Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Курсовой проект по дисциплине:

«Программирование»

Пояснительная записка к курсовой работе

Тема работы:

«Бот-секретарь для лабораторных работ»

Исполнитель

студент гр. 653502 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шпаков Н.И.

(подпись дата)

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Козуб В.Н.

(подпись дата)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка)

Минск

2017 год

**Содержание**

[**Введение 3**](#_Toc480910310)

[**1. Game maker: studio 3**](#_Toc480910311)

[**1.1 Игровой движок 3**](#_Toc480910312)

[**1.2 Game maker language 4**](#_Toc480910313)

[**2. Процесс разработки 6**](#_Toc480910314)

[**2.1 Начало зарождения игры 6**](#_Toc480910315)

[**2.2 Разработка игры 6**](#_Toc480910316)

[**2.2.1 Базовая механика 6**](#_Toc480910317)

[**2.2.2 Графика и музыка 8**](#_Toc480910318)

[**2.2.3 Искусственный интеллект 9**](#_Toc480910319)

[**2.2.4 Диалоговая система 11**](#_Toc480910320)

[**3.Структура проекта и игровая логика 13**](#_Toc480910321)

[**Заключение 17**](#_Toc480910322)

[**Список использованных источников 18**](#_Toc480910323)

# **Введение**

В современные дни все больше и больше распространяются любые виды игр, в особенности видео игры в связи с развитием информационных технологий.

Также можно отметить очень значительное развитие и технологий для самой разработки игр.

В наше время разработка для игровых приставок все больше напоминает написание программ для обычных компьютеров, разница между тем что творилось тогда и тем, что имеет сегодняшний разработчик колоссальна. С одной стороны, сегодня, технологии и культура разработки шагнули далеко вперед, позволив вести разработку на языках высокого уровня, с другой — в 80-е фирмы, выпускающие приставки, еще не до конца понимали, что следует дать сторонним разработчикам для создания игр. В то время большинству программистов приходилось довольствоваться ассемблером и лишь у некоторых везунчиков были компиляторы С, писать на котором тогда было огромное счастье и удовольствие. И поэтому если сейчас для очередной playstation можно получить документацию, пяток демо-дисков, а также мощную девелоперскую станцию, по сути гибрид самой приставки и компьютера общего назначения, пригодного для непосредственной разработки, то, например, в 80-е многие разработчики довольствовались книжечкой со спецификацией консоли. А вот за аппаратной частью приходилось идти на радиорынок.

Перед собой в данном проекте я поставил цель изучить средства программирования в game maker: studio на приемлемом уровне и пройти процесс создания игры с нуля как независимый разработчик одиночка. Для этого данный проект был выполнен также в рамках конкурса Ladum Dare 38 (хакатон), для того, чтобы протестировать свои возможности как программиста, где и за 48 часов непрерывного труда и был создан проект на текущей стадии. Учитывая актуальность данной сферы в сегодняшний день и также мой личный интерес к данной индустрии, я решил сделать свою курсовую работу именно как компьютерную игру, которая в дальнейшем будет доработана и выложена в сеть на бесплатной основе.

# **1. Game maker: studio**

## **1.1 Игровой движок**

1. Если затрагивать аспект технологий, то для данного проекта я использовал игровой движок Game maker: studio, который является практически наилучшим вариантом для разработки кросплатформенных игр в 2д измерении. Я сделал такой выбор из-за некоторых примеров действительно успешных проектов, написанных на этом игровом движке. Также данный игровой движок имеет и средства для создания игр и 3д, однако их использование не является оптимальным решением при создании своей игры. Среди конкурентов данному движку в программировании игр в двухмерном пространстве можно перечислить Unity, godot и прочие игровые движки. Game maker: studio использует для программирования свой уникальный язык который называется Game maker language, который, по моей практике, имеет не самые практичные средства отладки. Но подробнее про этот язык я расскажу в следующем тезисе.

## **1.2 Game maker language**

1. Язык GML очень по своему синтаксису напоминает С++, если бы не некоторые особенности. Во-первых, если приравнять оба языка, то GML имеет множество надстроек над собой, которые должны упростить разработку игр. Например: один из способов проитерироваться тысячу раз в С++ выглядит так:
2. for(int i = 0; i < 1000; i++)
3. //fo smth;
4. Все циклы до стандарта С++ 2011 существуют и в GML, но также есть и дополнительные упрощающие написание кода для игр циклы, по типу:
5. repeat(1000)
6. //do smth;
7. Это лишь один из примеров надстроек. Далее можно отметить полное отсутствие указателей в языке программирования GML, которые составляют практически основу С++. Однако язык программирования на этом движке по сути своей и не должен их иметь для выполнения поставленных целей. Здесь и начинается главное различие обоих языков, не считая других тонкостей, таких как нестрогая типизация переменных в GML. В самой среде game maker используется следующая логика: объекты создаются через пользовательский интерфейс, а их действия программируются уже непосредственно при действии каких-либо внешних побудителей. Также функции, которые будут в дальнейшем использоваться в кодах событий, пишутся отдельно в скриптах на языке GML. Например:
8. if(vspd == 0 && hspd == 0)
9. {
10. dir = point\_direction(x, y, mouse\_x, mouse\_y);

if((dir >= 315 && dir <= 360) || (dir >= 0 && dir <= 45))

sprite\_index = spr\_right;

else if(dir >= 45 && dir <= 135)

sprite\_index = spr\_up;

else if(dir >= 135 && dir <= 225)

sprite\_index = spr\_left;

else

sprite\_index = spr\_down;

var len = 0;

1. }
2. else
3. var len = spd;

if(hspd != 0 || vspd != 0)

{

dir = point\_direction(0, 0, hspd, vspd);

phy\_position\_x += lengthdir\_x(len, dir);

phy\_position\_y += lengthdir\_y(len, dir);

}

1. В данном примере реализуется код для отслеживания позиции курсора и непосредственно на получаемых данных персонаж меняет свой обзор путём смены спрайта, а также если клавиши в другом скрипте были нажаты, то будет произведено также и передвижение по координатам на проекции полученного вектора, умноженного на скорость персонажа.
2. Все выше изложенное и составляет по сути логику написания игр на самом игровом движке, а про саму игровую логику будет изложено несколькими тезисами далее.

# **2. Процесс разработки**

## **2.1 Начало зарождения игры**

1. На мероприятии Ladum Dare у меня было очень ограниченное время и в связи с этим мне потребовалось грамотно распределить свои усилия, с чем я немного, в конечном счете, не идеально справился. Хакатон для написания курсового проекта я выбрал не с проста, так как он предполагает собой очень усердную работу в течение двое суток, и это самый лучший способ сконцентрироваться на проекте и сделать его, предварительно до этого изучив игровой движок и его технологии разработки игр.
2. Тематика Ladum Dare 38 была “Маленький мир”. Для данной темы я придумал концепцию, которую полностью раскрывать не считаю должным, каждый в конце концов поймет ее по-своему, хотя задумка по сути была единственна. Но основная суть такова: есть главный герой, о котором ничего не известно, и, пробираясь по этажам таинственного замка, герой общается, взаимодействует и решает задачи чтобы добраться в конечном счете на последний этаж замка, который и завершит эту игру.
3. Сама игра по задумке должна проходиться не как классический представитель своего жанра, а как отдельный независимый рассказ, который будет интересно прочитать. По итогу игра должна будет занимать около 30 – 45 минут геймплея.

## **2.2 Разработка игры**

### **2.2.1 Базовая механика**

Продумав идею, я принялся за реализацию простейшей игровой механики, которая может быть принята за основу самой игры. Для начала, хотелось бы оговориться, что все ресурсы в игре самодельные, за исключением спрайта главного героя и мелодии в конце игры, которые в релизной версии будут полностью изменены, но суть останется примерно той же. В основе механики лежала простейшая возможность персонажа делать атаку, резкий рывок с красивой анимацией, а также его физическое взаимодействие с окружающей средой.

Для начала хотел бы еще оговориться и по поводу физики. В Game maker: studio имеется встроенная физика, которая подразумевает уже запрограммированное заранее взаимодействие объектов. В этой игре я использовал физическую модель взаимодействия предметов, но, тем не менее, можно запрограммировать ее и математическим способом, что я и использовал для своих тренировочных проектов:

if(place\_meeting(x, y + vsp, obj\_wall))

{

while(!place\_meeting(x, y + sign(vsp), obj\_wall))

y += sign(vsp);

vsp = 0;

}

if(place\_meeting(x + hsp, y, obj\_wall))

{

while(!place\_meeting(x + sign(hsp), y, obj\_wall))

x += sign(hsp);

hsp = 0;

}

X += hsp;

Y += vsp;

Данный код реализован в step\_event объекта player. Step\_event отслеживает каждый игровой кадр, заданный в настройках комнаты, и делает какое то действие. В данном событии сперва реализуется проверка на нажатие клавиш и построение на этих данных двух взаимоортогональных единичных или нулевых векторов:

left\_key = keyboard\_check(ord("A"));

rigth\_key = keyboard\_check(ord("D"));

up\_key = keyboard\_check(ord(“W”));

down\_key = keyboard\_check(ord(“S”));

hvector = rigth\_key - left\_key;

hsp = walk\_speed \* hvector;

vvector = up\_key - down\_key;

vsp = walk\_speed \* vvector;

На основе этих данных в первом блоке кода этого тезиса идет проверка на каждом шагу, а не будет ли по каждой из осей объект игрока пересекаться с объектом стены? Если же да, то мы будем прибавлять к текущим координатам объекта по 1, умноженной на направление вектора, до тех пор, пока наши координаты не будут стоять вплотную. Коллизия, а именно само взаимодействие объектов, определено в настройках спрайта (модели) объекта. Подобная проверка координат проводится для того, чтобы игрок не проходил сквозь стены, а последующее использовании функции sign(x) необходимо для того, чтобы игрок не застрял в этой стене, прибавив немного лишнего к координатам.

В проекте в рамках конкурса Ladum Dare я использовал физическое взаимодействие объектов. Объекту придается определенная форма, по которой будет происходить наша коллизия, и физика движка будет сама высчитывать что делать при этой коллизии. Для этого в событиях персонажа необходимо прописать события коллизии с объектом для взаимодействия, а в каждом шаге делать проверку на введенные данные и на основе них прибавлять к уже физическим координатам полученные значения, которые будут сами обработаны физикой Game maker: studio.

Для большей красоты и удобства кода в событии шага можно прописать выполнение некоторого состояния, которое в себе будет хранить функцию, в которой и будет реализован наш ввод данных и поведение персонажа на основании этих данных. Собственно, что я активно и использовал в этом проекте.

### **2.2.2 Графика и музыка**

Создав простые элементы механики, мне потребовалось очень много ресурсов для придания этому делу некоторой красоты. Поэтому действительно много времени было потрачено на отрисовку всех спрайтов, а затем последующее создание комнат на их основе.

Комнаты создавались по следующему принципу: имелись текстуры для фона, которые в зависимости от глубины слоев расставлялись по всей карте (рисунок 1). Далее, имея некоторую комнату, в ней уже расставлялись физические объекты вроде невидимых стен, с которыми и взаимодействовал, хотя и создается ощущение взаимодействия с отрисованными стенами. Это распространенная практика при создании игр с тайловой или около нее системой карт. Всего в моей игре 7 комнат, не считая комнаты для меню игры.

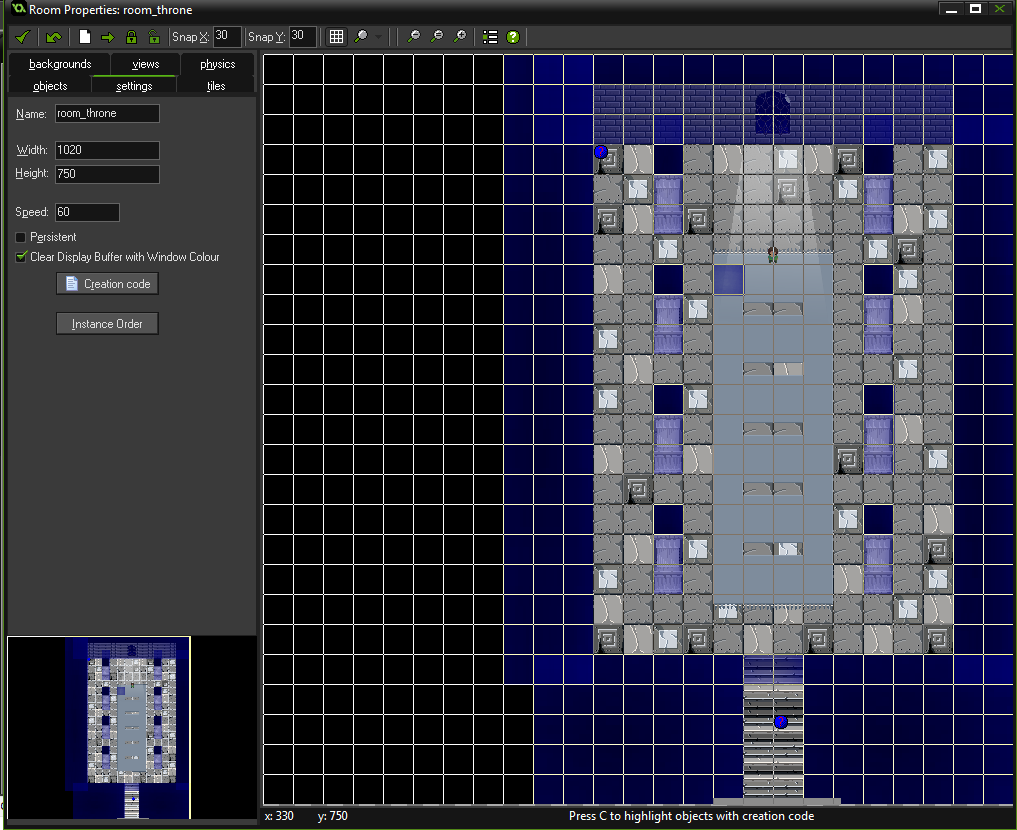


Рисунок 1. Редактор комнат Game maker: studio

После отрисовки всего этого дела потребовалась музыка, которая могла бы сделать замок более атмосферным. Написанием музыки ближе к концу конкурса занялся все-таки не я, а мой хороший друг, используя программное обеспечение fl studio.

### **2.2.3 Искусственный интеллект**

В итоге, создав систему уровней для игры, надо было приступать к программированию врагов и прочих систем. Про программирование таких мелочей, как улученная камера, тьма, красивые переходы и меню не имеет смысла, поэтому я сразу приступлю к чему-то более на сущему.

Во-первых так как враги обладают абсолютно той же физикой что и игрок, я решил упростить существенно себе жизнь и создал объект родитель для обоих форм жизни, который имел в себе базовые параметры и также свойства коллизий. В дальнейшем я просто унаследовал оба объекта от этого одного.

Далее, основы интеллекта оппонентов. Мне надо было их запрограммировать немного “по-умному”, чтобы они реагировали на меня только в определенном поле зрения и вне его двигались на короткие расстояния в случайном направлении, иногда останавливаясь. Реализовывал я всю эту схему с помощью специальных задержек и системы скриптов. В самом же событии кадра я делал выполнение текущего скрипта, который определялся через эти же самые скрипты. Для начала враг впадает в некоторое состояние, в котором он решает, что будет делать. У него есть один из двух выборов – либо стоять, либо пойти. В обоих скриптах также выполняется еще один дополнительный скрипт, который проверяет постоянно, не вошел ли игрок в радиус агрессии нашего врага.

Скрипт положения, в котором враг стоит:

///get\_idle\_state();

check\_for\_player();

Cкрипт движения в случайную сторону:

///get\_wander\_state()

check\_for\_player();

if(point\_direction(x, y, targetx, targety) > spd)

{

var dir = point\_direction(x, y, targetx, targety);

var hspd = lengthdir\_x(spd, dir);

var vspd = lengthdir\_y(spd, dir);

if(hspd != 0)

image\_xscale = sign(hspd);

phy\_position\_x += hspd;

phy\_position\_y += vspd;

}

Во втором скрипте мы также проверяем, будет ли расстояние до случайной точки карты больше, чем скорость врага. Если да, то мы делаем рассчеты по векторам и двигаемся в указанном направлении. Это сделано для того, чтобы если вдруг он достигнет этой отметки раньше, чем завершится таймер перехода в состояние выбора, то его бы не трясло в разные стороны, Также здесь мы отражаем нашу картинку для того, чтобы враг имел спрайт движения в своем направлении.

### **2.2.4 Диалоговая система**



Рисунок 2. Диалоги в игре

Так как основная составляющая игры – сюжет и атмосфера, то необходимо создать диалоговую систему. Ничто не передает историю лучше, чем диалоги (рисунок 2).

Так как свойством хранить в себе массив отображаемого текста будет иметь не только табличка, но и персонажи, то создадим отдельный объект с этим свойством, от которого будут унаследованы наши табличка и НИП. Далее создадим объект диалогового окна, который будет отрисовывать себя и текст. При создании этого объекта мы будем указывать его поле текста равным полю текста объекта со свойством содержания данных. Создаваться диалоговое окно будет при взаимодействии специального невидимого физического объекта, который порождает игрок при нажатии правой кнопки мыши. Далее при повторном нажатии будет выполняться перелистывание страниц текста. В случае же последней страницы, текст просто исчезнет через 12 секунд если не про взаимодействовать с ним в течении этого времени. Сперва разберем код события GUI\_draw объекта диалог:

text\_count += spd;

text\_visible = string\_copy(text[page\_count], 0, text\_count);

draw\_set\_font(font\_small);

draw\_set\_color(c\_black);

var xx = (x - view\_xview[0]) \* display\_scale;

var yy = (y - view\_yview[0]) \* display\_scale;

draw\_text\_ext(xx + margin - 48 \* display\_scale, yy + margin - 18 \* display\_scale, text\_visible, -1, width - (margin \* 2));

if(array\_length\_1d(text) - 1 == page\_count && destroy)

{

destroy = false;

alarm[0] = room\_speed \* 12;

}

Здесь мы устанавливаем стиль текста и высчитываем его координаты на экране, с учетом того, что в событии GUI\_draw началом системы координат является не верхний левый угол комнаты, а тот же угол, но уже самого экрана. Здесь же вступает в действие и display\_scale, переменная, которая хранится в объекте диалога и высчитанная по простой пропорции пользовательского экрана к настоящему отображаемому разрешению экрана.

Переменная text\_count каждый шаг отрисовки становится больше на велечину spd, которая задана в соответствии с эстетическими потребностями разработчика. Таким образом text\_count каждый шаг ограничивает text\_visible, однако через какое-то количество кадров оно становится равным количеству символов в тексте. Это создает медленную анимацию отрисовски текста в диалоговом окне.

Вызов диалогового окна происходит через событие взаимодействия объекта с текстом и физического объекта взаимодействия персонажа. Код этого события выглядит следующим образом:

with(other)

{

if(!instance\_exists(dialog))

{

dialog = instance\_create(x + xoffset, y + yoffset, obj\_dialog);

dialog.text = text;

}

else

{

dialog.text\_count = 0;

dialog.page\_count++;

if(dialog.page\_count > array\_length\_1d(dialog.text) - 1)

with(dialog)

instance\_destroy();

}

}

Первым делом, нужно обратить внимание на оператор with(other). Это одна из особенностей языка GML, которая позволяет обратиться к объекту, с которым проходит взаимодействие через взаимодействующий объект и с помощью этого можно пользоваться методами объекта взаимодействия как с самим взаимодействующим объектом.

Далее мы проверяем наличие объекта диалога, причем не самого диалога в данной комнате, а именно наличие не пустого значения в поле dialog объекта докладчика. Если оно пустое, то мы его создадим и присвоим его текстовому полю соответствующее поле нашего создателя. Иначе мы просто поменяем текущий показатель text\_count на ноль и увеличим текущую страницу, тем самым отразив на диалоге следующий элемент массива текста. Если вдруг текущая страница будет больше длины массива поля текста с отнятием единицы, то тогда мы удалим значение нашей переменной dialog.

Все просто и логично, однако подобных вещей в игре не мало, реализация которых заняла бы еще не мало времени, поэтому перейдем к последнему заголовку этого курсового проекта.

# **3.Структура проекта и игровая логика**

В этом тезисе я хотел бы немного подробнее разобрать вопрос игровой логики, который я затронул в своем первом тезисе. Основой игровой логики является подача некоторого действия в зависимости от действия на внешний побудитель. В каждой среде или же в зависимости от языка программирования - это реализовано по-разному, но суть остается той же. В Game maker: studio это реализовано через удобный пользовательский интерфейс событий, где разработчик уже и прописывает сами действия.

Остановимся подробнее на объекте игрока. У нас имеется событие create. Это событие вызывается при создании объекта игрока. В нем заложены основные поля данного объекта, в которые уже будут прописаны значения для использования или обращения к ним в дальнейшем. Например, та же скорость персонажа. Эта константа заложена при создании объекта, и мы можем ее использовать в любых скриптах которые направлены на вычисление данных внутри объекта игрока, например:

if(hspd != 0 || vspd != 0)

{

dir = point\_direction(0, 0, hspd, vspd);

phy\_position\_x += lengthdir\_x(len, dir);

phy\_position\_y += lengthdir\_y(len, dir);

}

Где len = spd, если игрок находится в движении, т е вектора по осям x и y не равны нулю. Данный фрагмент кода вырезан из скрипта движения персонажа.

Точно также мы закладываем состояния персонажа, которое означает его соответствующее действие на те же побудители, закладывая в step\_event совершение скрипта переменной state, содержащей соответствующую функцию. Переменная state в нашем случае меняется внутри скрипта движения игрока, и через другие события игрока возвращают в state движение игрока обратно.

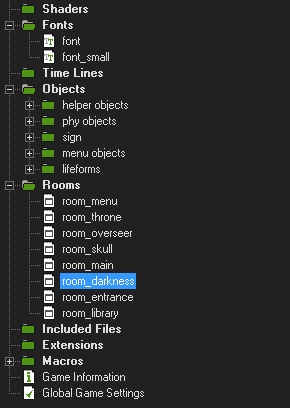
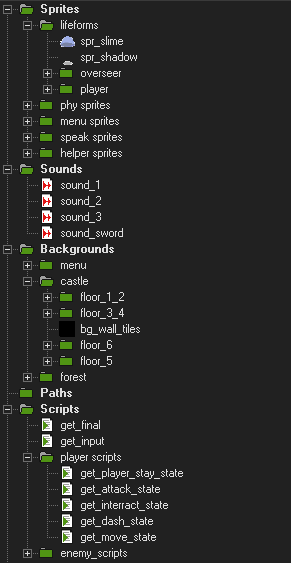


Рисунок 3 – 4. Структура файлов проекта

В самом же проекте было использовано очень много ресурсов, подробное строение папок проекта можно увидеть на рисунке (рисунок 3 - 4).

Для упрощения взаимодействия между объектами создавались также и дополнительные объекты, которые срабатывали как сенсоры для создания какого-либо игрового события.

И в заключение этого тезиса я хотел бы подметить, что работать в данной среде разработки действительно удобно и ее инструменты позволяют создавать высокоуровневые приложения которые могут конкурировать с другими аналогами на рынке. Как примеры успешных проектов на данном движке я могу привести такие игры, как Undertale, Hotline Miami и Hyper light drifter (рисунок 5). И этими играми список не ограничивается.



Рисунок 5. Hyper Light Drifter

# **Заключение**

В процессе написания курсовой работы я получил действительно бесценный опыт в той сфере программирования, которая притягивает меня больше всего. Я испытал свои навыке на площадке для разработчиков игр в режиме самостоятельного разработчика на 48 часов, понял свои ошибки, которые допускал, и прошел целиком от нуля до практически полной реализации игры самостоятельно. От зарождения идеи до создания игровых моделей, реализации этого в игровом движке и написания кода. За эти 48 часов очень интенсивной работы я сделал действительно большой шаг в этом направлении.

Я ознакомился с базовыми принципами игровой логики и для себя понял, что это дело мне нравится еще больше, чем мне казалось. При дальнейшей работе над игрой планируется сделать ее мир живым, геймплей несколько усложнить и также сделать ее как полноценную историю, которая может вызвать какие-то эмоции или чувства у игрока. В дальнейшем, когда эта игра будет полностью готова, она выпущена в сети на бесплатной основе.

А перспектива данного направления, учитывая распространение игр, пока заставляет уверенно смотреть вперед и продолжать заниматься этим делом.

# **Список использованных источников**

1. Habrahabr [электронный ресурс] – Режим доступа: <https://habrahabr.ru/post/200612/>. – Дата доступа: 25.04.2017.

2. Game maker: studio documentation [электронный ресурс] – Режим доступа: <https://docs.yoyogames.com/>. – Дата доступа: 21.04.2017.