ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ДОПУСТИТЬ К ЗАЩИТЕ
Профессор с возложенными обязанностями заведующего Кафедрой информационных систем в искусстве и гуманитарных науках

_____ (Борисов Н.В.)
“_____”_____20__ г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Направление 09.03.03 «Прикладная информатика»
Уровень Бакалавриат
Основная образовательная программа
«Прикладная информатика в области искусств и гуманитарных наук»

На тему
«*Разработка компьютерной игры «Unknown» на языке Java»*

Студентки *Козловой Анны Сергеевны*

_____
(подпись студента)

Руководитель: *канд. тех. наук, доцент, Посов Илья Александрович*

_____
(подпись руководителя)

Рецензент: *ген. директор, ООО «Центр мультимедиа 360», Столяров Денис Андреевич*

_____
(подпись рецензента)

Санкт-Петербург
2017

АННОТАЦИЯ

выпускной квалификационной работы

_____
(фамилия, имя, отчество)

название выпускной квалификационной работы
_____
_____

Отчет 37ст., 27 изображений, 11 источников.
ПРОГРАММИРОВАНИЕ, JAVA, КОМПЬЮТЕРНЫЕ ИГРЫ

Цель данной работы – получение навыков программирования на языке Java.

Задача – создание компьютерной игры «Неизведанное» по мотивам мультсериала «Over the Garden Wall».

В процессе работы использовались следующие программы - Java Development Kit (пакет разработчика на языке Java), IntelliJ IDEA (среда разработки программного обеспечения), Adobe Photoshop.

Автор работы _____ _____
подпись (фамилия, имя, отчество)

Руководитель работы _____ _____
подпись (фамилия, имя, отчество)

ОГЛАВЛЕНИЕ

ОПРЕДЕЛЕНИЯ

В данной работе используются следующие определения:

Класс – это элемент ПО, описывающий абстрактный тип данных и его частичную или полную реализацию.

Объект – 1) объект класса - «экземпляр» некоторого конкретного класса;

2) объект – термин, используемый для обозначения элементов произвольной категории.

Библиотека – сборник подпрограмм или объектов, используемых для разработки программного обеспечения (ПО).

ВВЕДЕНИЕ

Первые компьютерные игры появились еще на заре информационной эпохи, во времена ЭВМ. В современном мире игровая индустрия является наиболее быстрорастущей. Некоторые игры стали настолько популярны, что даже были перенесены на большой экран, вспомнить хотя бы Лару Крофт в исполнении Анджелины Джоли.

Целью данной выпускной квалификационной работы является получение навыков программирования на языке Java.

Для реализации этой цели была осуществлена попытка создать игру с нуля. Чаще всего начинающие разработчики используют готовые решения, коих сейчас достаточно, например «Unity».

Игра называется «Неизведанное», и относится к жанру 2D пазл-платформер.

Игра основана на мультсериале «Over the Garden Wall». В центре событий два брата, потерявшиеся в лесу. Старший из них, Вирт, является главным героем игры. Действие игры происходит во время последней серии мультфильма, которая также называется «Неизведанное». Главному герою нужно найти и спасти своего брата Грега из лап Зверя.

Так же вдохновением для меня стала игра «Limbo». От нее я позаимствовала жанр, концепцию и визуальный стиль.

Программный код на языке Java написан в среде разработки IntelliJ IDEA. Так же используется библиотека, о которой пойдет речь далее.

Жанр 2D платформер обычно означает, что локация на экране будет изображена сбоку, как бы в разрезе. Игроку дается возможность управлять

персонажем. Герой может бегать и прыгать при нажатии клавиш-стрелок. Для того чтобы пройти игру, нужно двигаться вперед и преодолевать препятствия. По пути будут встречаться разные предметы, некоторые из них можно взять в руки с помощью клавиши Shift. Так же герой может умереть, например, если упадет в яму. В таком случае игра перезапустится с предыдущей контрольной точки.

Рис. 1 Скриншот игры, начало

В конце игры герою предстоит встретиться с главным злодеем и победить его, чтобы спасти брата. По сюжету мультфильма Зверь предлагает Вирту сделку – заключить душу Грега в фонарь, и пока огонь в фонаре горит, Грег будет жив. Вирт почти соглашается, но потом понимает, что это обман. Он задумывается, почему этот фонарь так важен? Не потому ли, что это душа Зверя заключена в него? В мультфильме есть еще один персонаж, которого нет в игре, и он задувает фонарь. В игре это сделает сам Вирт. Зверь умирает, и ветки отпускают Грега.

Рис. 2 Скриншот игры, последняя сцена

Рис. 3 Скриншот игры, локация деревни

1. ФИЗИЧЕСКИЙ ДВИЖОК

Так как в данной игре особое внимание уделяется взаимодействию объектов, необходима была библиотека, симулирующая законы физики в реальном времени.

Была выбрана dyn4j, которая обеспечивает обработку соприкосновений объектов, оставляя возможность выбора графического рендера.

Физический движок позволяет создать некое виртуальное пространство, которое можно наполнить телами, и указать для него некие общие законы взаимодействия тел и среды, в той или иной мере приближенные к физическим, задавая при этом характер и степень взаимодействий. Так же можно контролировать такие величины как скорость, сила и импульс.

Тело — объект игровой физики, который определяется формой и набором параметров, таких как масса, плотность, коэффициент трения и другие.

Основная проблема, которую решает движок – так называемые коллизии, столкновения двух или более тел. Так как все тела в виртуальном мире – это наборы цифр, обнаружить что два объекта соприкоснулись можно только математическими расчетами. Движок берет на себя эти расчеты, и не допускает пересечения тел.

Между реальной и компьютерной физикой есть еще одно существенное различие. Реальная физика действует непрерывно, тогда как компьютерная, как и сам компьютер, действует дискретно, то есть ее нельзя вычислять непрерывно, необходимо разбить расчеты на шаги с определенным интервалом. Координаты тел меняются после каждого шага и объекты выводятся на экран.

2. СТРУКТУРА ПРОГРАММЫ

Запускающий класс создает фрейм, тот в свою очередь создает панель GamePanel, в которой находятся все основные действия.

Первым делом после создания панели запускается функция initializeWorld(), которая создает игровой мир и наполняет его объектами.

```
73      private void initializeWorld(double heroX, double heroY) throws IOException {
74          this.world = new World();
75          allObjects = new AllWorldGameObjects();
76
77          loadFloor();
78          for (Floor floor : floors) {
79              allObjects.addObject(floor);
80          }
81
82          kolobok = addGameObject(new Kolobok( radius: 0.5),  x: 21,  y: 6);
83          kolobok.body.applyForce(new Vector2( x: 100.0,  y: 0.0));
84
85          tree = addGameObject(new Tree( w: 1,  h: 14),  x: 64,  y: 10);
86          tree.body.rotate( theta: Math.PI / 6);
87
88          spikes = addGameObject(new InvisibleObject( w: 7,  h: 1.5),  x: 85.5,  y: 2);
89
90
91          greg = addGameObject(new Greg(),  x: 262,  y: 6);
92          branches = addGameObject(new Branches(),  x: 262,  y: 5.7);
93
94          beast = addGameObject(new Beast(),  x: 268,  y: 10);
95          hero = addGameObject(new Hero(world, allObjects), heroX, heroY);
96
97          light = addGameObject(new Light(),  x: 255,  y: 6);
98
99
100         listenCollisions(hero, spikes, () -> {
101                     if (hero.alive()) {
102                         hero.kill( death: true);
103                         hero.setMovementStart(System.nanoTime());
104                     }
105             }
106         );
107
108     }
```

Рис. 4 Фрагмент кода, функция initializeWorld()

Одновременно запускается бесконечный цикл gameLoop() в отдельном потоке, который будет постоянно считать текущее положение объектов в мире.

```
200     private void gameLoop() throws IOException {
201         long time = System.nanoTime();
202         long diff = time - this.last;
203 
204         this.last = time;
205         double elapsedTime = diff / NANO_TO_BASE; //сколько секунд прошло с прошлого раза
206 
207         if (hero.alive()) {
208             hero.move();
209             hero.processCarryButtonPress();
210         } else {
211             if (time - hero.getMovementStart() > 2e9) {
212                 Vector2 respawnPoint = getRespawnPoint();
213                 initializeWorld(respawnPoint.x, respawnPoint.y);
214             }
215         }
216 
217         camera.move(hero, elapsedTime);
218 
219         repaint();
220         this.world.update(elapsedTime);
221     }
```

Рис. 5 Фрагмент кода, функция gameLoop()

При каждом завершении этого цикла мир перерисовывается с помощью функции render().

```
242     private void render(Graphics2D g) {
243 
244         //определяем время, прошедшее с момента запуска программы, превращаем его в номер кадра.
245         // номер кадра передается как параметр draw() у всех GameObject
246         int frame = Timer.getFrameFrom(Timer.start);
247         Canvas canvas = new Canvas(g, camera);
248 
249         canvas.resetTransform();
250         camera.drawBackground(canvas);
251 
252 
253         for (GameObject object: allObjects.getList()){
254             canvas.transformBody(object.getBody());
255             object.draw(canvas, frame);
256             if (DEBUG_MODE) {
257                 object.drawDebug(canvas);
258                 setDebug(" coords: " + (int) hero.getBody().getWorldCenter().x + " , "
259                         + (int) hero.getBody().getWorldCenter().y); //только для героя
260             }
261         }
262 
263         canvas.kill();
264 
265         //рисуем текст дебаг
266         g.setFont(new Font( name: "Arial", Font.PLAIN,  size: 16));
267         g.drawString(debug,  x: 30,  y: 30);
268     }
```

Рис.6 Фрагмент кода, функция render()

3. ПЕРСОНАЖ

Персонаж является центром игры, именно через него игрок взаимодействует с миром. Также за ним постоянно следует камера.

Игровое тело героя это вертикальный прямоугольник с ограниченным вращением, чтобы персонаж не терял равновесие во время движения.

Рис. 7 Скриншот игры с включенным параметром DEBUG

Движется герой после нажатия кнопок вправо, влево и вверх или пробел для прыжка. Для реализации этой функции был создан слушатель клавиатуры HeroKeyListener, который регистрирует нажатия клавиш и меняет параметры, отвечающие за состояние героя в движении.

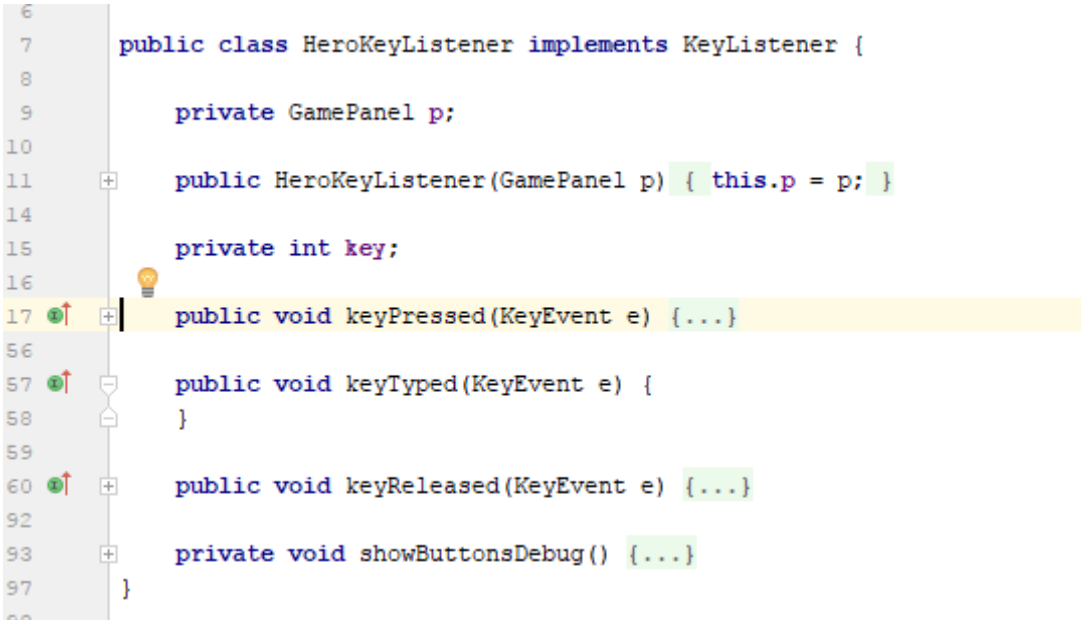

Рис. 8 Фрагмент кода, класс HeroKeyListener

Собственно смена положения героя происходит в методе move(), где расположены несколько вложенных условий, которые проверяют параметры на истинность или ложность, и придают импульсы физическому телу героя, если это необходимо.

```
130     public void move() {
131         Vector2 linearVelocity = body.getLinearVelocity();
132 
133         //идем вперед
134         if (go) {
135             right = true;
136             if (linearVelocity.x < 4) {  // тут было 3
137                 if (isHeroContactSomething())
138                     body.applyImpulse(new Vector2( x: 2,  y: 0));
139             }
140             if (carriedObject != null && !carriedObjectIsToTheRight)
141                 positionCarriedObject();
142         }
143 
144         //идем назад
145         if (goBack) {
146             right = false;
147             if (-4 < linearVelocity.x) {  // тут было -3
148                 if (isHeroContactSomething())
149                     body.applyImpulse(new Vector2( x: -2,  y: 0));
150             }
151             if (carriedObject != null && carriedObjectIsToTheRight)
152                 positionCarriedObject();
153         }
154 
155         //прыжок
156         if (jump) {
157             if (linearVelocity.y < 2)
158                 if (isHeroContactSomething()) {
159                     body.applyImpulse(new Vector2( x: 0,  y: 20)); // y был 15
160                 }
161         }
162 
163         //тормозим когда идем вперед
164         if (stopGo) {
165             if (linearVelocity.x < 0)
166                 body.setLinearVelocity( x: 0, linearVelocity.y);
167         }
168 
169         //тормозим когда идем назад
170         if (stopGoBack) {
171             if (0 < linearVelocity.x)
172                 body.setLinearVelocity( x: 0, linearVelocity.y);
173         }
174 
175         isHeroContactSomething = isHeroContactSomething();
176     }
```

Рис. 9 Фрагмент кода, метод move()

Так же герой должен отображаться на экране. Для анимации было нарисовано множество картинок для трех ситуаций, герой стоит, бежит или в прыжке. Для каждой ситуации по 8 кадров, получается 48.

Рис. 10 Кадры, обеспечивающие анимацию

Для того чтобы понять какой кадр сейчас следует отобразить, нужно еще раз посмотреть на параметры и выбрать нужную группу из 8 картинок. Чтобы соблюдать очередность этих восьми картинок, существует таймер, который считает кадры. Благодаря ему в каждый момент времени мы можем узнать текущий кадр как с начала игры (глобальный кадр), так и с какого-то определенного момента, который нужно предварительно засечь (локальный кадр). Для этого используется функция getFrameFrom(), где в скобках нужно указать время начала отсчета кадров.

Герой всегда использует локальный кадр начала движения, который необходимо разделить с остатком на число отрисованных изображений, чтобы получить номер текущей картинки.

Таким образом получается создать иллюзию походки и развивания плаща с помощью покадровой анимации.

Еще герой умеет держать предметы. Для этого есть еще один параметр, который становится истинным, если нажать клавишу Shift возле объекта, который можно нести. Для регистрации нажатия используется тот же слушатель клавиатуры, что и для движения. Как только оба условия удовлетворены, создается фиксированное соединение Joint между объектом и героем, при чем место связки зависит от того, куда смотрит герой. Связь живет пока зажата клавиша, с объектом можно свободно передвигаться. Как только кнопка отпущена, соединение удаляется, и объект падает.

Рис. 11 Вирт держит фонарь

Чтобы выглядело так, словно герой сам держит предмет, и при этом не рисовать еще 48 дополнительных картинок с вытянутой рукой, было решено рисовать руку отдельно от тела.

Рис. 12 Руки

Следующая функция героя – умирание. Был создан специальный слушатель коллизий(столкновений), который при возникновении такого столкновения между персонажем и объектом-сенсором меняет параметр смерти героя.

Этот параметр всегда проверяется перед движением и отрисовкой, так что управлять персонажем после смерти становится невозможно. Игра перезагружается. Мир создается заново, при этом герой появляется в ближайшей точке из списка контрольных точек.

Рис. 13 Скриншот игры, смерть героя

4. РЕЖИМ ОТЛАДКИ

Для того чтобы контролировать правильную отрисовку картинок, скорость героя, его местоположение был создан специальный режим отладки DEBUG_MODE. Это логическая переменная, отвечающая за состояние включен или выключен данный режим в данный момент.

При включенном DEBUG_MODE зеленым цветом обводятся границы всех объектов. Пол может включать еще и желтые промежутки, это означает что наклон данного отрезка слишком велик, и герой не сможет по нему ходить.

Так же в этом режиме в верхнем левом углу выводится строка, которая может отображать координаты героя, его скорость, а так же другие показатели.

Рис. 14 Скриншот игры в режиме DEBUG_MODE

5. ИГРОВЫЕ ОБЪЕКТЫ

Кроме героя в мире есть различные предметы, с которыми можно взаимодействовать.

Для каждого вида объектов создан отдельный класс, наследующий абстрактный класс GameObject.

```
7   public abstract class GameObject {
8       protected Body body;
9
10      protected boolean takeable = false;
11
12      public abstract void draw(Canvas canvas, int frame);
13
14      public void drawDebug(Canvas canvas) {
15          //get body, draw all fixtures inside
16      }
17
18      public Body getBody() { return body; }
21
22      public boolean isTakeable() { return takeable; }
25
26      public Vector2 getCarriedRightPoint() { return new Vector2( x: 0,  y: 0); }
29
30      public Vector2 getCarriedLeftPoint() { return new Vector2( x: 0,  y: 0); }
33  }
```

Рис. 15 Фрагмент кода, абстрактный класс GameObject

При создании объекта конкретного класса-наследника запускается конструктор, который создает тело. Этому телу можно придать различные свойства, изменить массу, сделать сенсором.

Так как каждое тело должно уметь рисоваться, в каждом конкретном классе должна быть переопределена функция draw(), которая так же как и у героя с помощью таймера находит нужный кадр и рисует его.

На данный момент существует 12 классов, описывающих тела.

К таким классам относятся: Beast, Greg, Branches, Light, Cart, CartWheel, Pumpkin, Stone, InvisibleObject, Tree, Stump, Text.

Light (фонарь), Tree (дерево), Pumpkin (тыква), Stone (булыжник) - обычные физические предметы, которых можно коснуться, толкнуть, но нельзя пересечься с ними. Некоторые еще можно взять в руки. У всех объектов GameObject есть поле takeable, которое может содержать два значения, либо true, либо false. По умолчанию оно ложно, но если его переопределить как положительное, объект можно будет взять и нести.

InvisibleObject это объекты-сенсоры, их нельзя потрогать, но можно определить, что герой их пересек. Эти объекты расставляются в местах, где герой может умереть, например в яме с кольями, если персонаж там оказывается, специальный слушатель сообщает об этом и игра заканчивается.

У этого класса функция draw() пустая, на экране он не отображается.

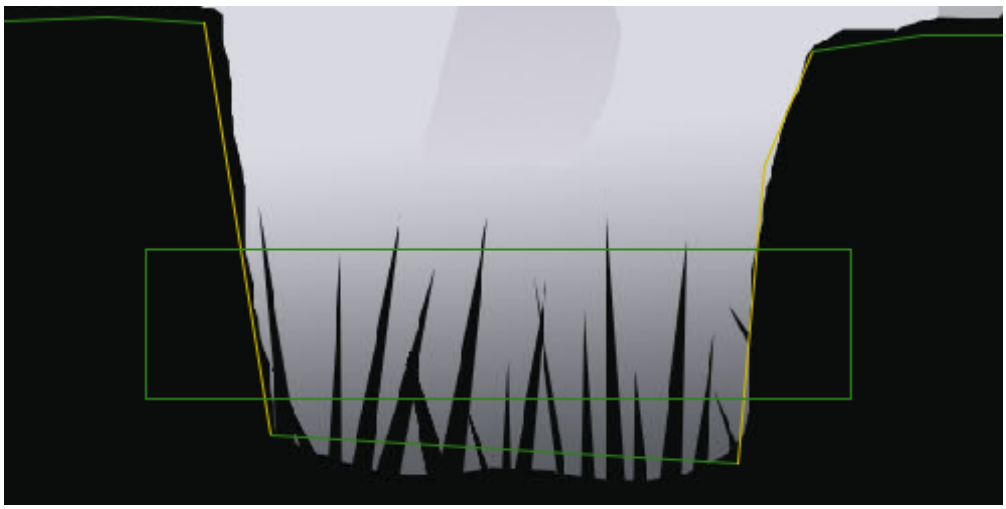

Рис. 16 Скриншот игры, объект InvisibleObject, режим DEBUG

Beast (Зверь) и Branches (ветви, путы) тоже являются сенсорами, но в отличие от предыдущего класса их задача только в том чтобы рисоваться,

не влияя на физический мир. Класс Beast отвечает за анимацию главного злодея, Зверя, а Branches это анимация ветвей опутывающих Грега.

Сложная анимация трех объектов в последней сцене была все так же отрисована покадрово.

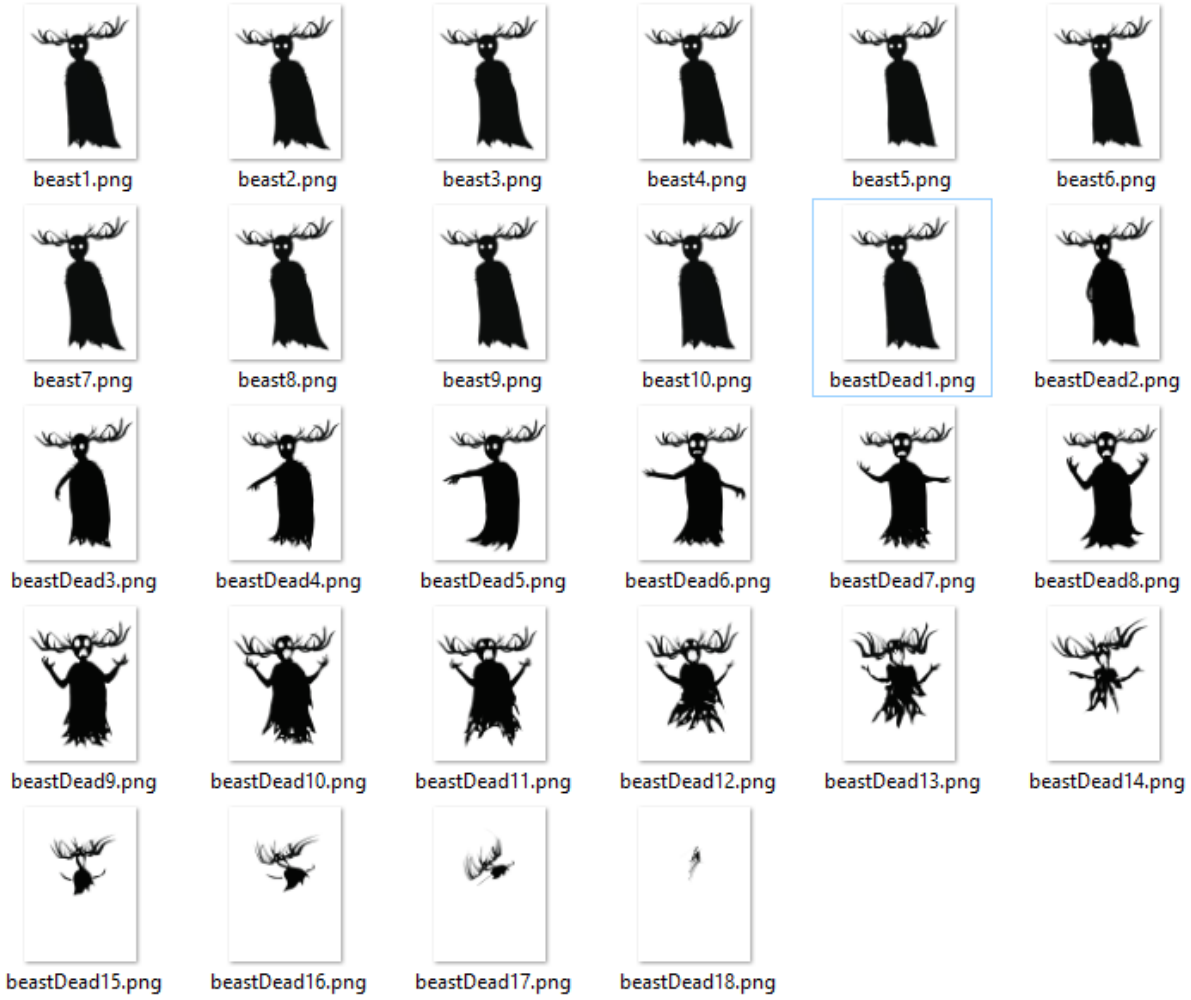

Рис. 17 Кадры, обеспечивающие анимацию Зверя

Для реализации нескольких анимаций подряд были созданы классы Animation и Animator. Объект класса Animation хранит только одну анимацию, тогда как у Зверя три таких анимации: первая – развивание плаща, зацикленная, вторая – собственно смерть, повторяется только один раз, и третья – пустая картинка.

Каждый такой объект класса Animation содержит картинки текущей анимации, знает какая анимация будет следующей (по умолчанию это ссылка на саму себя), а так же есть возможность установить действие после окончания анимации. В данном случае действие используется чтобы запустить анимацию следующего тела.

В классах Light и Branches все происходит по аналогии.

```
6   public class Animation {
7       private List<BufferedImage> images;
8       private Animation next;
9       private Runnable nextAction; //Runnable - произвольное действие
10
11      public Animation(List<BufferedImage> images) {
12          this.images = images;
13          next = this;
14      }
15
16      public Animation getNext() { return next; }
19
20      public void setNext(Animation next) { this.next = next; }
23
24      public Runnable getNextAction() { return nextAction; }
27
28      public void setNextAction(Runnable nextAction) { this.nextAction = nextAction; }
31
32      public List<BufferedImage> getImages() { return images; }
35  }
```

Рис. 18 Фрагмент кода, класс Animation

Для управления этими тремя анимациями нужен класс Animator. Объект этого класса у каждого тела всего один, и отвечает за то, какая конкретная картинка из какой анимации показывается в данный момент.

```
8   public class Animator {
9
10      private Animation currentAnimation;
11      private long animationStart;
12
13      public void start(Animation a) {
14          currentAnimation = a;
15          animationStart = System.nanoTime();
16      }
17
18      public BufferedImage getCurrentFrame() {
19          int frame = Timer.getFrameFrom(animationStart);
20          List<BufferedImage> images = currentAnimation.getImages();
21
22          while (frame >= images.size()) {
23              frame = frame - images.size();
24              animationStart = (long) (animationStart + images.size() * Timer.FRAME_TIME);
25
26              Runnable nextAction = currentAnimation.getNextAction();
27              if (nextAction != null)
28                  nextAction.run();
29
30              currentAnimation = currentAnimation.getNext();
31              images = currentAnimation.getImages();
32          }
33
34          return images.get(frame);
35      }
36  }
```

Рис. 19 Фрагмент кода, класс Animator

Так же в игре присутствует одно составное тело – телега. Состоит она из двух колес и каркаса. Колеса описаны в классе CartWheel, а каркас - в классе Cart. При создании использовались соединения WheelJoint. Создано два таких крепления, соединяющие каркас с каждым из колес в указанной точке.

```
115         // ТЕЛЕГА
116         Vector2 cartCenter = new Vector2( x: 370,  y: 5.4);
117         Vector2 wheelCenter1 = new Vector2( x: cartCenter.x - 1.5,  y: cartCenter.y - 1.4);
118         Vector2 wheelCenter2 = new Vector2( x: cartCenter.x + 1.5,  y: cartCenter.y - 1.4);
119
120         cart = addGameObject(new Cart( w: 5.5,  h: 2), cartCenter.x, cartCenter.y);
121         wheel1 = addGameObject(new CartWheel( radius: 1), wheelCenter1.x, wheelCenter1.y);
122         wheel2 = addGameObject(new CartWheel( radius: 1), wheelCenter2.x, wheelCenter2.y);
123
124         WheelJoint joint = new WheelJoint(cart.body, wheel1.body, wheelCenter1, wheelCenter1);
125         world.addJoint(joint);
126         WheelJoint joint2 = new WheelJoint(cart.body, wheel2.body, wheelCenter2, wheelCenter2);
127         world.addJoint(joint2);
```

Рис. 20 Фрагмент кода, создание телеги

Текст «THE END», появляющийся после прохождения игры, тоже является игровым объектом – сенсором. Так же он имеет бесконечную массу. Он рисуется только в том случае, если параметр gameEND стал истиной. Это произойдет при выполнении двух условий – Вирт и Грег соприкоснулись, при этом Зверь уже мертв.

Рис. 21 Скриншот игры, конец

6. КЛАСС ALL_WORLD_GAME_OBJECTS

Для хранения, обращения, и перебора всех объектов мира создан класс AllWorldGameObjects. Объект этого класса содержит список всех GameObjectов.

```
9   public class AllWorldGameObjects {
10
11      private Map<String, GameObject> nameToGameObject = new HashMap<>();
12      private Map<Body, GameObject> bodyToGameObject = new HashMap<>();
13      private ArrayList<GameObject> allWorldGameObjects = new ArrayList<>();
14
15
16      //TODO make all names be constants
17      public void addObject(GameObject gameObject, String name) {
18          //записать в map и в list
19
20          allWorldGameObjects.add(gameObject);
21          bodyToGameObject.put(gameObject.body, gameObject);
22          if (name != null)
23              nameToGameObject.put(name, gameObject);
24      }
25
26      //body -> GameObject
27      public GameObject getObjectByBody(Body body) { return  bodyToGameObject.get(body); }
30
31      public GameObject getByName(String name) { return nameToGameObject.get(name); }
34
35      public ArrayList<GameObject> getList() { return allWorldGameObjects; }
38  }
```

Рис. 22 Фрагмент кода, класс AllWorldGameObjects

Основная функция – выдавать объект класса наследующего GameObject соответствующий конкретному телу. Это необходимо, потому что при коллизии слушатель сообщает тела, которые столкнулись, и чтобы сделать что либо, нужно знать конкретные объекты, которым принадлежат эти тела.

Еще одна функция этого класса – выдавать объект по закрепленному за ним имени. Это используется в той части кода, где одному объекту нужно запустить действия другого объекта. Например, Зверь, после того как отработала анимация смерти, сообщает ветвям начать движение.

7. ПОЛЫ

В библиотеке dyn4j есть возможность задать координаты нескольких точек, и последовательно их соединить. В результате получится ломаная линия, которая станет полом.

Для человека довольно неудобно каждый раз прописывать координаты конкретной точки, поэтому было решено научить программу самостоятельно рассчитывать координаты по картинке.

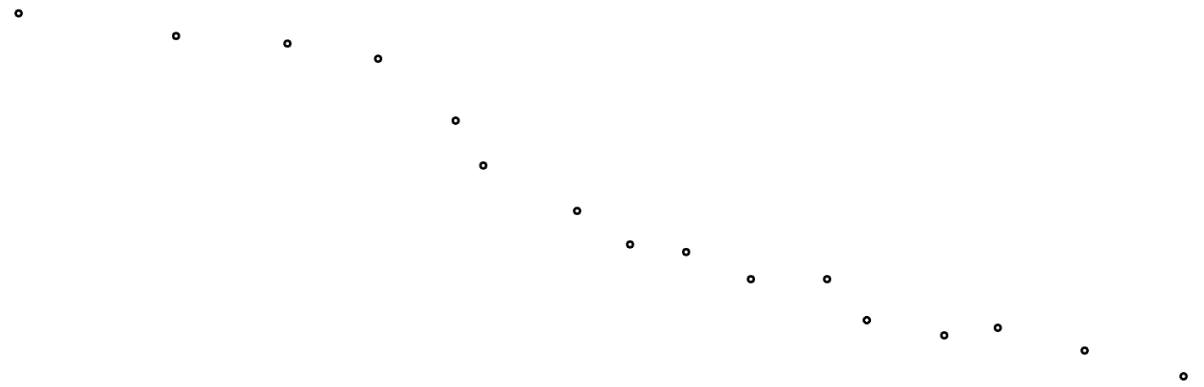

Рис. 23 Пол

Это пример картинки, которую должна обработать программа, чтобы создать пол. Начиная с левого верхнего угла, она перебирает каждый пиксель, и как только находит черный, сохраняет его, при условии что в радиусе 20 пикселей сохраненных точек пока нет.

Таким образом, мы нашли координаты всех точек, и можем перевести их в мировые. Но в игре должен быть не один пол, а несколько, последовательно соединенных.

У каждой картинки с точками для пола определенное название, например «bgFloor_0.png». Число в таком названии обозначает порядковый номер пола в цепочке полов. Следовательно, это нужно учитывать при переводе координат в мировые.

Сначала переведем в мировую систему координату X.

$$\frac{BGW}{x} = \frac{PW}{Px}$$

Допустим, PW – ширина картинки в пикселях, BGW – в мировых координатах, оно же длинна одного пола. Px – точка, найденный нами номер пикселя, а просто x – она же в мировых координатах, это значение нужно найти.

x = Px * BGW / PW

Но так как полов несколько, нужно прибавить произведение порядкового пола и BGW.

С координатой Y немного по другому. Так как в мировой системе этой игры отсчет Y начинается с левого нижнего угла, а в экранных координатах с левого верхнего, чтобы положение точки осталось неизменным, нам нужно вычислить расстояние, то есть количество пикселей, от низа экрана\картинки до точки. Это и будет Py.

$$\frac{BGH}{y} = \frac{PH}{Py}$$

y = Py * BGH / PH

8. КАМЕРА

Экран компьютера – это окно между виртуальным миром и реальным, и очень важно, чтобы оно постоянно находилось в нужном месте. В данном случае в центре всегда должен находиться персонаж.

Камера – это как раз тот класс, который отвечает за то, какую часть мира видно на экране и в каком масштабе. Ее задача – плавно следовать за героем, как бы догоняя.

```
15  public class Camera {
16
17      private Vector2 position = new Vector2( x: 0,  y: 0);
18
19      private BufferedImage black;
20      public final static BufferedImage NOTHING_IMG = new BufferedImage
21              ( width: 1920,  height: 1080, BufferedImage.TYPE_BYTE_BINARY);
22
23      private Map<BgCoords, BufferedImage> bgImagesCache = new HashMap<>();
24
25
26      public final static double SCREEN_W = 35;
27      public final static double SCREEN_H = 18;
28
29      public final static double BG_W = 35;
30      public final static double BG_H = 35.0 * 1080 / 1920; //примерно 20;
31
32      public final static int HERO_POSITION_H = 5; //от низа экрана до героя
33
34      public Camera() throws IOException {
35          black = ImageIO.read(new File( pathname: "images/bg/black.jpg"));
36      }
37
38      public List<BgCoords> getVisibleBackgrounds() {...}
55
56      private BufferedImage getBgImage(BgCoords bgC) {...}
74
75      private BufferedImage loadBgImage(BgCoords bgC) {...}
87
88      public void drawBackground(Canvas canvas) {...}
93
94      public void move(Hero hero, double elapsedTime){...}
112
113     public void setPosition(Vector2 position) { this.position = position; }
116
117     public Vector2 getPosition() { return position; }
120 }
```

Рис. 24 Отрывок кода, класс Camera

Камере всегда известно как свое положение, так и положение героя. Обладая данной информацией она может вычислить расстояние меду этими точками. Чем больше камера отстала, тем быстрее она догоняет персонажа, но по мере уменьшения расстояния уменьшается и скорость.

9. ФОН

Класс камера также отвечает за загрузку и отображение картинок фона.

Фон занимает основное пространство на экране и играет решающую роль в создании атмосферы в игре.

Он состоит из последовательности картинок одного размера. Проблема состоит в том, что таких картинок будет довольно много, и если хранить все в текущей памяти, это может негативно сказаться на быстродействии. То есть, одновременно хранить можно определенное количество фонов, и если требуется загрузить новые, необходимо удалить старые.

Для каждой точки в мировых координатах можно без проблем найти, на каком по счету фону она находится. Зная положение камеры, можно вычислить, где будут углы экрана. Таким образом можно найти все номера картинок, которые нужны для отрисовки этого экрана, максимум их будет 4.

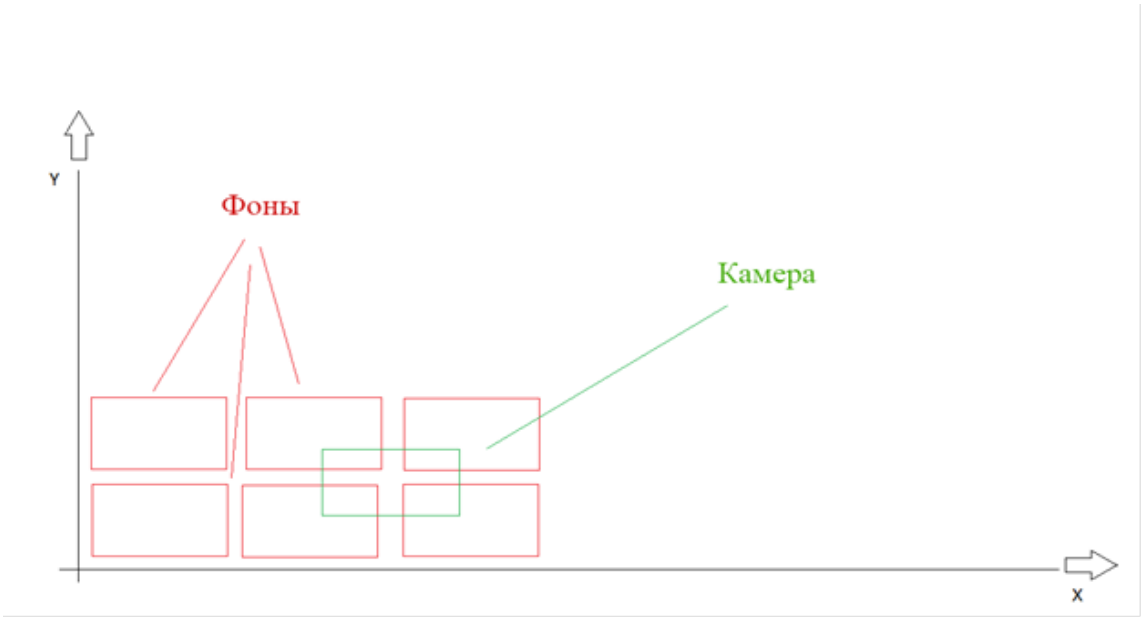

Рис. 25 Камера и фоны

Как было сказано ранее, некоторое количество картинок уже может быть загружено. Для хранения изображений использовался класс Map, который позволяет хранить элементы парами ключ - значение. Ключом в данном случае выступает координата фона, значением – сам фон. Для начала стоит поискать нужные картинки по ключам в данном списке. Если это не дало результата, искомое изображение можно загрузить с диска, добавить в Map, и не забыть удалить как можно дальнюю картинку от текущих координат, если требуется.

10. КООРДИНАТЫ И КЛАСС CANVAS

В рамках программного кода положение тела в пространстве – это его координаты. В виртуальном мире обычная привычная система координат, ноль в центре, X растет слева направо, Y - снизу вверх.

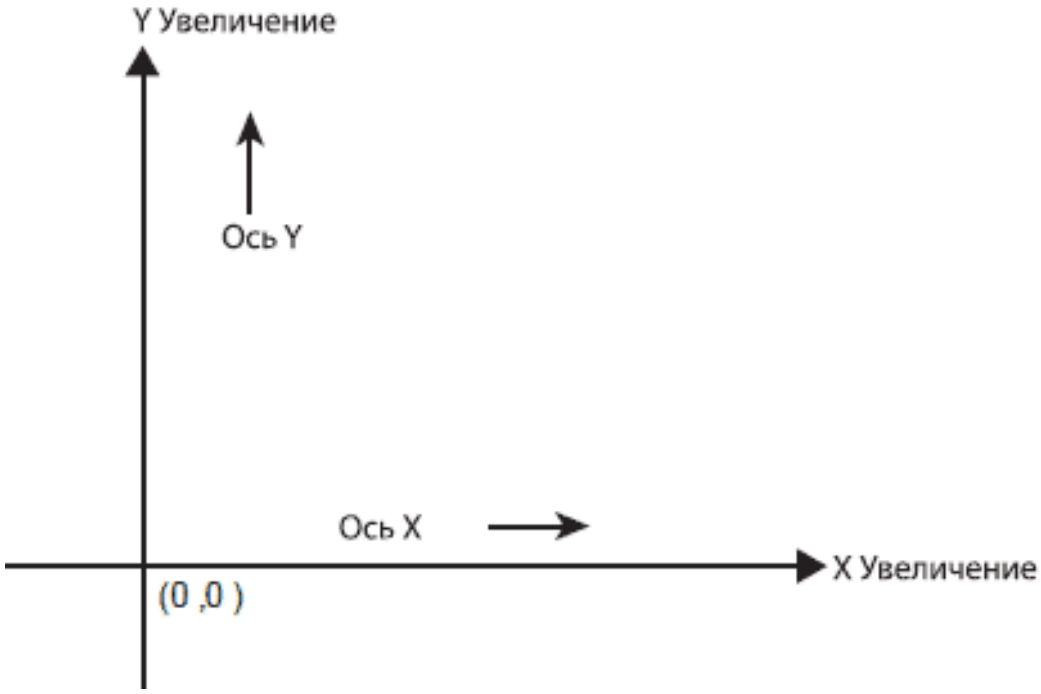

Рис. 26 Мировая система координат

Но чтобы отобразить что-то на экране, придется воспользоваться экранными координатами, которые отличаются от обычных. Ноль находится в левом верхнем углу, X растет так же направо, а Y наоборот, вниз.

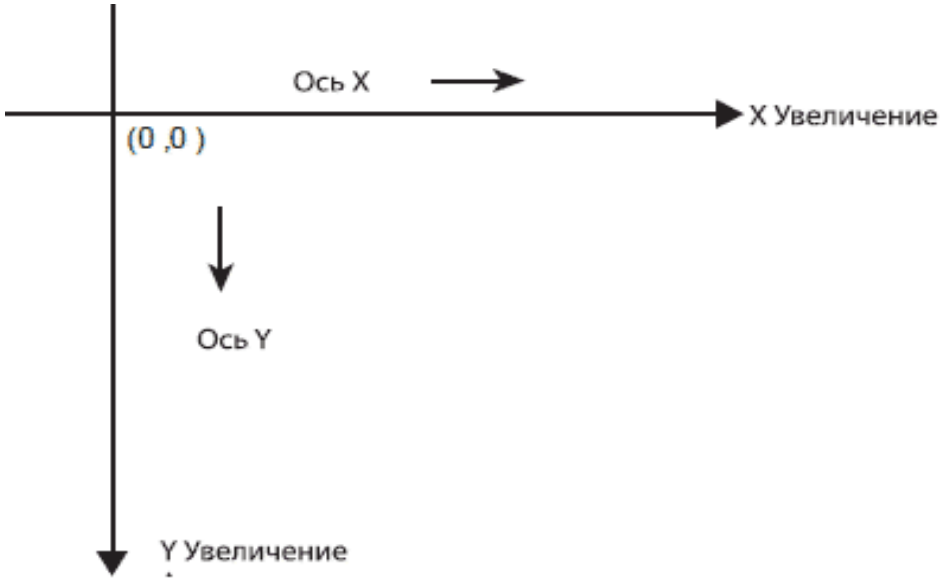

Рис. 27 Экранная система координат

Получается, что для расчета положения тел в мире используется мировая система координат, а для отрисовки – экранная. И чтобы нарисовать объект нужно его мировые координаты перевести в экранные.

Для начала разберемся с масштабом. Единицей в экранной системе координат является пиксель, в мировой это нечто похожее на реальный метр. Соответствие метр = пиксель не очень удобно, значит метры необходимо умножать на что-либо, в данном случае 1 метр отображается как 50 пикселей на экране.

Еще одно несоответствие - это направления. У координаты X направления совпадают, тогда как у Y противоположны, значит при переводе его нужно умножать на -1.

Есть еще и третья система координат – локальная, для каждого отдельного тела. Она используется при рисовании этого тела. За перевод между мировой системой координат и системой координат тела отвечает физический движок, потому что он вычисляет и хранит информацию о положении тела в пространстве: координатах центра и угле поворота.

Чтобы не думать об этих преобразованиях постоянно, проще вынести все эти вычисления в отдельный класс Canvas.

В методе render() создается объект этого класса, с помощью которого все тела рисуются там, где нужно.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была достигнута цель, которая заключалась в получении навыков программирования на языке Java.

Так же были изучены особенности применения библиотек, в частности библиотеки dyn4j.

Были освоены технологии создания компьютерных игр, изучены принципы проектирования уровней.

Результатом работы является игровая программа.

СПИСОК ЛИТЕРАТУРЫ

1. Java SE Documentation. [Электронный ресурс] // Oracle. – URL: https://docs.oracle.com/javase/7/docs/ (дата обращения: 15.03.2017).

2. dyn4j Documentation. [Электронный ресурс] // dyn4j. – URL: http://www.dyn4j.org/documentation/ (дата обращения: 18.05.2017).

3. Шилдт Г. Java. Полное руководство, 8-е изд.: пер. с англ.—М.: ООО “И.Д. Вильямс”, 2012.

4. Wright T. Fundamental 2D Game Programming with Java. Cengage Learning PTR, 2014. 656c.

5. Кадиков М. Дизайн уровней: теория и практика. [Электронный ресурс] // Интересное о дизайне уровней. – URL: http://level-design.ru/pro-ld-book-index/pro-ld-book-about/ (дата обращения: 30.04.2017).

6. Покадровая анимация в Фотошопе. [Электронный ресурс] // Уроки Фотошопа. – URL: http://photoshoplessons.ru/animation/pokadrovaja (дата обращения: 04.05.2017).

7. Компьютерные игры как искусство. [Электронный ресурс] // GamesisArt.ru. – URL: http://gamesisart.ru/index.html (дата обращения: 10.03.2017).

8. Введение в геймдизайн: Основные понятия и принципы проектирования игр. [Электронный ресурс] // vc.ru. – URL: https://vc.ru/p/gamedev-challenges (дата обращения: 10.03.2017).

9. Wallace J. Beginning Java 8 Games Development. Apress, 2014. 475c.

10. IntelliJ IDEA 2017.1 Help. [Электронный ресурс] // JetBrains.com. – URL: http://www.jetbrains.com/help/idea/2017.1/meet-intellij-idea.html (дата обращения: 18.03.2017).

11. Википедия [Электронный ресурс] // Свободная энциклопедия. – URL: https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B3%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0 (дата обращения: 22.03.2017).

Последний лист выпускной квалификационной работы

Выпускная квалификационная работа выполнена мною самостоятельно. Использованные в работе материалы из опубликованной научной, учебной литературы и Интернет имеют ссылки на них.

Отпечатано в ____ экземплярах.
Библиография ____ наименований.
Один экземпляр сдан на кафедру.

Козлова А. С.

Дата

Я, (ФИО), не возражаю против размещения на сайте Факультета искусств СПбГУ моей выпускной квалификационной работы и ее результатов

_____ _____ (расшифровка подписи)
Дата подпись