Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

«Игра Lode Runner для Linux»

по дисциплине

«Системное программное обеспечение вычислетельных машин»

Выполнил:

студент гр. 450502

Стаховский Антон Владимирович

Руководитель:

Лавникевич Д. А.

Минск, 2016

ЛИСТ ЗАДАНИЯ

Название темы:

«Игра Lode Runner для Linux»

Основные требования к программе:

- Наличие спрайтовой графики
- Большое количество уровней
- Сильный искусственный интеллект
- Наличие геймплея, знакомого по оригинальной игре

Рекомендуемые системные требования:

- Операционная система Gnu/Linux
- 64 МБ ОЗУ
- Разрешение экрана не ниже 1040x640
- Не менее 5 МБ свободного места на жестком диске
- Клавиатура

Для сборки из исходных кодов потребуются:

- Компилятор g++ не ниже версии 4.7.4
- Наличие следующих библиотек:
    - freeglut не ниже 2.5.2 версии
    - glu не ниже 7.2.2 версии

Для установки игры требуется сначала установить программу lode_decode, которая служит для декодирования уровней в ASCII код. По полученной после декодирования файла информации игра отрисовывает карту уровня. Установить lode_decode[1] и саму игру(steal_n_run[2]) можно используя комманду make. Makefile находятся в папках src/ и lode_decode/.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С каждым годом возрастает количество различных переизданий старых игр. Их цель — вернуть интерес ностальгирующих геймеров, а также заинтересовать новых игроков. Переиздания могут адаптировать игру под новые архитектуры ПК и консолей, расширяют список поддерживаемых платформ, улучшать геймплей и визуальную часть игры. Также большим спросом пользуются различные эмуляторы старых консолей.

Мною была выбрана цель сделать клон классической игры Lode Runner[3], используя современные технологии. Для разработки была выбрана OS Linux, язык C++ и графическая библиотека OpenGL.

Разработанная в ходе курсового проектирования игра имеет простое управление, которое понравится новым пользователям и будет привычно геймерам, игравшим в оригинальные игры серии. Графика имеет приятный 8-ми битный стиль. В игре были использованы спрайты из NES версии игры. Игра имеет крайне непритязательные системные требования, что позволяет запустить ее даже на старых ПК. Новыми игроками будет легко освоет игровой процесс, а опытные найдут некоторые интересные отличия от ставшего им привычного геймплея.

Следующие версии программы будут направлены на исправление существующих ошибок, добавление нового функционала, улучшение кроссплатформенности.

Приоритетные задачи:

- Улучшение логики AI
- Замена «рваной» поблочной логики перемещения на плавную попиксельную
- Добавление анимации и новых спрайтов
- Добавление звука

Из наиболее интересными улучшениями игры, которые будут реализованы после приоритетных задач и позволят выделить данный проект среди аналогов, представлются:

- Добавление поддержки геймпада
- Добавление многопользовательской игры на одном компьютере
- Сетевая игра
- Возможность выбора игровой темы
- Редактор уровней

Опыт в разработке данного программного продукта помог мне получить знания по разработке игр, разработке приложений с использованием

библиотек для низкоуровневой работы с графикой. Интересным также оказось изучение структуры графических форматов (BMP[4]) и использование полученных знаний на практике.

ОБЗОР ИСТОЧНИКОВ

1.1 *Обзор программных аналогов*

Ниже приведено описание типичного для игр серии гейплея.

Игрок управляет маленьким человечком и должен собрать всё золото, лежащее на данном уровне, избегая встречи роботами Весь уровень целиком виден на экране и состоит из кирпичных платформ, лестниц, а также натянутых верёвок, по которым можно двигаться, держась за них руками. Персонаж игры не может убивать роботов, но может создавать для них ямы в кирпичном полу— попавший в яму робот задерживается там на некоторое время. Герой может падать с любой высоты, не разбиваясь, но не может подпрыгивать. После того, как все ящики с золотом собраны, где-либо появляется лестница (или несколько), по которой нужно добраться до верха экрана— это приведёт к переходу на следующий уровень.

Первая игра серии была выпущена в 1983 году для Apple II. С этого времени было выпущено множество её клонов для различных платформ. Ниже приведены наиболее интересные из них.

- **AppleII версия —** первая версия игры. Затягивающий геймплей помог серии пережить более 30 лет.

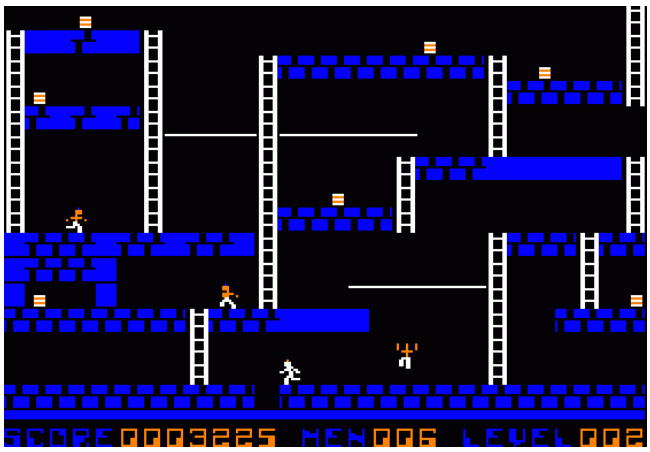

рисунок1 — оригинальная версия для AppleII

- **Xbox360 версия —** современная 3д версия игры. Имеет динамичный геймплей и сочную графику.

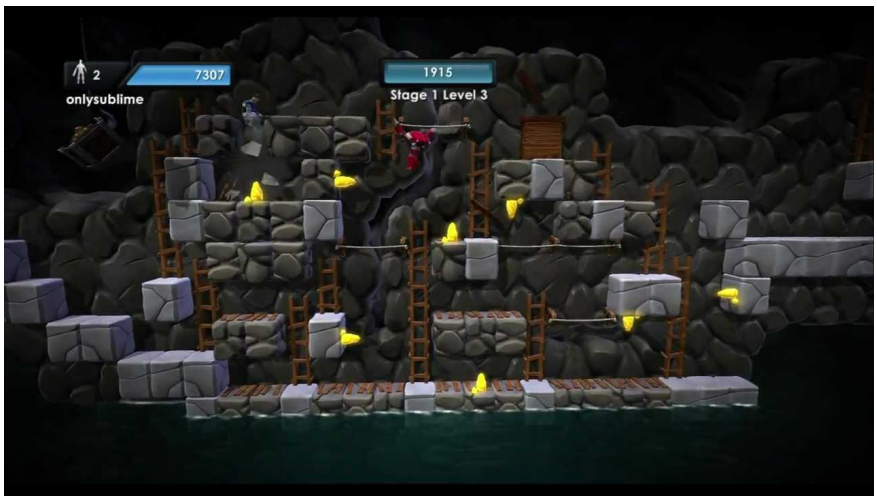

рисунок2 — версия для XBox360

1.2 *Основные технологии, использованные при разработке, и их краткий обзор:*

C++ — компилируемый статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции.

OpenGL — платформонезависимая библиотека для работы с 2d и 3d графикой.

G++ — свободно распространяемый компилятор языка C++.

Freeglut — библиотека для создания окон, содержащих opengl контекст, управление окнами. Средства для работы с мышью, клавиатурой, джойстиком.

1.3 *Функциональные требования к проекту:*

Игра является клоном Lode Runner.

Список требований к проекту:

- Наличие врагов
- Возможность игроку создавать ловушки (ямы)
- Наличие всех типов блоков из оригинальной игры

- Наличие огромного количества уровней
- Наличие спрайтов
- Физика перемещения из оригинальной игры

СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ

2.1 *Описание структуры классов*

В курсовом проекте содержатся 6 классов :

- Creature
- Enemy
- Player
- Field
- Painter
- Game

Краткое описание классов:

◆ Creature — абстрактный класс. Содержит логику передвижения существ. Наследниками данного класса являются классы Enemy и Player.

◆ Player — наследник класса Creature. Отвечает за управление игроком и логику поведения игрока. Объект данного класса управляет поведением врагов, попавших в ловушки, и самих ловушей.

◆ Класс Enemy – еще один наследник класса Creature. Содержит алгоритм преследования игрока.

◆ Painter – управляет отрисовкой игровых блоков. Также данный класс производит загрузку изображения, его парсинг, создание текстуры из сображения.

◆ Класс Game имеет методы для передачи управления объектам Enemy и Player по таймеру. Инициализирует данные, необходимые при старте игры, производит рестарт уровня в случае проигрыша.

◆ Field — производит загрузку необходимого уровня, управляет генерацией карты.

Графически отношения между классами программы представлены на структурной схеме (приложение А) и диаграмме классов (приложение Б).

2.2 *Краткое описание деление проекта на отдельные модули*

Для упрощения процесса разработки программа была разделена на классы, которые представляют собой законченные сущности. Для создания общего кода для классов Enemy и Player было применено наследование от

абстрактного класса Creature. Это помогло создать более ясную и удобную структуру программы.Для улучшения восприятия кода и его более удобной модификации, многие фрагменты кода были вынесены в отдельные функции.

Исходные коды проекта содержит описания каждого класса в отдельных файлах. Каждый класс был разбит на два файла — заголовочный файл (.hpp) и файл и основной (.cpp). Комманды для сборки находятся в файле Makefile. Такая разбивка улучшает читабельность кода и помогает пересобирать при компиляции только изменившиеся модули.

2.3 *Краткое описание сторонних программных компонент*

Для упрощения разработки была использована программа lode_decode[1].

Она декодирует файл с описанием уровней в ASCII формат. Данная программа была взята с github репозитория пользователя evgenybaskakov и изменена для лучшей совместимости с игрой.

ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данной главе будут представлены строение программы и её основные методы.

Одной из задач при написании приложения было создание программных абстракций, каждая из которых выполняет строго определенную функцию. Для этой цели было найдено решение в виде инкапсуляции данных в отдельные классы и их методы. Так же было использовано наследование для унификации кода. Это помогло существенно увеличить наглядность исходных кодов, уменьшить время, потраченное на отладку программы.

3.1 *Описание методов классов программы*

В функции main файла main файла main.cpp происходит инициализация среды и создание окна. Функции timer, enemy_timer управляют соответственно действиями героя и врагов по соответствующим им таймерам. Функции keyEvent служат для привязки действий к нажатию соответствующих кнопок. Функция display производит отрисовку и смену буферов, хранящих данные об изображении. Объекты классов Painter и Game создаются при старте программы.

3.1.1 *Описание работы класса Game*

Класс, управляющий всеми движужимися объектами. Содержит в себе объекты классов Field, Enemy, Player. Отвечает за передачу объектам событий от таймера, для этого служат методы enemy_tick и tick — для врагов и игрока соответственно.

При вызове конструктора класса Game происходит создание объектов Enemy и помещение их в вектор. Это требуется для проверки, находтится ли враг в ловушке.

Методы keyEvent и draw вызывают одноименные функции у объектов классов Player и Field.

3.1.2 *Описание работы класса Field*

Класс Field при вызове конструктора производит парсинг файла с картой и формирование соответствующего двумерного массива. Метод draw выполняет отрисовку всех элементов карты. Методы setBlock, getBlock, соответственно позволяют получать и изменять элементы массива — карты уровня. Функция drawBlock производит отрисовку выбранного блока.

3.1.3 *Описание работы класса Creature*

Этот класс является абстрактным родителем классов Player и Enemy. Содежит методы для установки и получения текущих, прошлых и возможных координат для объекта, также содержит информацию о блоках, находящихся на этих координатах. Также класс содержит догику поведения движущихся объектов(проверка для запрета прыжков, тест на пребывание в падении, тест на возможность перемещения на клетку и дальнейшее перемещение при удаче).

3.1.4 *Описание работы класса Player*

Абстракция, представляющая собой нашего виртуального игрока. Проверка состояния ловушек выполняется во время хода игрока. Для этого он получает ссылку на вектор врагов. Класс Player содержит методы setTrap, deleteTrap, checkTraps для установки, удаления и проверки ловушек соответственно.

Функция tick отвечает за ход игрока. В течении его хода могут происходить проверка ловушек, перемещение игрока, установка ловушек.

3.1.5 *Описание работы класса Enemy*

Класс отвечает за логику поведения врагов. Функция tick управляет логикой перемещения врага. Функция playerCatched проверяет, удалось ли врагу догнать игрока. При удаче управление передается объекту класса Game, и происходит рестарт уровня.

3.1.6 *Описание работы класса Painter*

Класс Painter управляет отрисовкой игрового поля. Для этого у него имеются следующие методы: imageLoad, loadGLtextures, square.

Метод imageLoad производит загрузку и парсинг 24битного bmp файла с целью получения из него информации о пикселях. Полученные данные будут использоваться при создании текстур.

LoadGLtextures производит создание текстуры из данных, полученных с помощью imageLoad.

Square производит отрисовку квадрата с добавлением поверх него соответствующего спрайта из текстуры.

РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 *Необходимые теоретические данные для разработки модулей*

При разработке программных модулей огромный вклад внес разбор исходных кодов небольших проектов на opengl[5]. Полученные знания помогли с пониманием основ работы с opengl, а также помогли с выбором начальной архитектуры проекта. Помощь в написании bitmap парсера оказала Википедия. С добавлением тестур в игру помогли материалы с сайта gamedev.net[6][7]. Информация по работе различных функиций opengl и freeglut бралась из оффициальной документации.

4.2 Описание работы некоторых ключевых алгоритмов

4.2.1 Описание работы алгоритма проверки ловушек

Проверка состояния установленных ловушек осуществляется в методе checkTraps класса Player.

Функия получает ссылку на объект класса Field для модификации блоков и ссылку на вектор с объектами класса Enemy для модификации состояния врагов.

Метод проверяет все имеющиеся ловушки. Закрывшаяся ловушка получает значение timeRemain = -1 и убирается с игрового поля. Оставшиеся ловушки с каждым ходом теряют значение timeRemain.

В момент timeRemain == 0 происходит закрытие ловушки. Если в ней до этого находился враг, то он появляется сверху закрытой ловушки. Если же там находился игрок, то происходит конец игры и функция checkTraps возвращает false. В методе tick происходит проверка возвращенного функцией значения. В случае смерти игрока в ловушке управление передается классу Game, который производит сброс данных об уровне и повторную генерацию уровня.

Данные о каждой ловушке хранятся в структурах digData. В структуре находятся оставшиеся время жизни timeRemain ловушки, координаты ловушки и тип блока, стоявшего на месте ловушки.

Алгоритм работы метода checkTraps представлен на блок схеме (приложение В).

4.2.2 Описание работы алгоритма установки ловушек

Установкой ловушек управляет класс Player. Установка производится в методе SetTrap. В качестве параметра метод принимает ссылку на объект класса Field.

Метод проверяет, является ли целевой блок кирпичным и не имеет ли он над собой лестницу. В случае нарушения одного из этих условий происходит немедленный возврат из функции.

Далее идет поиск свободной (с истекшим timeRemain) ловушки. В случае неудачи — возврат из функции.

Происходит инициализация координат ловушки, запоминание типа блока, на котором ее построили и установка времени жизни ловушки в максимальное значение.

4.2.3 Описание работы алгоритма проверки состояния игры, используя таймер игрока

Происходит проверка оставшегося количества золота. Если все золото собрано, то на поле появляются некоторые отсутсвовавшие до этого блоки: кирпичные, лестницы. С помощью этих блоков игрок может взобраться на верх лестницы, идущей на следующий уровень.

Идет передача управления игроку. Если функция, отвечающая за ход игрока возвращает ложь, значит игрок перешел на следующий уровень либо погиб. В таком случае сбрасываются все блоки памяти, хранящие информацию о состоянии текущего уровня и вызывается конструктур класса Field для новой генерации уровня. Разница между смертью игрока и завершением уровня заключается в увеличении значения переменной level класса Field при достижении конца уровня. По значению переменной level Field выбирает, какой уровень генерировать.

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

5.1 Базовые сведения по установке и запуску программы

Данное приложение является представляет собой клон популярной классической игры Lode Runner.

Рекомендуемые системные требования:

- Операционная система Gnu/Linux
- 64 МБ ОЗУ
- Разрешение экрана не ниже 1040x640
- Не менее 5 МБ свободного места на жестком диске
- Клавиатура

Для сборки из исходных кодов потребуются:

- Компилятор g++ не ниже версии 4.7.4
- Наличие следующих библиотек:
    - freeglut не ниже 2.5.2 версии
    - glu не ниже 7.2.2 версии

На данном этапе развития игра распространяется исключительно в виде исходных кодов. Получить их можно с репозитория steal_n_run на github.com. Форк программы lode_decode, который необходим для корректной работы игры необходимо скачать и поместить в папку с игрой.

Для установки следует собрать lode_decode и steal_n_run используя команду make. Makefile для lode_decode находится в lode_decode/. Для steal_n_run Makefile находится в steal_n_run/src.

5.2 Обзор основных элементов интерфейса программы

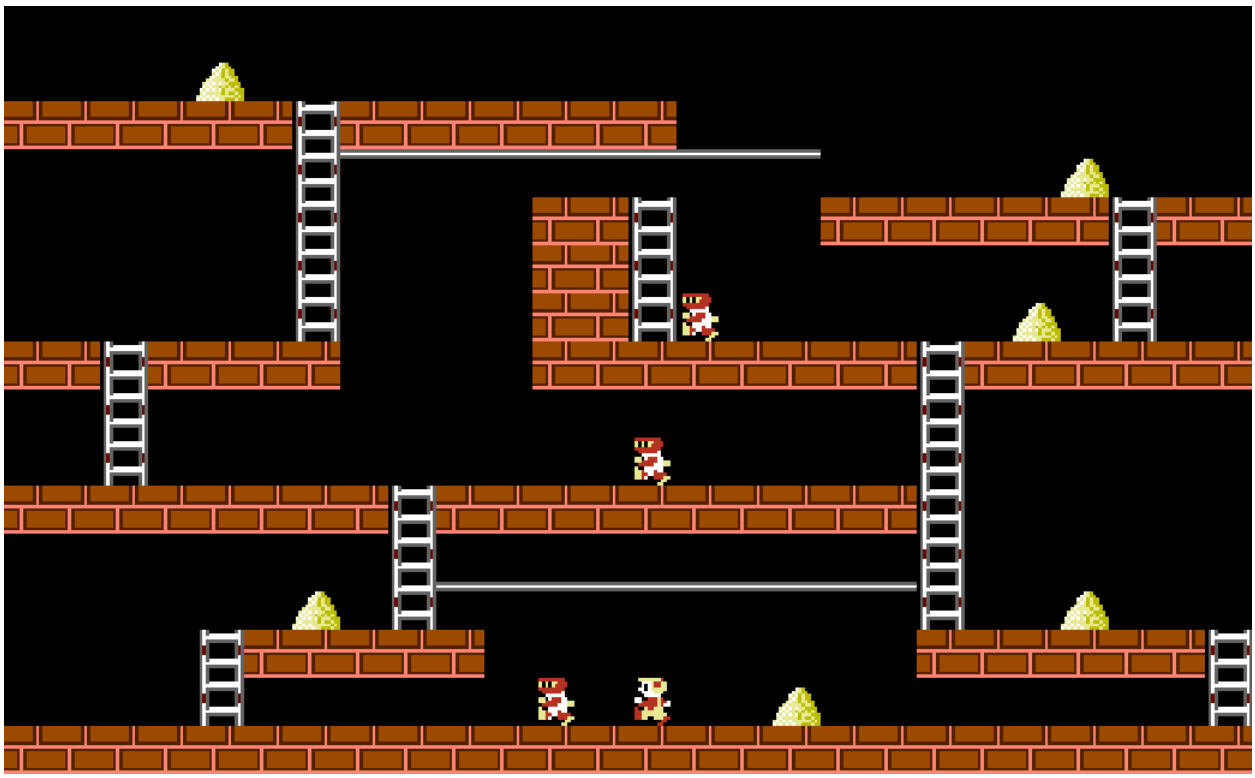

рисунок 3 — второй уровень. Начало уровня

При запуске приложения, происходит запуск игры с первого уровня. Для управления игроком используются клавиши со стрелками и кнопки z, x для рытья ямы слева или справа от игрока.

В отличии от оригинальной игры в steal_n_run логика работы ям(ловушек) имеет несколько отличий.

Клетка, на которой вырыта яма, считается пустой. Если под ней нет других занятых клеток или нижней границы экрана, то существа продолжают падение дальше. Если снизу есть кирпичный или бетонный блок, то объект застревает в яме. Если игрок оказался в яме, то, при ее закрытии, он умирает и происходит рестарт уровня. При закрытии ямы, в которой находится враг, враг удачно выбирается наружу.

Также в игре существуют ограничения на количество одновременно открытых ловушек.

Количество попыток прохождения уровня — неограниченно. После каждой смерти, игра начинается с текущего уровня.

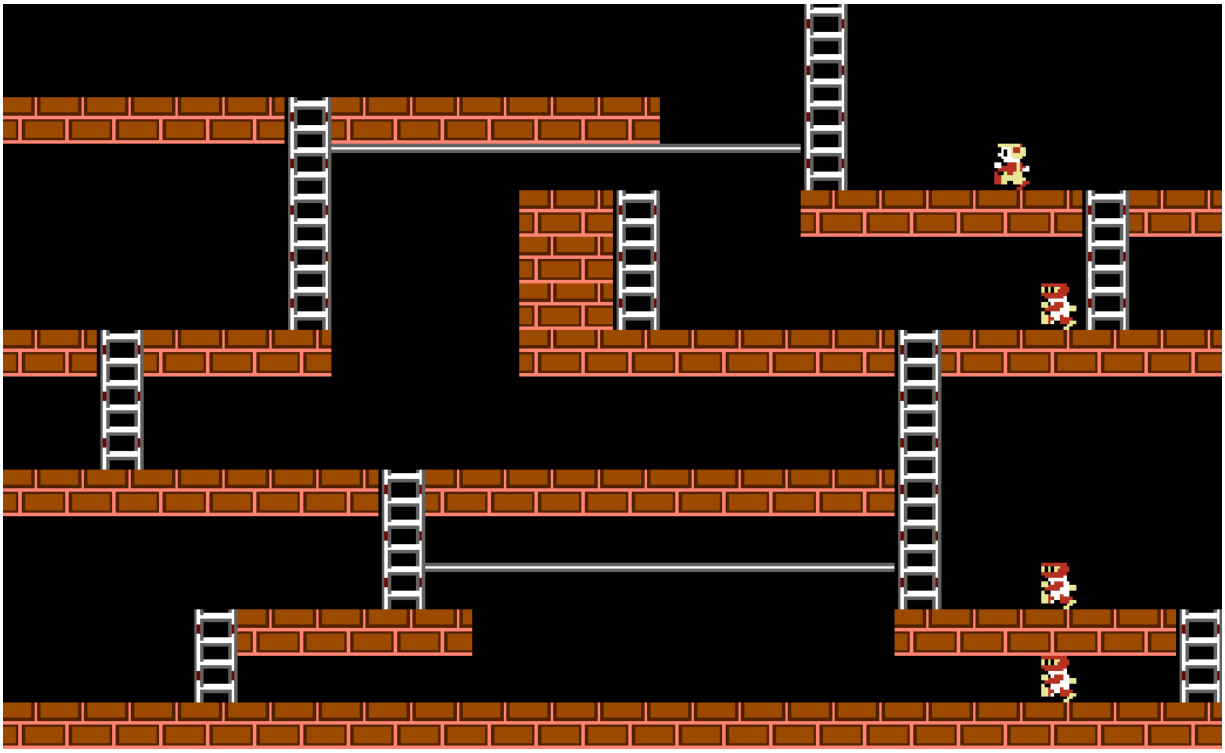

рисунок 4 — лестница на второй уровень

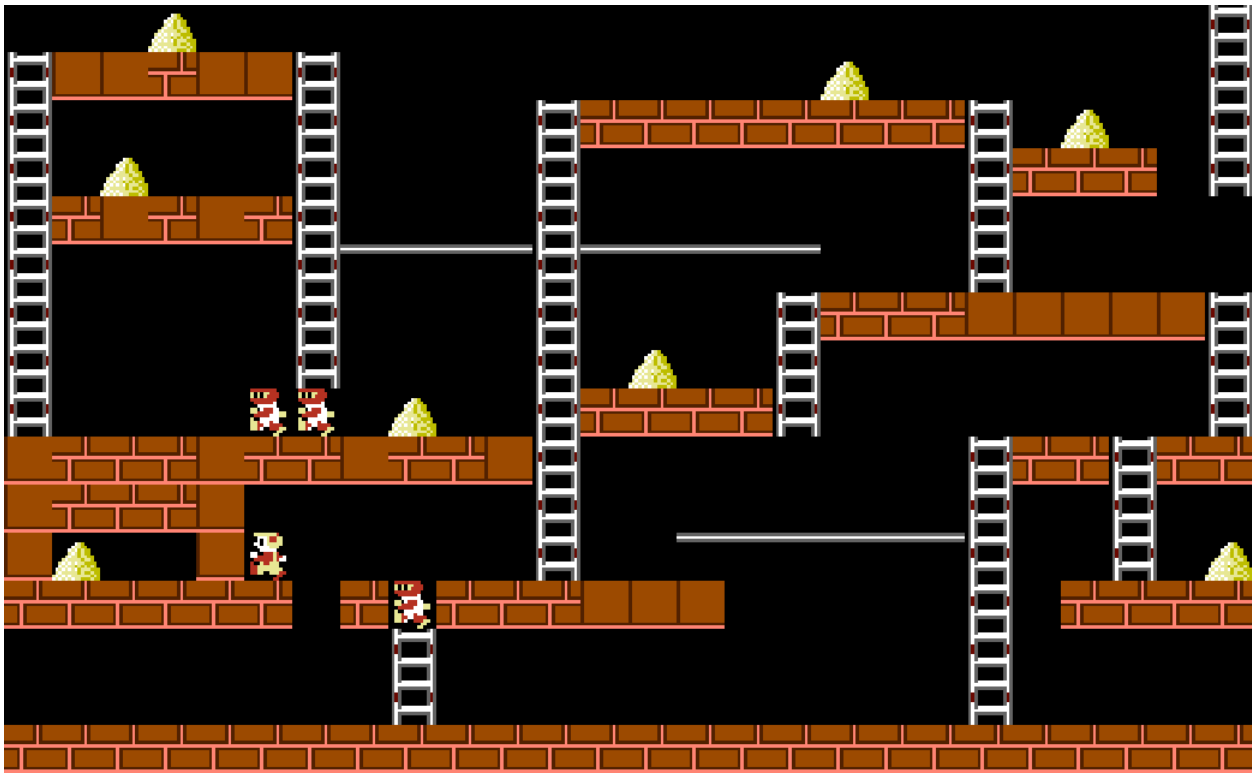

рисунок 5 — второй уровень. Справа от игрока – новая яма

В целях отладки и предоставления пользователю возможности просмотра логов игры, в консоли отображается информация о текстуре. В случае ошибки загрузки текстуры(например отсутствие файла с изображением) в консоль будет выведено соответствующее сообщение об ошибке.

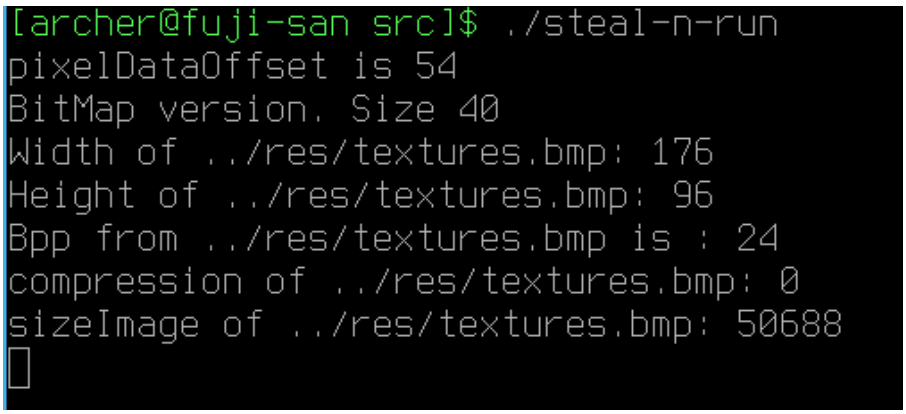

рисунок 6— логи игры

ТЕСТИРОВАНИЕ

При успешном запуске игры пользователь увидит окно с первым уровнем. В случае ошибки, причину можно найти по логам, представленным на рисунке 6.

В ходе разработки было выявлено огромное множество проблем, связанных с геймплеем. Многие из этих проблем были решены полностью, для решения других пришлось немного изменить логику игры. Например, существует проблема пересечения траекторий врагов. Проблема проявляется в создании фальшивого блока на месте пересечения их координат. На данном этапе разработки проблема была решена запретом врагов опережать друг друга. Полное решение проблемы будет построено после перестройки логики перемещения на попиксельную.

ЗАКЛЮЧЕНИЕ

На момент сдачи записки в игре работает базовый геймплей и имеются основные текстуры. Игра имеет простой и удобный интерфейс. В игре есть доступ к большой базе уровней.

На момент сдачи не удалось наделить ботов алгоритмами для поиска оптимального пути. Также не удалось выполнить попиксельное передвижение объектов. Это связано с нехваткой времени на разработку проекта и трудностями, возникшими при освоении opengl.

В дальнейшем в программе будут решены описанные выше проблемы, а также будут добавлены много нововведений:

- главное меню
- сохранение/загрузка
- анимация и звук
- новые текстуры
- кооператив
- очки и бонусы
- поддержка контроллеров
- поддержка новых платформ

Полученный в ходе курсового проектирования опыт пригодится в дальньнейшем при работе с графикой, написании игр.

Полученный проект может стать основой для создания достойного конкурента существующим играм Lode Runner. Этого можно достичь внедрением мультиплеерного режима. Для этого могут прегодится знания по работе с сетью, полученные в ходе прошлого курсового проекта — FTP клиента.

СПИСОК ИСТОЧНИКОВ

1. badabum007/lode_decode: Original LODE RUNNER level decoder [Электронный ресурс] – Электронные данные. – Режим доступа: https://github.com/badabum007/lode_decode
2. badabum007/steal_n_run: Simple "Lode runner" like game, written in C++, using openGL [Электронный ресурс] – Электронные данные. – Режим доступа: https://github.com/badabum007/steal-n-run
3. Lode Runner - Википедия [Электронный ресурс] – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Lode_Runner
4. BMP - Википедия [Электронный ресурс] – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/BMP
5. Writting Snake game in 10 minutes [Электронный ресурс] – Электронные данные. – Режим доступа: https://www.youtube.com/watch?v=GiZGEFBGgKU
6. GameDev.net Community Forums [Электронный ресурс] – Электронные данные. – Режим доступа: http://www.gamedev.net/page/resources/_/technical/opengl/rendering-efficient-2d-sprites-in-opengl-using-r2429
7. NeHe Productions: Texture Mapping [Электронный ресурс] – Электронные данные. – Режим доступа: http://nehe.gamedev.net/tutorial/texture_mapping/12038/