Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Системное программное обеспечение вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ЭМУЛЯТОР ИГРОВОЙ ПРИСТАВКИ DENDY

БГУИР КП 1–40 02 01 01 423 ПЗ

Студент А.Ф. Полховский

Руководитель А.А. Климец

МИНСК 2021

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ………………………………………………………………………..5

1 ОБЗОР ИСТОЧНИКОВ………………………………………………………...7

1.1 Виды эмуляции………………………………………………………...7

1.2 Постановка задачи…………………….……………………………….7

1.3 Структурные блоки эмулятора………………………………………..7

1.3.1 Процессор……………………………………………………..8

1.3.2 Архитектура NES…………………………………………….11

1.3.3 Устройство обработки картинок (PPU)…………………….11

1.3.4 Реализация скроллинга………………………………………14

2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ……………………………………...15

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ……………………………….16

3.1 Описание функционирования программы…………………………..16

3.2 Описание основных структур………………………………………..16

3.2.1 Структура «Sprite»………………………………………...…16

3.2.2 Структура «Sprites» ………………………………………….16

3.2.3 Структура «Semaphore»……………………………………...17

3.2.4 Структура «Map»…………………………………………….17

3.2.5 Структура «MapItem».……………………………………….17

3.2.6 Структура «Config»…………………………………………..18

3.2.7 Структура «PlayerControls»………………………………….18

3.2.8 Структура «Bomberman»…………………………………….19

3.2.9 Структура «Bomb»…………………………………………...20

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ……………………………...21

4.1 Функция загрузки рисуемой формы………………………………....21

4.2 Функции упорядоченного вывода в консоль..………………………21

4.3 Функция создания игровой бомбы………………………………......22

4.4 Функция, имитирующая разрушающий эффект от бомбы………...22

4.5 Функция обновляющая таймер бомбы………………………………22

4.6 Функция обновления бомб на карте…………………………………23

4.7 Функция отрисовки поля……………………………………………..23

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ…………………………………………25

ЗАКЛЮЧЕНИЕ………………………………………………………………….28

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ……………………………29

ПРИЛОЖЕНИЕ А……………………………………………………………….30

ПРИЛОЖЕНИЕ Б………………………………………………………………..31

ПРИЛОЖЕНИЕ В………………………………………………………………..32

ПРИЛОЖЕНИЕ Г………………………………………………………………..33

**ВВЕДЕНИЕ**

Эмулятор ­— это компьютер или программа, которая эмулирует или имитирует другой компьютер или программу. Типичным случаем использования эмулятора является запуск старых программ на современных цифровых устройствах. Часто эмуляторы используются для запуска старых игр с их помощью на новых системах, запуска игр, переведенных на языки, для которых не существует официальных версий, для модификации существующих игр, эмуляции работы операционных систем, эмуляции работы различных цифровых устройств. Эмулятор может быть очень полезным инструментом при создании программ для старых систем или же при создании пользовательских игр.

[Эмуляция](https://ru.wikipedia.org/wiki/%D0%AD%D0%BC%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F) — ресурсоёмкая задача, и может требовать от устройства намного большей производительности (скорости [процессора](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80), объёма оперативной [памяти](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BC%D1%8F%D1%82%D1%8C_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F))), чем производительность эмулируемой игровой приставки. Причем, чем сложнее система и выше точность эмуляции — тем большая производительность для неё требуется. От точности эмуляции зависит совместимость с запускаемыми в эмуляторе программами (играми) — например, если эмулятор не учитывает какой-то нюанс устройства системы, используемый в некотором количестве игр — эти игры не будут запускаться, или будут работать неправильно. Также от точности эмуляции зависит качество и достоверность получаемого изображения и звука.

Эмуляция тесно связана с концепцией виртуализации.

Виртуальные машины — это тип эмулятора, который может работать на базовом оборудовании хост-системы. Следовательно, потери производительности значительно меньше, но, в то же время, виртуальные машины могут быть ограничены в том, что они могут делать, по сравнению с исходной машиной.

Существуют различные типы эмуляторов, в которых используются различные методы эмуляции, но конечная цель всегда одна и та же: воспроизвести опыт использования оригинального оборудования или программного обеспечения. Иногда, некоторые эмуляторы даже превосходят производительность оригинального продукта и включают в себя дополнительные функции.

Создание эмулятора — сложная задача, требующая опыта программирования и знаний точной информации об устройстве эмулируемой системы. Поэтому в открытом доступе эмуляторы одной и той же системы могут различаться как в достоверности эмуляции, так и в требованиях к компьютеру/устройству.

Nintendo Entertainment System (NES) — восьмибитная игровая приставка, выпущенная 1983 году изначально в Японии, а в последствии и на территории Европы и США. Благодаря удачным аппаратным характеристикам и богатой библиотеке игр консоль снискала небывалый успех, фактически возродив игровой рынок после так называемого «кризиса» игровой индустрии. С начала 1990-х годов конкуренция со стороны технически более совершенных систем, таких как 16-битная Sega Mega Drive, положила конец господству NES.

Dendy, или Де́нди — игровая приставка, неофициальный аппаратный клон консоли третьего поколения Famicom (Nintendo Entertainment System).

Dendy выпускалась с конца [1992 года](https://ru.wikipedia.org/wiki/1992_%D0%B3%D0%BE%D0%B4) компанией [Steepler](https://ru.wikipedia.org/wiki/Steepler" \o "Steepler), собиралась на Тайване из [китайских](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D1%82%D0%B0%D0%B9%D1%81%D0%BA%D0%B0%D1%8F_%D0%9D%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%A0%D0%B5%D1%81%D0%BF%D1%83%D0%B1%D0%BB%D0%B8%D0%BA%D0%B0) комплектующих по заказу Steepler и была распространена в республиках [бывшего СССР](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%82%D1%81%D0%BE%D0%B2%D0%B5%D1%82%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE), особенно в [России](https://ru.wikipedia.org/wiki/%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D1%8F), [Украине](https://ru.wikipedia.org/wiki/%D0%A3%D0%BA%D1%80%D0%B0%D0%B8%D0%BD%D0%B0) и [Казахстане](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%B7%D0%B0%D1%85%D1%81%D1%82%D0%B0%D0%BD). Поскольку на постсоветском пространстве Famicom или NES официально никогда не продавались, Dendy, которая была широко распространена по доступной цене, снискала большую популярность.

Из-за такой доступности, название Dendy в бывшем СССР фактически стало синонимом Nintendo Entertainment System и Famicom. Даже приставки «Кенга» и «Сюбор», совместимые и появившиеся позже, [называли не иначе как «Денди»](https://ru.wikipedia.org/wiki/%D0%AD%D0%BF%D0%BE%D0%BD%D0%B8%D0%BC).

В курсовом проекте для реализации собственного эмулятора Dendy используется языки C/C++.

Помимо стандартных библиотек в программе используется библиотека SDL (Simple DirectMedia Layer), реализующая единый программный интерфейс к графической подсистеме, звуковым устройствам и средствам ввода.

**1 ОБЗОР ЛИТЕРАТУРЫ**

* 1. **Виды эмуляции**

Интерпретация. Интерпретатор дает возможность программно эмулировать все части консоли. Данный способ простой и вместе с тем наименее производителный**.**

Динамическая Рекомпиляция. Рекомпилятор довольно сложен в написании, но при этом значительно превосходит интерпретатор в производительности. Данный эмулятор может рекомпилировать инструкции эмулируемой машины в машинные инструкции вашего ПК. Проще говоря, рекомпилятор выступает в роли переводчика с одного машинного языка на другой.

В целях здравого смысла стоит выбрать способ интерпретации, в связи

со сложностью написания данной программы

* 1. **Постановка задачи**

Задача заключается в создании эмулятора NES для операционной системы Linux, который будет запускать желаемые образы игры и предоставлять возможность игрового взаимодействия.

В частности, это утверждает две главные задачи:

- реализовать компоненты приставки для запуска с помощью эмулятора большинства доступных игр в достаточном для комфортного игрового процесса виде;

- графически отобразить генерируемое эмулятором изображение и обеспечить пользовательский ввод;

**1.3 Структурные блоки эмулятора**

Создание эмулятора любого сложного устройства подразумевает отдельную эмуляцию каждого из его компонентов и дальнейшую настройку их взаимодействия друг с другом. Однако, перед тем как детально взглянуть на каждый компонент в отдельности, стоит обсудить общую архитектуру консоли.

В упрощенном виде, взаимодействие всех этих элементов заключается

в следующем: картридж, который содержит программу и игровую графику, подключается к шине. Процессор получает доступ к программе, а видеопроцессор к графическим элементам.

Обмениваясь через шину процессора данными, видеопроцессор спустя определенное количество тактов генерирует кадр, который впоследствии отображается на устройстве вывода.

Далее, рассмотрим все по отдельности.

**1.3.1 Процессор**

Процессор – «сердце» любого вычислительного устройства. В NES используется модифицированная версия процессора MOS Technology 6502 (далее – 6502). Эмуляция процессора довольно независима, потому его реализация в этом проекте потенциально может использоваться и в эмуляции других устройств (рисунок 1.1).



Рисунок 1.1 – Процессор

Шина позволяет процессору адресовать память через адреса. Адреса состоят уже не из восьми бит, как данные, а из 16, что позволяет адресовать 64 килобайта памяти (рисунок 1.2).

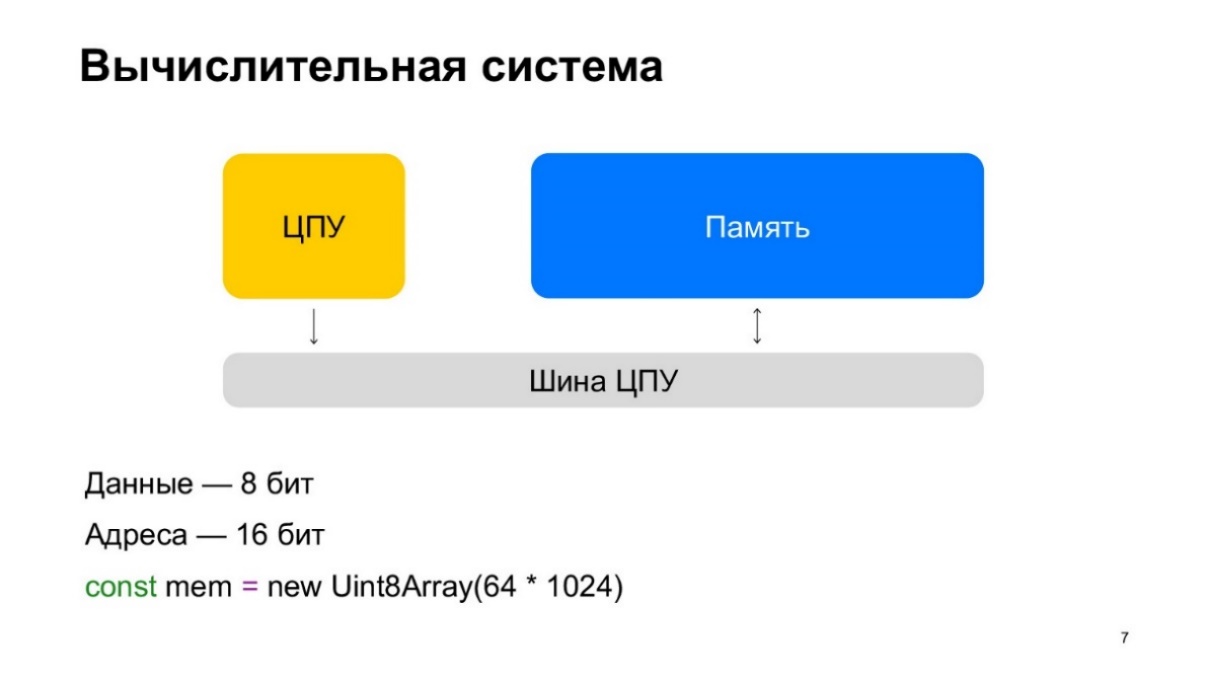


Рисунок 1.2 – Вычислительная система

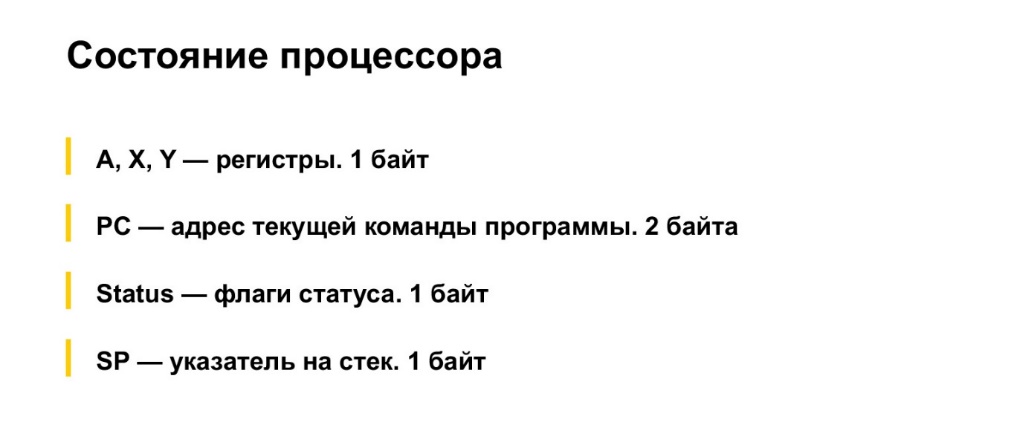


Рисунок 1.3 – Состояние процессора

В процессоре находится некое состояние (рисунок 1.3), тут есть три регистра — A, X, Y. Регистр — это как бы хранилище для промежуточных значений. Размер регистра — один байт или восемь бит. Это говорит о том, что процессор восьмибитный, он оперирует восьмибитными данными.  
Функционально эти регистры довольно независимые — их можно использовать как общие. Но у них появляется смысл, как, например, сложение, в регистре A получается результат и берется значение первого операнда.  
Что еще входит в состояние процессора? Есть регистр PC, который указывает на адрес текущей команды, поскольку адрес — это два байта.

Еще есть регистр Status, который указывает на флаги статуса. К примеру, если вычесть два значения и получилось отрицательное, то зажигается определенный бит в регистре флага.  
Наконец, есть SP, указатель на стек. Стек — это просто обычная память, она никак не отделена от всего остального, от всех остальных программ. Просто есть инструкция у процессора, который управляет этим указателем SP. Таким образом реализован стек.

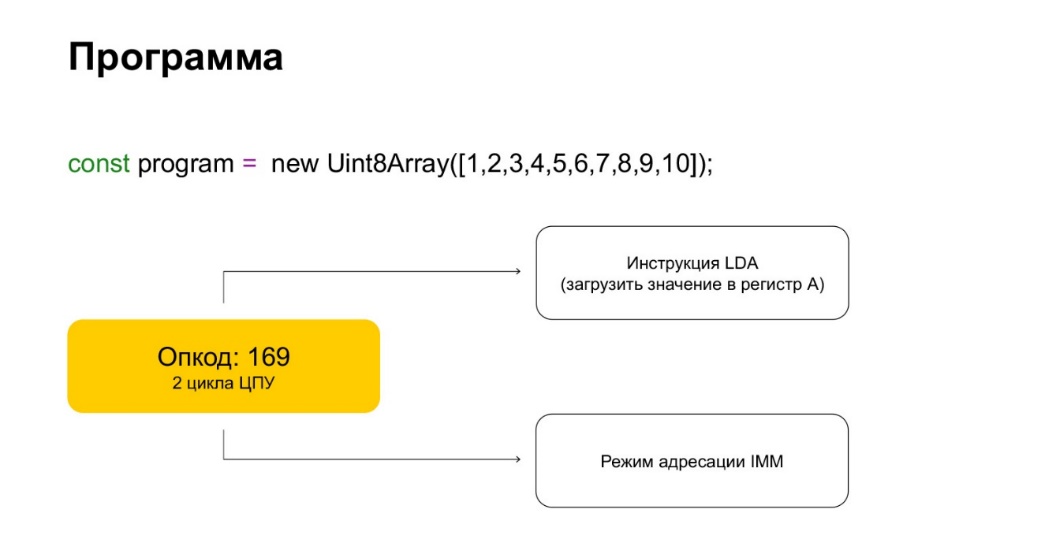


Рисунок 1.4 – Программа

Посмотрим, что же представляет собой такая программа (рисунок 1.4). Это некая последовательность байт. Она даже не обязательно должна быть последовательной. Сама программа может находиться в разных участках памяти. Можно представить программу — 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Это реальная программа на 6502. Каждый байт этой программы, каждая цифра в этом массиве представляет собой такую сущность, как опкод. Опкод — код операции. «то, опять же, обычное число.  
Например, есть опкод 169. Он кодирует в себе — во-первых, инструкцию. Инструкция при выполнении меняет состояние процессора, памяти и так далее, то есть состояние системы (рисунок 1.5).



Рисунок 1.5 – Архитектура Фон Неймана

Итак, процессор 6502 имеет:

- 8 регистров

- 3 прерывания

- 8 флагов

- 56 инструкций и 12 различных режимов адресации

- адресное пространство в 64 кБ (включая 2 кБ оперативной памяти (RAM), «отзеркаленной» до 8 кБ).

В таблице 1.1 приведена архитектура адресного пространства процессора 6502.

*Таблица 1.1.* *Адресное пространство процессора 6502*

|  |  |  |
| --- | --- | --- |
| **Адрес** | **Размер** | **Назначение** |
| $0000-$07FF | $0800 | 2 КБ внутренней оперативной памяти |
| $0800-$0FFF | $0800 | Зеркальное адресное пространство RAM |
| $1000-$17FF | $0800 |
| $1800-$1FFF | $0800 |

*Продолжение таблицы 1.1.*

|  |  |  |
| --- | --- | --- |
| $2000-$2007 | $0008 | Регистры PPU |
| $2008-$3FFF | $1FF8 | Зеркальное адресное пространство регистров PPU |
| $4000-$4017 | $0018 | Регистры APU |
| $4018-$401F | $0008 | Обычно не используется |
| $4020-$FFFF | $BFE0 | Пространство картриджа |

**1.3.2 Архитектура NES**

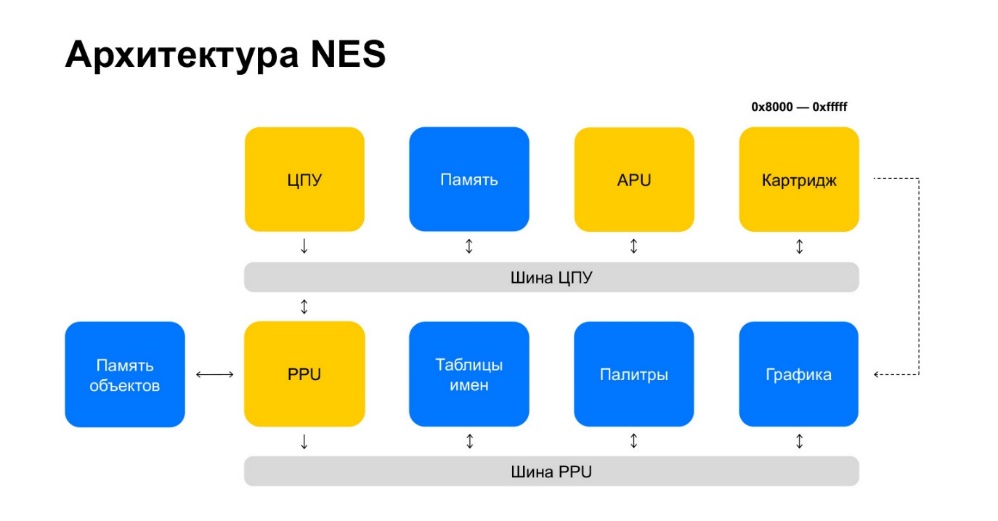


Рисунок 1.6 – Архитектура NES

Имеется CPU и его шина, как обычно. Есть два дополнительных килобайта памяти. Есть APU — устройство вывода звука. И есть картридж. Он помещен в верхний диапазон и поставляет данные о программе. Еще он поставляет данные графики, сейчас рассмотрим. Последняя штука, которая есть на шине CPU, — это PPU, picture processing unit, такая протовидеокарта. У PPU тоже есть своя шина, на которую смещаются таблицы имен, палитры и графические данные. Но графические данные поставляет картридж. Есть еще память объекта (рисунок 1.6).

**1.3.3 Устройство обработки картинок (PPU)**

.

Рисунок 1.7 – Устройство обработки картинок

Рассмотрим, как же работает устройство вывода картинок (рисунок 1.7). Потому что, если мы хотим сделать эмулятор, программа-минимум — сделать процессор и вот эту штуку, чтобы смотреть, как картинки и видеоигры выглядят.  
 Начнем с самой верхне-уровневой сущности — самой картинки. Она состоит из двух планов. Есть передний план, на котором размещаются более динамические сущности, и задний, где размещаются более статичные сущности вроде сцены. То есть передний план существует для более динамичных сущностей, а задний — для более статичных.

Картинка на растровом дисплее состоит из пикселей. Пиксели — это просто цветные точки. Как минимум нужны цвета. В NES есть системная палитра. Она состоит из 64 цветов, это, к сожалению, все цвета, которые NES способна воспроизводить. Но нельзя взять любой цвет из палитры. Для пользовательских палитр есть определенный диапазон в памяти, который, в свою очередь, тоже разбит на два таких поддиапазона.  
 Есть диапазон заднего и переднего плана. Каждый диапазон разбит на четыре палитры по четыре цвета. Например, задний план, нулевая палитра состоит из белого, синего, красного цветов. А четвертый цвет в каждой палитре всегда отсылает к прозрачному цвету, что позволяет нам делать прозрачный пиксель (рисунок 1.8).

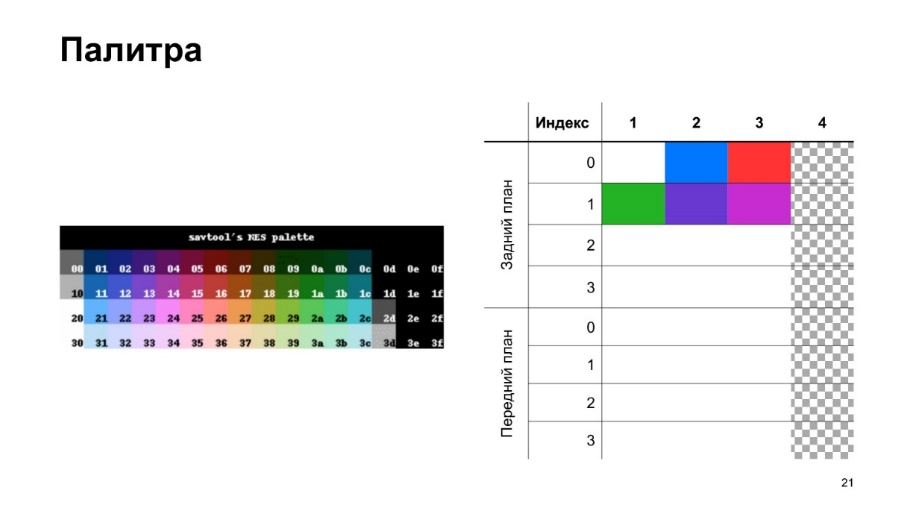


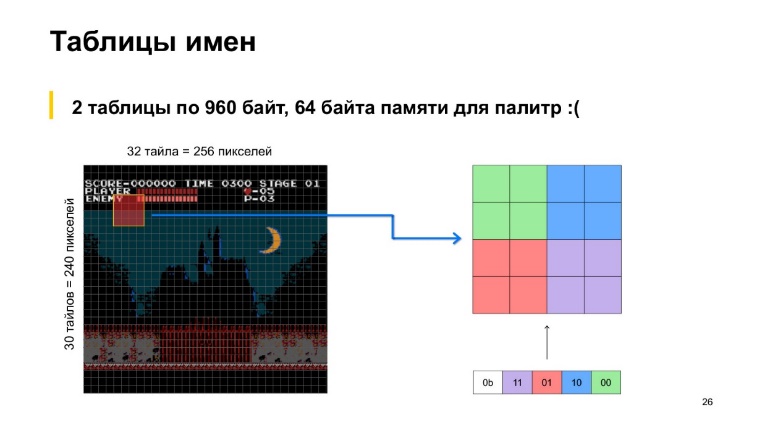
Рисунок 1.8 – Палитра цветов NES  


Рисунок 1.9 – Таблицы имен

Имеется таблица имен (рисунок 1.9). Имеется две, каждая по 960 байт, каждый байт отсылает нас к определенному тайлу.

Если представить эти 960 байт как матрицу, получится экран 32 на 30 тайлов. Разрешение NES будет 256 пикселей на 240 пикселей.  
Можно записать туда тайлы. Но как возможно заметить, тайлы не указывают палитру, с которой они должны отображаться.

Можно разные тайлы отображать с разными палитрами, и тоже нужно где-то хранить эту информацию. Но, к сожалению, всего 64 байта на одну таблицу имен для хранения информации о палитре.

И тут возникает проблема. Если разбить таблицу дальше, чтобы там было всего 64 значения, то получим квадраты четыре на четыре тайла, которые выглядят, мягко говоря, как квадратики, просто огромные порции экрана. Она была бы подчинена одной палитре, если бы не одно но: в подпалитре есть четыре палитры, и нам нужно всего два бита для указания нужной. Каждый этот байт из 64-х копирует информацию о палитре для сетки четыре на четыре. Но эта сетка разбивается на подсетки два на два.

То есть сетка два на два тайла привязана к одной палитре. Таковы ограничения в мире отображения фонов на Nintendo.

**1.3.4 Реализация скроллинга**

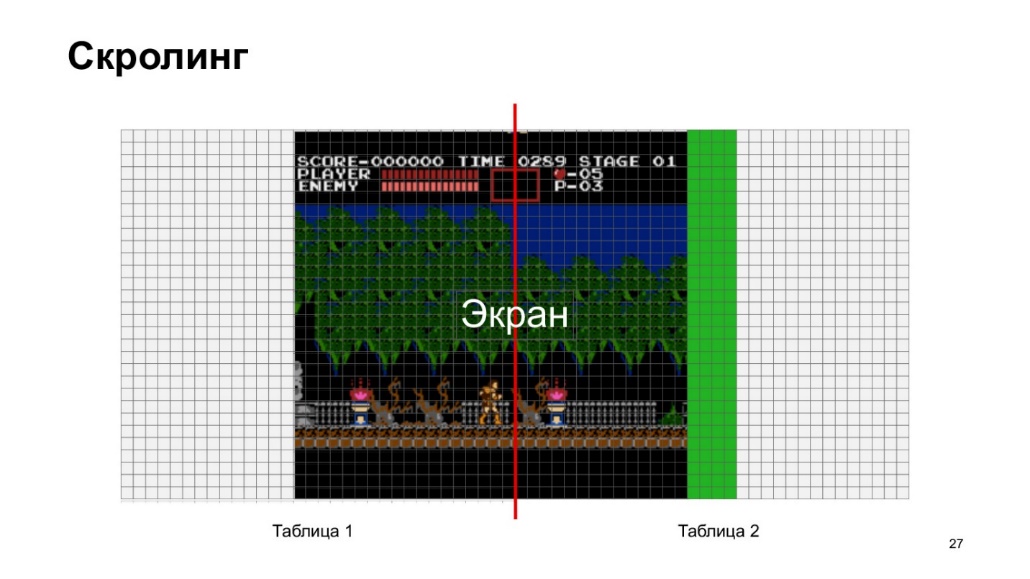
. 

Рисунок 1.10 – Скроллинг

Если вспомнить, например, «Марио» или Castlevania, то известно: если в этих играх герой будет двигаться направо, то мир как бы раскрывается вдоль экрана. Это сделано за счет скроллинга (рисунок 1.10).

Вспомним, что есть две таблицы имен, которые уже кодируют два экрана. И когда герой движется, мы как бы дозаполняем данные в таблицу имен, которые следуют далее.

Прямо на лету, когда герой движется, дозаполняемм таблицу имен.

Получается, можно указать, с какого тайла в таблице имен нужно начать отображать данные, и как мы будем их разворачивать по полоскам.

Весь фокус скроллинга — чтение из двух таблиц имен.  
То есть если выходить за одну таблицу имен по горизонтали, то начинаем автоматически читать из другой и т. д.

Но, важно, нельзя забывать заполнять данные.

# **2** **СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

В программе можно выделить шесть основных элементов: подготовительный блок, блок загрузки конфигурации, блок меню, блок игрового процесса, блок очистки памяти.

В подготовительном блоке создается окно ncurses, настраивается режим работы терминала, инициализируется файл логгера, куда будут записываться действия пользователей для обработки нажатых клавиш, инициализируется структура консоли для правильной отрисовки по длине и ширине.

В блоке загрузки конфигурации из текстовых файлов в структуру рисуемых объектов выгружаются формы из аски-символов, а также загружается конфигурация игровых модификаций и управления.

В блоке меню производится отрисовка меню (обновляется через каждые промежутки времени t, t > 1/3 секунды), обрабатывается нажатия клавиш пользователем для выбора количества живых игроков. При нажатии клавиши ввода, вызывается функция начала игры, при нажатии клавиши выхода, вызываем функцию очистки.

Далее, в блоке игрового процесса генерируется случайное поле с игроками на нем, после чего оно отрисовывается, в цикле обрабатывается нажатия клавиш, поле обновляется, действия повторяются. При нажатии клавиши выхода, переходим в меню, завершаем процессы и переходим в меню.

В блоке очистки завершаем процессы игрового меню, очищаем структуру с формами объектов, восстанавливаем терминал в исходное состояние.

Структурная схема программы предоставлена в приложении Б.

**3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

**3.1 Описание функционирования программы**

Структура, представляющая сущность игрового персонажа, Bomberman содержит в себе поле указателя на структуру map, обеспечивающее взаимодействие игрока с картой, поля, указывающие на то, жив ли игрок и может ли игрок установить бомбу, количество и мощность бомб у игрока, его координаты x y, ID, поля отвечающие за ИИ, поле потока.

Структура представляющая сущность игровой бомбы, Bomb, устанавливаемая игровым персонажем, т.е. Bomberman-ом, содержит в себе указатель на Bomberman-а установившего ее, указатель на игровую карту, координаты, а также мощность, или, приобретенную игроком, дальность поражения.

Синхронизация процессов осуществляется с помощью семафоров. В структурах сущностей, требующих синхронизации, имеется поле с указателем на структуру Semaphore, содержащей идентификаторы набора семафоров.

Для отрисовки объектов используются две структуры sprite и sprites, которые заполняются комбинацией аски-символов, читаемых с файлов, так что, первые два числа – ширина w и высота h объекта w\*h отрисованного ниже в этом файле.

**3.2 Описание основных структур**

**3.2.1 Структура «Sprite»**

Данная структура представляет собой структуру для считывания отрисовываемых объектов. Описание представлено в таблице 3.1.

*Таблица 3.1.* *Описание структуры «Sprite»*

|  |  |  |
| --- | --- | --- |
| Основные поля структуры «Sprite» | | |
| Имя | Тип | Описание |
| w | int | длинна объекта в аски-символах |
| h | int | высота объекта в аски-символах |
| data | wchar\_t\* | считываемый с файла объект |

**3.2.2 Структура «Sprites»**

Структура, содержащая в себе массив всех структур Sprite (отрисовываемых объектов), заполняемая функцией Sprites\_load (enum SPRITES name, char\* path), где name – перечисляемый тип объекта, а path – имя файла, с которого нужно считать символы.

**3.2.3 Структура «Semaphore»**

Данная структура сделана для удобства использования, предназначена для синхронизации процессов нашей игры и содержит только одно поле – поле идентификатора набора семафоров – semId.

**3.2.4 Структура «Map»**

Данная структура представляет собой структуру игровой карты. Описание представлено в таблице 3.2.

*Таблица 3.2. Описание структуры «Map»*

|  |  |  |
| --- | --- | --- |
| Основные поля структуры «Map» | | |
| Имя | Тип | Описание |
| w | Int | ширина игрового поля |
| h | Int | высота игрового поля |
| map | struct MapItem\* | указатель на структуру элементов карты |
| bombs | struct Bomb | массив структур сущностей игровых бомб |
| bombsSem | struct Semaphore\* | массив семафоров бомб |
| mapSem | struct Semaphore\* | семафор |
| Bombermans | struct Bomberman | массив сущностей игровых персонажей |

**3.2.5 Структура «MapItem»**

Данная структура представляет собой структуру свойств элементов на карте. Описание представлено в таблице 3.3.

*Таблица 3.3.* *Описание структуры «MapItem»*

|  |  |  |
| --- | --- | --- |
| Основные поля структуры «MapItem» | | |
| Имя | Тип | Описание |
| type | int | тип элемента |
| burnTime | int | время эффекта от взрыва |
| hasPlayer | int | счетчик игроков |
| hasBomb | int | счетчик бомб |
| hasBonus | int | счетчик бонусов |
| blockBombs | int | временное блокирование установки бомбы |

**3.2.6 Cтруктура «Config»**

Данная структура представляет собой структуру конфигурации, включающая таймер взрыва, время эффекта после взрыва и управление. Описание представлено в таблице 3.4.

*Таблица 3.4. Описание структуры «Config»*

|  |  |  |
| --- | --- | --- |
| Основные поля структуры «Config» | | |
| Имя | Тип | Описание |
| bombTimer | int | времени до детонации |
| bombFireTimer | int | таймер детонации |
| bombFireCenterTimer | int | время распределения взрыва |
| controls | struct PlayerControls | массив из структур управления для 4-ых игроков |

**3.2.7 Структура «PlayerControls»**

Данная структура представляет собой структуру управления. Описание представлено в таблице 3.5.

*Таблица 3.5. Описание структуры «PlayerControls»*

|  |  |  |
| --- | --- | --- |
| Основные поля структуры «PlayerControls» | | |
| Имя | Тип | Описание |
| up | int | движение вверх |
| down | int | движение вниз |
| left | int | движение влево |
| right | int | движение вправо |
| bomb | int | установка бомбы |

* + 1. **Структура «Bomberman»**

Данная структура представляет сущность игрока. Описание представлено в таблице 3.6.

*Таблица 3.6. Описание структуры «Bomberman»*

|  |  |  |
| --- | --- | --- |
| Основные поля структуры «Bomberman» | | |
| Имя | Тип | Описание |
| alive | int | флаг, обозначающий, что игрок жив |
| x | int | координата X |
| y | int | координата Y |
| id | int | ID игрока |
| direction | int | направление |
| shouldBomb | int | способность установить бомбу |
| showGrave | int | показывает, что игрок мертв |
| bombCount | int | количество бомб у игрока |
| bombPower | int | количество усилений бомбы игрока |
| map | struct Map\* | игровая карта |
| thread | pthread\_t | поле потока |
| isAi | int | флаг, обозначающий, является ли игрок ИИ |

*Продолжение таблицы 3.6*

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| tickAi | int | время ИИ |
| lastDirection | int | направление ИИ |

**3.2.9 Структура «Bomb»**

Данная структура представляет собой сущность игровой бомбы. Описание представлено в таблице 3.7.

*Таблица 3.7.* *Описание структуры «Bomb»*

|  |  |  |
| --- | --- | --- |
| Основные поля структуры «Bomb» | | |
| Имя | Тип | Описание |
| alive | int | флаг, указывающий на то, установлена ли бомба |
| x | int | координата X |
| y | int | координата Y |
| time | int | время до взрыва |
| power | int | мощность бомбы |
| map | struct Map\* | игровая карта |
| bomberman | struct Bomberman\* | игрок |

**4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

**4.1 Функция загрузки рисуемой формы**

Функция предназначенная для того, чтобы загружать форму объекта, который будет выводиться в консоль. При чем информация в файлах хранится в виде: ширина, высота фигуры, и сама фигура. Данные с файла считываются в соответствующей формате.

struct Sprite\* Sprite\_fromFile(char\* path){

FILE \*fp = fopen(path, "r");

if(fp==0){

return 0;

}

int w,h;

fscanf(fp, "%d %d", &w, &h);

struct Sprite\* spr = Sprite\_init(w,h);

int i = 0;

while(i<w\*h){

wchar\_t c;

fscanf(fp, "%lc", &c);

if(c!=L'\n'){

spr->data[i] = c;

i = i + 1;

}

}

fclose(fp);

return spr;

}

**4.2 Функции упорядоченного вывода форм в консоль**

Функции, которые выводят форму в консоль.

void Sprite\_draw(struct Sprite\* self, int x, int y){

if(self==0)

return;

int xx;

int yy;

for(yy=0; yy<self->h; yy++){

for(xx=0; xx < self->w; xx++){

Console\_drawChar(xx+x,yy+y,Sprite\_get(self,xx,yy));

}

}

}

void Console\_drawChar(int x,int y,wchar\_t c){

struct Console\* self = Console\_get();

if(x>=WINDOW\_WIDTH || y>=WINDOW\_HEIGHT || x<0 || y<0){

return;

}

self->consoleBuffer[y\*WINDOW\_WIDTH+x] = c;

}

**4.3 Функция создания игровой бомбы**

В данной функции заполняется структура бомбы, на карте помечается что на координатах x, y установлена бомба.

void Bomb\_create(Bomb\* self, int x, int y, Bomberman\* b){

self->alive = 1;

self->x = x;

self->y = y;

self->time = Config\_get()->bombTimer;

self->bomberman = b;

self->fakebomb = 0;

Map\_get(self->map,self->x,self->y)->hasBomb = 1;

}

**4.4 Функция, имитирующая разрушающий эффект от бомбы**

Передаваемые параметры – координаты игрока. Дальность взрыва зависит от приобретенной мощности. Если по всей длине мощности по x и по y на карте обнаружен игрок возвращает единицу.

int Bomb\_willAffectWave(struct Bomb\* self, int cx, int cy, int dx, int dy){

int power = self->bomberman->bombPower;

int d = 1;

while(d<=power){

int x = self->x+d\*dx;

int y = self->y+d\*dy;

if(x==cx && y==cy)

return 1;

if(Map\_wallType(self->map,x,y)==0){

}

else

if(Map\_wallType(self->map,x,y)==2){

return 0;

}else{

break;

}

d = d + 1;

}

return 0;

}

**4.5 Функция обновляющая таймер бомбы**

Если бомба еще существует, высчитываем секунду игрового времени.Если время вышло, либо если на ней эффект от другой, сдетонировать.

void Bomb\_update(struct Bomb\* self){

if(self->alive){

self->time = self->time - 1;

if(self->time <= 0 || Map\_get(self->map,self->x,self->y)->burnTime>0 || self->fakebomb){

Bomb\_explode(self);

}

}

}

**4.6 Функция обновления бомб на карте**

Управляет процессами каждой бомбы на игровом поле. Декрементирует время до детонации.

void Map\_update(struct Map\* self){

int i;

for(i=0;i<BOMB\_COUNT;i++){

Semaphore\_grab(self->bombsSem[i]);

Bomb\_update(&self->bombs[i]);

Semaphore\_release(self->bombsSem[i]);

}

int x,y;

for(y=0;y<self->h;y++){

for(x=0;x<self->w;x++){

self->map[y\*self->w+x].burnTime = self->map[y\*self->w+x].burnTime - 1;

self->map[y\*self->w+x].blockBombs = self->map[y\*self->w+x].blockBombs - 1;

}

}

}

**4.7 Функция отрисовки поля**

void Map\_draw(struct Map\* self){

int x,y;

for(y=0;y<self->h;y++){

for(x=0;x<self->w;x++){

if(self->map[y\*self->w+x].type==0){

if(self->map[y\*self->w+x].burnTime>0){

Sprite\_draw(Sprites\_get(S\_FIRE),x\*4,y\*2);

}else{

if(self->map[y\*self->w+x].hasBonus==0)

{

Sprite\_draw(Sprites\_get(S\_FLOOR),x\*4,y\*2);

}else

if(self->map[y\*self->w+x].hasBonus==1)

{

Sprite\_draw(Sprites\_get(S\_BONUS1),x\*4,y\*2);

}else

if(self->map[y\*self->w+x].hasBonus==2){

Sprite\_draw(Sprites\_get(S\_BONUS2),x\*4,y\*2);

}

}

}

if(self->map[y\*self->w+x].type==1){

Sprite\_draw(Sprites\_get(S\_WALL),x\*4,y\*2);

}

if(self->map[y\*self->w+x].type==2){

Sprite\_draw(Sprites\_get(S\_BRICKS),x\*4,y\*2);

}

}

}

int i;

for(i=0;i<4;i++){

Bomberman\_drawGrave(&self->bombermans[i]);

}

for(i=0;i<BOMB\_COUNT;i++){

Semaphore\_grab(self->bombsSem[i]);

Bomb\_draw(&self->bombs[i]);

Semaphore\_release(self->bombsSem[i]);

}

for(i=0;i<4;i++){

Bomberman\_draw(&self->bombermans[i]);

}

}

**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Для запуска программы, прежде скомпилируем ее командой make, вызывающей имеющийся в директории «Makefile». После чего в папке появятся исполняемые .o файлы (рисунок 5.1).

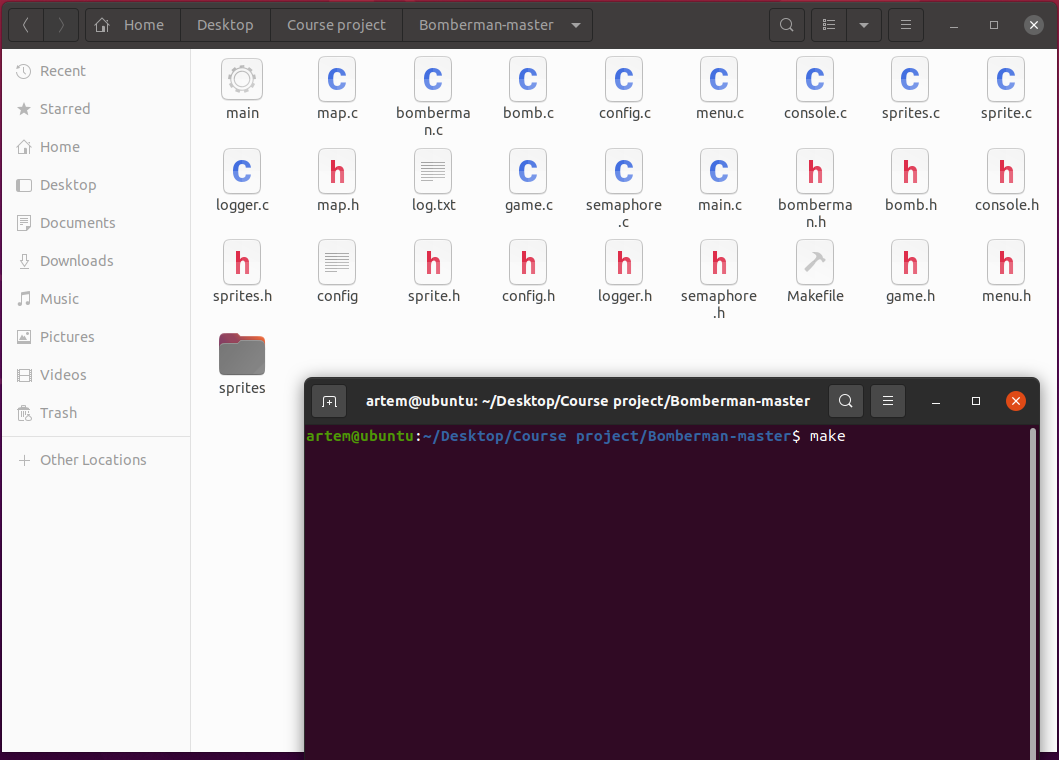


Рисунок 5.1 – Компиляция

Далее необходимо запустить файл «main.o». Для этого в терминале прописываем команду «./main» (рисунок 5.2).

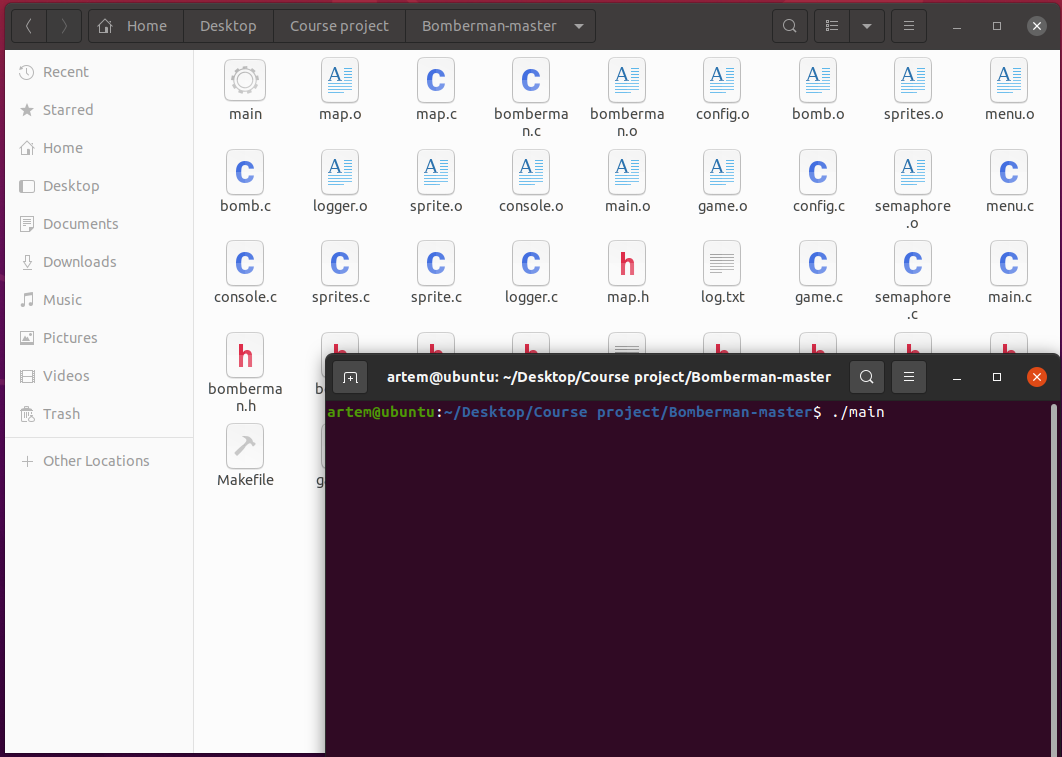


Рисунок 5.2 – Запуск главного исполняемого файла

Если для работы программы размер консольного окна будет мал, появится соответствующее предупреждение и программа завершится (рисунок 5.3).

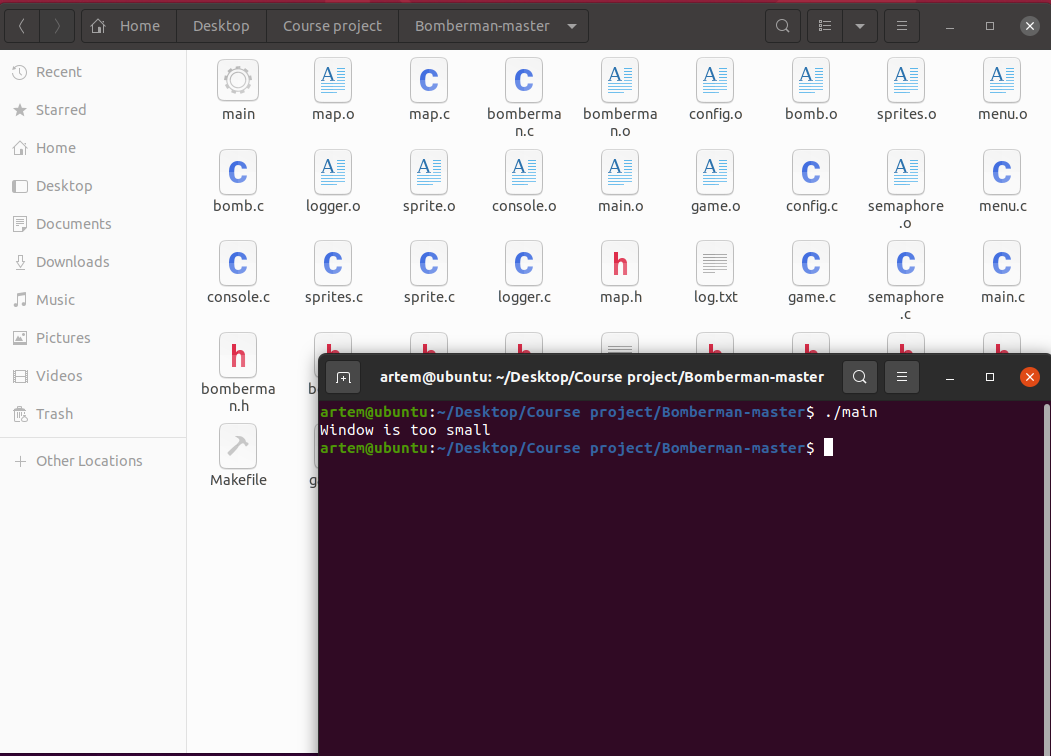


Рисунок 5.3 – Предупреждение

В главном меню можно выбрать количество живых игроков клавишами влево, вправо. Чтобы начать игру, необходимо нажать клавишу Enter (рисунок 5.4).

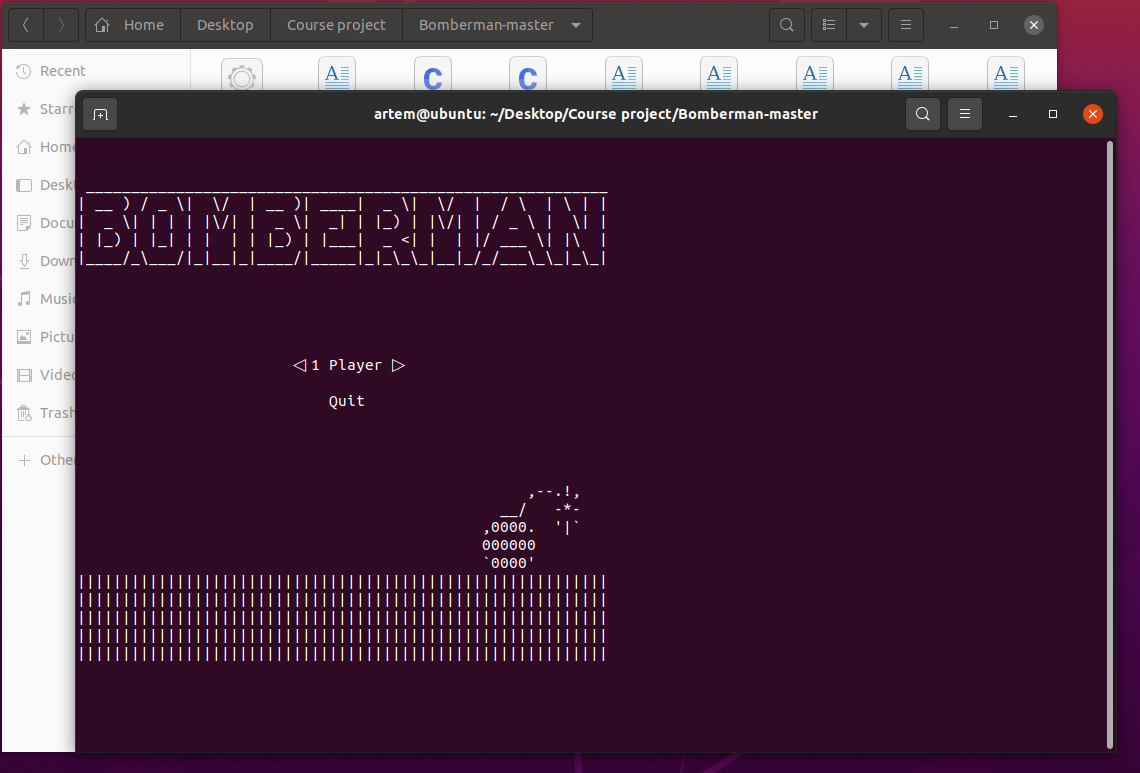


Рисунок 5.4 – Главное меню

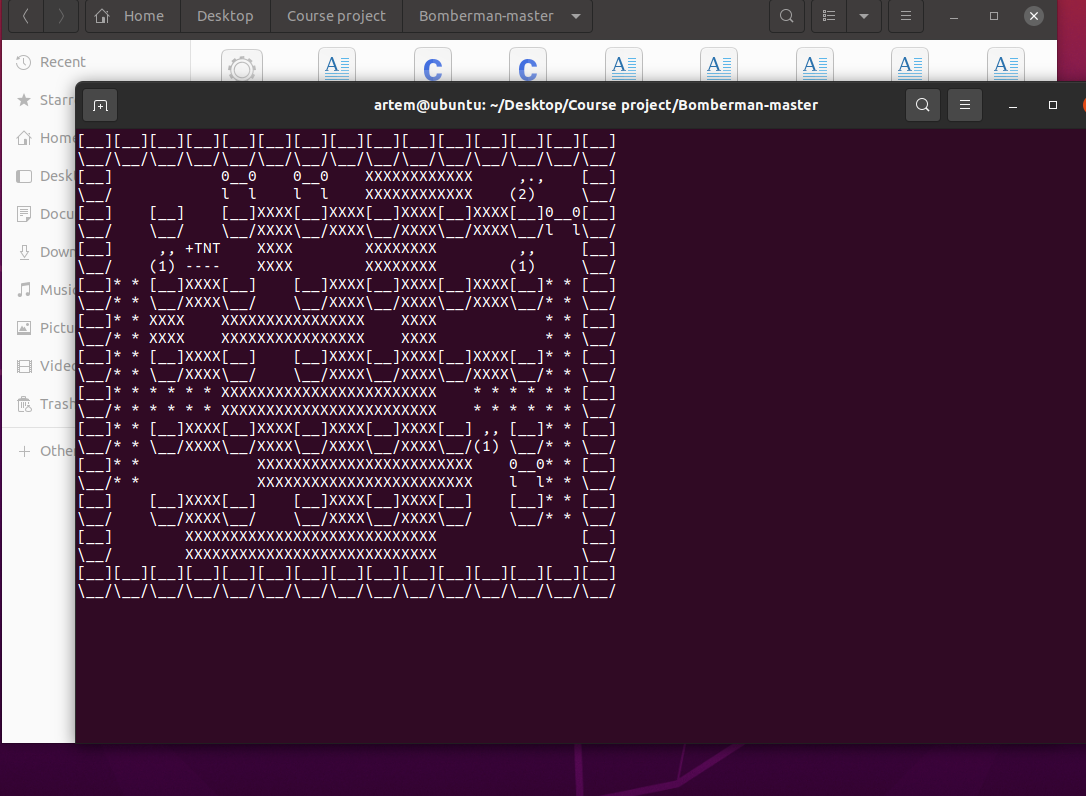
****

Рисунок 5.5 – Игровой процесс

Настройки управления можно посмотреть и изменить в файле «config.txt» (Рисунки 5.5 – 5.6).

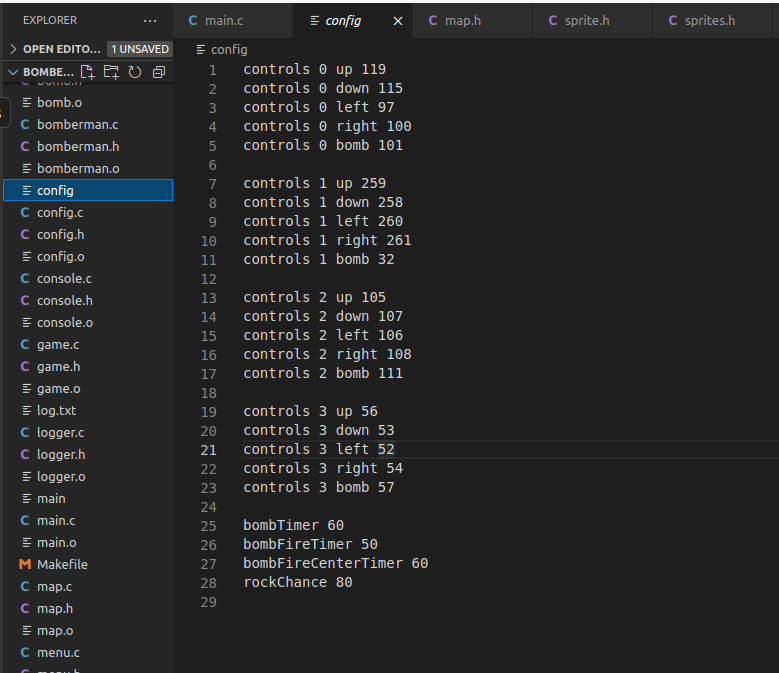


Рисунок 5.6 – Конфигурационный файл «config.txt»

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения данного курсового проекта был написан работоспособный эмулятор игровой приставки DENDY. Разработанный эмулятор тяжело назвать совершенным, имеющиеся модули можно улучшить и еще ближе приблизить их функционал к таковому у реальной консоли, либо добавить новые, в частности:

- аудио-процессор

- другие мапперы для запуска большего количества игр

- альтернативные устройства ввода

- модификации для консоли, которые выпускались и для настоящей, физической версии.

В результате, помимо готового приложения, были получены углубленные знания в области устройства и архитектуры ЭВМ. Были изучены работа процессора, его связь с остальными компонентами, способы генерации изображения.

Также, были расширены знания языка программирования С. Возможности этого языка в создании персональных программ, и, в частности, эмуляторов различных вычислительный систем очень впечатляют, поскольку язык предоставляет возможности взаимодействия с памятью на непосредственном уровне, что позволяет потенциально существенно повысить производительность эмулятора.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Язык программирования С++ [Электронный ресурс]. — Режим доступа: <https://ru.wikipedia.org/wiki/C%2B%2B> — Дата доступа: 09.04.2021

[2] Таненбаум Э., Современные операционные системы / Э. Таненбаум, Х. Бос. – Изд. 4-е. — СПб.: Питер, 2015. — 1120 с.

[3] Рожнова, Н. Г. Вычислительные машины, системы и сети. Дипломное проектирование: учебно-метод. пособие / Н. Г. Рожнова, Н. А. Искра, И. И. Глецевич. — Минск: БГУИР, 2014. — 96 с.: ил.

[4] Nesdev Wiki – неофициальная документация NES и руководство по созданию собственного эмулятора [Электронный ресурс]. — Режим доступа: http://wiki.nesdev.com/w/index.php/Nesdev\_Wiki — Дата доступа: 23.04.2021

# **ПРИЛОЖЕНИЕ А**

*(обязательное)*

Ведомость документов

# **ПРИЛОЖЕНИЕ Б**

*(обязательное)*

Схема структурная

# **ПРИЛОЖЕНИЕ В**

*(обязательное)*

Схема программы

# **ПРИЛОЖЕНИЕ Г**

*(обязательное)*

Листинг кода