Научное общество учащихся «Эврика»

Муниципальное бюджетное общеобразовательное учреждение

«Лицей №38»

Советского района г. Нижнего Новгорода

**Создание 2D игры на языке программирования Java.**

Выполнил: Паршин Даниил

ученик 9 класса

Научный руководитель:

Попова Н. Л.,

учитель информатики

Борисов Н.А.,

доцент кафедры программной инженерии

ННГУ им. Н.И.Лобачевского

Нижний Новгород

2020

Оглавление

[Введение 3](#_Toc34854983)

[1. Графика и написание спрайтов. 4](#_Toc34854984)

[2. Программы и инструменты для работы с графикой 7](#_Toc34854985)

[3. Производительность игры и требуемые ресурсы. 8](#_Toc34854986)

[4. Оценки и пожелания пользователей 10](#_Toc34854987)

[5. Сравнение с прототипом. 12](#_Toc34854988)

[6. Заключение. 13](#_Toc34854989)

[Список литературы. 14](#_Toc34854990)

# Введение

Знание языков программирования сегодня востребовано как никогда. Многие из нас мечтают связать свою дальнейшую жизнь с IT технологиями и погрузиться с головой в данную индустрию. Мы можем воплотить свои мечты и фантазии в реальность при помощи строк кода, написав какую-либо игру, мессенджер, приложение, сайт... Разве это не интересно? Сначала будет нелегко, возникнет много вопросов и сомнений, ошибок в коде и т.д. Но потом приятно получить результат – программу, которая работает по твоему алгоритму!

В качестве проекта я выбрал написание 2D игры. Игровая индустрия очень развита, написана масса игр в 3D-графике, как профессиональных, так и любительских, но нас привлекла старая советская электронная игра «Ну, погоди!» («Электроника ИМ-02»), созданная в 1984 году, это была самая известная и популярная игра из серии первых советских портативных игр.

**Объекты исследования:**

Объектами исследования являются: графический редактор AdobeIllustrator, язык программирования Java, среда разработки AndroidStudio и фреймворк для создания игр и приложений LibGDX1[[1]](#footnote-1).

**Задачи и цели:**

* Разобраться в программах, позволяющих создать данный проект, а именно: получить опыт работы с написанием 2D игр, научиться создавать простейшие векторные анимации.
* Проанализировать пожелания пользователя и максимально реализовать их в данном проекте.
* Создать игру, способную корректно работать на различных платформах (iOS, Android, Windows).
* Добиться максимальной производительности, реализовав метод очистки ненужных элементов в коде.
* Опробовать ООП2[[2]](#footnote-2) на практике.

**При проведении исследования применялись следующие методы:**

* Моделирование и измерение.
* Анализ и синтез.
* Изучение и сопоставление

**Значимость работы:**

Мы предоставляем пользователю возможность погрузится в прошлое с использованием новых технологий, поиграть в игру детства 80-х годов. Игра с таким же игровым процессом, но с измененным дизайном (основой дизайна являлся мультфильм «Wall-E3»).[[3]](#footnote-3)

**В процессе разработки:**

Спроектирована и создана 2D игра.

Получен опыт разработки приложений на языке Java с помощью AndroidStudio и LibGDX.

Научился создавать собственную игровую графику.

Приобретен опыт векторного рисования (использование геометрических примитивов).

Приобретен опыт использования графического редактора AdobeIllustrator.

В результате получилась хорошая и легко модифицируемая игра.

# 1. Графика и написание спрайтов.

Перед нами стояла задача сделать игру интересной увлекательной. Важно было создать качественную анимацию и графику. Поэтому первым этапом разработки являлось написание спрайт-листа. Было решено прорисовывать элементы интерфейса вручную, так как варианты, подобранные в интернете, не сочетались друг с другом. Начинающим художникам и дизайнерам, специализирующимся на играх и гейм-дизайне, стоит придерживаться следующего правила – стараться не использовать в палитре более шести цветов. Это позволяет создавать изображения наиболее гармоничными в восприятии. Перспективны в этом смысле программы векторной графики. Эти программы пользуются популярностью, так как многие 2D игры используют их. Например, Серия GeometryDash, SuperMarioRun, Badland и другие…

Для придания объема и реалистичности каждой детали изображения и лучшего визуального эффекта использовалась градиентная заливка, а также двухцветная заливка одного объекта для придания элементу тени (рис.1). В игру были добавлены уровни и новые элементы (рис. 2), который делятся на три типа: «нейтральные» (элементы, упрощающие игровой процесс, например, понижающие скорость падения метеоритов или отрицательных элементов...), «положительные» (элементы, которые игрок обязан поймать, иначе игра начнется заново, более того, эти элементы добавляют игроку очки) и «отрицательные» (элементы, которые игрок не должен ловить, соприкосновение персонажа с данным элементом приводит к game over). «Отрицательным» элементом является метеорит, в котором для создания эффекта горящего «хвоста» была реализована покадровая анимация (рис 1). Анимация была реализована следующим образом: создавался массив типа Texture, куда загружались спрайты одинакового размера, но с разным углом направления пламени. Чем больше кадров, тем лучше качество анимация. Затем массив в *методе render* перебирался по порядку с определенной частотой. И как только мы считывали последний индекс, то сразу приравнивали его 0, то есть считывали заново. В итоге получилась анимация, походящая на игру пламени. При контакте с поверхностью происходит взрыв. Здесь уже использовалась спрайт-анимация, которая реализуется хоть и сложнее, но объем кода намного меньше. Примеры других созданных объектов представлены на рис 1, 2, 3, 4:

|  |  |  |
| --- | --- | --- |
|  |  | |
| *Рис.1* | | |
|  |  | |
| *Рис.2* | | |
|  | |  |
| *Рис.3* | | |
|  | |  |
| *Рис.4* | | |

# 2. Программы и инструменты для работы с графикой

В работе была задействована лишь одна программа — AdobeIllustrator 2020. Нарисованные изображения экспортировались в PNG-формат без хинтовки. То есть в параметрах указывалось сглаживание пикселей, так как изображение теряло качество при его отрисовки в Anroid Studio графическим редактором (метод draw). Конечным этапом является помещение нарисованного спрайта в папку Assets, расположенную в нашем проекте. Там хранятся все ресурсы игры (медиа, звуки, фотографии и др.) (рис. 5). Можно непосредственно ссылаться на сам ресурс через его адрес.

|  |
| --- |
|  |
| *Рис 5.* |

Помимо графики, в программе загружены и аудио ресурсы в формате mp3. Аудио ресурсы делятся на два типа: Sound (как правило короткие) и Music (заметно больше и по объему, и по длительности воспроизведения). В игре используются различные звуковые сопровождения для каждого действия. Это позволило сделать игру интереснее и разнообразить игровой процесс, избавившись тем самым от монотонности.

# 3. Производительность игры и ресурсы.

Но стоит помнить, что чем больше графики и выше ее качество, тем больше нагрузка на устройство, а именно: повышается объем ненужных ресурсов, которые засоряют устройство и заставляют игру тормозить. Это, конечно же, влияет и на самого пользователя – у него пропадает желание играть в ваш продукт. Следовательно, каждому разработчику стоит обращать внимание на производительность, энергопотребление, частоту кадров (FPS) и объем памяти своего приложения. Речь сейчас идет больше об играх, так как именно там акцент идет на графику и ее качество. Поэтому, передо мной стояла максимально высвободить неиспользуемые ресурсы игры. Существуют приложения, способные проанализировать весь процесс работы программы, предоставляя разработчику сведения о количестве обращений пользователя к экрану, объеме энергии, затраченном за какое-либо время, скорости выполнения, количество кадров в секунду и т. п. Приведем пример использования ресурсов при работе игры в течение 1 минуты, представленных в виде диаграммы (рис. 6):

|  |
| --- |
|  |
| *Рис. 6* |

Мы видим, что большая часть памяти выделяется под графику. То есть метод очистки памяти не успевает очищать ненужные сцены, происходит переполнение хранилища, а в связи с этим и «лаги» программы. Пожалуй, это одна из самых больших и важных проблем в разработке. Для наглядности разобьем нашу диаграмму на несколько подразделов, таких как: CPU, MEMORY и ENERGY (рис. 7). Если провести анализ этого графика, то можно выявить следующие особенности:

|  |
| --- |
|  |
| Рис. 7 |

Позже, эта проблема была решена методом объявления Start\_Resource, где вызывались все пакеты, файлы и ресурсы лишь ОДИН РАЗ посредством переменной типа boolean. Это позволило добиться стабильных 201 Мб, а не 1.7 Гб. Как итог – программа перестала «вылетать» и «тормозить» на мобильных устройствах. Но все же, данная ошибка ярко демонстрирует последствия неправильно написанного кода, где инструмент Profider позволяет с легкостью контролировать каждое действие, выполненное программой. Память устройства и способы ее грамотного использования являются важными этапами в разработке, особенно играх. Именно поэтому я выбрал среду разработки AndroidStudio, которая предоставляет обширный набор инструментов и возможностей.

# 4. Оценки и пожелания пользователей

Для изучения мнения пользователей о предоставленной игре, был проведен опрос. Количество респондентов, опрошенных по теме составило 35 человек (наша фокус-группа). Опросник состоял из двух частей: нас интересовало отношение пользователя к технической составляющей игры (графика и анимация) и ее контентному наполнению (удовлетворенность персонажами, уровнями сложности и т.д.). Этот опрос необходим для того, чтобы выявить сильные и слабые стороны, проанализировать ошибки и постараться их устранить. Результаты опроса представлены диаграммами (рис.8,9)

|  |
| --- |
|  |
| *Рис.8* |

Из данных опроса видно, что наша игра, в целом, понравилась, наши друзья с удовольствием поиграли в игру их бабушек и дедушек. Я реализовал кнопки паузы и возобновления, что придало комфорт во время игры. Были добавлены уровни и звуки.

*Рис. 9*

В дальнейшем я буду улучшать игру, добавляя новых персонажей, более продуманный способ взаимодействия между игроком и персонажем (кнопки, джойстик и тому подобное...). Создание уровней сложности тоже не будет лишним, а наоборот, придаст еще больше сложности и интереса. В будущем, я постараюсь их реализовать как можно больше, сделав игру еще интереснее. Дальнейшей целью является переход от двумерной к трехмерной графике. Это позволит сделать игру еще реалистичнее, правдоподобнее. Самой главной цели я добился: сделали подарок своим близким и родным, которые смогут поиграть в игру их детства, но с более современным дизайном.

# 5. Сравнение с прототипом.

При сравнении интерфейса игры «Ну, погоди» (рис. 10) и нашей игры (рис. 11) хочется отметить следующее:

|  |
| --- |
|  |
| *Рис. 10* |

|  |
| --- |
|  |
| *Рис. 11* |

* Добавлено как сенсорное, так и управление при помощи физических клавиш.
* Добавлена пользовательская строка, которая намного информативнее (время, кол-во очков, и другое...).
* Добавлена анимация.
* Добавлены кнопки паузы и возобновления.
* Появилась возможность обмена данной игры по Bluetooth и запуск ее на различных устройствах.
* Появилась возможность замены как и графики, так и физики игры. Так как код в открытом доступе, то любой желающий может поучаствовать в создании и улучшении данного приложения, добавив свои изменения в программу.

# 6. Заключение.

В результате работы над проектом, было создана 2D игра, которая имела интересный и самобытный дизайн, соответствует большей части пользовательских ожиданий за исключением возможности вывода рейтинга лучших игроков и босса. Так же были решены задачи, поставленные перед выполнением проекта:

1. Изучены основы разработки 2D игр с использованием LibGDX.

2. Изучены основы верстки скриптов для 2D игры и их дальнейший экспорт в приложение. Получен опыт работы с AdobeIllustrator 2020.

3. Проанализированы пожелания пользователей, где были реализованы важные из них.

4. Разработана игра, в которой присутствуют базовые функции и геймплэй.

5. Изучены основы ООП и применение их на практике.

6. Создана простейшая покадровая анимация.

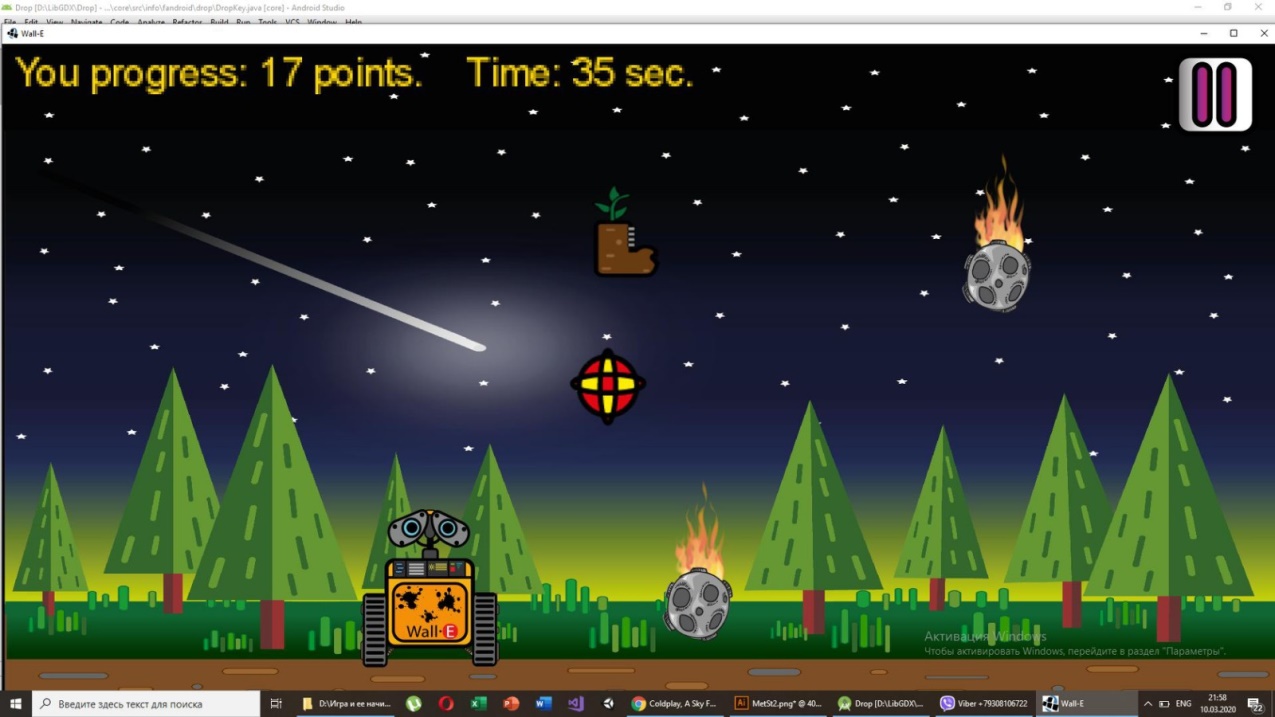
В дальнейшем я планирую расширить набор функций, предоставляемых моей программой по мере их разработки, например, реализовать возможность играть с другими игроками онлайн, перейти на 3D графику, реализовав ее в коде, добавить базу данных, где будут храниться результаты игроков.

# Список литературы.

1. Официальная страница LibGDX URL: <https://libgdx.badlogicgames.com/>
2. Статья по разработке приложения и гейм-дизайну URL: <https://o7planning.org/ru/10521/android-2d-game-tutorial-for-beginners>
3. Канал по разработке игр на Java URL: <https://www.youtube.com/channel/UCy8aONm_W21gtK2D8KKnINA>
4. 2D анимация с использование пакета LibGDX URL: <http://www.libgdx.ru/2014/09/2D-Animation.html>
5. Сайт разработчика и статьи по разработке URL: <https://developer.android.com/studio>

*Приложение 1.*

**Результат проекта**



*Приложение 2.*

**Листинг кода.**

Class DropKey.java:

**package** info.fandroid.drop;  
  
*// Процесс импортирования необходимых пакетов и файлов, классов, библиотек итд***import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Input;  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.audio.Music;  
**import** com.badlogic.gdx.audio.Sound;  
**import** com.badlogic.gdx.graphics.OrthographicCamera;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.GL20;  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch;  
**import** com.badlogic.gdx.math.MathUtils;  
**import** com.badlogic.gdx.math.Vector3;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.badlogic.gdx.utils.TimeUtils;  
**import** com.badlogic.gdx.math.Rectangle;  
**import** java.util.Iterator;  
**import** com.badlogic.gdx.graphics.g2d.Animation;  
**import** com.badlogic.gdx.graphics.g2d.TextureRegion;  
  
**public class** DropKey **implements** Screen { *//Создание и заполнение необходимых ресурсов* **public final int W** = Gdx.*graphics*.getWidth(); *//2100;* **public final int H** = Gdx.*graphics*.getHeight();*//1000;* **private int NUMB**;  
 **private int NUMB2**;  
 **private final** info.fandroid.drop.Drop **game**;  
 **private** OrthographicCamera **camera**;  
 **private** SpriteBatch **batch**;  
  
 **private** Texture **dropImage**; *// Игровые объекты* **private** Texture **drop2Image**;  
 **private** Texture **DOPING**;  
 **private** Texture **dropImage2**;  
  
 **private** Texture **st1**;  
 **private** Texture **st2**;  
 **private** Texture **st3**;  
 **private** Texture **st4**;  
 **private** Texture **st5**;  
 **private** Texture **st6**;  
  
 **private** Texture **gamoverImage**; *// Изображения состояния игры* **private** Texture **winImage**;  
 **private** Texture **newLevelImage**;  
  
 **private** Texture **heroImageMain**; *// Ссылки на ресурсы* **private** Texture **mainImage**;  
  
 **private** Texture **heroImage**; *// Текстыры персонажа* **private** Texture **heroImageL**;  
 **private** Texture **heroImageR**;  
  
 **private** Texture **areaImage**; *// Сцена* **private** Texture **progressImage**;  
  
 **private** Texture **retryImage**; *// Кнопки* **private** Texture **mainImagePause**;  
 **private** Texture **mainImageReplay**;  
  
 **private** Sound **dropSound**; *// Медиа итп* **private** Sound **doppingSound**;  
 **private** Music **loseMusic**;  
 **private** Music **backGroundnMusic**;  
 **private** Music **winMusic**;  
 **private** Rectangle **hero**;  
 **private** Vector3 **touchPos**;  
 **private** Array<Rectangle> **A\_BunchOfPlants**;  
 **private** Array<Rectangle> **A\_BunchOfRubbish**;  
 **private** Array<Rectangle> **A\_BunchOfDopping**;  
 **private** Array<Texture> **AnimationMeteor**;  
 **private** Rectangle **RetryState**;  
 **private** Rectangle **Cursor**;  
  
 **private long lastDropTime**;  
 **private long lastDropTime2**;  
 **private long lastDoppingTime**;  
 **private long ButtCheck**;  
 **private long timeCheck**;  
 **private long pointsCheck**;  
 **private long animationCheck**;  
 **private long timeCheckImage**;  
  
 **private boolean KEY** = **false**;  
 **private boolean END** = **false**;  
 **private boolean WIN** = **false**;  
 **private boolean LOSE** = **false**;  
 **private boolean BUTTONCHANGE** = **false**;  
 **private boolean ThreadStop** = **false**;  
 **private boolean NewLevel** = **false**;  
 **private boolean CheckLevel** = **false**;  
 **private boolean Boom** = **false**;  
  
 **private float rotate** = 360.0f;  
 **private int speed** = 350;  
 **private int CountLevel** = 0;  
 **private int speedAut**;  
 **private int animationFse** = 0;  
 **private int TIME** = 0;  
 **private int dropsKeys** = 0;  
 **private int random**;  
 **private int SpeedDifference** = 100;  
 **private int AddSpace** = 180;  
 **private int WinScore** = 50;  
 *//----------------------------------------* **private static final int *FRAME\_COLS*** = 5;  
 **private static final int *FRAME\_ROWS*** = 2;  
 **private** Animation **walkAnimation**;  
 **private** Texture **walkSheet**;  
 **private** TextureRegion[] **walkFrames**;  
 **private** TextureRegion **currentFrame**;  
 **private float stateTime**;  
 **private long BoomCh**;  
 **private int coordinates**;  
 **private int index** = 0;  
  
 *//-------------------------------------------------------* DropKey(**final** Drop gam) { *// Создание игры (объекта) и настройка камеры* **this**.**game** = gam;  
  
 **camera** = **new** OrthographicCamera();  
 **camera**.setToOrtho(**false**, **W**, **H**);  
  
 **batch** = **new** SpriteBatch();  
 **touchPos** = **new** Vector3();  
  
 *//---------------------------------------------------------- Загрузка PNG ресурсов* **areaImage** = **new** Texture(**"Area.png"**);  
 **progressImage** = **new** Texture(**"ProgressTable.png"**);  
  
 **heroImage** = **new** Texture(**"WalleDefault.png"**);  
 **heroImageL** = **new** Texture(**"WalleLeft.png"**);  
 **heroImageR** = **new** Texture(**"WalleRight.png"**);  
  
 **dropImage** = **new** Texture(**"cactus.png"**);  
 **dropImage2** = **new** Texture(**"Ognet.png"**);  
 **drop2Image** = **new** Texture(**"MetSt0.png"**);  
 **DOPING** = **new** Texture(**"Dopping.png"**);  
  
 **gamoverImage** = **new** Texture(**"GameOver.png"**);  
 **retryImage** = **new** Texture(**"RetryButton.png"**);  
 **mainImagePause** = **new** Texture(**"PauseButton.png"**);  
 **mainImageReplay** = **new** Texture(**"ReplayButton.png"**);  
 **winImage** = **new** Texture(**"winImage.png"**);  
 **newLevelImage** = **new** Texture(**"newLevel.png"**);  
  
 **st1** = **new** Texture(**"MetSt2.png"**);  
 **st2** = **new** Texture(**"MetSt3.png"**);  
 **st3** = **new** Texture(**"MetSt4.png"**);  
 **st4** = **new** Texture(**"MetSt5.png"**);  
 **st5** = **new** Texture(**"MetSt6.png"**);  
 **st6** = **new** Texture(**"MetSt7.png"**);  
 **walkSheet** = **new** Texture(Gdx.*files*.internal(**"boom.png"**)); *// #9  
  
 //--------------------------------------------- Загрузка аудиофайлов (ресурсов)* **dropSound** = Gdx.*audio*.newSound(Gdx.*files*.internal(**"dropSound.mp3"**));  
 **doppingSound** = Gdx.*audio*.newSound(Gdx.*files*.internal(**"DoppingSound.mp3"**));  
 **backGroundnMusic** = Gdx.*audio*.newMusic(Gdx.*files*.internal(**"AskyMusic.mp3"**));  
 **loseMusic** = Gdx.*audio*.newMusic(Gdx.*files*.internal(**"loseMusic.mp3"**));  
 **winMusic** = Gdx.*audio*.newMusic(Gdx.*files*.internal(**"musicWin.mp3"**));  
  
 TextureRegion[][] tmp = TextureRegion.*split*(**walkSheet**, **walkSheet**.getWidth()/***FRAME\_COLS***, **walkSheet**.getHeight()/***FRAME\_ROWS***); *// #10* **walkFrames** = **new** TextureRegion[***FRAME\_COLS*** \* ***FRAME\_ROWS***];  
 **for** (**int** i = 0; i < ***FRAME\_ROWS***; i++) {  
 **for** (**int** j = 0; j < ***FRAME\_COLS***; j++) {  
 **walkFrames**[**index**++] = tmp[i][j];  
 }  
 }  
 **walkAnimation** = **new** Animation(0.06f, **walkFrames**);  
 **stateTime** = 0f;  
  
 **heroImageMain** = **heroImage**; *// Выставление текстур по умолчанию* **mainImage** = **mainImagePause**;  
  
 **backGroundnMusic**.setLooping(**true**);  
 **backGroundnMusic**.play();  
 **backGroundnMusic**.setVolume(0.3f);  
  
 **hero** = **new** Rectangle(); *// Главный персонаж, создание и его координирование* **hero**.**x** = **W** / 2 - 195 / 2;  
 **hero**.**y** = 60;  
 **hero**.**width** = 170;  
 **hero**.**height** = 220;  
  
 **RetryState** = **new** Rectangle(); *// Кнопка начала игры заново* **RetryState**.**width** = 150;  
 **RetryState**.**height** = 150;  
 **RetryState**.**x** = **W** / 2 - 50;  
 **RetryState**.**y** = **H** - 400;  
  
 **Cursor** = **new** Rectangle(); *// Создание активной области курсора (пользовательского ввода)* **Cursor**.**width** = 45;  
 **Cursor**.**height** = 45;  
  
 **A\_BunchOfPlants** = **new** Array<Rectangle>(); *// Создание массивов игровых объектов* spawnPlant();  
 **A\_BunchOfRubbish** = **new** Array<Rectangle>();  
 spawnRubbish();  
 **A\_BunchOfDopping** = **new** Array<Rectangle>();  
  
 **timeCheck** = TimeUtils.*nanoTime*();  
 **timeCheckImage** = TimeUtils.*nanoTime*();  
 **pointsCheck** = TimeUtils.*nanoTime*();  
 **animationCheck** = TimeUtils.*millis*();  
  
 **AnimationMeteor** = **new** Array<Texture>();  
  
 **AnimationMeteor**.add(**st1**);  
 **AnimationMeteor**.add(**st2**);  
 **AnimationMeteor**.add(**st3**);  
 **AnimationMeteor**.add(**st4**);  
 **AnimationMeteor**.add(**st5**);  
 **AnimationMeteor**.add(**st6**);  
 }  
 *//------------------------------------------------------* **private void** spawnPlant() { *// Создание полей и обЪекта РАСТЕНИЕ* Rectangle PlantUnit = **new** Rectangle();  
  
 **if** (!**KEY**) {  
 PlantUnit.**x** = **W** / 2;  
 PlantUnit.**y** = **H** + 200;  
 }  
 **if** (**KEY**) {  
  
 PlantUnit.**x** = MathUtils.*random*(300, **W** - 300 - 100); *//Ключ начала игры* **NUMB** = (**int**) PlantUnit.**x**;  
 PlantUnit.**y** = **H**;  
 }  
 PlantUnit.**width** = 64;  
 PlantUnit.**height** = 64;  
  
 **if** (!**ThreadStop**) {  
 **A\_BunchOfPlants**.add(PlantUnit);  
 **lastDropTime** = TimeUtils.*nanoTime*();  
 }  
 } *// Создание Растения  
  
 //----------------------------------------------------------------------------------* **private void** spawnRubbish() { *// Создание полей и обЪекта МУСОР* Rectangle RubbishUnit = **new** Rectangle();  
 RubbishUnit.**y** = 1800;  
  
 **if** (!**KEY**) RubbishUnit.**x** = MathUtils.*random*(100, **W** - 200); *// Вступительная инициализация* **if** (**KEY**) {  
 RubbishUnit.**x** = MathUtils.*random*(100, **W** - 200);  
  
 **if** (RubbishUnit.**x** < **NUMB** + **AddSpace** && RubbishUnit.**x** > **NUMB** - **AddSpace**) { *//проверка на положение мусора в поле + его последующее положеник* **random** = MathUtils.*random*(0,2);  
 **switch** (**random**) {  
 **case** 1: RubbishUnit.**x** = MathUtils.*random*(100, **NUMB** - **AddSpace**); **break**;  
 **case** 2: RubbishUnit.**x** = MathUtils.*random*(**NUMB** + **AddSpace**, **W** - 200); **break**;  
 }  
 }  
 **if** (RubbishUnit.**x** == **NUMB**) RubbishUnit.**x** = MathUtils.*random*(100, **NUMB** - **AddSpace** - 100);  
  
 *//--------------------------------------------------------------------------------------* **if** (RubbishUnit.**x** > **W** - 200) { *// Последующая проверка и генерирование координат Мусора* RubbishUnit.**x** = MathUtils.*random*(100 + 300, **W** / 2);  
 }  
  
 **if** (RubbishUnit.**x** < 100) {  
 RubbishUnit.**x** = MathUtils.*random*(**W** / 2, **W** - 200 - 300);  
 }  
 }  
  
 RubbishUnit.**width** = 75;  
 RubbishUnit.**height** = 65;  
  
 **NUMB2** = (**int**) RubbishUnit.**x**;  
  
 **if** (!**ThreadStop**) {  
 **A\_BunchOfRubbish**.add(RubbishUnit);  
 **lastDropTime2** = TimeUtils.*nanoTime*();  
 }  
 }  
  
 **private void** spawnDoping() {  
  
 Rectangle DoppingUnit = **new** Rectangle();  
 DoppingUnit.**y** = **H** + 100;  
  
 DoppingUnit.**x** = MathUtils.*random*(100, **W** - 200);  
  
 **if**( DoppingUnit.**x** > **NUMB2** - **AddSpace** && DoppingUnit.**x** < **NUMB2** + **AddSpace**)  
  
 **random** = MathUtils.*random*(0,3);  
  
 **switch** (**random**) {  
 **case** 1:  
 DoppingUnit.**x** = MathUtils.*random*(**NUMB** + **AddSpace**, **W** - 200);  
 **break**;  
 **case** 2:  
 DoppingUnit.**x** = MathUtils.*random*(100, **NUMB2** - **AddSpace**);  
 **break**;  
 }  
  
 DoppingUnit.**width** = 64;  
 DoppingUnit.**height** = 64;  
  
 **if**(!**ThreadStop**) {  
 **A\_BunchOfDopping**.add(DoppingUnit);  
 **lastDoppingTime** = TimeUtils.*nanoTime*();  
 }  
 }  
 *//----------------------------------------------------* @Override  
 **public void** render(**float** delta) {  
  
 Gdx.*graphics*.getGL20().glClearColor(0.3f,0.3f,0.3f,1); *// Очистка игрового поля, закрашивание его в черный цвет для последующей анимации* Gdx.*gl*.glClear(GL20.***GL\_COLOR\_BUFFER\_BIT*** | GL20.***GL\_DEPTH\_BUFFER\_BIT***);  
  
 *//---------------------------------------------------------- Периодическая очистка массивов* **if**(**A\_BunchOfPlants**.**size** > 500){  
 **A\_BunchOfPlants**.clear();  
 }  
 **if**(**A\_BunchOfRubbish**.**size** > 500) {  
 **A\_BunchOfRubbish**.clear();  
 }  
 **if**(**A\_BunchOfDopping**.**size** > 500) {  
 **A\_BunchOfDopping**.clear();  
 }  
 *//----------------------------------------------------------* **camera**.update();  
 **game**.**batch**.setProjectionMatrix(**camera**.**combined**);  
 **game**.**batch**.begin();  
  
 **if**(**END**) { *// Отрисовка при проигрыше* **game**.**batch**.draw(**areaImage**, 0, 0, Gdx.*graphics*.getWidth(), Gdx.*graphics*.getHeight());  
 **game**.**batch**.draw(**mainImage**, **W** / 4, **H** / 3 + 150, **W** / 2, **H** / 3 + 70);  
 **game**.**batch**.draw(**retryImage**, **W** / 2 - 80, **H** / 4);  
 **game**.**batch**.draw(**heroImageMain**, **hero**.**x**, **hero**.**y**);  
  
 **if** (TimeUtils.*nanoTime*() - **timeCheck** > 800000000) { *// Анимация мигания надписи результатов* **game**.**font**.draw(**game**.**batch**, **"You progress: "** + **dropsKeys** + **" points."**, 20, **H** - 20);  
 **game**.**font**.draw(**game**.**batch**, **"Time: "** + **TIME** + **" sec."**, 700, **H** - 20);  
  
 **if** ((TimeUtils.*nanoTime*() - **timeCheck**) > 1800000000) {  
 **timeCheck** = 0;  
 **timeCheck** = TimeUtils.*nanoTime*();  
 }  
 }  
 }  
 **if** (!**END**) { *// Стандартная отрисовка* **game**.**batch**.draw(**areaImage**, 0, 0, Gdx.*graphics*.getWidth(), Gdx.*graphics*.getHeight()); *//Создание заднего фона* **game**.**font**.draw(**game**.**batch**, **"You progress: "** + **dropsKeys** + **" points."**, 20, **H** - 20);  
 **game**.**font**.draw(**game**.**batch**, **"Time: "** + **TIME** + **" sec."**, 700, **H** - 20);  
 **game**.**batch**.draw(**mainImage**, **W** - 160, **H** - 140);  
 **game**.**batch**.draw(**heroImageMain**, **hero**.**x**, **hero**.**y**);  
  
 BOOM();  
  
 **if** (TimeUtils.*nanoTime*() - **timeCheck** > 1000000000 && !**END**) {  
 **TIME**++;  
 **timeCheck** = 0;  
 **timeCheck** = TimeUtils.*nanoTime*();  
  
 }  
  
 **if**(**dropsKeys** == 30) {  
 **NewLevel** = **true**;  
 **ButtCheck** = 0;  
 }  
  
 **if** (**dropsKeys** == 31) {  
 **NewLevel** = **false**;  
 **dropImage** = **dropImage2**;  
 }  
  
 **if**(**TIME** < 67 && **dropsKeys** > 29) {  
  
 **if** (TimeUtils.*nanoTime*() - **ButtCheck** < 300000000) { *// Анимация мигания надписи* **game**.**batch**.draw(**newLevelImage**, **W** / 4 - 100, -100, 1200, 1000);  
 }  
 **if** ((TimeUtils.*nanoTime*() - **ButtCheck**) > 700000000) {  
 **ButtCheck** = 0;  
 **ButtCheck** = TimeUtils.*nanoTime*();  
 }  
 }  
  
 **if** (TimeUtils.*millis*() - **animationCheck** > 40) {  
  
 **animationCheck** = 0;  
 **animationCheck** = TimeUtils.*millis*();  
 **animationFse**++;  
  
 **if** (**animationFse** == 6) **animationFse** = 0;  
 }  
  
 *//-------------------------------------------------------------* **for** (Rectangle PlantUnit : **A\_BunchOfPlants**) { *// Отрисовка и очистка массива РАСТЕНИЙ* **game**.**batch**.draw(**dropImage**, PlantUnit.**x**, PlantUnit.**y**);  
 */\* game.batch.draw(dropImage, PlantUnit.x, PlantUnit.y, PlantUnit.getWidth() ,  
 PlantUnit.getHeight() , PlantUnit.getWidth() 1.5f ,  
 PlantUnit.getHeight() 1.8f, 1f, 1f, rotate,  
 0, 0, dropImage.getWidth(), dropImage.getHeight(),  
 false, false);  
 rotate--; \*/  
  
  
 // if (rotate < 0) rotate = 360.0f;* **if** (**A\_BunchOfPlants**.**size** > 20) {  
 **for** (**int** a = 0; a <= 10; a++) {  
 **A\_BunchOfPlants**.removeIndex(a);  
 }  
 }  
 }  
  
 **for** (Rectangle RubbishUnit : **A\_BunchOfRubbish**) { *// Отрисовка и очистка массива МУСОРА* **drop2Image** = **AnimationMeteor**.get(**animationFse**);  
 **game**.**batch**.draw(**drop2Image**, RubbishUnit.**x**, RubbishUnit.**y**);  
  
 **if** (**A\_BunchOfRubbish**.**size** > 20) {  
 **for** (**int** b = 0; b <= 10; b++) {  
 **A\_BunchOfRubbish**.removeIndex(b);  
 }  
 }  
 }  
  
 **for** (Rectangle DoppingUnit : **A\_BunchOfDopping**) { *// Отрисовка и очистка массива МУСОРА* **game**.**batch**.draw(**DOPING**, DoppingUnit.**x**, DoppingUnit.**y**);  
  
 **if** (**A\_BunchOfDopping**.**size** > 20) {  
 **for** (**int** r = 0; r <= 10; r++) {  
 **A\_BunchOfDopping**.removeIndex(r);  
 }  
 }  
 }  
 }  
 **game**.**batch**.end();  
 *//-------------------------------------------- Мониторинг нажатий и обработка направления движения* **if** (!**END** && !**ThreadStop**) {  
  
 **if** (Gdx.*input*.isTouched() && Gdx.*input*.getX() < **W** / 2) {  
  
 **touchPos**.set(Gdx.*input*.getX(), Gdx.*input*.getY(), 0);  
 **camera**.unproject(**touchPos**);  
 **heroImageMain** = **heroImageL**;  
 **hero**.**x** -= 16;  
  
 }  
  
 **if** (Gdx.*input*.isTouched() && Gdx.*input*.getX() > **W** / 2 && **Cursor**.**y** > 150) {  
  
 **touchPos**.set(Gdx.*input*.getX(), Gdx.*input*.getY(), 0);  
 **camera**.unproject(**touchPos**);  
 **heroImageMain** = **heroImageR**;  
 **hero**.**x** += 16;  
  
 }  
  
 **if** ((!Gdx.*input*.isTouched() || (Gdx.*input*.isKeyPressed(Input.Keys.***LEFT***) && Gdx.*input*.isKeyPressed(Input.Keys.***RIGHT***))) && (TimeUtils.*nanoTime*() - **timeCheckImage** > 1000000000)) {  
 **heroImageMain** = **heroImage**;  
 **timeCheckImage** = 0;  
 }  
 *//---------------------------------------------* **if** (Gdx.*input*.isKeyPressed(Input.Keys.***LEFT***)) {  
  
 **heroImageMain** = **heroImageL**;  
 **hero**.**x** -= 750 \* Gdx.*graphics*.getDeltaTime();  
 }  
  
 **if** (Gdx.*input*.isKeyPressed(Input.Keys.***RIGHT***)) {  
  
 **heroImageMain** = **heroImageR**;  
 **hero**.**x** += 750 \* Gdx.*graphics*.getDeltaTime();  
 }  
 }  
  
 **if**(Gdx.*input*.getX() == **W** / 2) **heroImageMain** = **heroImage**;  
  
 *//------------------------------------- Уровни и дополнительные анимации* **if** (**dropsKeys** == **WinScore**) {  
  
 **mainImage** = **winImage**;  
 **WIN** = **true**;  
  
 **try** {  
 GamState();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 *//-------------------------- Отслеживание границ игрового поля* **if** (**hero**.**x** < 0) **hero**.**x** = 0;  
 **if** (**hero**.**x** > **W** - 190) **hero**.**x** = **W** - 190;  
  
 *//----------------------------------------------------- Рассчет координаты пользовательского ввода и обработка кнопки* **Cursor**.**x** = Gdx.*input*.getX();  
 **Cursor**.**y** = Gdx.*input*.getY();  
  
 **if** ((**Cursor**.**x** > **W** - 200 && **Cursor**.**y** < 200) && (Gdx.*input*.isTouched() || Gdx.*input*.isCursorCatched()) && !**BUTTONCHANGE** && !**END** && ((TimeUtils.*nanoTime*() - **ButtCheck** > 500000000))) {  
  
 **mainImage** = **mainImageReplay**;  
 **BUTTONCHANGE** = **true**;  
 **ThreadStop** = **true**;  
 **ButtCheck** = 0;  
 **ButtCheck** = TimeUtils.*nanoTime*();  
  
 pause();  
 }  
  
 **if** ((**Cursor**.**x** > **W** - 200 && **Cursor**.**y** < 200) && (Gdx.*input*.isTouched() || Gdx.*input*.isCursorCatched()) && **BUTTONCHANGE** && !**END** && (TimeUtils.*nanoTime*() - **ButtCheck** > 500000000)) {  
  
 **mainImage** = **mainImagePause**;  
 **BUTTONCHANGE** = **false**;  
 **ThreadStop** = **false**;  
 **ButtCheck** = 0;  
 **ButtCheck** = TimeUtils.*nanoTime*();  
  
 resume();  
 }  
 *//---------------------------------------------------- Создание нового мусора, процесс подсчета времени* **if** ((TimeUtils.*nanoTime*() - **lastDropTime** > 2100000000 && !**ThreadStop**) && !**NewLevel**) { *//Создание нового сапога, процесс подсчета времени* **lastDropTime** = 0;  
 **KEY** = **true**;  
 spawnPlant();  
 }  
 **if** ((TimeUtils.*nanoTime*() - **lastDropTime2** > 1300000000 && !**ThreadStop**) && !**NewLevel**) {  
  
 **lastDropTime2** = 0;  
 **KEY** = **true**;  
 spawnRubbish();  
 }  
 **if**((TimeUtils.*nanoTime*() / 20 - **lastDoppingTime** / 20 > 1800000000 && !**ThreadStop** && **dropsKeys** > 15) && !**NewLevel**) {  
  
 **lastDoppingTime** = 0;  
 spawnDoping();  
 }  
  
 *//------------------------------------ Создание анимаций и движения объектов* **if** (!**ThreadStop** || !**END**) {  
  
 Iterator<Rectangle> iterator = **A\_BunchOfPlants**.iterator();  
  
 **while** ( iterator.hasNext()) {  
  
 Rectangle PlantUnit = iterator.next();  
  
 **if** (!**END** && !**ThreadStop**) {  
 PlantUnit.**y** -= (**speed** - **SpeedDifference**) \* Gdx.*graphics*.getDeltaTime();  
 }  
  
 **if** (PlantUnit.**y** + 64 < 0) {  
  
 **mainImage** = **gamoverImage**;  
 **LOSE** = **true**;  
 **try** { GamState(); }  
 **catch** (InterruptedException e) { e.printStackTrace();}  
 }  
  
 **if** (PlantUnit.overlaps(**hero**)) {  
  
 **dropsKeys**++;  
 **dropSound**.play();  
 **speed** += 3;  
 iterator.remove();  
 }  
 }  
  
 Iterator<Rectangle> iterator2 = **A\_BunchOfRubbish**.iterator();  
 **while** ( iterator2.hasNext()) {  
  
 Rectangle RubbishUnit = iterator2.next();  
  
 **if** (!**END** && !**ThreadStop**) {  
 RubbishUnit.**y** -= **speed** \* Gdx.*graphics*.getDeltaTime();  
 }  
  
 **if**(RubbishUnit.**y** <= 100) {  
 **coordinates** = (**int**) RubbishUnit.**x**;  
 }  
  
 **if** (RubbishUnit.**y** < 10) {  
 iterator2.remove();  
 **Boom** = **true**;  
 **BoomCh** = 0;  
 **BoomCh** = TimeUtils.*nanoTime*();  
 }  
  
 **if**(**Boom** && TimeUtils.*nanoTime*() - **BoomCh** > 510000000) {  
 **Boom** = **false**;  
 **stateTime** = 0f;  
 }  
  
 **if** (RubbishUnit.overlaps(**hero**)) {  
  
 **mainImage** = **gamoverImage**;  
 **LOSE** = **true**;  
 **try** { GamState(); }  
 **catch** (InterruptedException e) { e.printStackTrace(); }  
 }  
 }  
  
  
 Iterator<Rectangle> iterator3 = **A\_BunchOfDopping**.iterator();  
 **while** ( iterator3.hasNext()) {  
  
 Rectangle DoppingUnit = iterator3.next();  
  
 **if** (!**END** && !**ThreadStop**) {  
 DoppingUnit.**y** -= (**speed** - 80) \* Gdx.*graphics*.getDeltaTime();  
 }  
  
 **if** (DoppingUnit.**y** < -300) {  
 iterator3.remove();  
 }  
  
 **if** (DoppingUnit.overlaps(**hero**)) {  
  
 iterator3.remove();  
 **speed** -= 40;  
 **doppingSound**.play(0.15f);  
 }  
 }  
 }  
 }  
 *//-----------------------------------------------------------------* @Override  
 **public void** resize(**int** width, **int** height) { **camera**.setToOrtho(**false**);}  
  
 @Override  
 **public void** pause() {  
  
 **backGroundnMusic**.pause();  
 **speedAut** = **speed**;  
 **speed** = 0;  
 **ThreadStop** = **true**;  
 **lastDropTime2** = 0;  
 **lastDropTime** = 0;  
 **lastDoppingTime** = 0;  
 }  
  
 @Override  
 **public void** resume() {  
  
 **backGroundnMusic**.play();  
 **speed** = **speedAut**;  
 **speedAut** = 0;  
 **ThreadStop** = **false**;  
 **lastDropTime2** = 0;  
 **lastDropTime** = 0;  
 **lastDoppingTime** = 0;  
 }  
  
 @Override  
 **public void** hide() {}  
  
 **private void** BOOM() {  
 **if**(**coordinates** != 0 && **Boom**) {  
 **stateTime** += Gdx.*graphics*.getDeltaTime();  
 **for** (**int** i = 0; i < ***FRAME\_ROWS*** ; i++) {  
 **for** (**int** j = 0; j < ***FRAME\_COLS***; j++) {  
 **currentFrame** = (TextureRegion) **walkAnimation**.getKeyFrame(**stateTime**, **true**);  
 **game**.**batch**.draw(**currentFrame**, **coordinates** - 25, 20);  
 }  
 }  
 }  
 }  
  
 **private void** GamState() **throws** InterruptedException {  
  
 **END** = **true**;  
 **dropSound**.dispose();  
 **backGroundnMusic**.dispose();  
 show();  
  
 *//////////////////// Работа с кнопками* **if**((**Cursor**.overlaps(**RetryState**) && (Gdx.*input*.isTouched() || Gdx.*input*.isCursorCatched())) ) {  
  
 dispose();  
 **game**.setScreen(**new** DropKey(**game**));  
 **camera**.update();  
 }  
 }  
 *///////////////////////* @Override  
 **public void** dispose() { *// Очистка памяти* **dropImage**.dispose();  
 **drop2Image**.dispose();  
 **DOPING**.dispose();  
 **heroImage**.dispose();  
 **heroImageL**.dispose();  
 **heroImageR**.dispose();  
 **dropSound**.dispose();  
 **doppingSound**.dispose();  
 **backGroundnMusic**.dispose();  
 **areaImage**.dispose();  
 **loseMusic**.dispose();  
 **winMusic**.dispose();  
 **A\_BunchOfRubbish**.clear();  
 **A\_BunchOfPlants**.clear();  
 **A\_BunchOfDopping**.clear();  
 **AnimationMeteor**.clear();  
 **st1**.dispose();  
 **st2**.dispose();  
 **st3**.dispose();  
 **st4**.dispose();  
 **st5**.dispose();  
 **st6**.dispose();  
 **walkSheet**.dispose();  
  
 }  
 @Override  
 **public void** show() { *// Фоновая музыка и ее воспроизведение* **if**(**WIN**) **winMusic**.play();  
 **if**(**LOSE**) **loseMusic**.play();  
 **else backGroundnMusic**.play();  
 }  
}

Class MainMenuScreen.java:

package info.fandroid.drop;  
  
import com.badlogic.gdx.Gdx;  
import com.badlogic.gdx.Screen;  
import com.badlogic.gdx.graphics.OrthographicCamera;  
import com.badlogic.gdx.graphics.Texture;  
  
public class MainMenuScreenimplements Screen {  
  
final info.fandroid.drop.Dropgame;  
OrthographicCameracamera;  
static Texture areaImage;  
  
public MainMenuScreen (final info.fandroid.drop.Drop gam) {  
this.game= gam;  
camera = new OrthographicCamera();  
camera.setToOrtho(false, Gdx.graphics.getWidth(),Gdx.graphics.getHeight()); //2100 1000  
}  
  
@Override  
public void show() {}  
  
@Override  
publicvoidrender(floatdelta) {  
  
areaImage= newTexture("Area.png");  
camera.update();  
game.batch.setProjectionMatrix(camera.combined);  
  
game.batch.begin();  
  
game.batch.draw(areaImage, 0,0, Gdx.graphics.getWidth(), Gdx.graphics.getHeight()); //Созданиезаднегофона

game.font2.draw(game.batch, "WELCOME TO WALL-E!", 790,850);  
game.font2.draw(game.batch, "BE ATTANTION: DO NOT LOSE A SINGLE POINT!", 450,750);

game.font2.draw(game.batch, "IF YOU ARE READY, THEN TAB TO PLAY THE GAME! ", 400,650);

game.font2.draw(game.batch, "GOOD LUCK!!! ", 900,550);  
  
game.batch.end();  
  
if(Gdx.input.justTouched() || Gdx.input.isCursorCatched()) {

dispose();  
game.setScreen(new DropKey(game));  
 }  
 }  
  
@Override  
public void resize(int width, int height) {}  
  
@Override  
public void pause() { }  
  
@Override  
public void resume() {}  
  
@Override  
public void hide() {}  
  
@Override  
public void dispose() {}  
}

Class Drop.java:

package info.fandroid.drop;  
  
import com.badlogic.gdx.Game;  
import com.badlogic.gdx.graphics.Color;  
import com.badlogic.gdx.graphics.g2d.BitmapFont;  
import com.badlogic.gdx.graphics.g2d.SpriteBatch;  
  
public class Drop extends Game {  
  
SpriteBatchbatch;  
BitmapFontfont;  
BitmapFontfont2;  
  
@Override  
public void create() {  
  
  
batch = new SpriteBatch(); // Настройкацветаиразмерашрифта  
font = new BitmapFont();  
font2 = new BitmapFont();  
font.setColor(Color.GOLD);  
font2.setColor(Color.GOLD);  
font.getData().setScale(4,4);  
font2.getData().setScale(4,4);  
this.setScreen(new MainMenuScreen(this));  
 }  
  
@Override  
public void render() { // Отображение  
super.render();  
 }  
  
@Override  
public void dispose() { // Очистка  
super.dispose();  
batch.dispose();  
font.dispose();  
font2.dispose();  
 }  
}

1. Ф[реймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для создания игр и приложений, позволяет писать кроссплатформенные игры и приложения используя один код. [↑](#footnote-ref-1)
2. Объектно-ориентированное программирование. [↑](#footnote-ref-2)
3. Полнометражный компьютерный анимационный научно-фантастический фильм 2008года, созданныйPixarAnimationStudios. [↑](#footnote-ref-3)