Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ЭВМ

Отчёт

по лабораторной работе

**игра «Hell guardians»**

Выполнил: Проверил:

ст. гр. 450502 Кухарчук И. В.

Стаховский А.В.

Минск, 2016

**Скриншоты игры**

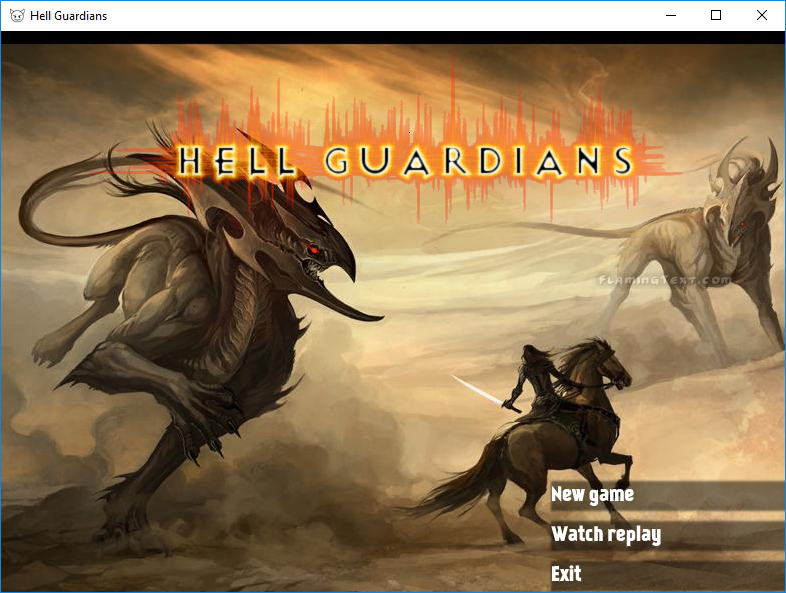
****

Рисунок 1 – Главное меню на операционных Windows

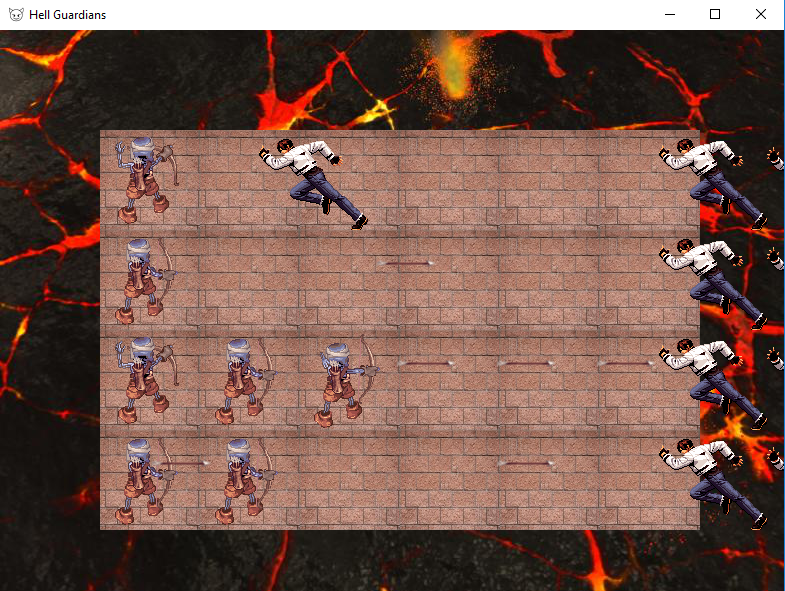


Рисунок 2 – Вид игрового экрана на операционных системах Windows



Рисунок 3 – Главное меню на операционных системах Linux

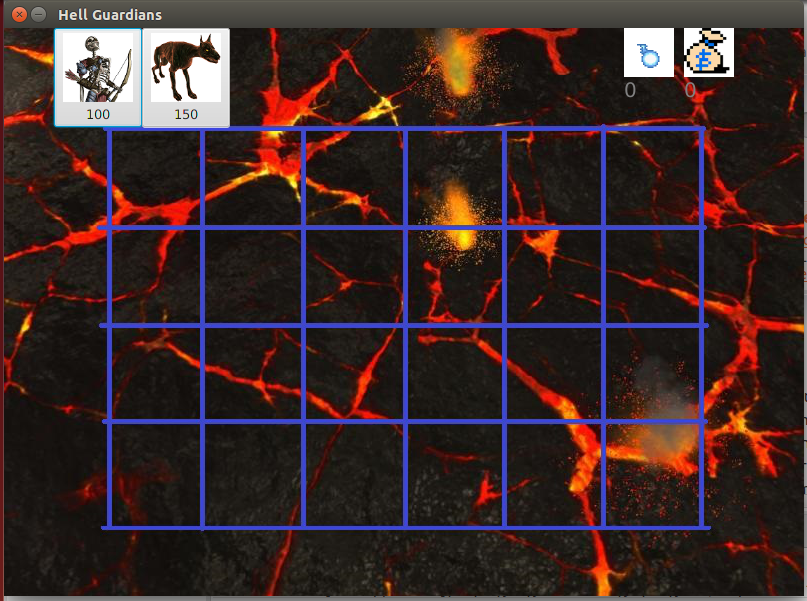


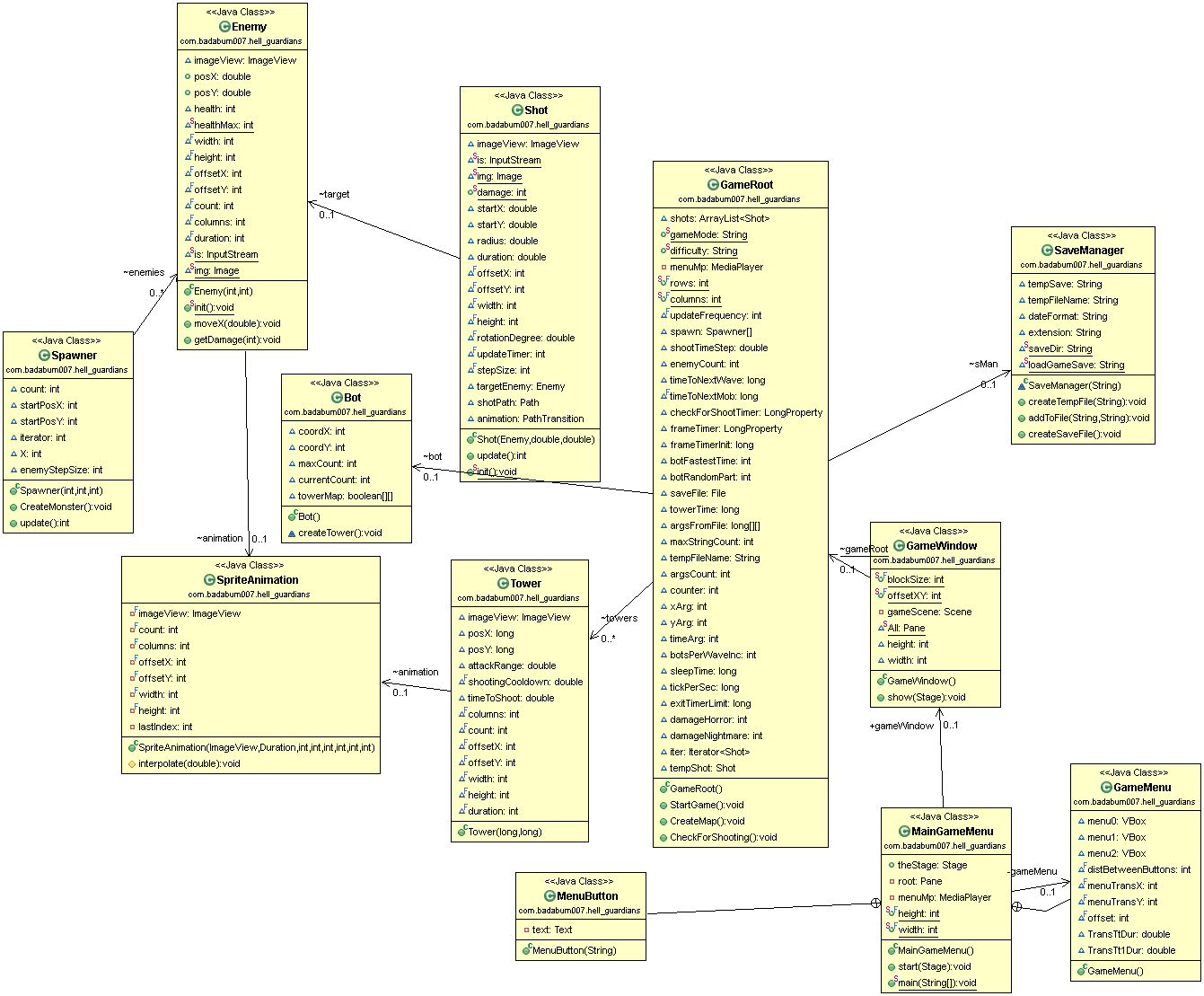
Рисунок 4 – Вид игрового экрана на операционных системах Linux

**Игровой процесс**

Игра основывается на Tower Defense механике, использованной в игре «Plants vs Zombies». Игра «Hell guardians» проходит в режиме «игры на выживание». Игрок строит «башни» для защиты от бесконечно прибывающих врагов. За каждого убитого врага начисляются очки и ресурсы для постройки новых башен. С течением времени количество врагов растет. Для успешной игры игроку требуется грамотно организовать оборону. Цель игрока — набрать как можно больше очков. При достижении врагами башен происходит нанесение урона сооружениям игрока. Если количество полученного урона равно или превышает лимит прочности башни, то сооружение разрушается. На месте разрушенной башни можно построить новую.

Игровое поле представляет из себя сетку. Игрок может строить башни в свободных ячейках сетки. Враги передвигаются по прямой через центр клеток. По достижению врагами конца игрового поля, наступает конец игры.

**Диаграмма классов**



Описание классов:

* MainGameMenu – класс, отвечающий за отрисовку стартового меню программы
* GameMenu – класс, являющийся контейнером для кнопок. Управляет поведением кнопок
* MenuButton – обеспечивает создание одиночной кнопки
* GameWindow – создает и настраивает игровую сцену, передает дальнейшее управление игрой объекту класса GameRoot
* GameRoot – основной класс. Управляет процессом игры: генерация врагов, обновление их позиции, управление ботом, управление поведением башен. Также содержит класс для показа и установки игровых очков. Функцию установки осуществляет асинхронно специальный поток. Содержит ArrayList объектов классов Tower, Shot, Enemy, Spawner.
* Spawner – отвечает за генерацию врагов в определенной точке. Также осуществляет обновление позиций всех врагов, создавшихся в заданной точке.
* Enemy – описание врага и его поведения. Содержит количество здоровья врага, метод отвечающий за его перемещение, метод, осуществляющий смерть врага, при падении уровня здоровья ниже критического.
* Tower – класс, описывающий башню
* Block – параметры клетки игрового поля. Блоки являются площадками для строения башен. При нажатии на свободный блок игрок может построить башню.
* SpriteAnimation – управление спрайтовой анимацией башен и врагов
* Bot – логика бота, который может играть вместо игрока. Шаг бота наступает согласно таймеру в GameRoot и случайному значению из определенного диапазона. Бот строит башню на первую свободную клетку(Block) от начала случайно выбранной линии.
* Shot – параметры выстрела, генерируемого башней. Выстрел генерируется в объекте класса GameRoot. Выстрел происходит при готовности башни (Tower) и нахождении врага на линии перед башней. При столкновении выстрела с первым врагом (объектом класса Enemy), выстрел уничтожается и врагу наносится урон.
* SaveManager – работа с сохранением и воспроизведением игр. При запуске игры создается временный файл. При создании башни пишется запись в файл. При окончании содержимое переписывается в файл сохранения, имя которого имеет заданный формат.

**Блок-схема основного алгоритма**



**Сохранение игры**

Для сохранения игры использовался следующий формат записи:

|  |  |  |
| --- | --- | --- |
| Количество урона, наносимого выстрелом | | |
| 20 | | |
| Позиция башни по Х | Позиция башни по Y | Время постройки |
| 400 | 200 | 276 |
| 400 | 300 | 318 |
|  | | |

Пример сохранения:

|  |  |  |
| --- | --- | --- |
| 20 | | |
| 400 | 200 | 276 |
| 400 | 300 | 318 |
| 100 | 400 | 703 |
| 100 | 300 | 744 |
|  | | |

Для хранения данных о сохранениях используется папка saves. Во время игры данные сохраняются во временный файл, из которого потом записываются в отформатированном виде в сохранение. Имя файла сохранения имеет следующий формат: ггггММдд\_ччммсс (дата и время постройки) + количество построенных башен. Сохранения сохраняются с расширением .sav и содержат все необходимые данные для точного воспроизведения игры. Сохранение происходит автоматически после каждой игры. Выбрать сохранение для воспроизведения можно перейдя в диалоговое окно через кнопку “Watch replay” из главного меню.

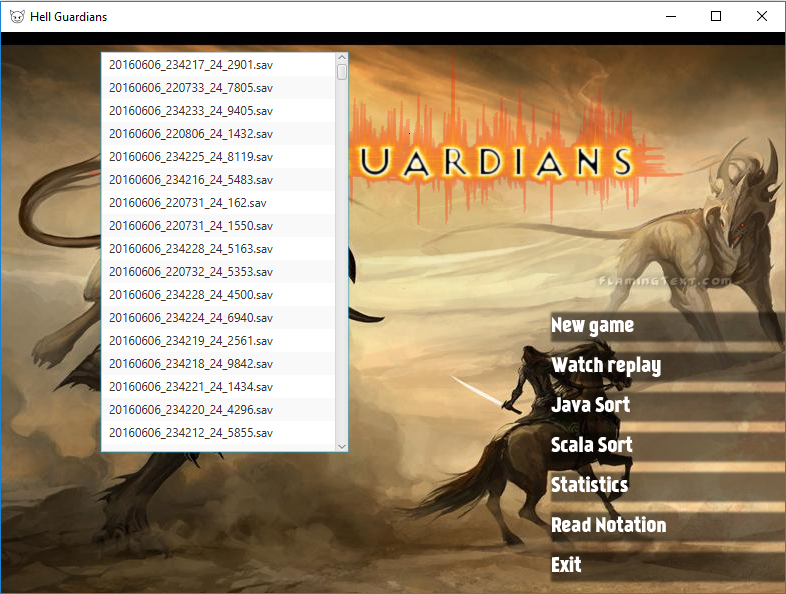
**Многопоточность**



Клиентский поток выполняет проверку количества здоровья у врагов. При уровне здоровья ниже критического происходит смерть врага – удаление его из игрового поля. За каждого убитого врага игроку начисляются очки. Для их установки процесс-клиент уведомляет о смерти врага процесс-сервер, который производит установку очков. Установка очков происходит асинхронно, что ускоряет работу приложения.

**Сортировка**

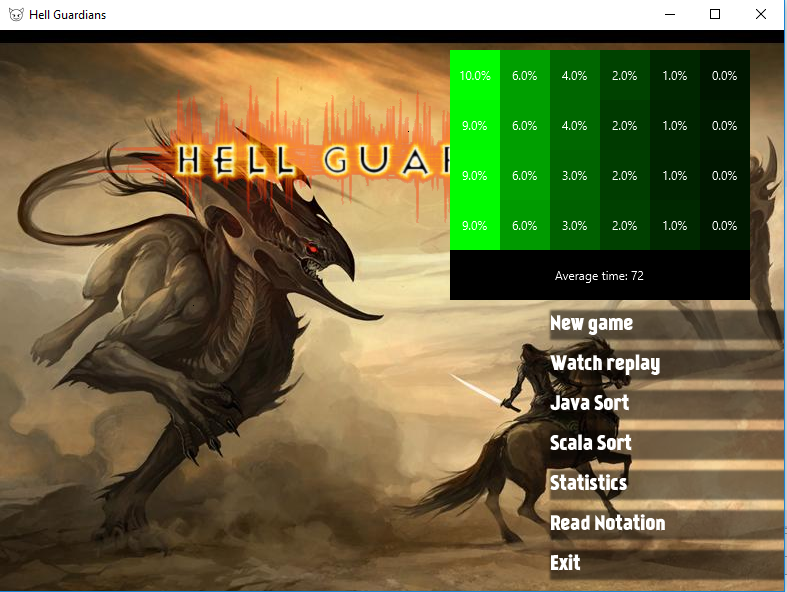
Для сортировки файлов используются пункты меню “Java Sort” и “Scala Sort”.





Для сортировки использовался алгоритм qsort в его нефункциональной форме. При первой сортировке время заметно больше из-за прогрузки всех файлов сохранений. Тестирование производилось на списке из более чем на 100 тысячах файлов для более точных замеров времени. Сортировки проходят примерно за равное время, т.к в обоих случаях используется одинаковая реализация qsort. Отсортированные файлы отображаются на экране в виде списка, время сортировок выводится на консоль.

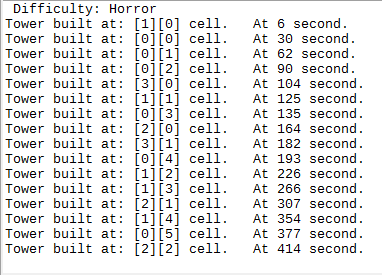
**Статистика**

****

Статистика представляет собой мини-кару игрового поля. Число в ячейках соответствует отношению количества построенных на этой ячейке башен за все игры к количеству игр. Для наглядности добавлено также цветовое представление этого отношения. Ярчайшие клетки были наиболее часто были застроены.

Снизу карты присутствует строка, в которой отражено среднее время постройки башен.

**Преобразование нотации**



Для преобразования нотации в более понятный вид использовался Scala объект NotationReader. Данные сохранения были дополнены текстовыми комментариями, а также данные были переведены в более удобный для чтения формат. Просмотреть нотацию любого сохранения можно используя пункт меню Read Notation.

Для формирования списка с данными из файла сохранения использовался Scala класс SNotList. Парсинг данных файла и показ нотации производился в функции showSaveNotation класса SaveManager.