуть с них; прыжки по диагонали	Корректная работа	Пройден
5.	Проверить корректность работы со стенками	Попытка выйти за границы карты	Змейка остаётся у стены	Пройден
6.	Проверить корректность работы столкновения с врагом	Приближение к врагу такое, что змейка его касается	Получение урона и мигание красным	Пройден
7.	Проверить корректность работы столкновения с огненным шаром	Приближение к огненному шару такое, что змейка его касается	Получение урона и мигание красным	Пройден
8.	Проверить корректность работы	Подбор зелья неуязвимости и попытки сбора	Мигание синим и не получение урона	Пройден

	состояния неуязвимости	ядовитых цветков, касания врагов и огненных шаров		
9.	Проверить корректность работы взаимодействия с платформами	Прыжки на платформу и с платформы, движение по платформе, езда на ней без движения	Корректная работа	Пройден
10.	Проверить правильность перехода на новый уровень	Нажатие кнопки «Новая игра» и касание двери перехода на новый уровень	Загрузка и отображение нового уровня	Пройден
11.	Проверить корректность работы сбора сердечек и мышек(шариков) и их отображения	Сбор сердечек, мышек и шариков	Отображение собранного количества в правом нижнем углу экрана	Пройден
12.	Проверить корректность работы ситуации смерти игрока	Сбор большого количество цветков; нахождение рядом с врагом в ожидании смерти	Змейка мигает красным, теряет сердечки(в правом нижнем углу) и игра прекращается (двигаться больше нельзя)	Пройден
13.	Проверить корректность работы концовки с успешным прохождением игры	Сбор достаточного количества предметов	Отображение праздничного поздравления и конец игры с выходом в меню	Пройден
14.	Проверить корректность работы концовки с неуспешным прохождением игры	Сбор недостаточного количества предметов	Отображение презрительной надписи и конец игры с выходом в меню	Пройден

15.	Проверить корректность сохранения в файл	Прохождение игры до второго уровня, выход из игры и открытие файла score.txt	Корректные данные в файле	Пройден
16.	Проверить корректность работы загрузки данных из файла	Прохождение игры до второго уровня, выход из игры и повторный вход	При нажатии кнопки «Продолжить» загружается второй уровень и отображается имевшееся ранее количество общих очков и жизней	Пройден
17.	Проверить уничтожение пуль при столкновении со стеной	Приближение к врагу, ожидание выхода пули, следование за ней, пока она не коснётся стены	Уничтожение пули	Пройден

4.2. Анализ полученных результатов.

В ходе разработки било получено игровое средство, которое удовлетворяет поставленным изначально требованиям с некоторыми дополнениями. Все тесты пройдены успешно.

В итоге достигнуто:

а) Привлекательный пользовательский интерфейс:

- имеются пояснения по управлению;
- управление является стандартным;
- цвета отражаемых объектов гармонируют друг с другом, создавая приятную зрению картину;
- забавный персонаж, умеющий менять цвет.

б) Корректная работа игровых функций и процедур:

- столкновения с врагами, пулями, платформами, цветами работают верно;
- состояния неуязвимости и получения урона работают корректно;
- персонаж ускоряется в прыжке при падении вниз;
- персонаж не выходит за границы карты;
- камера следует за персонажем при движении;
- отображается прогресс уровня во время игры.

в) Корректная работа с файлами:
- сохранение в файл прогресса;
- загрузка прогресса из файла;
- загрузка карт уровней из файла.

г) Наличие звукового сопровождения:
- при получении урона;
- при уничтожении врагов;
- при переходе на новый уровень;
- и др.

5. РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

При входе в программу отображается игровое меню. При первом запуске программы можно прочитать историю, чтобы понять, в чём дело. В истории описано управление и встречаемые объекты. Там же сказано, что персонаж пользователя – змейка.

В процессе игры змейка ползает по пустыне, встречает врагов и сражается с ними, может наткнуться на ядовитое растение (рис. 5.а).), бродит по дорожкам, находит тайники.

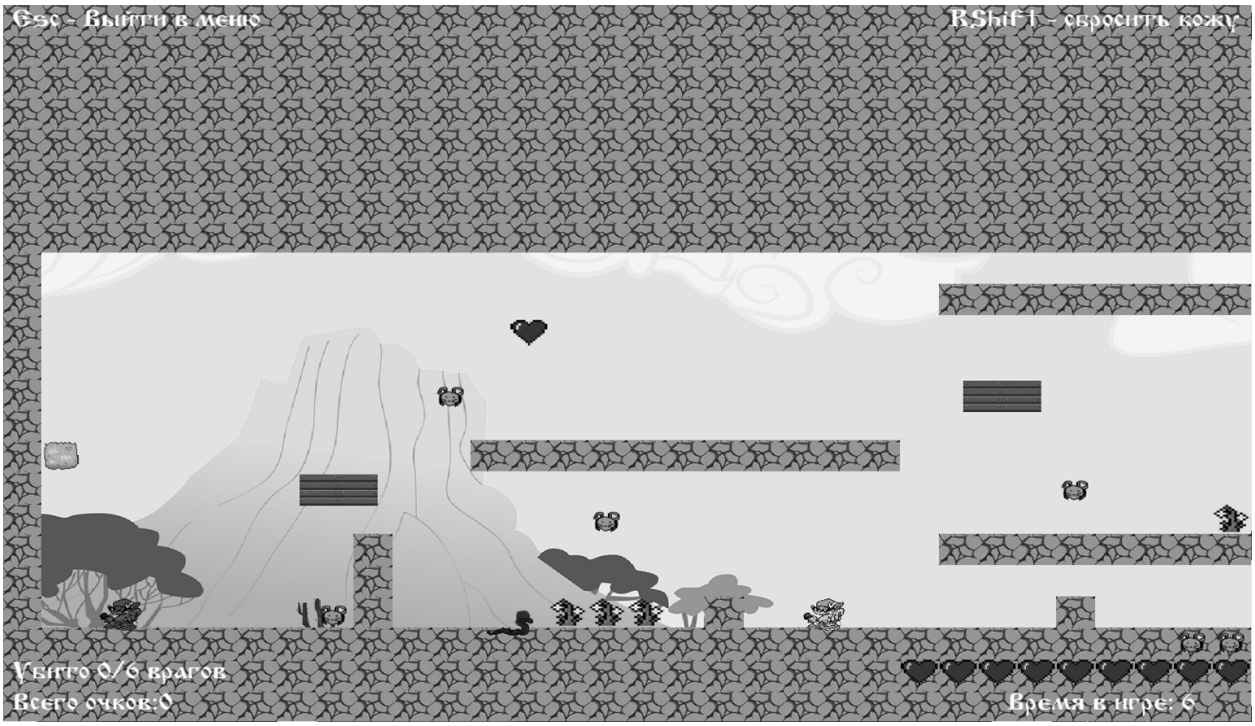

рис. рис. 5.а). Игровой процесс (змея ранена).

Предлагается стандартное управление – с помощью стрелочек на клавиатуре. Змейка может прыгать по нажатию клавиши вверх и двигаться влево и вправо по нажатию соответствующих стрелочек.

Цель игры – набрать побольше мышек и других предметов, чтобы получить приз (какой именно, скрывается). Количество собранных предметов отображается в правом нижнем углу. У змейки имеется запас жизней (изначально их 10) и её могут ранить. Количество жизней отображается в правом нижнем углу. Когда змея ранена, она мигает красным цветом и теряет жизни. Её можно ранить тремя способами:

а) съесть ядовитое растение (коснуться его);

б) получить удар от врага (коснуться его);

в) пораниться об огненный шар (их выпускает особый вид врагов, описанный ниже).

Враги встречаются двух видов:

1) Простой разноцветный враг. Он быстрый, но особо глупый. Ползает из стороны в сторону.

2) Коричневый враг. Этот враг гораздо хитрее. Если он чувствует близость змейки, начинает выпускать огненные шары с частотой в определённое количество секунд, которые ранят персонажа. Они уничтожаются, если касаются персонажа или стенки. До того они могут лететь очень долго, за ними стоит следить.

Чтобы уничтожить врага, достаточно на него один раз прыгнуть сверху. Это может быть как во время прыжка по нажатию клавиши «вверх», так и при падении с высоты. Необходимо помнить, что нельзя убить двух врагов сразу. Надо подождать, пока они пойдут в разные стороны, и пойти в бой после этого.

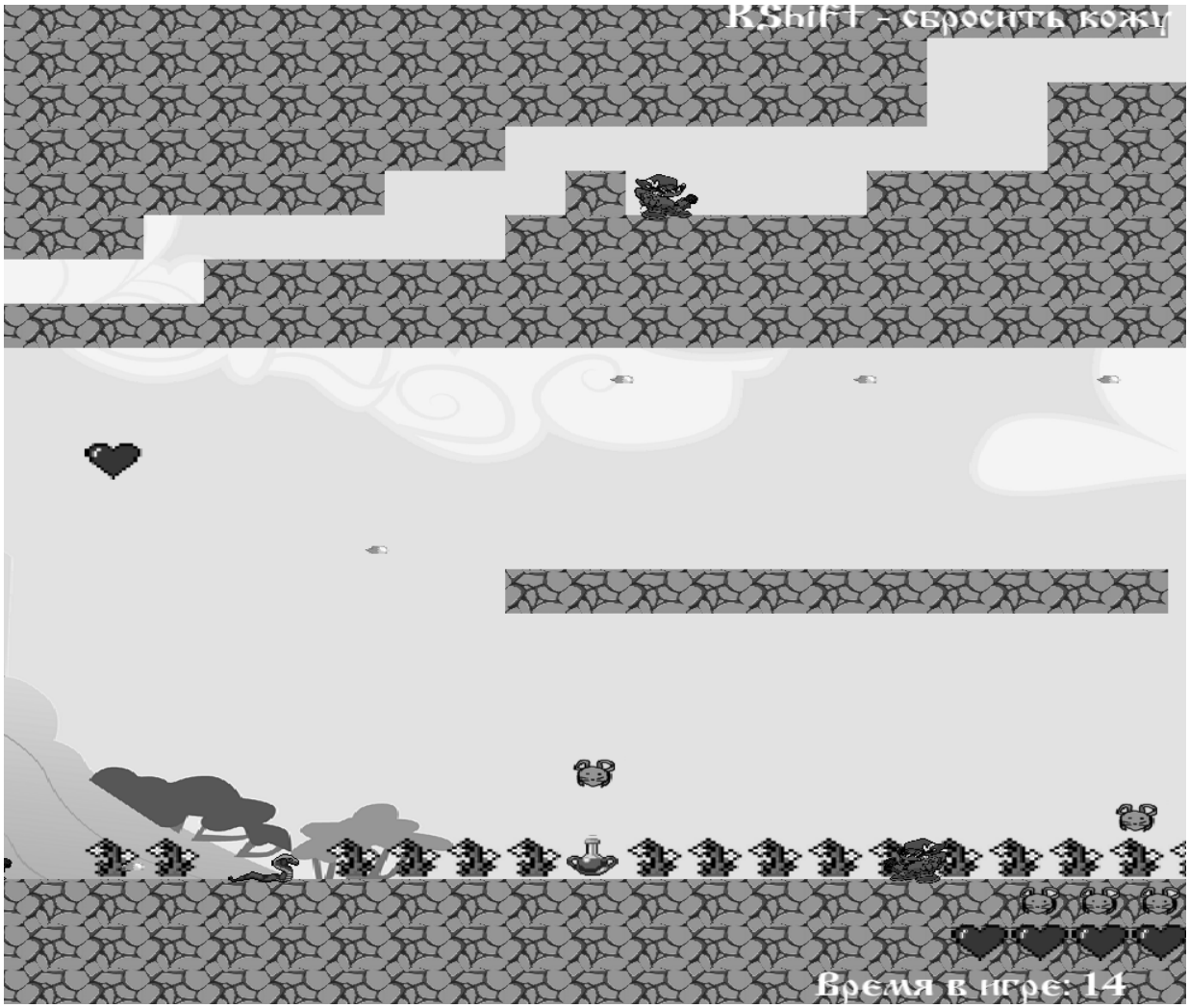

рис. 5. б)

В ситуациях, когда до чего-то сложно добраться (рис 5.б).), следует обратить внимание на то, какие предметы недалеко находятся. Змейке на

пути может встретиться зелье неуязвимости – маленькая колба синего цвета. Если змейка съест его, то не будет получать урон в течение нескольких секунд. В это время она будет мигать синим цветом (если сама змея синяя, можно это не заметить, потому требуется внимательность).

Следует думать, до всех ли предметов можно добраться, и могут ли враги где-то прятаться или слиться с фоном.

Помощниками в достижении труднодоступных мест являются движущиеся платформы (но не стоит забывать и о смекалке). Это дорожки, по которым тоже можно ходить, но они перемещаются с места на место и могут куда-то подвезти змейку. Враги тоже могут по ним ходить, потому нужно быть бдительным.

ЗАКЛЮЧЕНИЕ

В ходе работы над курсовым проектом было разработано игровое средство в жанре 2D - платформера.

Были изучены литературные источники и аналоги игр, позволяющие осуществить интуитивно понятный пользовательский интерфейс и организовать работу с данными. Были исследованы способы реализации подобных программ и выбран подходящий. Полученная программа может быть удобна в использовании даже неопытному пользователю, так как имеются пояснения по управлению.

Реализовано управление игровым персонажем, взаимодействие с окружающими его объектами, отображение их на карте.

По итогу разработки пользователь может насладиться лёгкой игрой с приятной графикой, напоминающей всемирно известные игры прошедшего времени. Он может изучать уровни, отыскивая тайники, а может получить приз без особых напряжений. Предложено также сохранение прогресса.

Программное средство было успешно протестировано и испытано, проверено на соответствие начальным требованиям. Реализованные алгоритмы работают корректно.

Проект возможно развивать и дальше. Можно добавить больше уровней, сделать особых врагов – боссов, за победу над которыми получается дополнительная награда. Интересно расширить количество игроков до двух с возможностью соревнования.

Цель работы считаю достигнутой.

СПИСОК ЛИТЕРАТУРЫ

[1] Documentation of SFML 2.5.1 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.sfml-dev.org/documentation/2.5.1/index.php

[2] Ravesli C++ [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ravesli.com/uroki-cpp/

[3] kychka-pc | SFML [Электронный ресурс]. – Электронные данные. – Режим доступа: https://kychka-pc.ru/

[4] SFML Game Development By Example / Raimondas Pupius. – Packt Publishing, 2015. – 523 p.

ПРИЛОЖЕНИЕ А.ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp

```
#include <SFML/Graphics.hpp>
#include "map.h"
#include "View.h"
#include "Game.h"
using namespace sf;
int main()
{
	RenderWindow window(VideoMode(1600, 920), "WORKS!", Style::Close);
	setValues();

	//игра
	setsounds();
	maintheme.play();
	gameState = inmenu;
	play(window, levelNumber);
	saveValues();

	return 0;
}
```

Game.h

```
#include <list>;
#include "menu.h"
#include "Music.h"

//всякие глобальности
bool rshift;
//с уровнями всякое
int levelNumber = 1;
const int kolLevel = 3;
const int specialLevel = 3;
//

class Creature{
public:
	float x, y;
	float w, h, dx, dy, speed, movetime, ch = 0;
	enum { left, right, jump, up, down, stay } state;
	bool shoots;
	int shootingtime;
	bool life, moves, onground;
	int health;
	Texture texture;
```

```
	Sprite sprite;
	String name;
	Creature(Image &image, float X, float Y, float W, float H, String Name){
		x = X;
		y = Y;
		w = W;
		h = H;
		name = Name;
		movetime = 0;
		shootingtime = 0;
		ch = 0;
		dx = 0;
		dy = 0;
		speed = 0;
		moves = false;
		onground = false;
		life = true;
		health = 100;
		texture.loadFromImage(image);
		sprite.setTexture(texture);
		sprite.setOrigin(w / 2, h / 2);
	}

	FloatRect getRect(){
		return FloatRect(x, y, w, h);
	}

	virtual void update(float time) = 0;
};

class Player :public Creature{
public:
	float colx = 0, coly = 0, timeColor, sped = 0.22, injuredTime = 0, invulnTime = 0;
	int score = 0;
	bool injured = false, canclose = false, invuln = false;
	enum spriteState { left, right, jump, up, down, stay };
	spriteState state;
	//int colx = 48 * 6, coly = 0;
	float CurrentFrame = 0;
	bool oldonground = onground, onplatf = false;
	int strangev = 17;


	Player(Image &image, float X, float Y, float W, float H, String Name) :Creature(image,
X, Y, W, H, Name){
		colx = gcolx;
```

```
		coly = gcoly;
		timeColor = 0;
		state = stay;
		score = 0;
		health = hp;
		Name = "Player";
		sprite.setTextureRect(IntRect(0 + colx, 0 + coly, w, h));
		speed = sped;
		//onground = true;
	}

	void interactWithMap(float Dx, float Dy){
		int i, j;

		for (i = x / 40; i < (x + w) / 40; i++)
		for (j = y / 40; j < (y + h) / 40; j++){
			if (Map[j][i] == '0'){
				if (Dx < 0) { x = i * 40 + 40; dx = 0; }
				if (Dx > 0) { x = i * 40 - w; dx = 0; }
				if (Dy > 0) { y = j * 40 - h; dy = 0; oldonground = onground;
onground = true; if (!oldonground) sped = 0.22; }
				if (Dy < 0) { y = j * 40 + 40; dy = 0; }
			}

			if (Map[j][i] == 'm'){
				Map[j][i] = ' ';
				score++;
				bonussound.play();
			}

			if (Map[j][i] == 'n'){
				Map[j][i] = ' ';
				score++;
				bonussound.play();
			}

			if (Map[j][i] == 'f'){
				Map[j][i] = ' ';
				if ((!invuln)) { health -= 1; hisssound.play(); }
				if (!invuln) injured = true;
			}

			if (Map[j][i] == 'h'){
				Map[j][i] = ' ';
				health++;
				bonussound.play();
```

```
                }

                if (Map[j][i] == 'd'){
                        canclose = true;
                        doorsound.play();
                }

                if (Map[j][i] == 'z'){
                        Map[j][i] = ' ';
                        invuln = true;
                        bonussound.play();
                }
        }

}

void control(float time){
        if (colx == 6 * 48) { strangev = 10; }
        else { strangev = 17; }

        if (Keyboard::isKeyPressed(Keyboard::Right)) {
                speed = sped;
                state = right;
                CurrentFrame += 0.005*time;
                if (CurrentFrame > 3) CurrentFrame -= 3;
                sprite.setTextureRect(IntRect(48 * int(CurrentFrame) + colx, 96 + coly +
strangev, 48, 48 - strangev));
                h = 25;
                w = 40;
        }

        if (Keyboard::isKeyPressed(Keyboard::Left)) {
                speed = sped;
                state = left;
                CurrentFrame += 0.005*time;
                if (CurrentFrame > 3) CurrentFrame -= 3;
                sprite.setTextureRect(IntRect(48 * int(CurrentFrame) + colx, 48 + coly +
strangev, 48, 48 - strangev));
                h = 25;
                w = 40;
        }

        if ((Keyboard::isKeyPressed(Keyboard::Up)) && (onground)) {
                dy = -0.9;
                if (!onplatf) sped = 0.26;
                state = jump;
```

```
                onground = false;

        }

        if (life){
                if (Keyboard::isKeyPressed(Keyboard::F)){

                        if (timeColor > 0.5){
                                colx += 48 * 3;
                                if (colx >= 48 * 12){
                                        colx = 0;
                                        coly += 48 * 4;
                                        if (coly >= 48 * 4 * 2) coly = 0;
                                        gcolx = colx; gcoly = coly;
                                }
                                timeColor = 0;
                        }
                }

        }
}

void update(float time){
        if (life) control(time);
        else sprite.setColor(Color::Red);
        switch (state){
        case right: dx = speed; break;
        case left: dx = -speed; break;
        case stay: break;
        case jump: break;
        case up: break;
        case down: break;
        }

        sprite.setOrigin(w / 2, h / 2);

        x += dx * time;
        interactWithMap(dx, 0);
        y += dy * time;
        interactWithMap(0, dy);

        sprite.setPosition(x + w / 2, y + h / 2);
        if (health <= 0){ oversound.play(); life = false; }
        if (!moves)
                speed = 0;
        else speed = sped;
```

```
            if (life) setviewcoordinates(x, y);
            timeColor += time * 0.001;
            if (life)
            {
                if (injured){
                    injuredTime += time * 0.001;
                    if ((int)(injuredTime * 8) % 2) sprite.setColor(Color::White);
                    else sprite.setColor(Color::Red);
                    if (injuredTime > 1.5){
                        injuredTime = 0;
                        injured = false;
                        sprite.setColor(Color::White);
                    };
                }

                if (invuln){
                    invulnTime += time * 0.001;
                    if ((int)(invulnTime * 4) % 2) sprite.setColor(Color::White);
                    else sprite.setColor(Color(59, 88, 252));
                    if (invulnTime > 5){
                        invulnTime = 0;
                        invuln = false;
                        sprite.setColor(Color::White);
                    };
                }
            }

            dy = dy + 0.0025*time;
        }
};

class Enemy :public Creature{
public:
        float ch = 0;

        Enemy(Image &image, float X, float Y, float W, float H, String Name)
:Creature(image, X, Y, W, H, Name){
            ch = 0;
            if (name == "easy"){
                sprite.setTextureRect(IntRect(0, 0, w, h));
                ch = 0.1;
                w *= ch * 0.8;
                h *= ch * 0.8;
                sprite.setScale(-ch, ch);
                dx = 0.15;
```

```
                speed = 0.15;
                state = right;
            }

            if (name == "easyb"){
                sprite.setTextureRect(IntRect(0, 0, w, h));
                ch = 0.1;
                w *= ch * 0.8;
                h *= ch * 0.8;
                sprite.setScale(-ch, ch);
                dx = 0.05;
                speed = 0.05;
                state = right;
                shoots = false;
                sprite.setColor(Color(168, 104, 70));
            }
        }

        void interactionWithMap(float Dx, float Dy){
            for (int i = x / 40; i < std::min((x + w) / 40, (float)map_width); i++)
            for (int j = y / 40; j < std::min((y + h) / 40, (float)map_height); j++){
                if (Map[j][i] == '0'){
                    if (Dx < 0) { x = i * 40 + 40; dx = speed; state = right;
sprite.setScale(-ch, ch); }
                    if (Dx > 0) { x = i * 40 - w; dx = -speed; state = left;
sprite.setScale(ch, ch); }
                    if (Dy > 0) { y = j * 40 - h; dy = 0; }
                    if (Dy < 0) { y = j * 40 + 40; dy = 0; }
                }
            }
        }

        void update(float time){
            if ((name == "easy") || (name == "easyb")){
                x += dx * time;
                interactionWithMap(dx, 0);
                y += dy * time;
                interactionWithMap(0, dy);
                if (life) sprite.setPosition(x + w / 2, y + h / 2);
                if (health <= 0) life = false;
                dy = dy + 0.0025*time;
            }
        }
};

class MovingPlatform : public Creature{
```

```
public:
	int hightime = 2000;
	MovingPlatform(Image &image, float X, float Y, float W, float H, String Name)
:Creature(image, X, Y, W, H, Name){
		w *= 0.2353;
		h *= 0.2353 / 4;
		sprite.setScale(0.2353, 0.2353);
		sprite.setTextureRect(IntRect(402, 26, W, H));
		if (name == "longmovingPlatform") hightime = 6170;
		dx = 0.08;
		sprite.setOrigin(w / 2, h / 2);
	}

	void update(float time){
		x += dx* time;
		movetime += time;
		if (movetime > hightime){
			movetime = 0;
			dx *= -1;
		}
		sprite.setPosition(x + w / 2, y + h / 2);
	}
	FloatRect getRect(){
		if (name == "longmovingPlatform") return FloatRect(x, y, w + 10, h);
		return FloatRect(x, y, w + 20, h);
	}
};

class Bullet :public Creature{
public:
	int direction;

	Bullet(Image &image, float X, float Y, float W, float H, String Name, int dir)
:Creature(image, X, Y, W, H, Name){
		x = X;
		y = Y;
		direction = dir;
		speed = 0.15;
		w = 16;
		h = 7;
		life = true;
	}


	void update(float time)
	{
```

```
                switch (direction)
                {
                case 1: dx = speed; dy = 0;   sprite.setScale(1, 1); break;//state == right
                case 0: dx = -speed; dy = 0;  sprite.setScale(-1, 1); break;//

                }

                x += dx*time;//само движение пули по x

                if (x <= 0) x = 1;
                if (y <= 0) y = 1;
                if (x >= map_width * 40) x = map_width * 40 - 2;
                if (y >= map_height * 40) y = map_height * 40 - 2;


                for (int i = x / 40; i < (x + w) / 40; i++)
                for (int j = y / 40; j < (y + h) / 40; j++) {
                        if (Map[j][i] == '0') if (FloatRect(i * 40, j * 40, 40,
40).intersects(getRect())) //если этот объект столкнулся с пулей,
                        {
                                life = false;// то пуля умирает
                        }
                }

                sprite.setPosition(x + w / 2, y + h / 2);//задается позицию пуле
        }
};

bool Game(RenderWindow &window, int &levelNumber){
        rshift = false;
        //menu(window);
        //RenderWindow window(VideoMode(1600, 920), "WORKS!"/*, Style::Fullscreen*/);

        view.reset(FloatRect(0, 0, 1280, 920));

        createLevel(levelNumber);

        //для фона и
        Image map_image, map_image2;
        map_image.loadFromFile("images/text.png");
        map_image.createMaskFromColor(Color::White);
        map_image2.loadFromFile("images/text4.png");

        Texture map, map2, mainBackground;
        map.loadFromImage(map_image);
        map2.loadFromImage(map_image2);
```

```
	if(levelNumber!= specialLevel) mainBackground.loadFromFile("images/mainBg.png");
	else mainBackground.loadFromFile("images/specialBg.png");

	Sprite s_map, s_map2, mainBg(mainBackground);
	s_map.setTexture(map);
	s_map2.setTexture(map2);

	//if (levelNumber != specialLevel)


	//шрифты
	Font font;
	font.loadFromFile("files/CyrilicOld.TTF");
	Text text("", font, 30);
	text.setFillColor(Color::White);
	Text text2("", font, 90);
	text2.setFillColor(Color::Red);

	//игрок

	int colx = 0, coly = 0, jumpBool = 0, change = 0, px = 0, py = 0;
	Image pimage;
	pimage.loadFromFile("images/snakes.png");
	for (int i = 0; i < map_height; i++)
	for (int j = 0; j < map_width; j++){
		if (Map[i][j] == 's'){ px = j * 40; py = i * 40; break; }
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
	}
	Player p(pimage, px, py, 48, 48 - 5, "p");
	setviewcoordinates(p.sprite.getPosition().x, p.sprite.getPosition().y);
	mainBg.setPosition(view.getCenter().x - 640, view.getCenter().y - 460);
	//враги
	Image eimage;
	eimage.loadFromFile("images/rat3.png");
	eimage.createMaskFromColor(Color::White);
	int kolEnemies = 0;
	int kolKilled = 0;

	//платформы
	Image platfImage;
	platfImage = map_image2;

	//пули
	Image bulletImage;
	bulletImage.loadFromFile("images/bullet.png");
	bulletImage.createMaskFromColor(Color::Black);
```

```
	//звуки
	//setsounds();

	//тут со списком врагов
	std::list <Creature*> enemies;
	std::list<Creature*>::iterator list_i;
	for (int i = 0; i < map_height; i++)
	for (int j = 0; j < map_width; j++){
		if (Map[i][j] == 'e'){
			enemies.push_back(new Enemy(eimage, j * 40, i * 40, 450, 430,
"easy"));
			kolEnemies++;
		}

		if (Map[i][j] == 'b'){
			enemies.push_back(new Enemy(eimage, j * 40, i * 40, 450, 430,
"easyb"));
			kolEnemies++;
		}
	}

	std::list <MovingPlatform*> platforms;
	std::list<MovingPlatform*>::iterator listp_i;
	for (int i = 0; i < map_height; i++)
	for (int j = 0; j < map_width; j++){
		if (Map[i][j] == 'p'){
			platforms.push_back(new MovingPlatform(platfImage, j * 40, i * 40,
340, 170, "movingPlatform"));
		}

		if (Map[i][j] == 'l'){
			platforms.push_back(new MovingPlatform(platfImage, j * 40, i * 40,
170, 170, "longmovingPlatform"));
		}
	}

	std::list <Bullet*> bullets;
	std::list<Bullet*>::iterator listb_i;

	//для концовки
	Clock clock2;
	float wholeTime = 0, sx, sy;
	Image santaImage;
	santaImage.loadFromFile("images/santaall.png");
```

```
santaImage.createMaskFromColor(Color(166, 202, 240));
Texture santaTexture;
santaTexture.loadFromImage(santaImage);
Sprite santaSprite(santaTexture);
santaSprite.setTextureRect(IntRect(0, 0, 680, 476));
santaSprite.setScale(0.5, 0.5);
sx = view.getCenter().x - 1000;
sy = view.getCenter().y + 440;
//для

//время
Clock clock, gameClock;
int gameTime = 0;

setviewcoordinates(p.x, p.y);
while (window.isOpen())
{

	//////////////////////////////////////для концовки
	if ((levelNumber > kolLevel) && (totalscore >= 15)){
		//p.speed = 0;
		p.life = false;
		santaSprite.setPosition(sx, sy);
		window.draw(santaSprite);
			//clock2.restart();
			if (wholeTime < 6){

				float time2 = clock.getElapsedTime().asMilliseconds();
				wholeTime += time2 / 1000;
				sx += 0.1 * time2;
				santaSprite.setPosition(sx, sy);

			}

			if ((wholeTime >= 6) && (wholeTime < 13)){
				santaSprite.setTextureRect(IntRect(680, 0, 680, 476));
				float time2 = clock.getElapsedTime().asMilliseconds();
				wholeTime += time2 / 1000;
			}
			if (wholeTime >= 13) return false;

	}
	//////////////////////////////////////////////////////

	mainBg.setPosition(view.getCenter().x - 640, view.getCenter().y - 460);
	float time = clock.getElapsedTime().asMicroseconds();
```

```
		if (p.life) gameTime = gameClock.getElapsedTime().asSeconds();
		else { view.move(0.1, 0); mainBg.setPosition(view.getCenter().x - 640,
view.getCenter().y - 460);; /*continue;*/ }
		clock.restart();
		time = time / 1000;

		//Event event;
		while (window.pollEvent(event))
		{
			if (event.type == Event::Closed){
				window.close();
				//hp = p.health;
				return false;
			}

		}

		//закрываем, если у двери
		if (p.canclose == true) { gcolx = p.colx; gcoly = p.coly; hp = p.health; totalscore
+= p.score; return true; }
		gcolx = p.colx; gcoly =
p.coly;////////////////////////////////////////////////////////////////////////////////////////////////////
		if (!p.life) scrollwasd(time);

		//удаляем лишних врагов из списка
		for (list_i = enemies.begin(); list_i != enemies.end();){
			(*list_i)->update(time);

			if (!(*list_i)->life) list_i = enemies.erase(list_i);
			else {
				if ((*list_i)->name == "easyb"){
					if ((((*list_i)->x - p.x < 400) && ((*list_i)->x - p.x > 0)
&& ((*list_i)->state == 0)) || ((-(*list_i)->x + p.x < 400) && (-(*list_i)->x + p.x > 0) &&
((*list_i)->state == 1))){


						//if ((*list_i)->state == 1) std::cout << "right";


						if ((*list_i)->shootingtime == 0) (*list_i)->shoots =
true;
						(*list_i)->shootingtime += time;
						if ((*list_i)->shootingtime > 1500){ (*list_i)-
>shootingtime = 0; }
						if ((*list_i)->shoots){
							firesound.play();
```

```
                                                bullets.push_back(new Bullet(bulletImage,
(*list_i)->x, (*list_i)->y + 19, 16, 7, "bullet", (*list_i)->state));
                                                (*list_i)->shoots = false;
                                        }
                                }
                        }
                        list_i++;
                }
        }

        //пули
        for (listb_i = bullets.begin(); listb_i != bullets.end();){
                (*listb_i)->update(time);
                //std::cout << "here";
                if (!(*listb_i)->life) listb_i = bullets.erase(listb_i);
                else listb_i++;
        }
        for (listb_i = bullets.begin(); listb_i != bullets.end(); listb_i++){
                if ((*(*listb_i)).getRect().intersects(p.getRect())){
                        if ((*listb_i)->name == "bullet"){
                                if ((!p.injured) && (!p.invuln)) { p.health -= 1;
hisssound.play(); }
                                if(!p.invuln)p.injured = true;
                                (*listb_i)->life = false;

                        }
                }
        }


        //платформы
        for (listp_i = platforms.begin(); listp_i != platforms.end();){
                (*listp_i)->update(time);
                if (!(*listp_i)->life) listp_i = platforms.erase(listp_i);
                else listp_i++;
        }

        //проверяем столкновения с врагами
                                                //((int)p.y / 40 == (int)
(*list_i)->y / 40 - 1))
        for (list_i = enemies.begin(); list_i != enemies.end(); list_i++){
                if ((*(*list_i)).getRect().intersects(p.getRect())){
                        if (((*list_i)->name == "easy") || ((*list_i)->name == "easyb")){
                                if ((p.dy > 0) && ((p.onground == false) || (p.y + p.h <
(*list_i)->y + (*list_i)->h))){
                                        //(*list_i)->dx = 0;
```

```
                                        p.dy = -0.6;
                                        (*list_i)->health = 0;
                                        kolKilled++;
                                        hitsound.play();
                                }
                                else {
                                        if ((!p.injured) && (!p.invuln)) p.health -= 1;
                                        if (!p.invuln){ p.injured = true; hisssound.play(); }
                                }
                        }
                }
        }

        //столкновения с платформами
        for (listp_i = platforms.begin(); listp_i != platforms.end(); listp_i++){
                if ((*(*listp_i)).getRect().intersects(p.getRect())){
                        if (((*listp_i)->name == "movingPlatform") || ((*listp_i)->name
== "longmovingPlatform")){
                                if ((p.y - (*listp_i)->y < -p.h / 2) && (p.dy > 0) &&
((p.sprite.getPosition().x + 20 - (*listp_i)->sprite.getPosition().x > 0))){
                                        p.dy = 0;
                                        p.x += (*listp_i)->dx * time;
                                        p.onground = true;
                                        p.onplatf = true;
                                }
                                else {
                                        /*if ((!p.injured) && (!p.invuln)) p.health -= 1;
                                if(!p.invuln)p.injured = true;*/
                                        p.onplatf = false;
                                }
                        }
                        else p.onplatf = false;
                }
                else p.onplatf = false;
        }

        ////////////////////////////////////////////////////////////////////////////////////////////////////
        //рисую карту
        //рисую всё, что только рисую
        window.setView(view);
        window.clear(Color(238, 226, 206));
        window.draw(mainBg);
        for (int i = 0; i < map_height; i++)
        for (int j = 0; j < map_width; j++){
                if ((Map[i][j] == '0') || (Map[i][j] == '9')) {
                        s_map2.setScale(0.2353, 0.2353);
```

```
                                s_map2.setTextureRect(IntRect(402, 405, 170, 170));
s_map2.setPosition(j * 40, i * 40);
                                window.draw(s_map2); continue;
                        }
                        //if (Map[i][j] == '0') { s_map.setTextureRect(IntRect(0, 0, 40, 40)); }
                        if (Map[i][j] == 'f') { s_map.setTextureRect(IntRect(120, 40, 40, 40)); }
                        if (Map[i][j] == 'm') s_map.setTextureRect(IntRect(80, 40, 40, 40));
                        if (Map[i][j] == 'g') s_map.setTextureRect(IntRect(0, 200, 40, 40));
                        if (Map[i][j] == ' ') s_map.setTextureRect(IntRect(120, 320, 40, 40));
                        if (Map[i][j] == 'h') s_map.setTextureRect(IntRect(120, 0, 40, 40));
                        if (Map[i][j] == 'd') s_map.setTextureRect(IntRect(119, 80, 23, 40));
                        if (Map[i][j] == 'z') s_map.setTextureRect(IntRect(120, 160, 40, 40));
                        if (Map[i][j] == 'n') s_map.setTextureRect(IntRect(120, 120, 40, 40));

                        s_map.setPosition(j * 40, i * 40);
                        window.draw(s_map);
                        s_map.setScale(1, 1);
                }

                //рисую сердечки
                s_map.setTextureRect(IntRect(120, 0, 40, 40));
                int dx = 0;
                for (int i = 0; i < p.health; i++){
                        s_map.setPosition(view.getCenter().x + 600 - dx, view.getCenter().y +
375);
                        window.draw(s_map);
                        dx += 40;
                }

                //рисую прогресс в плане мышек
                if (levelNumber != specialLevel) s_map.setTextureRect(IntRect(80, 40, 40, 40));
                else s_map.setTextureRect(IntRect(120, 120, 40, 40));
                dx = 0;
                for (int i = 0; i < p.score; i++){
                        s_map.setPosition(view.getCenter().x + 600 - dx, view.getCenter().y +
335);
                        window.draw(s_map);
                        dx += 40;
                }

                //рисую текст времени
                char st[10], sst[30] = "Время в игре: ", s2t[30] = "Всего очков:", s3t[25] =
"RShift - сбросить кожу", s4t[20] = "Esc - Выйти в меню";
                _itoa_s(gameTime, st, 10);
                strcat_s(sst, st);
                text.setString(sst);
```

```
                text.setPosition(view.getCenter().x + 390, view.getCenter().y + 415);
                window.draw(text);
                //и текст очков
                _itoa_s(totalscore, st, 10);
                strcat_s(s2t, st);
                text.setString(s2t);
                text.setPosition(view.getCenter().x - 630, view.getCenter().y + 415);
                window.draw(text);
                //и управление
                text.setString(s4t);
                text.setPosition(view.getCenter().x - 630, view.getCenter().y - 460);
                window.draw(text);
                text.setString(s3t);
                text.setPosition(view.getCenter().x + 330, view.getCenter().y - 460);
                window.draw(text);
                //и текст убитых врагов
                char senemies[30] = "Убито ", svsp[4], senemies2[10] = " врагов", sslash[2] =
"/";
                _itoa_s(kolKilled, svsp, 10);
                strcat_s(senemies, svsp);
                strcpy_s(svsp, sslash);
                strcat_s(senemies, svsp);
                _itoa_s(kolEnemies, svsp, 10);
                strcat_s(senemies, svsp);
                strcat_s(senemies, senemies2);
                text.setString(senemies);
                text.setPosition(view.getCenter().x - 630, view.getCenter().y + 375);
                window.draw(text);


                for (list_i = enemies.begin(); list_i != enemies.end(); list_i++){
                        window.draw((*list_i)->sprite);
                }

                for (listp_i = platforms.begin(); listp_i != platforms.end(); listp_i++){
                        window.draw((*listp_i)->sprite);
                }

                for (listb_i = bullets.begin(); listb_i != bullets.end(); listb_i++){
                        window.draw((*listb_i)->sprite);
                }

                //текст концовки
                ////////////////для концовки
                if ((levelNumber > kolLevel) && (totalscore < 15)){
                        p.life = false;
```

```
                    char stroka[35] = "МОГЛИ НАБРАТЬ \n   И ПОБОЛЬШЕ!!!";
                    text2.setString(stroka);
                    text2.setPosition(view.getCenter().x - 350, view.getCenter().y - 140);
                    window.draw(text2);
                }

                if (Keyboard::isKeyPressed(Keyboard::Escape)) { gameState =
inmenu;/*menu(window);*/ return false; }
                if (Keyboard::isKeyPressed(Keyboard::RShift)) { /*hp = p.health;*/ rshift =
true; return true; }
                if(p.life) p.update(time);
                //рисую змейку и отображаю всё
                window.draw(p.sprite);

                for (int i = 0; i < map_height; i++)
                for (int j = 0; j < map_width; j++){
                    if ((Map[i][j] == '9')) {
                        s_map2.setScale(0.2353, 0.2353);
                        s_map2.setTextureRect(IntRect(402, 405, 170, 170));
s_map2.setPosition(j * 40, i * 40);
                        window.draw(s_map2);
                    }
                }

                //window.draw(mainBg);
                if (levelNumber > kolLevel) window.draw(santaSprite);
                window.display();
        }
        return false;
}

void setValues(){
        std::ifstream fin;
        fin.open("files/score.txt", std::ios::in);
        fin >> hp;
        fin >> totalscore;
        fin >> levelNumber;
        fin >> gcolx;
        fin >> gcoly;
        fin.close();
        //тут из файла собираем значения
}

void saveValues(){
        std::ofstream fout;
        fout.open("files/score.txt", std::ios::out | std::ios::trunc);
```

```
	fout << hp << ' ';
	fout << totalscore << ' ';
	fout << levelNumber<< ' ' << gcolx << ' ' << gcoly;
	fout.close();
	//тут из файла собираем значения
}

void play(RenderWindow & window, int &levelNumber){
	while (window.isOpen()){
		switch (gameState){
			case inmenu:
				menu(window);
				break;

			case inrest:
				levelNumber = 1;
				totalscore = 0;
				hp = 10;
				gameState = ingame;
				break;

			case ingame:
				if (levelNumber > kolLevel) {
					Music specMusic;
					specMusic.openFromFile("files/specMusic.ogg");
					if (totalscore >= 15) {
						specMusic.play();
						maintheme.setVolume(20);
					}
					Game(window, levelNumber);
					if (totalscore >= 15){
						specMusic.stop();
						maintheme.setVolume(100);
					}
					gameState = inmenu;
				}
				else if (Game(window, levelNumber)){
					if (!rshift) {
						levelNumber++;
					}
					if (levelNumber > kolLevel) {
						Music specMusic;
						if (totalscore >= 15) {
							maintheme.setVolume(20);

	specMusic.openFromFile("files/specMusic.ogg");
```

```
                                        specMusic.play();
                                    }
                                    Game(window, levelNumber);
                                    if (totalscore >= 15) {
                                        specMusic.stop();
                                        maintheme.setVolume(100);
                                    }
                                    gameState = inmenu;
                                }
                                else play(window, levelNumber);
                            }
                            break;

                        case inend:
                            saveValues();
                            window.close();
                            break;
                        }
                }
}
```

Music.h

```
#include <SFML/Audio.hpp>
sf::SoundBuffer hitbuff, hissbuff, firebuff, bonusbuff, overbuff, doorbuff;
sf::Music maintheme;
sf::Sound hitsound, hisssound, firesound, bonussound, doorsound, oversound;
void setsounds(){
        hitbuff.loadFromFile("files/hit.ogg");
        hissbuff.loadFromFile("files/hiss1.ogg");
        firebuff.loadFromFile("files/fire.ogg");
        bonusbuff.loadFromFile("files/bonus.ogg");
        overbuff.loadFromFile("files/over.ogg");
        doorbuff.loadFromFile("files/door.ogg");

        hitsound.setBuffer(hitbuff);
        hisssound.setBuffer(hissbuff);
        firesound.setBuffer(firebuff);
        bonussound.setBuffer(bonusbuff);
        oversound.setBuffer(overbuff);
        doorsound.setBuffer(doorbuff);

        maintheme.openFromFile("files/maintheme.ogg");
        maintheme.setLoop(true);
        //maintheme.setVolume(50);
}
```

Menu.h

```
Event event;
enum { inmenu, inrest, ingame, inauth, inend, instory } gameState;
int hp = 10, totalscore = 0;
int gcolx = 0, gcoly = 0;
#include <fstream>

void menu(RenderWindow & window) {
	Image menuIm;
	menuIm.loadFromFile("images/rmenu.png");
	menuIm.createMaskFromColor(Color(237, 28, 36));
	Texture menuTexture1, menuTexture2, menuTexture3, menuTexture4, menuTexture5,
menuBackground, storyText;
	menuTexture1.loadFromImage(menuIm);
	menuTexture2.loadFromImage(menuIm);
	menuTexture3.loadFromImage(menuIm);
	menuTexture4.loadFromImage(menuIm);
	menuTexture5.loadFromImage(menuIm);

	menuBackground.loadFromFile("images/des1.png");
	storyText.loadFromFile("images/story.png");

	Sprite menu1(menuTexture1), menu2(menuTexture2), menu3(menuTexture3),
menu4(menuTexture4), menu5(menuTexture5), menuBg(menuBackground),
storySprite(storyText);
	bool isMenu = 1;
	int menuNum = 0;

	menuBg.setPosition(0, 0);

	menu1.setPosition(600, 150);
	menu1.setTextureRect(IntRect(0, 333, 382, 111));
	menu2.setPosition(600, 271);
	menu2.setTextureRect(IntRect(0, 0, 382, 111));
	menu3.setPosition(600, 392);
	menu3.setTextureRect(IntRect(0, 111, 382, 111));
	menu4.setPosition(600, 513);
	menu4.setTextureRect(IntRect(0, 222, 382, 111));

	window.setView(window.getDefaultView());
	while (isMenu)
	{
		window.clear(Color(238, 226, 206));
		menu1.setColor(Color::White);
		menu2.setColor(Color::White);
		menu3.setColor(Color::White);
```

```
menu4.setColor(Color::White);

menuNum = 0;
if (IntRect(menu1.getPosition().x, menu1.getPosition().y, 382,
111).contains(Mouse::getPosition(window))) { menu1.setColor(Color(251, 148, 45));
menuNum = 1; }
if (IntRect(menu2.getPosition().x, menu2.getPosition().y, 382,
111).contains(Mouse::getPosition(window))) { menu2.setColor(Color(251, 148, 45));
menuNum = 2; }
if (IntRect(menu3.getPosition().x, menu3.getPosition().y, 382,
111).contains(Mouse::getPosition(window))) { menu3.setColor(Color(251, 148, 45));
menuNum = 3; }
if (IntRect(menu4.getPosition().x, menu4.getPosition().y, 382,
111).contains(Mouse::getPosition(window))) { menu4.setColor(Color(251, 148, 45));
menuNum = 4; }

while (window.pollEvent(event))
{
    if (event.type == Event::Closed)
    {
        window.close();
        isMenu = false;
    }
}

if (Mouse::isButtonPressed(Mouse::Left))
{
    switch (menuNum){
    case 1:
        window.draw(storySprite);
        window.display();
        while (!Keyboard::isKeyPressed(Keyboard::Escape));
        break;

    case 2:
        isMenu = false;
        gameState = inrest;
        break;

    case 3:
        gameState = ingame;
        isMenu = false;
        break;


    case 4:
```

```
                    isMenu = false;
                    gameState = inend;
                    break;
                }

            }

                window.draw(menuBg);
                window.draw(menu1);
                window.draw(menu2);
                window.draw(menu3);
                window.draw(menu4);

                window.display();
            }
}
```

Map.h

```
#include <SFML/Graphics.hpp>
#include <fstream>

std::ifstream fin;
sf::String Map[60];
int map_height;
int map_width;
void createLevel(int level){
    switch (level)
    {
    case 1: fin.open("files/level 1.txt", std::ios::in); break;
    case 2: fin.open("files/level 2.txt", std::ios::in); break;
    case 3: fin.open("files/level 3.txt", std::ios::in); break;
    default: fin.open("files/level 3.txt", std::ios::in); break;//тут должен быть последний
уровень!
    break;
    }

    char ss[181];
    int k = -1, l;
    while (fin){
        fin.getline(ss, 181, '\n');
        if (strlen(ss) != 0) l = strlen(ss);

        k++;
        Map[k] = ss;
    }
    map_width = l;
```

```
        map_height = k;
        fin.close();
}
```

View.h

```
#include <SFML/Graphics.hpp>
using namespace sf;

View view;
void setviewcoordinates(float x, float y){
        float tx = x, ty = y;
        if (x > (map_width - 16) * 40) tx = (map_width - 16) * 40;
        if (x < 640) tx = 640;
        if (y >(map_height - 12) * 40) ty = (map_height - 12) * 40 + 20;
        if (y < 460) ty = 460;

        view.setCenter(tx, ty);
}

void scrollwasd(float time){
        if (Keyboard::isKeyPressed(Keyboard::A)){

                view.move(-0.1 * time, 0);
        }
        if (Keyboard::isKeyPressed(Keyboard::W)){
                view.move(0, -0.1 * time);
        }
        if (Keyboard::isKeyPressed(Keyboard::S)){
                view.move(0, 0.1 * time);
        }
        if (Keyboard::isKeyPressed(Keyboard::D)){
                view.move(0.1 * time, 0);
        }
}
```

ПРИЛОЖЕНИЕ Б.ВЕДОМОСТЬ

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 1–40 01 01 521 ПЗ	Пояснительная записка	52с.
	Графические документы	
ГУИР 051005-21 СП	Реализация игры «Змейка». Схема программы.	Формат А1

					БГУИР КП 1-40 01 01 221 Д1		
Изм.	Л.	№ докум.	Подп.	Дата	Реализация игры «Змейка». Ведомость курсового проекта	Лист	Листов
Разраб.		Станкевич Е.Ю.		27.12.21		Т 52	52
Пров.		Болтак С.В.		27.12.21		Кафедра ПОИТ гр. 051005	