Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики и технологии программирования

Дисциплина: Программирование

К ЗАЩИТЕ ДОПУСТИТЬ

\_\_\_\_\_\_\_\_\_\_И. В. Удовин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе  
на тему

Игра «Flappy bird»

Студент

Руководитель

А. Д. Климович

И. В. Удовин

МИНСК 2020

Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики и технологии программирования

Дисциплина: Программирование

УТВЕРЖДАЮ

Заведующий кафедрой ИиТП

\_\_\_\_\_\_\_\_\_\_\_\_ Н. А. Волорова

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_2020 г.

ЗАДАНИЕ

по курсовой работе студента  
Климовича Артемия Денисовича

1. Тема работы: Игра «Flappy bird»
2. Срок сдачи студентом законченной работы: 1 июня 2020 г.
3. Содержание пояснительной записки (перечень подлежащих разработке вопросов):

Введение. Основная часть. 1 Общие положения. 2 Аспекты игровой разработки 3 Игровой движок Unity. 4 Ход игровой программы. 4.1 Геймплей игры. 4.2 Механика движения игрового объекта «Bird». 4.3 Игровой уровень. 4.4 Создание глубины с помощью эффекта параллакса. 4.5 Игровой процесс. 4.6 Пользовательский интерфейс(UI). 4.7 Звук. 4.8 Тестирование. 5 Код игровой программы. Заключение. Список использованных источников.

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов курсовой работы | Объём этапа, % | Срок выполнения этапа | Примечания |
| Проектирование архитектуры (описание системы с точки зрения пользователя) | 9 | 20.03-25.03 |  |
| Подробный проект (спецификации) | 36 | 26.03-15.04 |  |
| Кодирование и отладка | 43 | 16.04-11.05 |  |
| Тестирование модулей | 7 | 12.05-18.05 |  |
| Интеграция (объединение модулей в систему) | 5 | 19.05-20.05 |  |

Дата выдачи задания: 06 марта 2020 г.

Руководитель И. В. Удовин

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ \_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 5

ОСНОВНАЯ ЧАСТЬ 6

1 Общие положения 6

2 Аспекты разработки игровой программы 7

3 Игровой движок Unity 7

4 Ход работы игровой программы 8

4.1 Геймплей игры 8

4.2 Механика движения игрового объекта «Bird» 9

4.3 Игровой уровень 10

4.4 Создание глубины с помощью эффекта параллакса 12

4.5 Игровой процесс 14

4.6 Пользовательский интерфейс(UI) 16

4.7 Звук 17

4.8 Тестирование 18

5 Код игровой программы 18

ЗАКЛЮЧЕНИЕ 19

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20

**ВВЕДЕНИЕ**

Цель курсовой работы – разработать игровой программный продукт под названием «Urban bird» по механике «Flappy bird», применить язык программирования C# на практике.

C# является универсальным, безопасным в отношении типов, объектно-ориентированным языком программирования. Цель C# заключается в обеспечении продуктивности работы программистов. Для этого в языке соблюдается баланс между простотой, выразительностью и производительностью. С самой первой версии главным архитектором языка C# был Андерс Хейлсберг (создатель Turbo Pascal и архитектор Delphi). Язык C# нейтрален в отношении платформ и работает с различными компиляторами, специфическими для платформ, наиболее заметной из которых считается Microsoft .NET Framework для Windows. [1]

С применением C# и .NET Framework можно создавать динамические веб-страницы, приложения Windows Presentation Foundation, веб-службы XML, компоненты для распределенных приложений, компоненты для доступа к базам данных, классические настольные приложения Windows и многое другое.

Этот язык предоставляет средства для кодирования практически любого типа программного обеспечения или компонентов для платформы Windows. C# и среда .NET привели к революционным изменениям в способе написания разработчиками программ и сделали программирование приложений для Windows гораздо более простым, чем когда-либо.

**ОСНОВНАЯ ЧАСТЬ**

1. **Общие положения**
   1. Для разработки игры используется движок Unity.
   2. За концепцию взята механика игры «Flappy bird».
   3. Применяется язык программирования C# в игровой сфере.
   4. В ходе написания игрового кода изучены многие возможности языка C#, для чего и как его применять.
   5. Реализована начальная кодовая база для развития данного проекта:

* Стартовое меню;
* меню проигрыша;
* экран загрузки;
* механизм движения и анимации игрового объекта «Bird»;
* изменение игрового окружения (эффект параллакса);
* динамическая генерация уровня;
* подсчёт очков;
  1. Разработка курсовой работы с целью создания игровой программы проведена на базе факультета ИиТП БГУИР.

1. **Аспекты разработки игровой программы**

В разработке игровой программы существуют такие аспекты, как:

* Проектирование
  + - Цель: идея, жанр, сеттинг.
    - Средство: программный код, игровой движок.
* Творчество
  + - Игровая механика: объекты, управление, физический движок, ИИ.
    - Уровни: расстановка объектов (левелдизайн).
    - Графика: арты, 2D, 3D модели, анимации, фоны, спецэффекты, оформление экрана и меню.
    - Звук: звуковые эффекты, музыка, озвучка.
* Издание
  + - Отшлифовка: устранение ошибок (бета-версия). [4]

1. **Игровой движок Unity**

Движок игры – это ее основное ядро, базовое программное обеспечение, на основе которого строятся все остальные составляющие игры, программный код, который может использоваться для создания вариаций игры или даже совершенно нового игрового мира.

Впервые определение появилось в середине девяностых годов, когда начали появляться игры, похожие на главный шутер того времени – «Doom». В то же время в свободном доступе начали появляться игровые движки, на основе которых и сторонние разработчики, и обычные пользователи могли пробовать писать собственные игры.

С тех пор игровые движки становились все более сложными технически, длинными и насыщенными по своему программному коду. Но при этом, как и в начале своего существования, они содержат в себе жестко фиксированные данные:

* Игровую логику;
* Физику объектов;
* Правила отрисовки объектов;
* Геймплей в целом.

«Поверх» движка прописываются все остальные составляющие игры, и их немало. Поэтому даже при использовании одного и того же движка в итоге виртуальные миры получаются совершенно различными.

Выбор игрового движка Unity для реализации данного проекта сделан, основываясь на двух факторах:

* низкий порог вхождения;
* чрезвычайно производительный визуальный рабочий процесс.

Также Unity обладает модульной системой компонентов, которая используется для конструирования игровых объектов. «Компоненты» в такой системе представляют собой комбинируемые пакеты функциональных элементов, поэтому объекты создаются как наборы компонентов, а не как жёсткая иерархия классов. В результате получается альтернативный подход к объектно-ориентированному программированию. [2]

1. **Ход игровой программы**
2. **Геймплей игры**

Игра «Flappy bird» имеет игровой процесс с участием 2D-графики. Цель игры состоит в управлении полётом птицы, которая непрерывно передвигается между рядами зелёных труб. При столкновении с ними происходит завершение игры. Управление производится касанием экрана, при котором птица совершает небольшой рывок вверх. При отсутствии рывков птица падает из-за силы тяжести, и игра также завершается. Очки набираются при каждом успешном перелёте между двумя трубами. Геймплей игры «Urban bird» полностью базируется на «Flappy bird».

1. **Механика движения игрового объекта «Bird»**

Объект птица представлен на рисунке 4.2.1. Движение птицы на игровом поле реализовано с помощью метода класса Bird, который использует компонент UnityEngine.Rigidbody2D. Rigidbody2D - это основной компонент, подключающий физическое поведение для 2D объекта. [5]

С прикреплённым Rigidbody, объект немедленно начнёт реагировать на гравитацию. Если добавлен один или несколько компонентов Collider, то при коллизиях (столкновениях) объект будет передвигаться. Поле типа Vector2 Rigidbody.velocity содержит вектор скорости твёрдого тела.

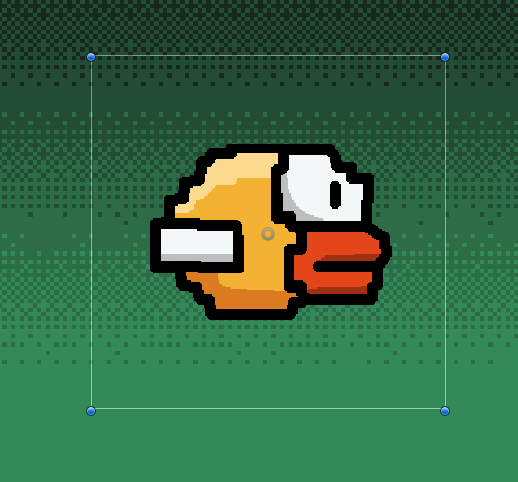


Рисунок 4.2.1 – Игровой объект «Bird»

private void Jump()

{

birdRigidbody2D.velocity = Vector2.up \* JUMP\_AMOUNT;

// дополнительный код, не относящийся к данному пункту

}

Таким образом, птица двигается только по оси Oy.

1. **Игровой уровень**

Координата x игрового объекта «Bird» не изменяется. Для создания эффекта движения относительно оси Ox двигаются игровой объект «Pipe», а также задний фон, создавая эффект параллакса. В скрипте «Level.cs» описана логика динамической генерации уровня. Трубы, представленные на рисунке 4.3.1, создаются методом CreateGapPipes(float gapY, gapSize, float xPosition):

private void CreateGapPipes(float gapY, float gapSize, float xPosition)

{

CreatePipe(gapY - gapSize \* .5f, xPosition, true);

CreatePipe(CAMERA\_ORTHO\_SIZE \* 2f - gapY - gapSize \* .5f, xPosition, false);

pipesSpawned++;

// дополнительный код, не относящийся к данному пункту

}

Трубы создаются парами в точке за экраном, координата X которой задана константой PIPE\_SPAWN\_X\_POSITION, оставляя пустое пространство, размер которого задан параметром gapSize, который уменьшается по ходу игры.

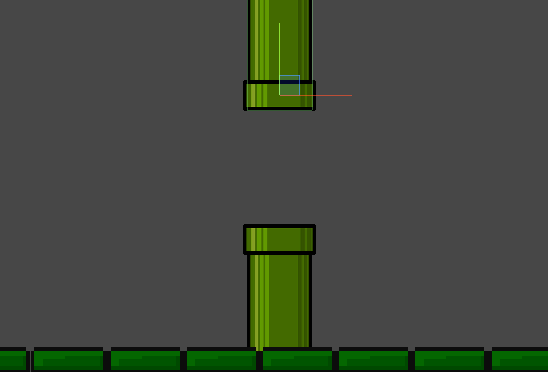
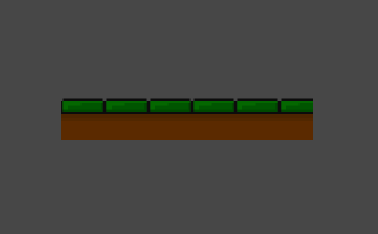


Рисунок 4.3.1 – Игровой объект «Pipe» Рисунок 4.3.2 – Игровой объект «Ground»

Как выглядит сцена за пределами экрана, показано на рисунке 4.3.3. Трубы двигаются методом HandlePipeMovement() в отрицательном направлении оси Ox с фиксированной скоростью, заданной константой PIPE\_MOVE\_SPEED, и уничтожаются за экраном, после прохождения точки, координата X которой задана константой PIPE\_DESTROY\_X\_POSITION, освобождая память:

private void HandlePipeMovement()

{

for (int i=0; i<pipeList.Count; i++)

{

Pipe pipe = pipeList[i];

pipe.Move();

// дополнительный код, не относящийся к данному пункту

if (pipe.GetXPosition() < PIPE\_DESTROY\_X\_POSITION)

{

pipe.DestroySelf();

pipeList.Remove(pipe);

i--;

}

}

}

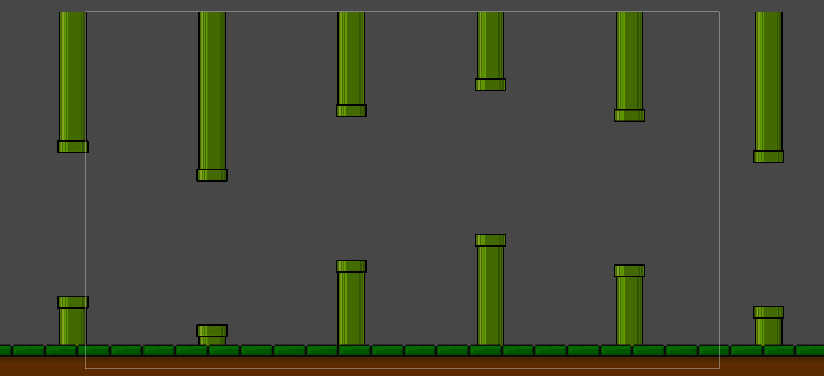


Рисунок 4.3.3 – Появление игровых объектов «Pipe»

Вызов данного метода осуществляется в функции MonoBehaviour.Update(), которая вызывается в Unity каждый кадр:

private void Update()

{

// дополнительный код, не относящийся к данному пункту

HandlePipeMovement();

HandlePipeSpawning();

HandleGround();

// дополнительный код, не относящийся к данному пункту

}

Метод HandleGround() работает по тому же принципу, что и HandlePipeMovement(), только работает с игровым объектом «Ground», который представляет землю. Он показан на рисунке 4.3.2.

1. **Создание глубины с помощью эффекта параллакса**

Паралла́кс (греч. «смена, чередование») — изменение видимого положения объекта относительно удалённого фона в зависимости от положения наблюдателя. [6]

В контексте 2D игры с видом сбоку это движение различных слоёв игровой сцены с разной скоростью, которая зависит от скорости движения камеры. В этой игровой программе камера статична, поэтому вместо скорости движения камеры будет использоваться скорость движения игровых объектов «Pipe». Игровой объект «Background» обладает 4 дочерними игровыми объектами: «StaticBackground», «FarBackground», «MiddleBackground» и «CloseBackground». Каждый из них обладает компонентом UnityEngine.SpriteRenderer, который позволяет размещать на игровой сцене спрайт. Таким образом, задний план игровой сцены является наложением четырёх изображений друг на друга. Они представлены на рисунках 4.4.1 – 4.4.4.

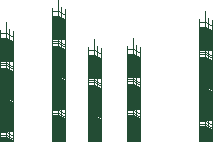




Рисунок 4.4.1 – Игровой объект «StaticBackground» Рисунок 4.4.2 – Игровой объект «FarBackground»

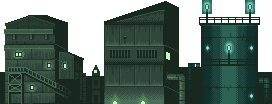




Рисунок 4.4.3 – Игровой объект «MiddleBackground» Рисунок 4.4.4 – Игровой объект «CloseBackground»

«StaticBackground» не двигается относительно камеры. Все остальные дочерние объекты «Background» обладают скриптом «Parallax.cs». Метод Update() класса Parallax двигает объекты, причём скорость их движения зависит от поля ParallaxEffect, значение которого изменяется в пределах от 0 до 1 и различно для разных слоёв игровой сцены: чем ближе план, тем больше значение его ParallaxEffect.

private void Update()

{

transform.Translate(-speed \* ParallaxEffect \* Time.deltaTime, 0, 0);

if(transform.position.x < -length)

{

Vector2 pos = new Vector2(transform.position.x + length, 0f);

transform.position = pos;

}

}

При выходе за пределы экрана объекты телепортируются вперёд на расстояние, равное длине спрайта, чтобы этого не было заметно.

1. **Игровой процесс**

Смысл игрового процесса заключается в правиле, согласно которому при столкновении птицы с верхним краем экрана, нижнем краем экрана или трубой игра заканчивается и набранный счёт фиксируется. Это можно реализовать с помощью коллайдеров – компонентов, которые определяют форму объекта для обработки физический коллизий. [7]

В Unity определено несколько их типов, в данном проекте были использованы компоненты UnityEngine.CircleCollider2D и UnityEngine.BoxCollider2D. При столкновении любых типов коллайдеров двух объектов у каждого вызывается метод OnTriggerEnter2D(Collider2D collider). У класса Bird этот метод имеет такую реализацию:

private void OnTriggerEnter2D(Collider2D collider)

{

birdRigidbody2D.bodyType = RigidbodyType2D.Static;

// дополнительный код, не относящийся к данному пункту

OnDied?.Invoke(this, EventArgs.Empty);

}

Объект «Bird» становиться статичным, и запускается событие OnDied, благодаря которому все скрипты могут “узнать” о завершении игры.

Подсчёт пройденных труб происходит в методе HandlePipeMovement() в классе Level:

private void HandlePipeMovement()

{

for (int i=0; i<pipeList.Count; i++)

{

Pipe pipe = pipeList[i];

bool isToTheRightOfBird = pipe.GetXPosition() > BIRD\_X\_POSITION;

// дополнительный код, не относящийся к данному пункту

if (isToTheRightOfBird && pipe.GetXPosition() <= BIRD\_X\_POSITION && pipe.IsBottom())

{

pipesPassedCount++;

// дополнительный код, не относящийся к данному пункту

}

// дополнительный код, не относящийся к данному пункту

}

}

В игровом процессе было выделено 3 главных состояния: ожидание старта игры, игра, конец игры. Ожидание игры – это первое состояние. Игра начинается с первым вводом и заканчивается при столкновении птицы с препятствием.

public enum State

{

WaitingToStart,

Playing,

Dead

}

Во время первого состояния игровые объекты не создаются, а задний движется медленнее. При старте игры движение фона ускоряется, объекты начинают генерироваться на игровом поле. При завершении все объекты останавливаются, а дополнительные перестают создаваться, фон полностью останавливается, вызывается меню завершения игры. Все состояния представлены на рисунках 4.5.1 – 4.5.3.

Рисунок 4.5.1 – ожидание старта игры Рисунок 4.5.2 – процесс игры

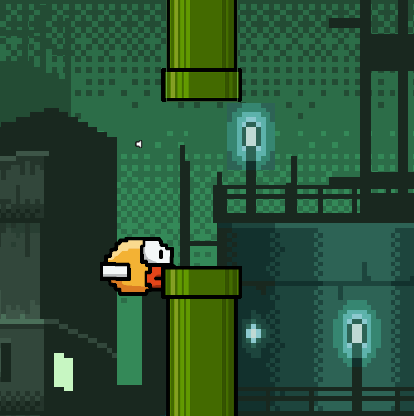


Рисунок 4.5.3 – конец игры

Все скрипты реагируют на смену состояния благодаря событиям OnDied и OnStartedPlaying класса Bird.

1. **Пользовательский интерфейс(UI)**

Интерфейс — «общая граница» между отдельными системами, через которую они взаимодействуют; совокупность средств и правил, обеспечивающих взаимодействие отдельных систем.

Интерфейс пользователя(UI) — интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы. [8]

Графический интерфейс представлен интерфейсом во время игры, главным меню, загрузочным экраном и меню завершения игры. Они представлены на рисунках 4.6.1 – 4.6.4.

Интерфейс реализован с помощью компонента UnityEngine.Canvas – абстрактного пространства, в котором производится настройка и отрисовка UI. Все UI-элементы должны быть потомками игровых объектов, к которым присоединен Canvas. Использованы элементы UI.Button, UI.Image и UI.Text. [9]

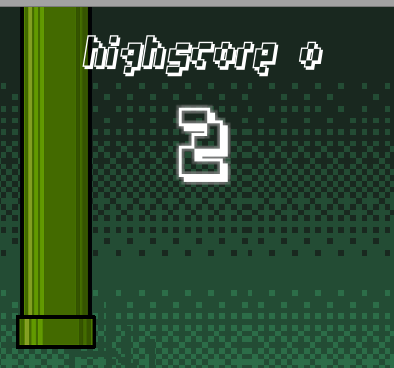
 

Рисунок 4.6.1 – главное меню Рисунок 4.6.2 – интерфейс во время игры



Рисунок 4.6.3 – меню завершения игры Рисунок 4.6.4 – экран загрузки

1. **Звук**

Звук воспроизводится при нажатии на экран и при увеличении счёта с помощью компонента AudioModule.AudioSource(источник звука). Он воспроизводит Audio Clip в сцене. Аудио клипы (Audio Clips) содержат данные аудио, используемые в источниках аудио. [10]

Звук управляется классом SoundManager и воспроизводится его методом PlaySound(Sound sound):

public static void PlaySound(Sound sound)

{

GameObject gameObject = new GameObject("Sound", typeof(AudioSource));

AudioSource audioSource = gameObject.GetComponent<AudioSource>();

audioSource.PlayOneShot(GetAudioClip(sound));

}

Например, звук нажатия на экран воспроизводится в методе Jump() класса Bird:

private void Jump()

{

// дополнительный код, не относящийся к данному пункту

SoundManager.PlaySound(SoundManager.Sound.BirdJump);

}

1. **Тестирование**

Для проверки работы программы используются тестовые сценарии в компонентном, интеграционном и системном тестировании.

* При модульном (компонентном) тестировании проверяются отдельные небольшие части приложения. При выполнении данного тестирования могут проверяться отдельные функции или методы классов, сами классы.
* При интеграционном тестировании проверяется взаимодействие между несколькими частями приложения.
* Системное тестирование направлено на проверку всего приложения как единого целого, собранного из частей, проверенных на двух предыдущих стадиях. Здесь появляется возможность полноценно взаимодействовать с приложением с точки зрения конечного пользователя, применяя множество других видов тестирования. [11]

1. **Код программы**

Код размещён на сайте <https://github.com/Artegful/UrbanBird>.

**ЗАКЛЮЧЕНИЕ**

Итогом курсовой работы стал игровой продукт «Urban bird», созданный на платформе Unity.

В процессе проведения курсовой работы изучены возможности применения языка C# в игровой сфере. Актуальность работы состоит в том, что в дополнении к общему курсу изучения языка C# рассмотрены возможности игрового движка Unity и новые способы применения C# на практике.

В результате исследования были выявлены следующие недостатки полученного программного обеспечения:

* + 1. примитивность игрового продукта;
    2. отсутствие уникальных черт продукта, отличающих его от остальных «Flappy bird-like» игр.

В результате созданный игровой продукт требует большего количества времени и средств, а также улучшения творческого аспекта.

Достоинствами данного продукта является следующее:

1. Механика движения птицы, основанная на физике Unity
2. Возможность сохранения рекордов игрока
3. Динамически генерируемое окружение
4. Анимация и звук
5. Интуитивно понятный интерфейс
6. Кроссплатформенность

Таким образом, данный игровой продукт может использоваться в определенном круге обращения.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

Албахари, Джозеф, Албахари, Бен. C# 7.0. Справочник. Полное описание языка.: Пер. с англ. – СпБ.: ООО «Альфа-книга», 2018. – 1024 с.

Хокинг Дж. Unity в действии. Мультиплатформенная разработка на C#.: Пер. с англ. И. Рузмайкиной. − СПб.: Питер, 2016. − 336 с.

Бонд, Джереми. Unity и C#. Геймдев от идеи до реализации.: – СПб.: Питер, 2019. – 928 с.

UnityEngine.RigidBody2D – Unity Scripting API [Электронный ресурс]. – 2019. – Режим доступа: <https://docs.unity3d.com/ru/current/ScriptReference/Rigidbody2D.html>

Параллакс – Википедия [Электронный ресурс]. – 2019. – Режим доступа: <https://ru.wikipedia.com/wiki/Параллакс>

Unity – Manual: Colliders [Электронный ресурс]. – 2020. – Режим доступа: <https://docs.unity3d.com/Manual/CollidersOverview.html>

Интерфейс пользователя – Википедия [Электронный ресурс]. – 2020. – Режим доступа: [https://ru.wikipedia.org/wiki/ Интерфейс\_пользователя](https://ru.wikipedia.org/wiki/%20Интерфейс_пользователя)

Холст(Canvas) – Unity Manual [Электронный ресурс]. – 2018. – Режим доступа: <https://docs.unity3d.com/ru/current/Manual/class-Canvas.html>

Audio Source – Unity Manual [Электронный ресурс]. – 2018. – Режим доступа: <https://docs.unity3d.com/ru/current/Manual/class-AudioSource.html>

Куликов, С. C. Тестирование программного обеспечения. Базовый курс.: Минск: Четыре четверти, 2017. − 312 с.