|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ** | | | | | | | | | | | | |
| **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** | | | | | | | | | | | | |
| **«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| Институт информационных технологий и управления в технических системах | | | | | | | | | | | | | |
| (полное название института) | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| кафедра «Информационные системы» | | | | | | | | | | | | | |
| (полное название кафедры) | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Пояснительная записка** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| к курсовой работе | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| на тему | **игра ”Потерянный”** | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| по дисциплине | | **Объектно-ориентированное программирование** | | | | | | | | | | |
|  |  | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| Выполнил: студент | | | III | | курса, группы: | | | | **ИС/б-17-2-о** | | | |
|  | | | | | | | | | | | | |
| Направления подготовки (специальности) | | | | | | | | | 09.03.02 | | | |
|  | | | | | | | | | | | | |
| Информационные системы и технологии | | | | | | | | | | | | |
| (код и наименование направления подготовки (специальности)) | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| профиль (специализация) | | | | | |  | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Черняева Никиты Георгиевича** | | | | | | | | | | | | |
| (**фамилия, имя, отчество студента**) | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| Дата допуска к защите « | | | |  | | | » |  | | 20 | 19 | г. |
|  | | | |  | | |  |  | |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Руководитель | |  |  |  |
|  | (подпись) | |  | (инициалы, фамилия) |

1. АННОТАЦИЯ

Пояснительная записка к курсовому проектированию по дисциплине «Объектно-ориентированное программирование» для студентов направления 09.03.02 – «Информационные системы и технологии».

Пояснительная записка содержит описание процесса проектирования и кодирования курсового проекта, состоящего в разработке игры «Потерянный» с использованием объектного подхода.

|  |
| --- |
|  |

### СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc25781523)

[1 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc25781524)

[1.1 Цель разработки 5](#_Toc25781525)

[1.2 Постановка задачи 5](#_Toc25781526)

[1.3 Ограничения, условия выполнения и функционирования 5](#_Toc25781527)

[2 РАЗРАБОТКА ОБЪЕКТНОЙ МОДЕЛИ 6](#_Toc25781528)

[2.1 Построение иерархии классов 6](#_Toc25781529)

[2.2 Построение информационной модели 7](#_Toc25781530)

[2.3 Жизненный цикл программы 8](#_Toc25781531)

[2.4 Жизненный цикл объекта 9](#_Toc25781532)

[2.5 Диаграмма переходов состояний 9](#_Toc25781533)

[2.6 Диаграмма потоков данных и действий 10](#_Toc25781534)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 11](#_Toc25781535)

[3.1 Обоснование выбора языка программирования 11](#_Toc25781536)

[3.2 Общее описание взаимодействия программных модулей и описание классов и методов 11](#_Toc25781537)

[4 Интерфейс пользователя 14](#_Toc25781538)

[ЗАКЛЮЧЕНИЕ 20](#_Toc25781539)

[ПЕРЕЧЕНЬ ССЫЛОК 21](#_Toc25781540)

[ПРИЛОЖЕНИЕ А 22](#_Toc25781541)

**ВВЕДЕНИЕ**

В ходе курсового проектирования ведется разработка программы под названием «Потерянный»: 2D-платформера, разработанной на основании технического задания.

Целью данного курсового проекта является изучение современного подхода к программированию на основе объектно-ориентированной технологии и создание приложений с помощью этого подхода. Для достижения поставленной цели были определены следующие задачи:

− определение основных функциональных требований;

− структурирование системы, выделение основных систем и определение порядка их взаимодействия;

− разработка базовой модели управления взаимодействия подсистем и компонентов.

Структура работы состоит из программы и пояснительной записки. В первом разделе пояснительной записки описывается цель разработки и задачи, которые необходимо выполнить. Второй раздел описывает проектное решение: выделение объектов, построение иерархии классов, информационной модели, описание жизненного цикла программы, построение диаграммы переходов состояний и потоков данных и действий. В третьем разделе описывается реализация проекта: описание взаимодействия программных модулей, реализации основных классов и их методов. В четвертом разделе описывается интерфейс пользователя. В заключении делаются выводы о соответствии программы техническому заданию.

* 1. ПОСТАНОВКА ЗАДАЧИ

## 1.1 Цель разработки

Целью курсового проекта является изучение современного подхода к программированию на основе объектно-ориентированной технологии, приобретение навыков написания программ на языке с поддержкой ООП.

Тематика данной курсовой работы – компьютерная игра в жанре “2D Платформер”.

## 1.2 Постановка задачи

Игровой процесс основан на том, что пользователь управляет главным персонажем – человеком, который потерялся в лесу. Необходимо выбраться, перемещаясь по локации с различными врагами.

В игре присутствуют различные виды врагов. Одни статические, другие могут перемещаться.

Игра заканчивается победой, если игрок выбирается из леса, и поражением, если игрок падает, либо враги убивают игрока.

## 1.3 Ограничения, условия выполнения и функционирования

Требование к среде исполнение: наличие клавиатуры для управления персонажем и мыши для управления меню.

Не предусматривается использование данной игры на мобильных устройствах. Приложение не требует наличие интернета. Программа работает на OC Windows.

* 1. РАЗРАБОТКА ОБЪЕКТНОЙ МОДЕЛИ
     1. Построение иерархии классов

В ходе анализа проекта были выделены основные классы. На рисунке 2.1 изображена диаграмма классов. Из этой диаграммы видно, что присутствуют иерархия. Классы Intro, Guide, Menu, Death, Win являются производными от класса DynamicBackground. Классы WinBlock и EnemyDynamic наследуются от класса EnemyStatic, а он в свою очередь наследуется от Person.

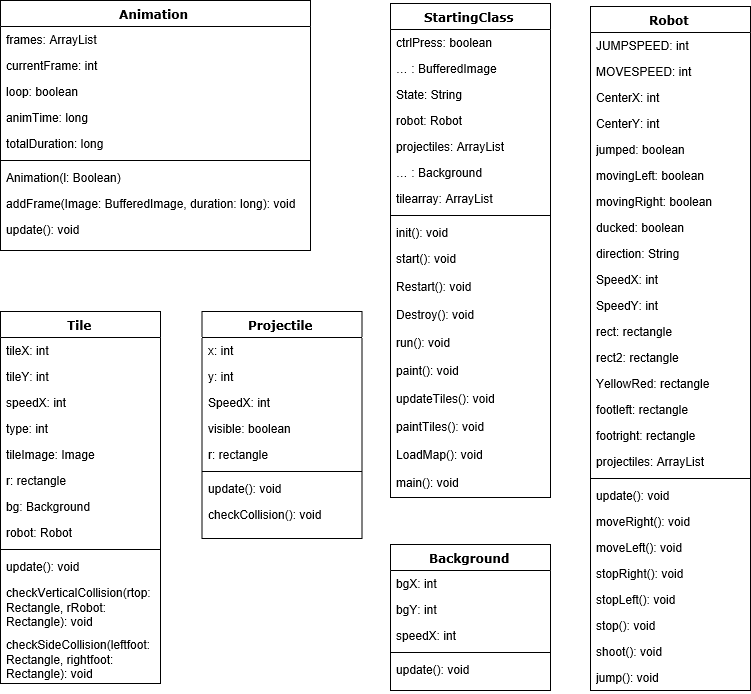


Рисунок 2.1 – Диаграмма классов

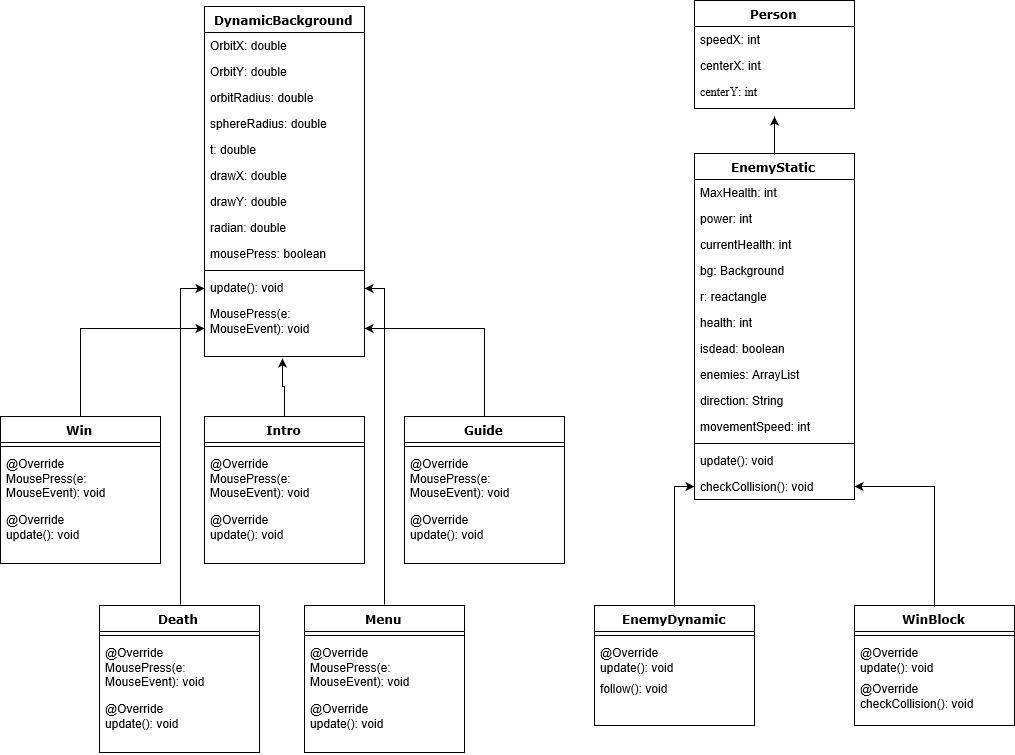


Рисунок 2.1 – Лист 2

**2.2 Построение информационной модели**

Рассмотрим диаграмму, представленную на рисунке 2.1 и построим информационную модель игры. Она представлена на рисунке 2.2. Данная информационная модель содержит основные классы, определенные на начальном этапе разработки при абстрагировании и выделение объектов. Так же отображены основные связи классов, что помогает понять принципы их взаимодействия.

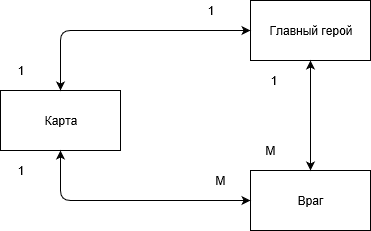


Рисунок 2.2 – Информационная модель

**2.3 Жизненный цикл программы**

В ходе работы приложения меняет своё состояние – это называется жизненным циклом. Опишем жизненный цикл нашей игры на рисунке 2.3. На диаграмме жизненного цикла можно проследить стадии работы приложения.

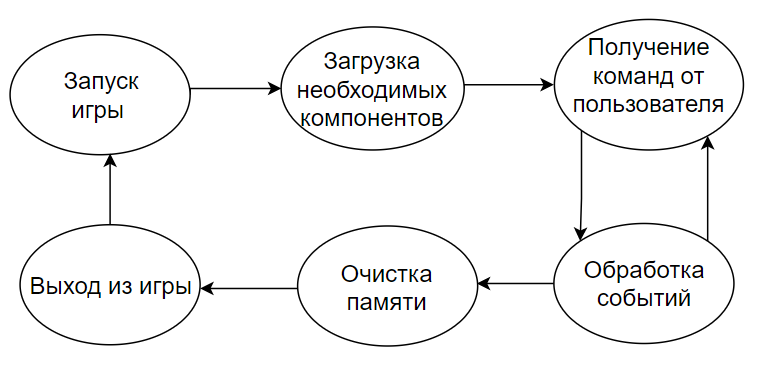


Рисунок 2.3 – Жизненный цикл программы

**2.4 Жизненный цикл объекта**

В программе создается множество объектов, одним из них является главный герой. На рисунке 2.4 представлен его жизненный цикл. Диаграмма жизненного цикла объекта позволяет проследить период его существования и его работу.

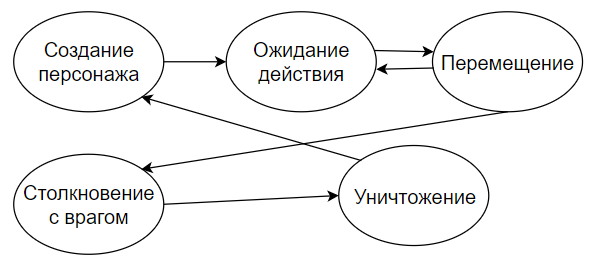


Рисунок 2.4 – Жизненный цикл объекта

**2.5 Диаграмма переходов состояний**

На рисунке 2.5 представлена диаграмма переходов состояний врага, на ней изображены состояния, в которых находится объект. При запуске игры враг создается. Затем если главный герой попадает в него, то враг переходит в состояние ранен, а если его здоровье равно нулю, враг уничтожается.

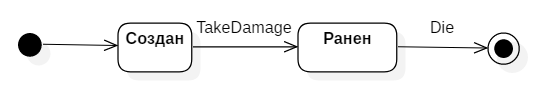


Рисунок 2.5 – Диаграмма переходов состояний

**2.6 Диаграмма потоков данных и действий**

На рисунке 2.6 изображена диаграмма потоков данных главного героя. Диаграмма показывает, какие потоки данных влияют на состояние счета очков главного героя. Входящими данными является информация о столкновении о коллайдер кристалла. Исходящими же данными являются счет персонажа.



Рисунок 2.6 – Диаграмма потоков данных

**3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

**3.1 Обоснование выбора языка программирования**

Для реализации проекта был выбран язык программирования Java.

Java является объектно-ориентированным языком программирования. Отличительной особенностью Java в сравнении с другими языками программирования общего назначения является обеспечение высокой продуктивности программирования, нежели производительность работы приложения или эффективность использования им памяти.

Огромное преимущество Java заключается в том, что на этом языке можно создавать приложения, способные работать на различных платформах. Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода.

**3.2 Общее описание взаимодействия программных модулей и описание классов и методов**

В ходе разработки было написано 16 классов, которые находятся в различных файлах.

В проекте существует иерархия и связи между классами. Из представленной выше диаграммы классов видно, что модули наследуют атрибуты и методы друг друга.

Класс DynamicBackground представляет собой логику меню, делая картинку на заднем фоне движущейся, оставляя при этом кнопки статичными.

Классы Intro, Menu, Guide, Death, Win наследуются от класса DynamicBackground, переопределяют некоторые поля и реализуют различные возможности меню.

Класс Intro создает окно, отображающееся при нажатии на кнопку "Intro".

Класс Menu создает окно, отображающееся при открытии меню.

Класс Guide создает окно, отображающееся при нажатии на кнопку "Guide".

Класс Death создает окно, отображающееся при поражении.

Класс Win создает окно, отображающееся при выигрыше.

Класс Animation реализует анимацию игрока, врагов, чтобы игра выглядела более динамично и приятно.

Класс создает Background задний фон игры.

Класс Projectile отвечает за создание выстрелов и попадание пуль во врагов.

Класс Enemy создает врагов, метод follow() позволяет им двигаться.

Класс Robot создает главного героя, в этом классе прописаны методы для передвижения и стрельбы.

Класс Tile определяет плитку на карте.

Класс Person реализует основные методы для создания активного объекта.

Класс EnemyStatic наследуется от класса Person расширяя его методами для создания неподвижного врага.

Класс EnemyDynamic наследуется от класса EnemyStatic добавляя к нему метод для передвижения врага.

Класс WinBlock наследуется от класса EnemyStatic и создает невидимый объект, при соприкосновении с которым главный герой выигрывает.

Класс StartingClass является главным, из него запускается программа. Он содержит такие важные методы:

init() - инициализирует все переменные.

start() - создание всех фреймов и вызов метода loadMap().

run() - работы программы.

gameUpdate() - обновление игры в зависимости от действий.

paint() - создание визуального изображения.

loadMap() - загрузка карты.

keyReleased(), keyPressed(), mousePressed(), mouseReleased() – обрабатывают события при работе с клавиатурой и мышью.

**4 Интерфейс пользователя**

При запуске игры перед пользователем появляется главное меню, в котором он может выбрать пункт играть в игру, прочитать "интро", прочитать "гайд" по игре или пункт выйти из приложения (рисунок 4.1).



Рисунок 4.1 – Главное меню

При нажатии на кнопку «Intro» игроку открывается текст, который повествует историю главного героя.(рисунок 4.2).



Рисунок 4.2 – Окно, появляющееся при нажатии на кнопку «Intro»

При нажатии на кнопку «Guide» игроку открывается текст, рассказывает как играть в игру.(рисунок 4.3).



Рисунок 4.3 – Окно, появляющееся при нажатии на кнопку «Guide»

При нажатии на кнопку «Play» начинается игра.(рисунок 4.4).



Рисунок 4.4 – Начало игры

Главный герой может приседать(рисунок 4.5), прыгать(рисунок 4.6), идти налево или направо и стрелять(рисунок 4.7).



Рисунок 4.5 – Главный герой в приседе



Рисунок 4.6 – Главный герой в прыжке



Рисунок 4.7 – Стрельба главного героя

В игре существуют враги(рисунок 4.8), которые могут убить главного героя при соприкосновении с ним. Когда герой погибает появляется соответствующий текст и две кнопки для того, чтобы начать уровень заново или выйти из игры (рисунок 4.9).



Рисунок 4.8 – Изображение врага



Рисунок 4.9 – Текст и кнопки, отображающиеся при поражении

Также на карте расположены некоторые объекты, которые мешают, либо помогают главному герою проходить уровень, например - рисунок 4.10, где показаны ямы, которые надо перепрыгивать.



Рисунок 4.10 – Ямы

Когда игрок проходит всю карту и выбирается из леса, он выигрывает - рисунок 4.11.



Рисунок 4.11 – Текст и кнопки, отображающиеся при выигрыше

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта была выполнена разработка программы в соответствии с постановкой задачи на курсовое проектирование по теме по дисциплине “Объектно-ориентированное программирование”.

Была разработана игра в жанре “2D Платформер”.

В ходе выполнения работы был решен ряд поставленных задач. Сначала был выполнен анализ предметной области, затем был спроектирована иерархия классов. Следующей задачей была программирование и разработка графической составляющей игры. Протестировав наше приложение, был сделан вывод, что она работает корректно.

Так же в ходе выполнения курсовой работы были закреплены понятия объектно-ориентированного программирования, объектно-ориентированного анализа и объектно-ориентированного подхода, а также закреплены навыки программирования на объектно-ориентированном языке

Было приобретено умение написания объектно-ориентированных программ с графическим интерфейсом, разработки иерархии, было достигнуто умение оформления программ с использованием информационных моделей, моделей состояний и жизненных циклов.

Из всех проделанных этапов можно сделать вывод, что объектно-ориентированный подход повышает гибкость программы для дальнейшего усовершенствования.

По результатам тестирования разработанного продукта и составленной документации к нему, можно сделать вывод, что задача, поставленная на курсовое проектирование, была выполнена в полном объеме.

# **ПЕРЕЧЕНЬ ССЫЛОК**

1. Arlow, J. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design [Текст] / J. Arlow, I. Neustadt. - 2-е изд. - Boston: Addison-Wesley Professional, 2005. - 624 с.

2. Фримен, Э. Паттерны проектирования [Текст] / Э. Фримен, Э. Робсон. - Санкт-Петербург: Питер, 2019 - 656 с.

3. Fowler, M. Patterns of Enterprise Application Architecture [Текст] / M. Fowler, D. Rice, M. Foemmel. - Boston: Addison-Wesley, 2011 - 559 с.

4. Буч Г. Обьектно-ориентирванное проектирование с примерами/Г.Буч-М.Конокорд,1992-519с.

5. Эккель Б. Философия Java / Эккель Б. - 4-е изд. - Санкт-Петербург: Питер, 2019 - 1168с.

**ПРИЛОЖЕНИЕ А**

**ТЕКСТ ПРОГРАММЫ**

Класс Animation.java

package Game;

import java.awt.Image;

import java.awt.image.BufferedImage;

import java.util.ArrayList;

public class Animation {

@SuppressWarnings("rawtypes")

ArrayList frames;

int currentFrame;

boolean loop;

private long animTime;

private long totalDuration;

@SuppressWarnings("rawtypes")

public Animation(boolean l) {

loop = l;

frames = new ArrayList();

totalDuration = 0;

synchronized (this) {

animTime = 0;

currentFrame = 0;

}

}

@SuppressWarnings("unchecked")

public synchronized void addFrame(BufferedImage image, long duration) {

totalDuration += duration;

frames.add(new AnimFrame(image, totalDuration));

}

public synchronized void update(long elapsedTime) {

if (frames.size() > 1) {

animTime += elapsedTime;

if (!(loop && currentFrame == frames.size() - 1)) {

if (animTime >= totalDuration) {

animTime = animTime % totalDuration;

currentFrame = 0;

}

while (animTime > getFrame(currentFrame).endTime) {

currentFrame++;

}

}

}

}

public synchronized BufferedImage getImage() {

if (frames.size() == 0) {

return null;

} else {

return getFrame(currentFrame).image;

}

}

private AnimFrame getFrame(int i) {

return (AnimFrame) frames.get(i);

}

private class AnimFrame {

BufferedImage image;

long endTime;

public AnimFrame(Image image, long endTime) {

this.image = (BufferedImage) image;

this.endTime = endTime;

}

}

}

Класс Background.java

**package** Game;

**public** **class** Background {

**private** **int** bgX, bgY, speedX;

**public** Background(**int** x, **int** y) {

bgX = x;

bgY = y;

speedX = 0;

}

**public** **void** update() {

bgX += speedX;

**if** (bgX < -1920 \* 3) {

bgX = 1920;

}

**if** (bgX > 1920 \* 2) {

bgX = -1920 \* 2;

}

}

**public** **int** getBgX() {

**return** bgX;

}

**public** **int** getBgY() {

**return** bgY;

}

**public** **int** getSpeedX() {

**return** speedX;

}

**public** **void** setBgX(**int** bgX) {

**this**.bgX = bgX;

}

**public** **void** setBgY(**int** bgY) {

**this**.bgY = bgY;

}

**public** **void** setSpeedX(**int** speedX) {

**this**.speedX = speedX;

}

}

Класс Death.java

**package** Game;

**import** java.awt.event.MouseEvent;

**public** **class** Death **extends** DynamicBackground {

**public** **double** orbitX = -110; /\* x-coordinate in orbit's center \*/

**public** **double** orbitY = -60; /\* y-coordinate in orbit's center \*/

@Override

**public** **void** update() {

radian = orbitSpeed \* t;

drawX = orbitX + orbitRadius \* Math.*cos*(radian);

drawY = orbitY + orbitRadius \* Math.*sin*(radian);

t += 1;

}

**void** mousePress(MouseEvent e) {

**int** mx = e.getX();

**int** my = e.getY();

StartingClass.*restart*();

**if** (mx > 60 && mx < 210 && my > 350 && my < 390) {

StartingClass.*State* = "menu";

} **else** **if** (mx > 60 && mx < 210 && my > 400 && my < 440) {

StartingClass.*State* = "game";

}

}

}

Класс DynamicBackground.java

**package** Game;

**abstract** **class** DynamicBackground {

**public** **double** orbitX; // x-coordinate in orbit's center \*/

**public** **double** orbitY; // y-coordinate in orbit's center \*/

**public** **double** orbitRadius = 20;

**public** **double** orbitSpeed = 0.03;

**public** **double** sphereRadius = 5;

**public** **double** t;

**public** **double** drawX;

**public** **double** drawY;

**public** **double** radian;

**boolean** mousePress;

**public** **abstract** **void** update();

}

Класс EnemyDynamic.java

**package** Game;

**import** java.util.ArrayList;

**public** **class** EnemyDynamic **extends** EnemyStatic{

**public** **static** ArrayList<EnemyDynamic> *enemies* = **new** ArrayList<EnemyDynamic>();

**public** EnemyDynamic(**int** centerX, **int** centerY) {

**super** (centerX, centerY);

}

// Behavioral Methods

**public** **static** **void** update() {

**for** (EnemyDynamic i : *enemies*) {

i.follow();

i.centerX += i.speedX;

i.speedX = i.bg.getSpeedX() \* 5 + i.movementSpeed;

i.r.setBounds(i.centerX - 30, i.centerY - 10, 85, 60);

**if** (i.r.intersects(Robot.*yellowRed*))

i.checkCollision();

}

}

**public** **void** follow() {

**if** (centerX - StartingClass.*getRobot*().getCenterX() > 650) {

**this**.movementSpeed = 0;

} **else** **if** (StartingClass.*getRobot*().getCenterX() - centerX > 650) {

**this**.movementSpeed = 0;

} **else** **if** (Math.*abs*(StartingClass.*getRobot*().getCenterX() - centerX) < 5) {

**this**.movementSpeed = 0;

} **else** {

**if** (StartingClass.*getRobot*().getCenterX() >= centerX) {

**this**.direction = "right";

**this**.movementSpeed = 2;

} **else** {

**this**.direction = "left";

**this**.movementSpeed = -2;

}

}

}

}

Класс EnemyStatic.java

package Game;

import java.awt.Rectangle;

import java.util.ArrayList;

public class EnemyStatic extends Person{

protected int maxHealth, currentHealth, power;

protected Background bg = StartingClass.getBg1();

public Rectangle r = new Rectangle(0, 0, 0, 0);

public int health = 5;

protected boolean isdead;

public static ArrayList<EnemyStatic> enemies = new ArrayList<EnemyStatic>();

protected int movementSpeed;

public String direction = "left";

public EnemyStatic(int centerX, int centerY) {

setCenterX(centerX);

setCenterY(centerY);

}

public static void update() {

for (EnemyStatic i : enemies) {

i.centerX += i.speedX;

i.speedX = i.bg.getSpeedX() \* 5 + i.movementSpeed;

i.r.setBounds(i.centerX - 30, i.centerY - 10, 85, 60);

if (i.r.intersects(Robot.yellowRed))

i.checkCollision();

}

}

protected void checkCollision() {

if (r.intersects(Robot.rect) || r.intersects(Robot.rect2)) {

StartingClass.State = "dead";

}

}

public void attack() {

}

public int getMaxHealth() {

return maxHealth;

}

public int getCurrentHealth() {

return currentHealth;

}

public int getPower() {

return power;

}

public Background getBg() {

return bg;

}

public void setMaxHealth(int maxHealth) {

this.maxHealth = maxHealth;

}

public void setCurrentHealth(int currentHealth) {

this.currentHealth = currentHealth;

}

public void setPower(int power) {

this.power = power;

}

public void setBg(Background bg) {

this.bg = bg;

}

public boolean getIsDead() {

return isdead;

}

public void setIsDead(boolean s) {

isdead = s;

}

}

Класс Guide.java

**package** Game;

**import** java.awt.event.MouseEvent;

**public** **class** Guide **extends** DynamicBackground {

**public** **double** orbitX = -20; /\* x-coordinate in orbit's center \*/

**public** **double** orbitY = -20; /\* y-coordinate in orbit's center \*/

@Override

**public** **void** update() {

radian = orbitSpeed \* t;

drawX = orbitX + orbitRadius \* Math.*cos*(radian);

drawY = orbitY + orbitRadius \* Math.*sin*(radian);

t += 1;

}

**void** mousePress(MouseEvent e) {

**int** mx = e.getX();

**int** my = e.getY();

**if** (mx > 60 && mx < 210 && my > 350 && my < 390) {

StartingClass.*State* = "guide";

} **else** **if** (mx > 20 && mx < 170 && my > 700 && my < 740) {

StartingClass.*State* = "menu";

}

}

}

Класс Intro.java

**package** Game;

**import** java.awt.event.MouseEvent;

**public** **class** Intro **extends** DynamicBackground {

**public** **double** orbitX = -50; /\* x-coordinate in orbit's center \*/

**public** **double** orbitY = -50; /\* y-coordinate in orbit's center \*/

@Override

**public** **void** update() {

radian = orbitSpeed \* t;

drawX = orbitX + orbitRadius \* Math.*cos*(radian);

drawY = orbitY + orbitRadius \* Math.*sin*(radian);

t += 1;

}

**void** mousePress(MouseEvent e) {

**int** mx = e.getX();

**int** my = e.getY();

**if** (mx > 60 && mx < 210 && my > 300 && my < 340) {

StartingClass.*State* = "intro";

} **else** **if** (mx > 20 && mx < 170 && my > 700 && my < 740) {

StartingClass.*State* = "menu";

}

}

}

Класс Menu.java

**package** Game;

**import** java.awt.event.MouseEvent;

**public** **class** Menu **extends** DynamicBackground {

**public** **double** orbitX = -700; /\* x-coordinate in orbit's center \*/

**public** **double** orbitY = -600; /\* y-coordinate in orbit's center \*/

@Override

**public** **void** update() {

radian = orbitSpeed \* t;

drawX = orbitX + orbitRadius \* Math.*cos*(radian);

drawY = orbitY + orbitRadius \* Math.*sin*(radian);

t += 1;

}

**void** mousePress(MouseEvent e) {

**int** mx = e.getX();

**int** my = e.getY();

**if** (mx > 60 && mx < 210 && my > 250 && my < 290) // Play button

StartingClass.*State* = "game";

**else** **if** (mx > 60 && mx < 210 && my > 300 && my < 340) // intro button

StartingClass.*State* = "intro";

**else** **if** (mx > 60 && mx < 210 && my > 350 && my < 390) // guide button

StartingClass.*State* = "guide";

**else** **if** (mx > 60 && mx < 210 && my > 450 && my < 490)

System.*exit*(1);

}

}

Класс Person.java

**package** Game;

**public** **class** Person {

**protected** **int** speedX, centerX, centerY;

**public** **int** getSpeedX() {

**return** speedX;

}

**public** **int** getCenterX() {

**return** centerX;

}

**public** **int** getCenterY() {

**return** centerY;

}

**public** **void** setSpeedX(**int** speedX) {

**this**.speedX = speedX;

}

**public** **void** setCenterX(**int** centerX) {

**this**.centerX = centerX;

}

**public** **void** setCenterY(**int** centerY) {

**this**.centerY = centerY;

}

}

Класс Projectile.java

package Game;

import java.awt.Rectangle;

import java.util.ArrayList;

public class Projectile {

private int x, y, speedX;

private boolean visible;

private Rectangle r;

public Projectile(int startX, int startY, boolean right) {

x = startX;

y = startY;

if (right)

speedX = 30;

else

speedX = -30;

visible = true;

r = new Rectangle(0, 0, 0, 0);

}

public void update() {

x += speedX;

r.setBounds(x, y, 10, 5);

if (x > 1366 || x < 0) {

visible = false;

r = null;

} else

checkCollision();

}

private void checkCollision() {

@SuppressWarnings("rawtypes")

ArrayList tiles = StartingClass.getTileArray();

for (int i = 0; i < tiles.size(); i++) {

Tile p = (Tile) tiles.get(i);

if (r != null) {

if (r.intersects(p.getRectangle()) && (p.getType() == 2 || p.getType() == 3 || p.getType() == 6)) {

if (Robot.getDirection() == "right")

x = p.getTileX() - 35;

else if (Robot.getDirection() == "left")

x = p.getTileX() + 50;

visible = false;

}

}

}

for (EnemyDynamic i : EnemyDynamic.enemies) {

if (r.intersects(i.r)) {

visible = false;

if (i.health > 0) {

i.health -= 1;

}

if (i.health == 0) {

i.setIsDead(true);

}

}

}

for (EnemyStatic i : EnemyStatic.enemies) {

if (r.intersects(i.r)) {

visible = false;

if (i.health > 0) {

i.health -= 1;

}

if (i.health == 0) {

i.setIsDead(true);

}

}

}

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public int getSpeedX() {

return speedX;

}

public boolean isVisible() {

return visible;

}

public void setX(int x) {

this.x = x;

}

public void setY(int y) {

this.y = y;

}

public void setSpeedX(int speedX) {

this.speedX = speedX;

}

public void setVisible(boolean visible) {

this.visible = visible;

}

}

Класс Robot.java

package Game;

import java.awt.Rectangle;

import java.util.ArrayList;

public class Robot {

// Constants are Here

final int JUMPSPEED = -17;

final int MOVESPEED = 5;

private int centerX = 500;

private int centerY = 560;

private boolean jumped = false;

private boolean movingLeft = false;

private boolean movingRight = false;

private boolean ducked = false;

private static String direction = "right";

private int speedX = 0;

private int speedY = 0;

public static Rectangle rect = new Rectangle(0, 0, 0, 0);

public static Rectangle rect2 = new Rectangle(0, 0, 0, 0);

public static Rectangle yellowRed = new Rectangle(0, 0, 0, 0);

public static Rectangle footleft = new Rectangle(0, 0, 0, 0);

public static Rectangle footright = new Rectangle(0, 0, 0, 0);

private Background bg1 = StartingClass.getBg1();

private Background bg2 = StartingClass.getBg2();

private Background bg3 = StartingClass.getBg3();

private Background bg4 = StartingClass.getBg4();

private ArrayList<Projectile> projectiles = new ArrayList<Projectile>();

public void update() {

if (speedX == 0) {

bg1.setSpeedX(0);

bg2.setSpeedX(0);

bg3.setSpeedX(0);

bg4.setSpeedX(0);

}

if (centerX <= 800 && speedX > 0) {

centerX += speedX;

}

if (centerX >= 400 && speedX < 0) {

centerX += speedX;

}

if (speedX > 0 && centerX > 800) {

bg1.setSpeedX(-MOVESPEED / 5);

bg2.setSpeedX(-MOVESPEED / 5);

bg3.setSpeedX(-MOVESPEED / 5);

bg4.setSpeedX(-MOVESPEED / 5);

}

if (speedX < 0 && centerX < 400) {

bg1.setSpeedX(MOVESPEED / 5);

bg2.setSpeedX(MOVESPEED / 5);

bg3.setSpeedX(MOVESPEED / 5);

bg4.setSpeedX(MOVESPEED / 5);

}

// Updates Y Position

centerY += speedY;

// Handles Jumping

speedY += 1;

if (speedY > 3) {

jumped = true;

}

rect.setRect(centerX - 55, centerY - 63, 35, 70);

rect2.setRect(rect.getX(), rect.getY() + 70, 35, 70);

yellowRed.setRect(centerX - 92, centerY - 80, 110, 180);

footleft.setRect(centerX - 60, centerY - 33, 5, 95);

footright.setRect(centerX - 20, centerY - 33, 5, 95);

}

public void moveRight() {

direction = "right";

if (ducked == false) {

speedX = MOVESPEED;

}

}

public void moveLeft() {

direction = "left";

if (ducked == false) {

speedX = -MOVESPEED;

}

}

public void stopRight() {

setMovingRight(false);

stop();

}

public void stopLeft() {

setMovingLeft(false);

stop();

}

private void stop() {

if (isMovingRight() == false && isMovingLeft() == false) {

speedX = 0;

}

if (isMovingRight() == false && isMovingLeft() == true) {

moveLeft();

}

if (isMovingRight() == true && isMovingLeft() == false) {

moveRight();

}

}

public void jump() {

if (jumped == false) {

speedY = JUMPSPEED;

jumped = true;

}

}

public void shoot() {

Projectile p;

if (getDirection() == "right")

p = new Projectile(centerX - 5, centerY - 5, true);

else

p = new Projectile(centerX - 5, centerY - 5, false);

projectiles.add(p);

}

public int getCenterX() {

return centerX;

}

public int getCenterY() {

return centerY;

}

public boolean isJumped() {

return jumped;

}

public int getSpeedX() {

return speedX;

}

public int getSpeedY() {

return speedY;

}

public void setCenterX(int centerX) {

this.centerX = centerX;

}

public void setCenterY(int centerY) {

this.centerY = centerY;

}

public void setJumped(boolean jumped) {

this.jumped = jumped;

}

public void setSpeedX(int speedX) {

this.speedX = speedX;

}

public void setSpeedY(int speedY) {

this.speedY = speedY;

}

public boolean isDucked() {

return ducked;

}

public void setDucked(boolean ducked) {

this.ducked = ducked;

}

public boolean isMovingRight() {

return movingRight;

}

public void setMovingRight(boolean movingRight) {

this.movingRight = movingRight;

}

public boolean isMovingLeft() {

return movingLeft;

}

public void setMovingLeft(boolean movingLeft) {

this.movingLeft = movingLeft;

}

@SuppressWarnings("rawtypes")

public ArrayList getProjectiles() {

return projectiles;

}

public static String getDirection() {

return direction;

}

}

Класс Tile.java

package Game;

import java.awt.Image;

import java.awt.Rectangle;

public class Tile {

private int tileX, tileY, speedX, type;

public Image tileImage;

private Robot robot = StartingClass.getRobot();

private Background bg = StartingClass.getBg1();

private Rectangle r;

public Tile(int x, int y, int typeint) {

tileX = x \* 40;

tileY = y \* 40;

type = typeint;

r = new Rectangle();

if (type == 2)

tileImage = StartingClass.grasstop;

else if (type == 3)

tileImage = StartingClass.tiledirt;

else if (type == 4)

tileImage = StartingClass.tilestone;

else if (type == 5)

tileImage = StartingClass.tiletree;

else if (type == 6)

tileImage = StartingClass.tilerock;

else {

type = 0;

}

}

public void update() {

speedX = bg.getSpeedX() \* 5;

tileX += speedX;

r.setBounds(tileX, tileY, 40, 40);

if (r.intersects(Robot.yellowRed) && type != 0) {

checkVerticalCollision(Robot.rect, Robot.rect2);

checkSideCollision(Robot.footleft, Robot.footright);

}

for (EnemyDynamic i : EnemyDynamic.enemies) {

if (r.intersects(i.r) && (type == 2 || type == 3 || type == 6)) {

i.setSpeedX(0);

if (i.getCenterX() < tileX)

i.setCenterX(tileX - 55);

else if (i.getCenterX() > tileX)

i.setCenterX(tileX + 56);

}

}

}

public int getTileX() {

return tileX;

}

public void setTileX(int tileX) {

this.tileX = tileX;

}

public int getTileY() {

return tileY;

}

public void setTileY(int tileY) {

this.tileY = tileY;

}

public Rectangle getRectangle() {

return r;

}

public int getType() {

return type;

}

public Image getTileImage() {

return tileImage;

}

public void setTileImage(Image tileImage) {

this.tileImage = tileImage;

}

public void checkVerticalCollision(Rectangle rtop, Rectangle rbot) {

if (type == 2 || type == 3 || type == 6) {

if (rtop.intersects(r)) {

robot.setCenterY(tileY + 100);

robot.setSpeedY(-robot.JUMPSPEED);

}

if (rbot.intersects(r)) {

robot.setJumped(false);

robot.setSpeedY(0);

robot.setCenterY(tileY - 75);

}

}

}

public void checkSideCollision(Rectangle leftfoot, Rectangle rightfoot) {

if (type == 2 || type == 3 || type == 6) {

if (leftfoot.intersects(r)) {

robot.setCenterX(tileX + 100);

robot.setSpeedX(0);

} else if (rightfoot.intersects(r)) {

robot.setCenterX(tileX + 15);

robot.setSpeedX(0);

}

}

}

}

Класс Win.java

**package** Game;

**import** java.awt.event.MouseEvent;

**public** **class** Win **extends** DynamicBackground {

**public** **double** orbitX = -110; /\* x-coordinate in orbit's center \*/

**public** **double** orbitY = -60; /\* y-coordinate in orbit's center \*/

@Override

**public** **void** update() {

radian = orbitSpeed \* t;

drawX = orbitX + orbitRadius \* Math.*cos*(radian);

drawY = orbitY + orbitRadius \* Math.*sin*(radian);

t += 1;

}

**void** mousePress(MouseEvent e) {

**int** mx = e.getX();

**int** my = e.getY();

StartingClass.*restart*();

**if** (mx > 60 && mx < 210 && my > 350 && my < 390) {

StartingClass.*State* = "menu";

} **else** **if** (mx > 60 && mx < 210 && my > 400 && my < 440) {

StartingClass.*State* = "game";

}

}

}

Класс WinBlock.java

**package** Game;

**import** java.util.ArrayList;

**public** **class** WinBlock **extends** EnemyStatic{

**public** **static** ArrayList<WinBlock> *wins* = **new** ArrayList<WinBlock>();

**public** WinBlock(**int** centerX, **int** centerY) {

**super**(centerX, centerY);

}

// Behavioral Methods

**public** **static** **void** update() {

**for** (WinBlock i : *wins*) {

i.centerX += i.speedX;

i.speedX = i.bg.getSpeedX() \* 5 + i.movementSpeed;

i.r.setBounds(i.centerX - 30, i.centerY - 10, 85, 60);

**if** (i.r.intersects(Robot.*yellowRed*))

i.checkCollision();

}

}

**protected** **void** checkCollision() {

**if** (r.intersects(Robot.*rect*) || r.intersects(Robot.*rect2*)) {

StartingClass.*State* = "win";

}

}

}

Класс StartingClass.java

package Game;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.awt.image.BufferedImage;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class StartingClass extends JPanel implements Runnable, KeyListener, MouseListener {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private boolean ctrl\_press;

ArrayList<?> projectiles;

static StartingClass starter = new StartingClass();

private static Robot robot;

private BufferedImage currentSprite, c0, c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, s0, s1, s2, s3, s4, s5, s6,

s7, s8, s9, s10, s11, crouch\_r0, crouch\_r1, crouch\_r2, crouch\_r3, crouch\_l0, crouch\_l1, crouch\_l2,

crouch\_l3, menu, bulletr, bulletl, characterJumped\_r, characterJumped\_l, background1, background2, el, e2l,

er, e2r, play, exit, backButton, introback, introfront, guideback,

guidefront, guideButton, introButton, start, deathimg, winimg, menubutton, playagain;

private static Background bg1, bg2, bg3, bg4;

public Animation anim\_r, anim\_l, anim, anim\_still, anim\_still\_r, anim\_still\_l, crouchdown\_r, crouchdown\_r1,

crouchdown\_l;

public static Animation hanim\_l, hanim\_r;

public static BufferedImage tiledirt, grasstop, tilestone, tiletree, tilerock;

private static ArrayList<Tile> tilearray = new ArrayList<Tile>();

static String State = "start";

Menu MENU;

Intro INTRO;

Guide GUIDE;

Death DEATH;

Win WIN;

public static void restart() {

robot.getProjectiles().clear();

tilearray.clear();

EnemyDynamic.enemies.clear();

WinBlock.wins.clear();

EnemyStatic.enemies.clear();

bg1 = new Background(0, 0);

bg2 = new Background(1920, 0);

bg3 = new Background(-1920, 0);

bg4 = new Background(-1920 \* 2, 0);

robot = new Robot();

try {

starter.loadMap("data/map1.txt");

} catch (IOException e) {

e.printStackTrace();

}

}

public void init() throws Exception {

setFocusable(true);

addKeyListener(this);

addMouseListener(this);

// Image Setups

characterJumped\_r = ImageIO.read(new File("data/jumpedr.png"));

characterJumped\_l = ImageIO.read(new File("data/jumpedl.png"));

background1 = ImageIO.read(new File("data/bg1.png"));

background2 = ImageIO.read(new File("data/bg2.png"));

tiledirt = ImageIO.read(new File("data/tiledirt.png"));

grasstop = ImageIO.read(new File("data/grasstop.png"));

tilestone = ImageIO.read(new File("data/tilestone.png"));

tiletree = ImageIO.read(new File("data/tiletree.png"));

tilerock = ImageIO.read(new File("data/tilerock.png"));

bulletr = ImageIO.read(new File("data/bulletr.png"));

bulletl = ImageIO.read(new File("data/bulletl.png"));

c0 = ImageIO.read(new File("data/walk/walk-r (1).png"));

c1 = ImageIO.read(new File("data/walk/walk-r (2).png"));

c2 = ImageIO.read(new File("data/walk/walk-r (3).png"));

c3 = ImageIO.read(new File("data/walk/walk-r (4).png"));

c4 = ImageIO.read(new File("data/walk/walk-r (5).png"));

c5 = ImageIO.read(new File("data/walk/walk-r (6).png"));

c6 = ImageIO.read(new File("data/walk/walk-r (7).png"));

c7 = ImageIO.read(new File("data/walk/walk-r (8).png"));

c8 = ImageIO.read(new File("data/walk/walk-r (9).png"));

c9 = ImageIO.read(new File("data/walk/walk-r (10).png"));

c10 = ImageIO.read(new File("data/walk/walk-r (11).png"));

c11 = ImageIO.read(new File("data/walk/walk-r (12).png"));

s0 = ImageIO.read(new File("data/walk/0.png"));

s1 = ImageIO.read(new File("data/walk/1.png"));

s2 = ImageIO.read(new File("data/walk/2.png"));

s3 = ImageIO.read(new File("data/walk/3.png"));

s4 = ImageIO.read(new File("data/walk/4.png"));

s5 = ImageIO.read(new File("data/walk/5.png"));

s6 = ImageIO.read(new File("data/walk/6.png"));

s7 = ImageIO.read(new File("data/walk/7.png"));

s8 = ImageIO.read(new File("data/walk/8.png"));

s9 = ImageIO.read(new File("data/walk/9.png"));

s10 = ImageIO.read(new File("data/walk/10.png"));

s11 = ImageIO.read(new File("data/walk/11.png"));

crouch\_r0 = ImageIO.read(new File("data/crouch/r0.png"));

crouch\_r1 = ImageIO.read(new File("data/crouch/r1.png"));

crouch\_r2 = ImageIO.read(new File("data/crouch/r2.png"));

crouch\_r3 = ImageIO.read(new File("data/crouch/r3.png"));

crouch\_l0 = ImageIO.read(new File("data/crouch/1.png"));

crouch\_l1 = ImageIO.read(new File("data/crouch/2.png"));

crouch\_l2 = ImageIO.read(new File("data/crouch/3.png"));

crouch\_l3 = ImageIO.read(new File("data/crouch/4.png"));

// Menu Images

menu = ImageIO.read(new File("data/menu.png"));

play = ImageIO.read(new File("data/play.png"));

exit = ImageIO.read(new File("data/exit.png"));

guideButton = ImageIO.read(new File("data/guideButton.png"));

introButton = ImageIO.read(new File("data/introButton.png"));

start = ImageIO.read(new File("data/intro1.png"));

deathimg = ImageIO.read(new File("data/deathimg.png"));

winimg = ImageIO.read(new File("data/winimg.png"));

menubutton = ImageIO.read(new File("data/menuButton.png"));

playagain = ImageIO.read(new File("data/playagainButton.png"));

backButton = ImageIO.read(new File("data/backButton.png"));

// Intro stuff

introback = ImageIO.read(new File("data/introback.png"));

introfront = ImageIO.read(new File("data/introfront.png"));

// Guide stuff

guideback = ImageIO.read(new File("data/guideback.png"));

guidefront = ImageIO.read(new File("data/guidefront.png"));

el = ImageIO.read(new File("data/heliboy.png"));

e2l = ImageIO.read(new File("data/heliboy2.png"));

er = ImageIO.read(new File("data/heliboyl.png"));

e2r = ImageIO.read(new File("data/heliboy2l.png"));

hanim\_l = new Animation(false);

hanim\_l.addFrame(el, 500);

hanim\_l.addFrame(e2l, 500);

hanim\_r = new Animation(false);

hanim\_r.addFrame(er, 500);

hanim\_r.addFrame(e2r, 500);

crouchdown\_r = new Animation(true);

crouchdown\_r.addFrame(crouch\_r0, 20);

crouchdown\_r.addFrame(crouch\_r1, 20);

crouchdown\_r.addFrame(crouch\_r2, 20);

crouchdown\_r.addFrame(crouch\_r3, 20);

crouchdown\_l = new Animation(true);

crouchdown\_l.addFrame(crouch\_l0, 20);

crouchdown\_l.addFrame(crouch\_l1, 20);

crouchdown\_l.addFrame(crouch\_l2, 20);

crouchdown\_l.addFrame(crouch\_l3, 20);

anim\_r = new Animation(false);

anim\_r.addFrame(c0, 50);

anim\_r.addFrame(c1, 50);

anim\_r.addFrame(c2, 50);

anim\_r.addFrame(c3, 50);

anim\_r.addFrame(c4, 50);

anim\_r.addFrame(c5, 50);

anim\_r.addFrame(c6, 50);

anim\_r.addFrame(c7, 50);

anim\_r.addFrame(c8, 50);

anim\_r.addFrame(c9, 50);

anim\_r.addFrame(c10, 50);

anim\_r.addFrame(c11, 50);

anim\_l = new Animation(false);

anim\_l.addFrame(s0, 50);

anim\_l.addFrame(s1, 50);

anim\_l.addFrame(s2, 50);

anim\_l.addFrame(s3, 50);

anim\_l.addFrame(s4, 50);

anim\_l.addFrame(s5, 50);

anim\_l.addFrame(s6, 50);

anim\_l.addFrame(s7, 50);

anim\_l.addFrame(s8, 50);

anim\_l.addFrame(s9, 50);

anim\_l.addFrame(s10, 50);

anim\_l.addFrame(s11, 50);

anim = anim\_r;

currentSprite = c0;

}

public void start() {

bg1 = new Background(0, 0);

bg2 = new Background(1920, 0);

bg3 = new Background(-1920, 0);

bg4 = new Background(-1920 \* 2, 0);

robot = new Robot();

MENU = new Menu();

INTRO = new Intro();

GUIDE = new Guide();

DEATH = new Death();

WIN = new Win();

// Initialize Tiles

try {

loadMap("data/map1.txt");

}

catch (IOException e) {

e.printStackTrace();

}

Thread thread = new Thread(this);

thread.start();

}

@Override

public void run() {

while (true) {

switch (State) {

case "start":

break;

case "menu":

MENU.update();

break;

case "dead":

DEATH.update();

break;

case "win":

WIN.update();

break;

case "intro":

INTRO.update();

break;

case "guide":

GUIDE.update();

break;

case "game":

gameUpdate();

break;

}

repaint();

try {

Thread.sleep(17);

}

catch (InterruptedException e) {

e.printStackTrace();

}

}

}

@SuppressWarnings("static-access")

public void gameUpdate() {

robot.update();

bg1.update();

bg2.update();

bg3.update();

bg4.update();

EnemyDynamic.update();

EnemyStatic.update();

WinBlock.update();

updateTiles();

hanim\_l.update(50);

hanim\_r.update(50);

@SuppressWarnings("rawtypes")

ArrayList projectiles = robot.getProjectiles();

for (int i = 0; i < projectiles.size(); i++) {

Projectile p = (Projectile) projectiles.get(i);

if (p.isVisible() == true) {

p.update();

} else {

projectiles.remove(i);

}

}

for (int i = 0; i < EnemyDynamic.enemies.size(); i++) {

if (EnemyDynamic.enemies.get(i).getIsDead())

EnemyDynamic.enemies.remove(i);

}

for (int i = 0; i < EnemyStatic.enemies.size(); i++) {

if (EnemyStatic.enemies.get(i).getIsDead())

EnemyStatic.enemies.remove(i);

}

if (robot.isJumped() && robot.getDirection() == "right") {

currentSprite = characterJumped\_r;

}

else if (robot.isJumped() && robot.getDirection() == "left") {

currentSprite = characterJumped\_l;

}

else if ((robot.getDirection() == "right") && (robot.getSpeedX() == 0) && robot.isDucked() == false) {

currentSprite = c0;

}

else if ((robot.getDirection() == "left") && (robot.getSpeedX() == 0) && robot.isDucked() == false) {

currentSprite = s0;

}

else {

if (robot.getSpeedX() < 0) {

anim = anim\_l;

}

else if (robot.getSpeedX() > 0) {

anim = anim\_r;

}

else if (robot.isDucked() && robot.getDirection() == "right") {

anim = crouchdown\_r;

}

else if (robot.isDucked() && robot.getDirection() == "left") {

anim = crouchdown\_l;

}

currentSprite = anim.getImage();

anim.update(10);

}

if (robot.getCenterY() > 1400) {

State = "dead";

}

}

@Override

public void paint(Graphics g) {

switch (State) {

case "start":

g.drawImage(start, 0, 0, this);

break;

case "menu":

try {

g.drawImage(menu, (int) MENU.drawX, (int) MENU.drawY, this);

g.drawImage(play, 60, 250, this);

g.drawImage(introButton, 60, 300, this);

g.drawImage(guideButton, 60, 350, this);

g.drawImage(exit, 60, 450, this);

}

catch (Exception e) {

}

break;

case "dead":

g.drawImage(deathimg, (int) DEATH.drawX, (int) DEATH.drawY, this);

g.drawImage(menubutton, 60, 350, this);

g.drawImage(playagain, 60, 400, this);

break;

case "win":

g.drawImage(winimg, (int) WIN.drawX, (int) WIN.drawY, this);

g.drawImage(menubutton, 60, 350, this);

g.drawImage(playagain, 60, 400, this);

break;

case "intro":

g.drawImage(introback, (int) INTRO.drawX, (int) INTRO.drawY, this);

g.drawImage(introfront, 52, 70, this);

g.drawImage(backButton, 20, 700, this);

break;

case "guide":

g.drawImage(guideback, (int) GUIDE.drawX, (int) GUIDE.drawY, this);

g.drawImage(guidefront, 478, 170, this);

g.drawImage(backButton, 20, 700, this);

break;

case "game":

g.drawImage(background1, bg1.getBgX(), bg1.getBgY(), this);

g.drawImage(background2, bg2.getBgX(), bg2.getBgY(), this);

g.drawImage(background2, bg3.getBgX(), bg3.getBgY(), this);

g.drawImage(background1, bg4.getBgX(), bg4.getBgY(), this);

paintTiles(g);

paintProjectiles(g);

paintEnemies(g);

paintEnemiesStatic(g);

g.drawImage(currentSprite, robot.getCenterX() - 61, robot.getCenterY() - 63, this);

break;

}

}

private void updateTiles() {

for (int i = 0; i < tilearray.size(); i++) {

Tile t = (Tile) tilearray.get(i);

t.update();

}

}

private void paintEnemies(Graphics g) {

for (EnemyDynamic i : EnemyDynamic.enemies) {

if (i.getCenterX() > -50 && i.getCenterX() < 1366) {

if (i.direction == "right")

g.drawImage(hanim\_r.getImage(), i.getCenterX() - 48, i.getCenterY() - 48, this);

else if (i.direction == "left")

g.drawImage(hanim\_l.getImage(), i.getCenterX() - 48, i.getCenterY() - 48, this);

}

}

}

private void paintEnemiesStatic(Graphics g) {

for (EnemyStatic i : EnemyStatic.enemies) {

if (i.getCenterX() > -50 && i.getCenterX() < 1366) {

if (i.direction == "right")

g.drawImage(hanim\_r.getImage(), i.getCenterX() - 48, i.getCenterY() - 48, this);

else if (i.direction == "left")

g.drawImage(hanim\_l.getImage(), i.getCenterX() - 48, i.getCenterY() - 48, this);

}

}

}

@SuppressWarnings("static-access")

private void paintProjectiles(Graphics g) {

projectiles = robot.getProjectiles();

for (int i = 0; i < projectiles.size(); i++) {

Projectile p = (Projectile) projectiles.get(i);

if (robot.getDirection() == "right")

g.drawImage(bulletr, p.getX(), p.getY(), this);

else if (robot.getDirection() == "left")

g.drawImage(bulletl, p.getX(), p.getY(), this);

}

}

private void paintTiles(Graphics g) {

for (int i = 0; i < tilearray.size(); i++) {

Tile t = (Tile) tilearray.get(i);

g.drawImage(t.getTileImage(), t.getTileX(), t.getTileY(), this);

}

}

@SuppressWarnings({ "unchecked", "unused" })

private void loadMap(String filename) throws IOException {

@SuppressWarnings("rawtypes")

ArrayList lines = new ArrayList();

int width = 0;

int height = 0;

BufferedReader reader = new BufferedReader(new FileReader(filename));

while (true) {

String line = reader.readLine();

// no more lines to read

if (line == null) {

reader.close();

break;

}

if (!line.startsWith("!")) {

lines.add(line);

width = Math.max(width, line.length());

}

}

height = lines.size();

for (int j = 0; j < 20; j++) {

String line = (String) lines.get(j);

for (int i = 0; i < width; i++) {

if (i < line.length()) {

char ch = line.charAt(i);

if (ch == '\*') {

EnemyDynamic.enemies.add(new EnemyDynamic(i \* 40, j \* 40 - 80));

} else if (ch == '+') {

EnemyStatic.enemies.add(new EnemyStatic(i \* 40, j \* 40 - 80));

} else if (ch == '=') {

WinBlock.wins.add(new WinBlock(i \* 40, j \* 40 - 80));

} else{

Tile t = new Tile(i, j, Character.getNumericValue(ch));

tilearray.add(t);

}

}

}

}

}

@Override

public void keyPressed(KeyEvent e) {

if (State == "start") {

if (e.getKeyCode() == KeyEvent.VK\_SPACE)

State = "menu";

}

if (State == "intro" || State == "guide") {

switch (e.getKeyCode()) {

case KeyEvent.VK\_ESCAPE:

State = "menu";

break;

}

}

if (State == "game") {

switch (e.getKeyCode()) {

case KeyEvent.VK\_ESCAPE:

State = "menu";

break;

case KeyEvent.VK\_UP:

System.out.println("Move up");

break;

case KeyEvent.VK\_DOWN:

robot.setDucked(true);

break;

case KeyEvent.VK\_LEFT:

robot.moveLeft();

break;

case KeyEvent.VK\_RIGHT:

robot.moveRight();

break;

case KeyEvent.VK\_SPACE:

robot.jump();

break;

case KeyEvent.VK\_ENTER:

State = "game";

break;

case KeyEvent.VK\_CONTROL:

ctrl\_press = true;

}

}

}

@Override

public void keyReleased(KeyEvent e) {

switch (e.getKeyCode()) {

case KeyEvent.VK\_UP:

break;

case KeyEvent.VK\_DOWN:

if (State == "game") {

crouchdown\_r.currentFrame = 0;

crouchdown\_l.currentFrame = 0;

robot.setDucked(false);

}

break;

case KeyEvent.VK\_LEFT:

if (State == "game")

robot.stopLeft();

break;

case KeyEvent.VK\_RIGHT:

if (State == "game")

robot.stopRight();

break;

case KeyEvent.VK\_SPACE:

if (State == "game") {

robot.setMovingLeft(false);

robot.setMovingRight(false);

}

break;

case KeyEvent.VK\_CONTROL:

if (State == "game") {

if (ctrl\_press) {

robot.shoot();

}

}

break;

}

}

@Override

public void keyTyped(KeyEvent e) {

}

@Override

public void mouseClicked(MouseEvent me) {

}

@Override

public void mousePressed(MouseEvent me) {

switch (State) {

case "dead":

DEATH.mousePress(me);

break;

case "win":

WIN.mousePress(me);

break;

case "menu":

MENU.mousePress(me);

break;

case "intro":

INTRO.mousePress(me);

break;

case "guide":

GUIDE.mousePress(me);

}

}

@Override

public void mouseReleased(MouseEvent me) {

if (State == "menu")

MENU.mousePress = false;

if (State == "intro")

MENU.mousePress = false;

if (State == "guide")

GUIDE.mousePress = false;

}

@Override

public void mouseEntered(MouseEvent me) {

}

@Override

public void mouseExited(MouseEvent me) {

}

// Getters

public static Background getBg1() {

return bg1;

}

public static Background getBg2() {

return bg2;

}

public static Background getBg3() {

return bg3;

}

public static Background getBg4() {

return bg4;

}

public static Robot getRobot() {

return robot;

}

@SuppressWarnings("rawtypes")

public static ArrayList getTileArray() {

return tilearray;

}

// Main Function

public static void main(String[] args) {

JFrame frame = new JFrame("LOST");

frame.setSize(1366, 768);

frame.setBackground(Color.black);

frame.add(starter);

frame.setUndecorated(true);

frame.setVisible(true);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

try {

starter.init();

starter.start();

}

catch (Exception e) {

e.printStackTrace();

}

}

}