**

Колледж космического машиностроения и технологии

**КУРСОВОЙ ПРОЕКТ**

**По дисциплине «Прикладное программирование»**

**Тема: «Реализация компьютерной игры типа «Марио» на Unity»**

Выполнил студент

Братцев Максим Алексеевич

Группа П2-16

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(подпись)

\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Королев, 2019

Оглавление

[Задание на выполнение курсового проекта 3](#_Toc11486202)

[Введение 4](#_Toc11486203)

[**1.** **Теоретическая часть** 5](#_Toc11486204)

[1.1. Этапы создания компьютерной игры 5](#_Toc11486205)

[1.1.1. Проектирование 5](#_Toc11486206)

[1.1.2. Творчество 7](#_Toc11486207)

[1.1.3. Издание 9](#_Toc11486208)

[**2.** **Проектная часть** 11](#_Toc11486209)

[2.1. Выбор инструментария 11](#_Toc11486210)

[2.1.1. Игровой движок Unity 11](#_Toc11486211)

[2.1.2. Язык программирования C# 11](#_Toc11486212)

[2.1.3. Достоинства и недостатки языка C# 11](#_Toc11486213)

[2.2. Основные требования к проекту 12](#_Toc11486214)

[2.3. Выбор инструмента для написания проекта 12](#_Toc11486215)

[2.4. Список компонентов и модулей приложения 13](#_Toc11486216)

[2.5. Игра 15](#_Toc11486217)

[2.5.1. Основы 15](#_Toc11486218)

[2.5.2. Создание игровой сцены (уровня) 15](#_Toc11486219)

[2.5.3. Создание игрового персонажа 18](#_Toc11486220)

[2.5.4. Управление персонажем 20](#_Toc11486221)

[2.5.5. Камера 20](#_Toc11486222)

[2.5.6. Главное меню 21](#_Toc11486223)

[**3.** **Организационная часть** 22](#_Toc11486224)

[3.1. Назначение программы 22](#_Toc11486225)

[3.2. Порядок установки программного средства 22](#_Toc11486226)

[3.3. Руководство оператора 22](#_Toc11486227)

[3.4. Минимальные системные требования. 27](#_Toc11486228)

[Заключение 28](#_Toc11486229)

[Список используемой литературы 29](#_Toc11486230)

[Приложение 1. Листинг игрового персонажа (Player.cs) 30](#_Toc11486231)

[Приложение 2. Листинг контроллера камеры 36](#_Toc11486232)

[Приложение 3. Листинг главного меню 37](#_Toc11486233)

# Задание на выполнение курсового проекта

Разработать 2D-игру жанра «платформер» на языке программирования C# с использованием игрового движка Unity.

# Введение

Данный курсовой проект посвящен созданию 2D-игры на движке Unity. Выбором такого жанра игры, как платформер, послужили: возможность понимания основных механик в создание игра, творчество в создание игры.

Цель курсового проекта заключалась в понимании основных принципов создание 2D-игры и изучения игрового движка Unity.

Для осуществления обозначенной цели служат следующие задачи:

1. Изучение литературы и документации по Unity
2. Создание игрового персонажа
3. Создания компонентов для игры
4. Создания интерфейса программы
5. Написание кода
6. Описание руководства оператора
7. **Теоретическая часть** 
   1. Этапы создания компьютерной игры
      1. Проектирование

**Жанр**

Жанр игры необходимо выбрать в самом начале в обязательном порядке. Жанр и будет основным направлением развития игры.



Рис. Жанры игр

Жанр (Рис. 1) – это своеобразный фундамент всей игры. В зависимости от выбранного жанра различается получаемый игровой опыт и основные игровые механики.

**Сеттинг**

Игровые жанры определяют лишь основные действия, которые будут совершать игроки в процессе игры, тем самым они отвечают только на вопрос "ЧТО?". На вопросы "ГДЕ?" и "КОГДА?" отвечает другая основная характеристика игры – сеттинг.

**Сеттинг** – это принадлежность игры к какой-то сюжетной теме или к определённому виртуальному миру.

Существует множество видов сеттингов, от средневекового фэнтези до научной фантастики, в каждом виде присутствует свои определённые правила и условности.

**Программный код**

Программный код играет роль инструмента, он представляет собой каркас (скелет), на который будут нанизываться результаты всех последующих этапов разработки.

Прежде всего, мы должны выбрать язык программирования, который нам наиболее подходит оперировать двухмерными объектами в пространстве. По ходу разработки придётся изучить все форматы изображений и аудиофайлов, всевозможные кодеки и кодировки.

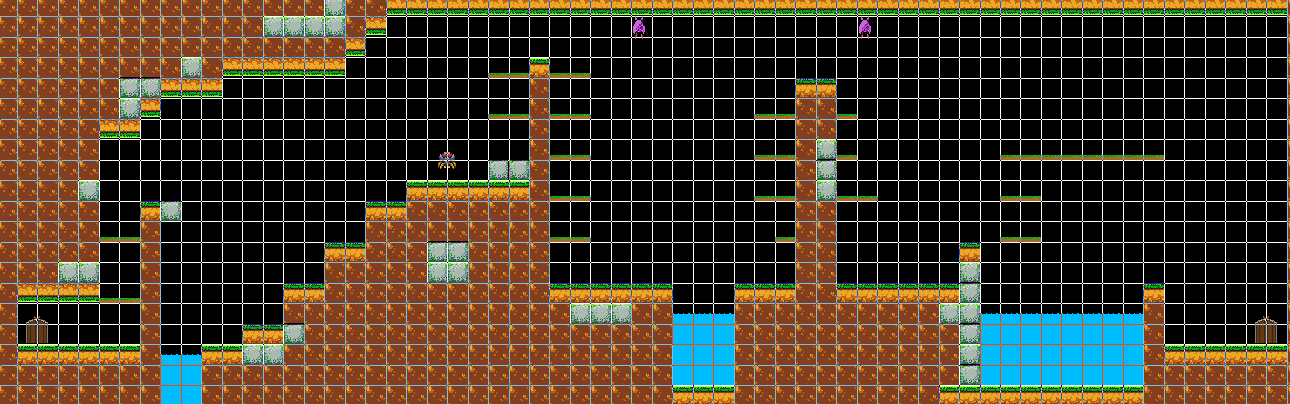
* + 1. Творчество

**Игровая механика**

**Игровая механика** – совокупность определенных методов взаимодействия игры с игроком. По сути своей, игровая механика представляет собой свод правил, по которым будет функционировать игра, то есть ответами на вопросы, по типу «Что будет, если я сделаю, то или это действие?».

**Уровни**

Созданные игровые объекты расставляются в отдельных виртуальных пространствах – уровнях (локациях) (рис. 4). Игры чаще всего содержат множество отдельных уровней, переход между которыми происходит по ходу сюжета.

 Рис. 2 Шаблон уровня

На каждом отдельном уровне расставляются игровые объекты, стенки, платформы, декорации, фоны. Уровни создаются в играх всех жанров.

**Оформление**

На данном этапе происходит улучшение графической составляющей игры.

Арты

Для начала нужно создать образы героев, врагов, игровых предметов, задних фонов. Первоначально они рисуются либо на бумаге, либо на компьютере с использованием графического планшета (рис. 5). А потом с помощью образов создается персонаж.

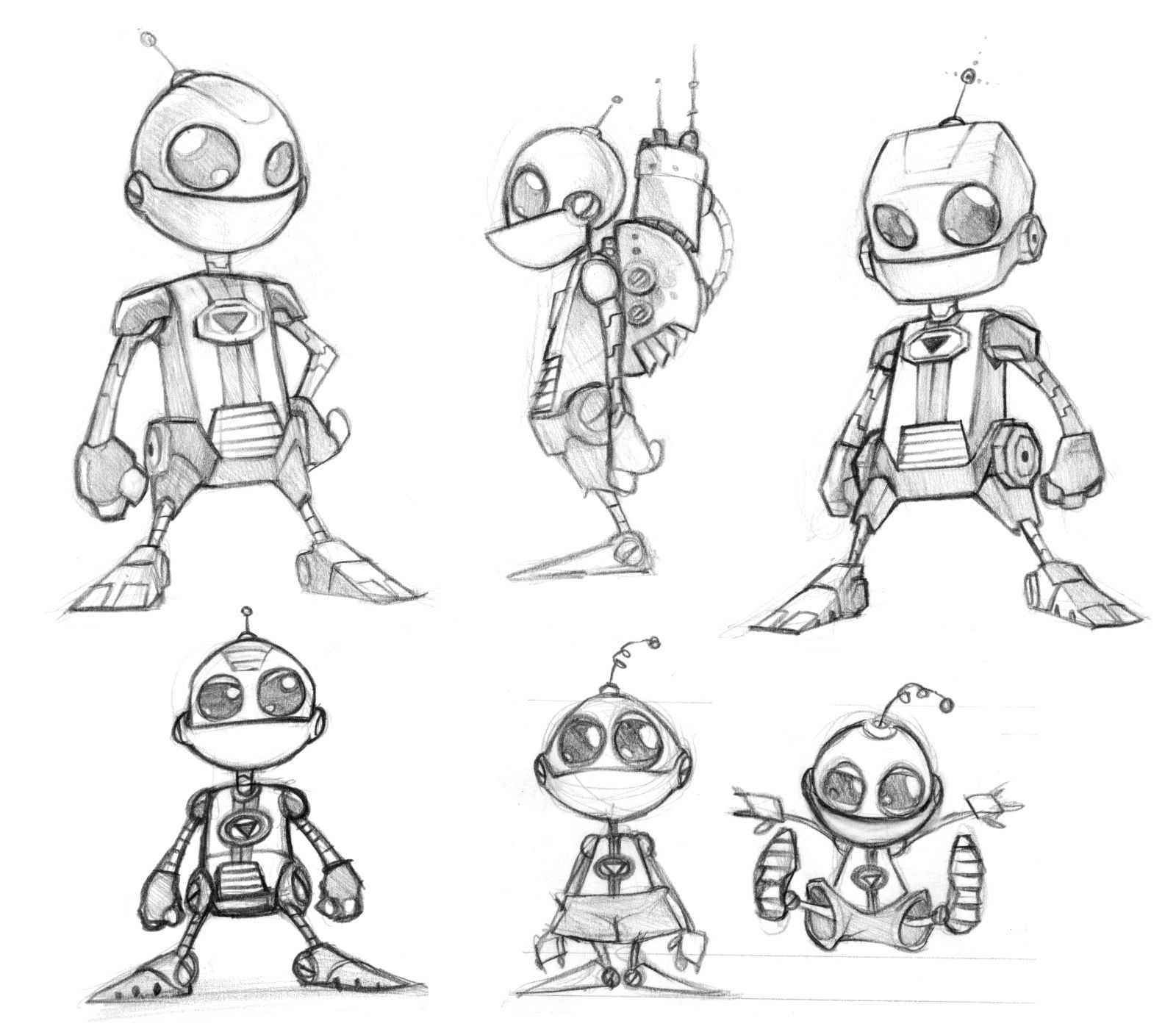


Рис. 3 Пример образа персонажа

Анимации

Для нестатических игровых объектов создаются анимации. Анимации делают игру живой и добавляют динамики, как игровому персонажем, так и окружению.

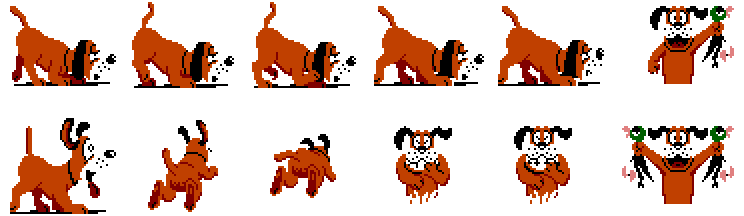


Рис. 4 Пример покадровой анимации

Фоны

Чаще всего фон – это статичное изображение или изображение, имеющее небольшие анимированные объекты. Фон придаёт окружению глубины

* + 1. Издание

Процесс разработки большой игры построен таким образом, что различными её элементами занимаются различные специалисты. На начальном этапе игра представляет собой разрозненный набор творческих наработок в различных областях искусства: изображения, звуки, 3D-модели, архитектура, тексты, сценки, видеовставки, оформление. С помощью программных средств разрозненные объекты соединяются в единую сложную систему.

**Сведение материала (a-версия)**

При построении игры на игровом движке объединение объектов происходит постепенно с самого начала процесса. Пока игра не собрана до конца, её называют альфа версией. В этот момент уже можно заниматься тестированием отдельных уровней, скриптов и прочих механизмов.

На этом этапе уже технически возможно выпустить демо-версию или хотя бы видеоролик с игровым процессом, чтобы заранее привлечь игроков к своему проекту.

**Устранение ошибок (b-версия)**

Когда игра полностью собрана, остаётся лишь устранить получившиеся ошибки (bugs). Они появляются в любом случае, так как игра – это система со сложной структурой. Сами элементы игры наглядны и просты, но связи между ними настолько сложны и витиеваты, что процесс отладки и устранения ошибок может занимать до 40% всего времени разработки проекта. Полностью собранная, но ещё не проверенная на ошибки игра называется бета версией.

1. **Проектная часть**
   1. Выбор инструментария
      1. Игровой движок Unity

Unity — межплатформенная среда разработки компьютерных игр. Unity позволяет создавать приложения, работающие под более чем 20 различными операционными системами, включающими персональные компьютеры, игровые консоли, мобильные устройства, интернет-приложения и другие.

* + 1. Язык программирования C#

C# —язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Разработан в 1998—2001 годах группой инженеров под руководством Андерсa Хейлсбергa в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET. Компилятор с C# входит в стандартную установку самой .NET, поэтому программы на нём можно создавать и компилировать даже без инструментальных средств вроде Visual Studio.

* + 1. Достоинства и недостатки языка C#

К многочисленным преимуществам этого языка относят:

* поддержку подавляющего большинства продуктов Microsoft;
* для небольших компаний и некоторых индивидуальных разработчиков бесплатными являются такие инструменты, Visual Studio, облако Azure, Windows Server, Parallels Desktop для Mac Pro и многие другие;
* большое количество синтаксического сахара, представляющего собой специальные конструкции, разработанные для понимания и написания кода. Они не имеют значения при компиляции;
* порог вхождения у языка C# низкий. Его синтаксис имеет много схожего с другими языками программирования, благодаря чему облегчается переход для программистов. Язык C# считается наиболее понятным и подходящим для новичков;
* сегодня в любом регионе России имеется много вакантных мест на должность C#-программиста.

Кроме многочисленных плюсов этот язык программированияимеет некоторые недостатки. Среди них следует выделить:

* приоритетная ориентированность на Windows платформу;
* язык бесплатен только для небольших фирм, индивидуальных программистов, стартапов и учащихся . Крупной компании покупка лицензионной версии этого языка обойдется в круглую сумму;
* в языке осталась возможность использования оператора безусловного перехода.
  1. Основные требования к проекту

Проект должен запускаться в полноэкранном режиме. С элементами управления, возможностью настройки разрешения экрана.

**Стадии и этапы разработки:**

* Разработка механики игры
* Разработка уровней
* Сведение материала (a-версия)
  1. Выбор инструмента для написания проекта

Для написания проекта мной были выбраны: C# и Unity.

Игровой движок Unity предназначен для написания мультимедийных приложений, таких как игры. Unity отлично подходит для создания простейших 2D-игр и C# один из возможных языков программирования с помощью которого можно писать скрипты под этот движок.

* 1. Список компонентов и модулей приложения

Компоненты:

* Animation – компоненты с анимацией
* Font – компоненты с шрифтом
* Scene – компоненты с игровыми сценами
* Script – компоненты со скриптами
* Sprite – компоненты спрайтов

Модули:

* Airflow.cs – модуль класса объекта воздушный поток
* BoxScript.cs – модуль класса объекта коробки
* CameraController.cs – модуль управления камеры
* Coin.cs – модуль класса объекта монета
* Damping.cs – модуль для затухающих объектов
* End.cs – модуль окончания игры
* GameController.cs – модуль отвечающий за игровую логику
* Glowworm.cs – модуль класса объекта светлячка
* Location5.cs – модуль локации №5 на игровой сцене
* MainMenu.cs – модуль главного меню
* MovePlatform.cs – модуль отвечающий за передвижения движущихся платформ
* Pause.cs – модуль меню паузы
* Platform.cs – модуль базового класса движущихся платформ
* PlatformHorizontal.cs – модуль класса горизонтальных платформ
* PlatformVertical.cs – модуль класса вертикальных платформ
* Player.cs – модуль класса игрового персонажа
* Pazzle1.cs, Pazzle2.cs, Pazzle3.cs – модуль управления головоломок.
* RetractableDoor.cs – модуль класса объекта убирающийся двери
* SlidingDoor.cs – модуль класса объекта скользящая дверь
* Trigger.cs – модуль базового класса триггер
* TriggerBlock.cs – модуль класса блока с триггером
* TriggerButton.cs – модуль класса кнопки с триггером
* TriggerField.cs – модуль класса поля с триггером

* 1. Игра
     1. Основы

Проект в Unity делится на сцены — отдельные файлы, содержащие свои игровые миры со своим набором объектов, сценариев, и настроек. Сцена в Unity имеет сетку и координатную плоскость, объекты, расположенные на сцене, имеют координаты в соответствии с этой плоскостью, для этого у них есть компонент Transform, который является обязательный для каждого объекта.

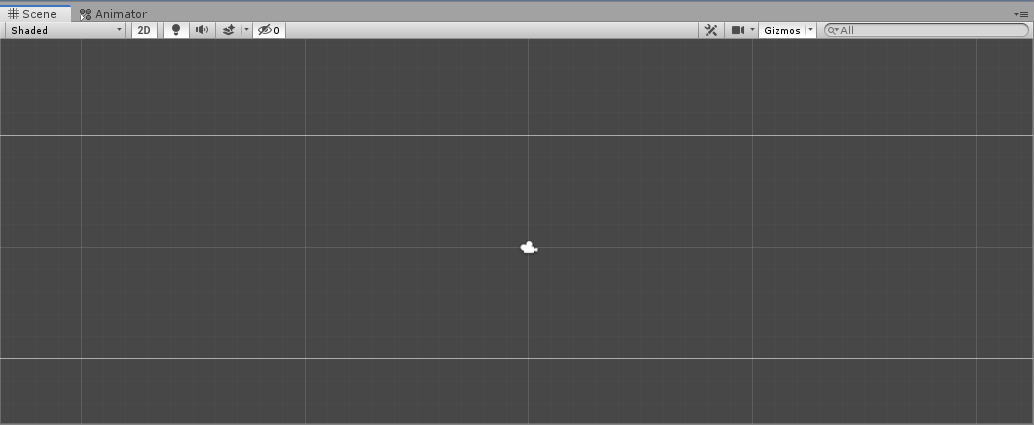


Рис. 5 Окно сцены в Unity

* + 1. Создание игровой сцены (уровня)

В первую очередь требуется создать сцену, на которой и будет происходить весь игровой процесс.

В окне Hierarchy создаём пустой объект, который будет отображён в окне сцены. Это можно сделать, щёлкнув правой кнопкой мыши по пустому пространству окна Hierarchy, а затем выбрать Create Empty.

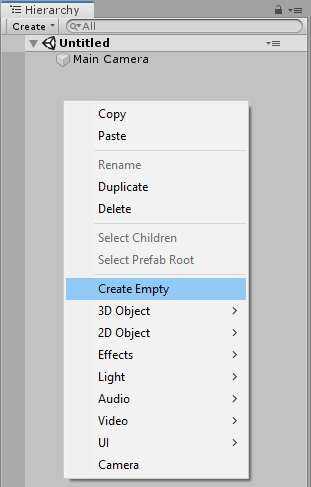


Рис. 6 Создание пустого объекта через окно Hierarchy

К этому объекту можно добавить компоненты, такие как Sprite Renderer (для отображения объекта в виде определённого спрайта в игре) и Box Collider 2D (для коллизии объекта в двухмерном пространстве).

На Рис. 7 представлен вид объекта в окне Inspector, в этом окне видны все компоненты выделенного объекта и параметры этих компонентов.

Осталось добавить в параметр Sprite компонента Sprite Renderer файл в требуемом формате, чтобы объект приобрёл вид этого спрайта, а также подкорректировать размер коллизии (Box Collider 2D) под размер спрайта.

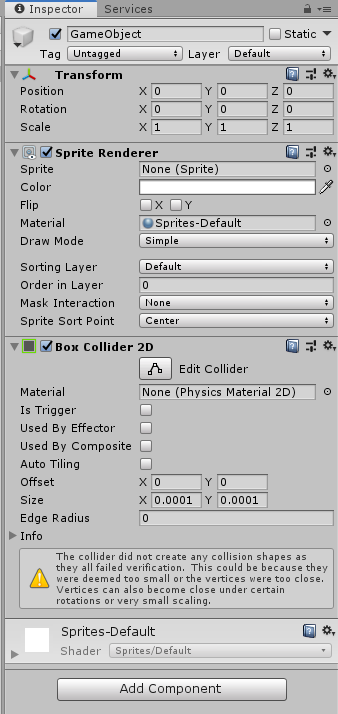


Рис. 7 Компоненты созданного объекта

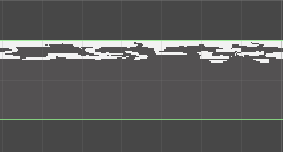


Рис. 8 Вид объекта в окне Scene

Теперь, когда объекты отображаются на сцене и имеют коллизию, можно создать первую локацию (Рис. 9). Остаётся сохранить созданную сцену в специально отведённой директории. Сцены в Unity имеют расширение .unity

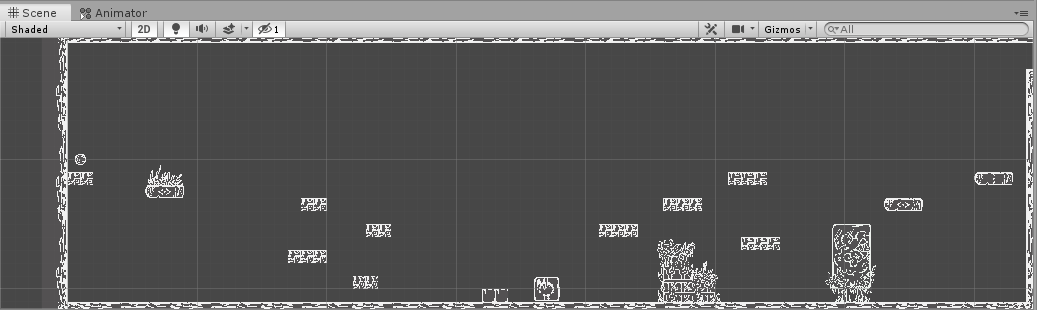


Рис. 9 Первая локация на уровне.

* + 1. Создание игрового персонажа

Игровой персонаж представляет собой объект с множеством компонентов, такими как:

* Компонент спрайта (Sprite Renderer), влияющий на отображение главного героя в игровом пространстве.
* Компонент твёрдого тела (Rigidbody 2D), задающий физику.
* Компонент коллайдера (Box Collider 2D), позволяющий взаимодействовать с объектами в игровом мире.
* Компонент анимации (Animator), проигрывающий определённые анимации при заданных условиях.
* Компонент скрипта (Player (Приложение 1)), осуществляющий логику игрового персонажа, его управления реализовано с помощью класса Input. Для чтения информации с оси используется Input.GetAxis() со стандартным обозначением осей: "Horizontal".

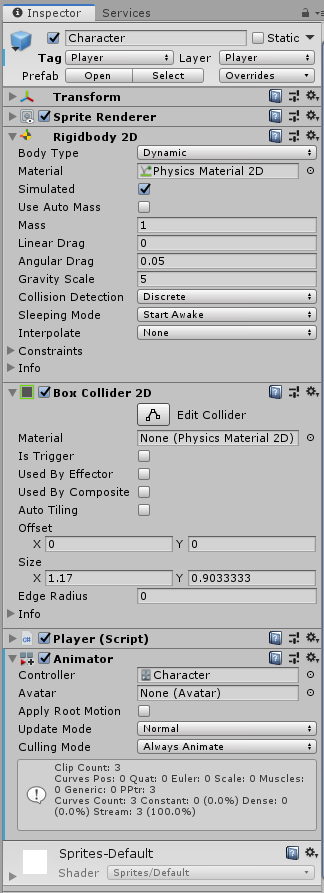


Рис. 0 Компоненты игрового персонажа

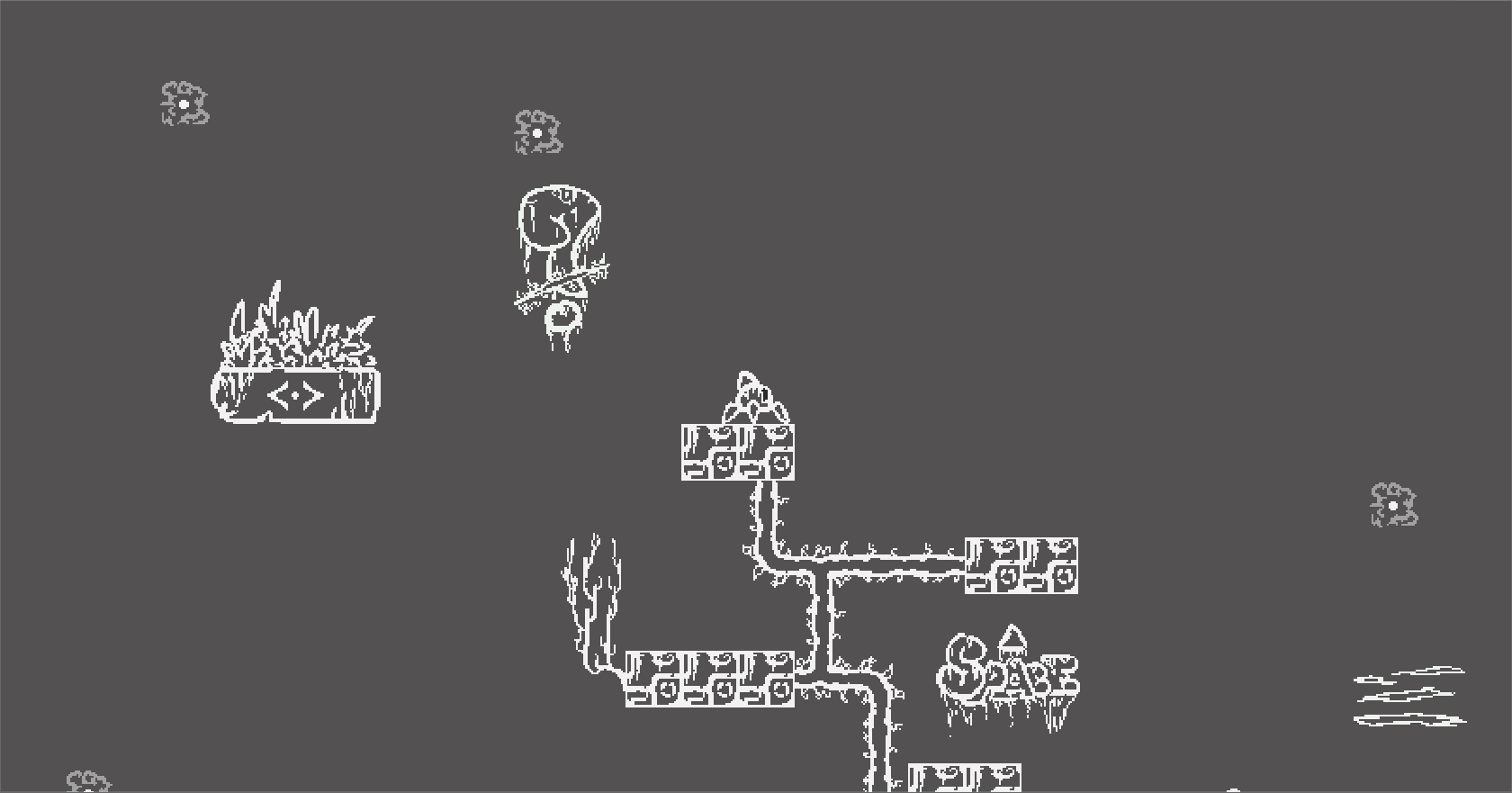
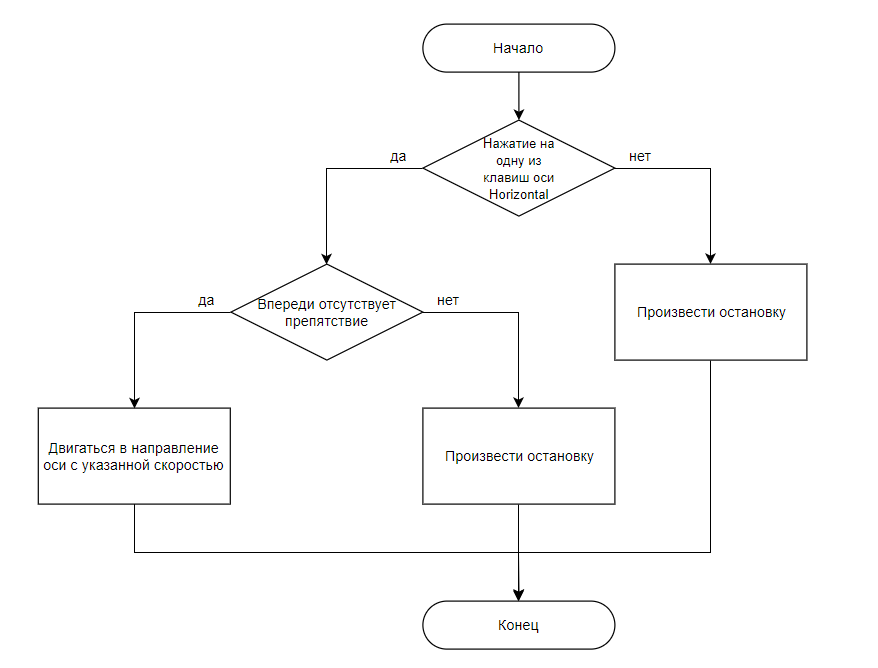


Рис. 11 Персонаж в игровом мире

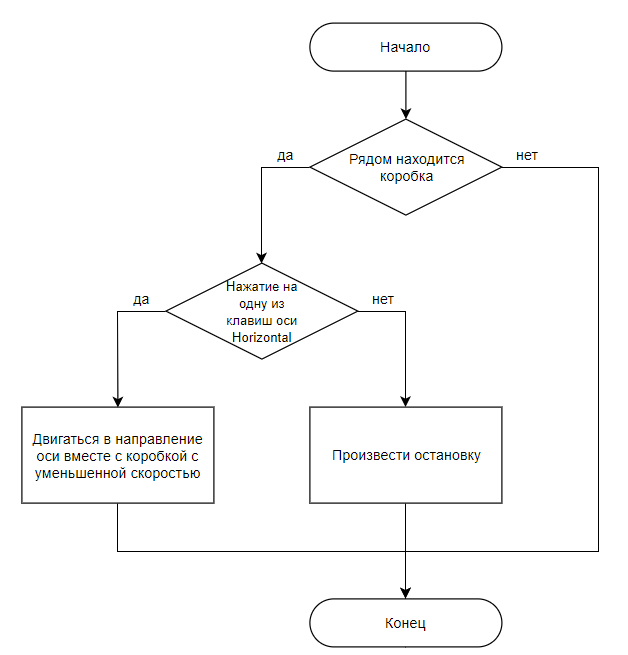
* + 1. Управление персонажем

Управление персонажем осуществляют методы класса Player:

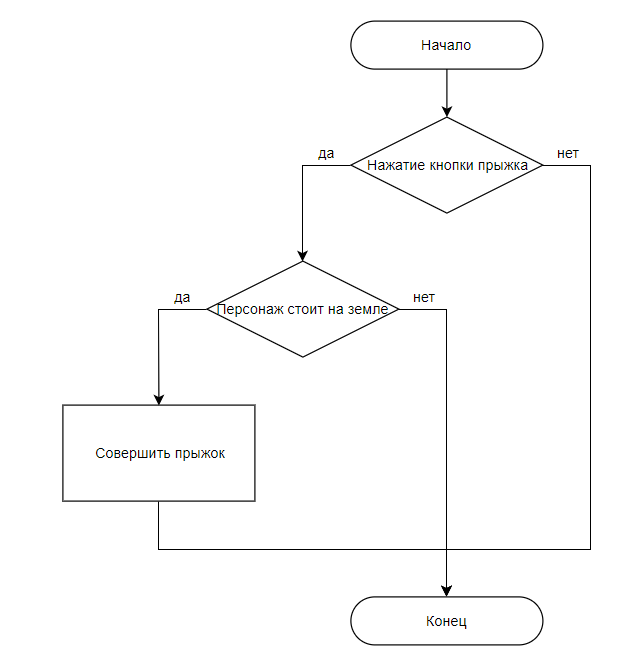
* Move() – передвижение по горизонтали. За перемещение отвечают клавиши «A» и «←» для передвижение в левую сторону, «D» и «→» для передвижения в правую сторону.
* MoveBox() – передвижение по горизонтали вместе с коробкой. При соприкосновении с коробкой и с зажатой клавишей «Ctrl» игровой персонаж передвигается вместе с ней.
* Jump() – прыжок. Осуществляется нажатием клавиши «space».



Блок-схема 1. Метод Move()



Блок-схема . Метод MoveBox()



Блок-схема . Метод Jump()

* + 1. Камера

После того как мы создали блок управления героем, замечаем что персонаж может уйти за границу экрана, а экран не следует за ним. Что бы это исправить был создан скрипт камеры CameraController.cs (Приложение 2). Данный скрипт плавной централизует камеру на заданном объекте – цели. Эта цель может изменяться на протяжении игры, тем самым оставляя в поле зрении игрока самое главное.

* + 1. Главное меню

Главное меню представляет собой отдельную сцену, на которой находятся пользовательский интерфейс, в моём случае это текст – название игры и кнопки: старта игры, выхода из игры.

При нажатии на кнопку старта сцена сменится с главного меню на сцену с игровым уровнем, а кнопка выхода закроет игру.

За организацию логики главного меню отвечает скрипт MainMenu.cs (Приложение 3), а конкретнее методы StartPressed() и ExitPressed().

1. **Организационная часть**
   1. Назначение программы

Данная программа предназначена для проведения досуга (компьютерная игра).

* 1. Порядок установки программного средства

Установку данной игры производится путём скачивания папки хранящей в себе .exe и игровые файлы.

Для открытия игры требуется запустить exe-файл.

* 1. Руководство оператора

Программа имеет простой и понятный интерфейс, что делает ее более доступной пользователю.

После запуска программы откроется вступительная заставка характерная для бесплатной версии движка Unity (Рис. 11).



Рис. 12 Вступительная заставка движка игры

Затем открывается основное меню (рис. 12):



Рис. 13 Главное меню игры

В главном меню есть выбор из двух кнопок:

* Start – запускает игру
* Exit – выходит из игры

После запуска игры игрок попадает на уровень (Рис. 13)

Управление:

* «A» и «←» для передвижение в левую сторону
* «D» и «→» для передвижения в правую сторону
* «Пробел» для прыжка
* зажатие «Ctrl» для передвижения с коробкой
* «R» для перезапуска комнаты с головоломкой
* «Esc» для паузы
* «C» для нажатия на кнопку

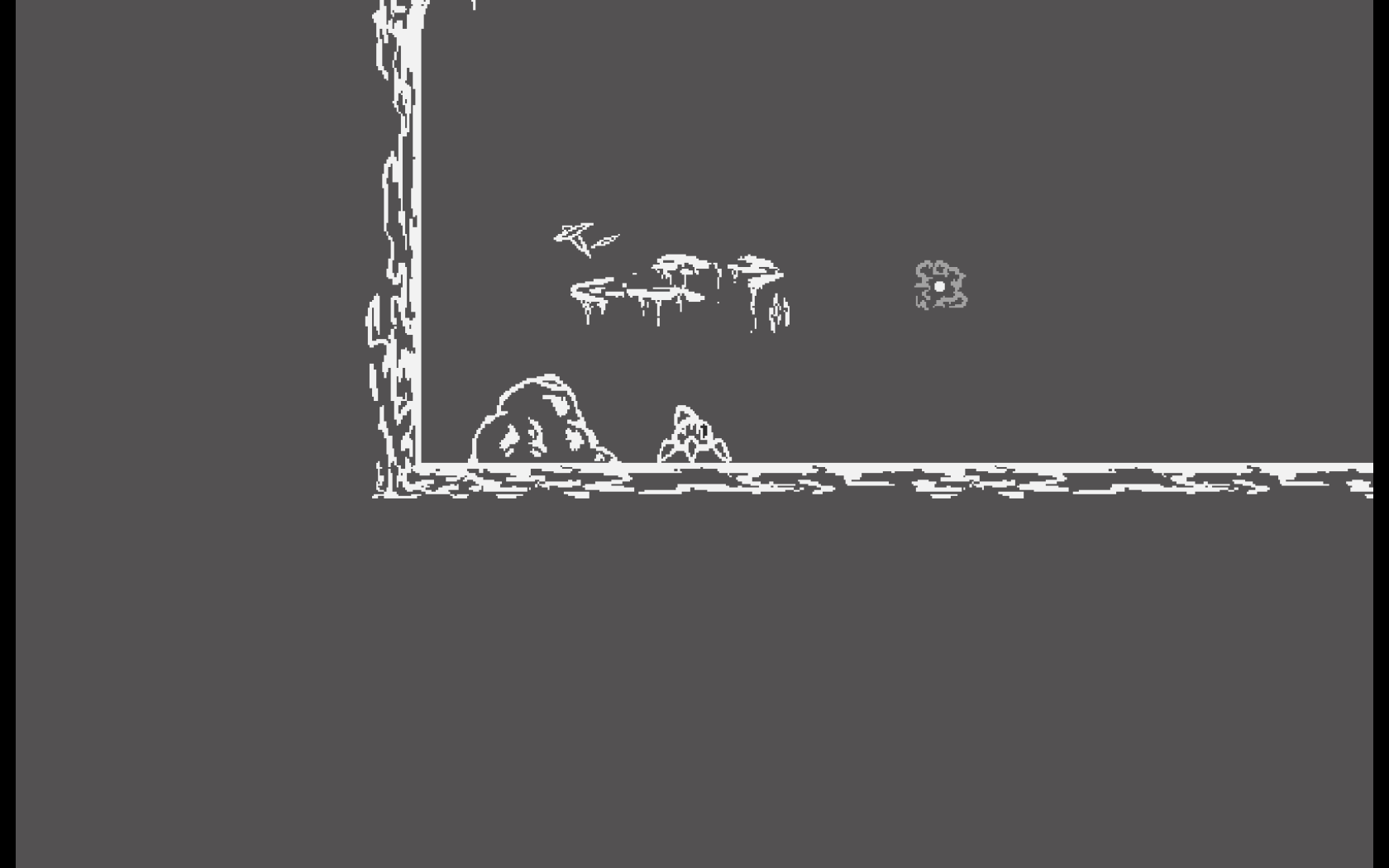


Рис. 14 Игра

Для прохождения игры требуется разрешить все головоломки на уровне, в конце последней будет конец игры.

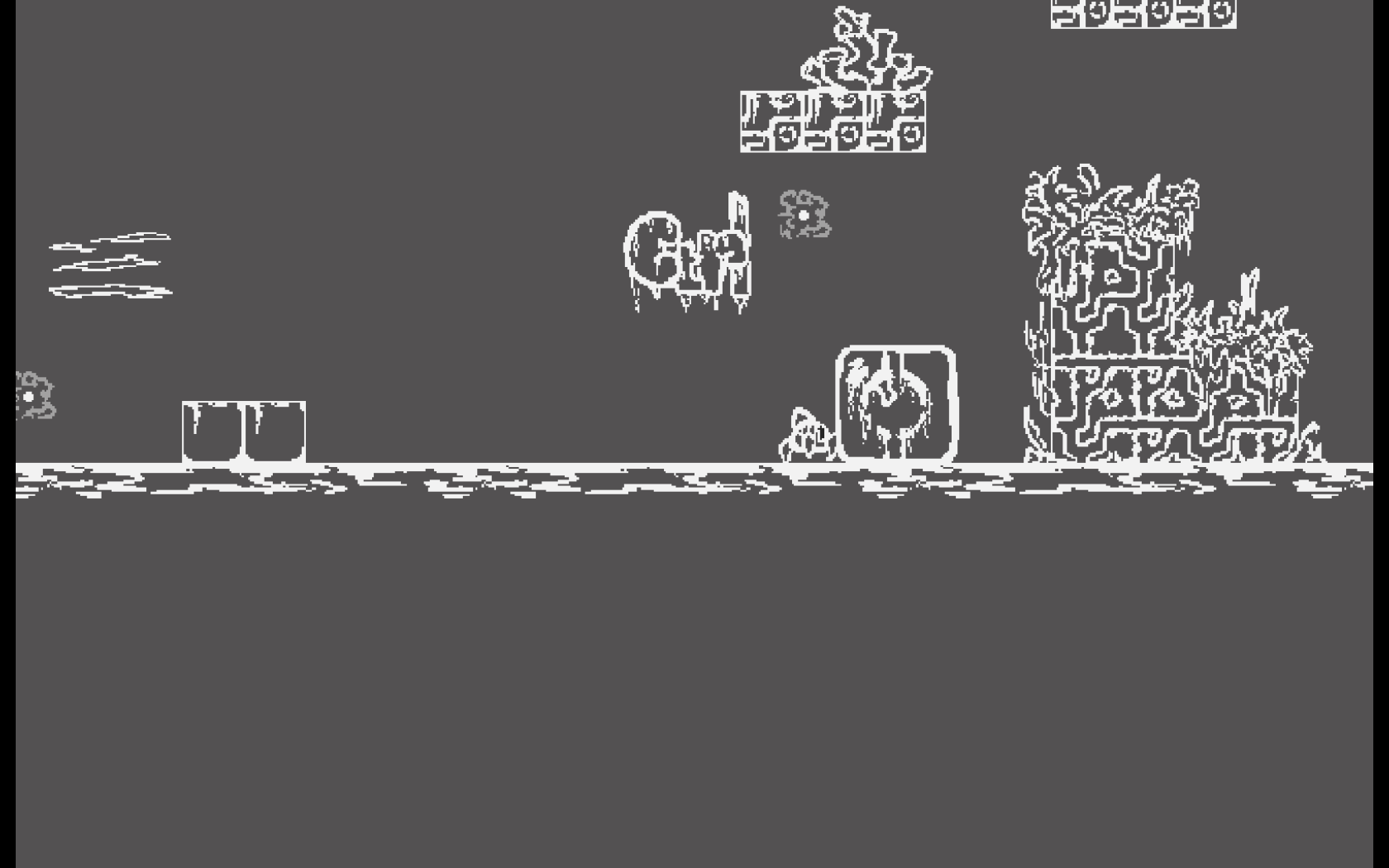


Рис. 15 Обучение толканию коробки

Уровень разбит на 5 локаций, 3 из которых являются головоломками. Для того чтобы попасть к первой головоломке игроку предстоит пройти первую локацию, которая является ознакомительной и учит игрока управлению и основам игры.

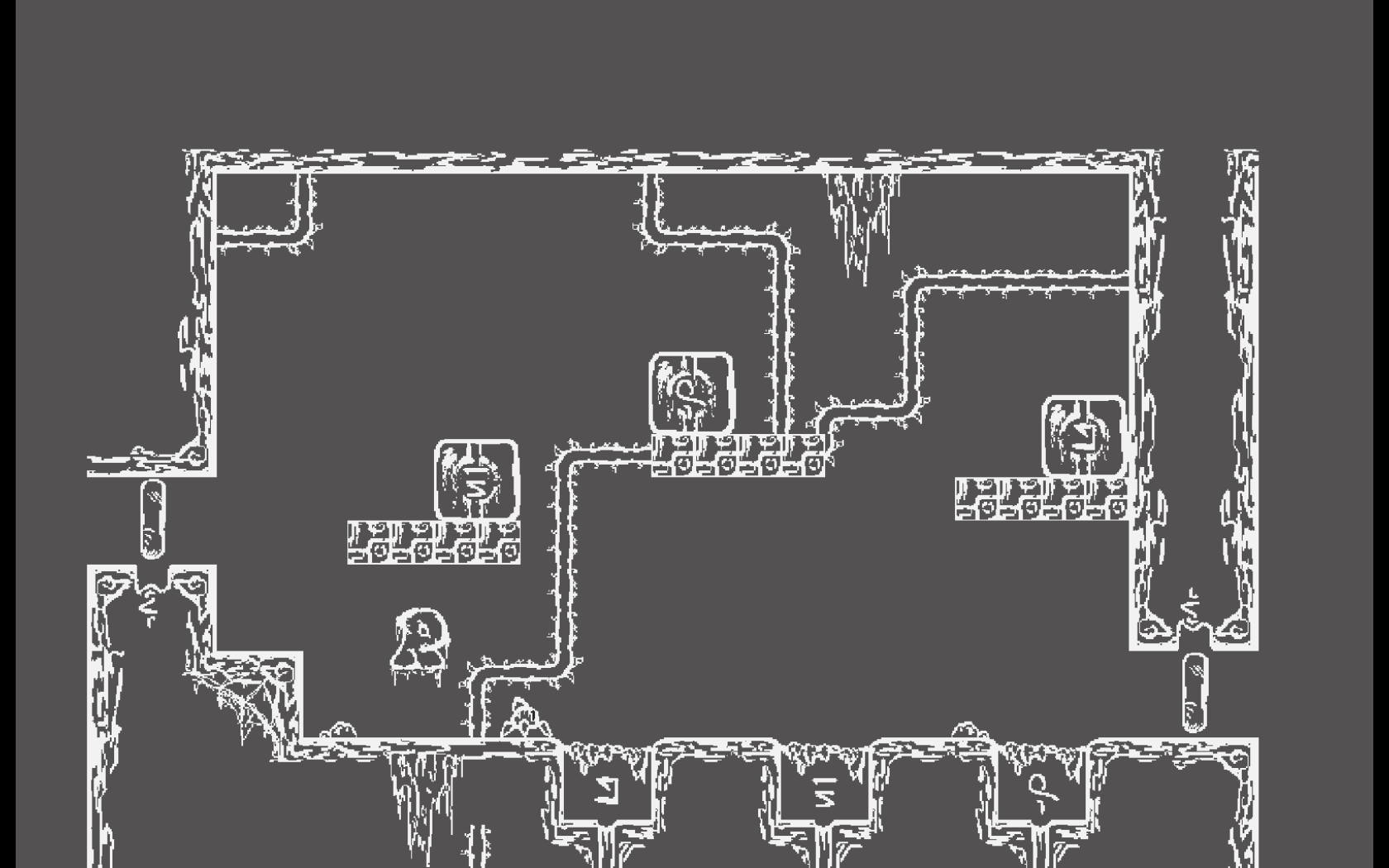


Рис. 16 Первая головоломка

Для прохождения первой головоломки (Рис. 15) требуется расположить каждую из коробки в место с тем же знаком, что и на ней. После этого дверь откроется и даст возможность пройти в следующую локацию.

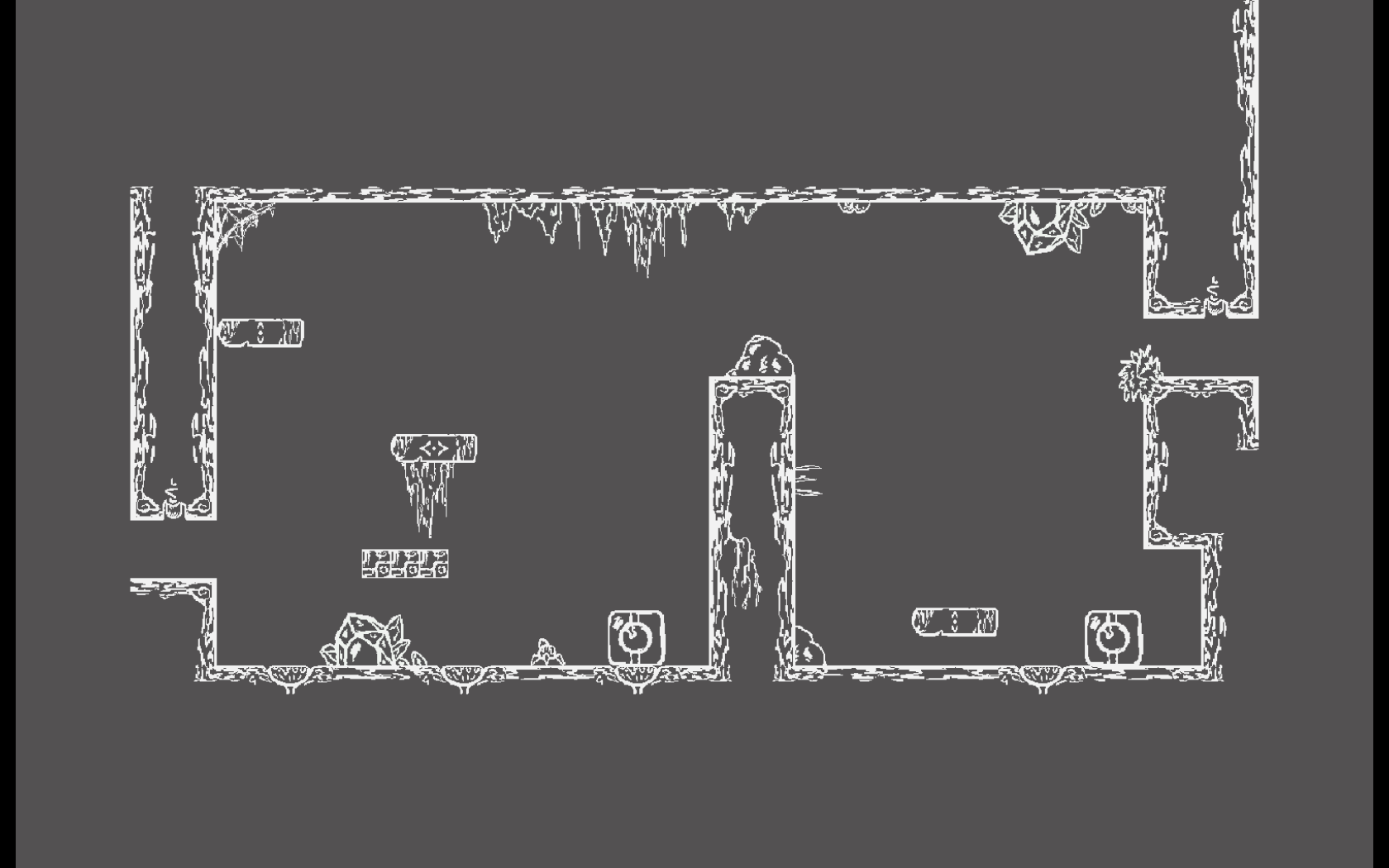


Рис. 17 Вторая головоломка

Чтобы пройти вторую головоломку (Рис. 16) надо перемещая коробку по плитам активирующие платформы.

После прохождения второй головоломки, игрок попадает и так называемую «зону отдыха» (Рис. 17) - безопасную локацию, где он может ненадолго отдохнуть

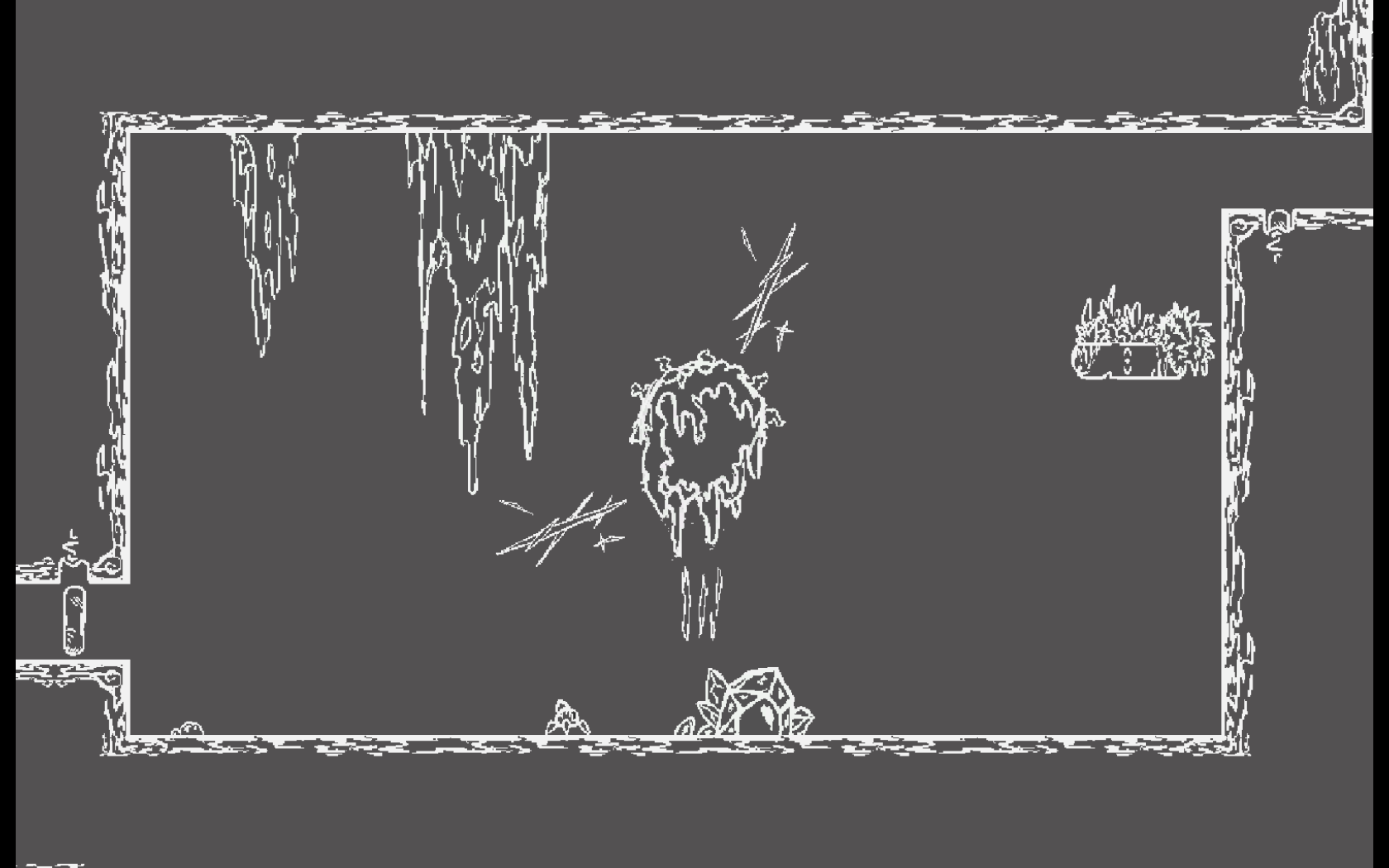


Рис. 18 Локация 4 - Зона отдыха

Для перехода на следующую локацию игроку необходимо запрыгнуть на платформу.

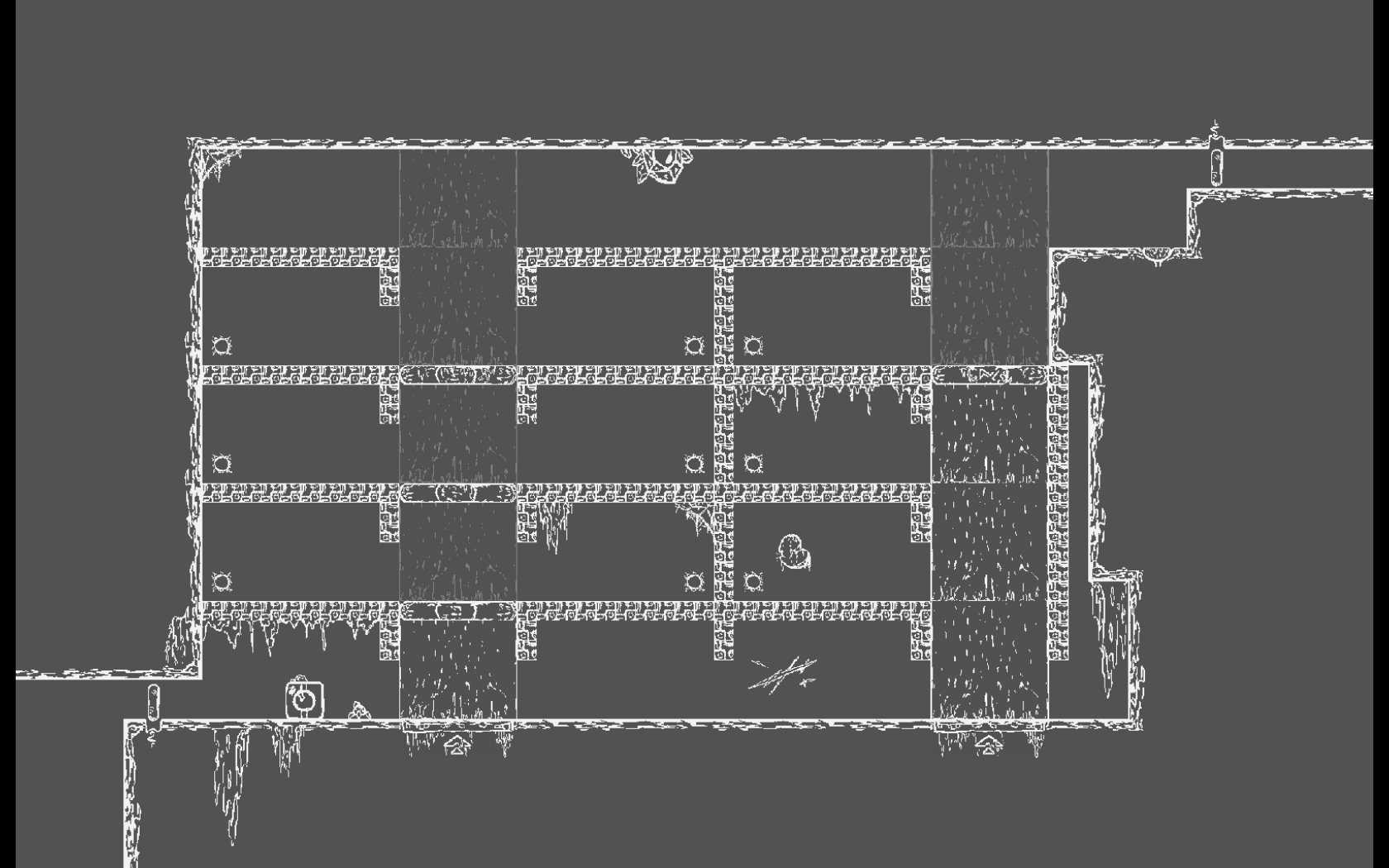


Рис. 19 Третья головоломка

Данная головоломка является последней, для её прохождения требуется донести коробку из начала в конец локации. В этом помогут воздушные потоки, которые способны поднимать как коробку, так и игрока. А также будет препятствовать двери, которые открываются и закрываются от кнопок (две из которых являются сломанными и не работают). Каждая кнопка подсоединена к нескольким дверям и требуется найти правильную комбинацию для решения головоломки.

И вот после прохождения последний головоломки игрок попадает в длинный коридор (Рис. 19). Камера начинает приближаться, а экран тускнеть и после того как всё исчезнет, откроется главное меню и игра будет считаться пройденной.

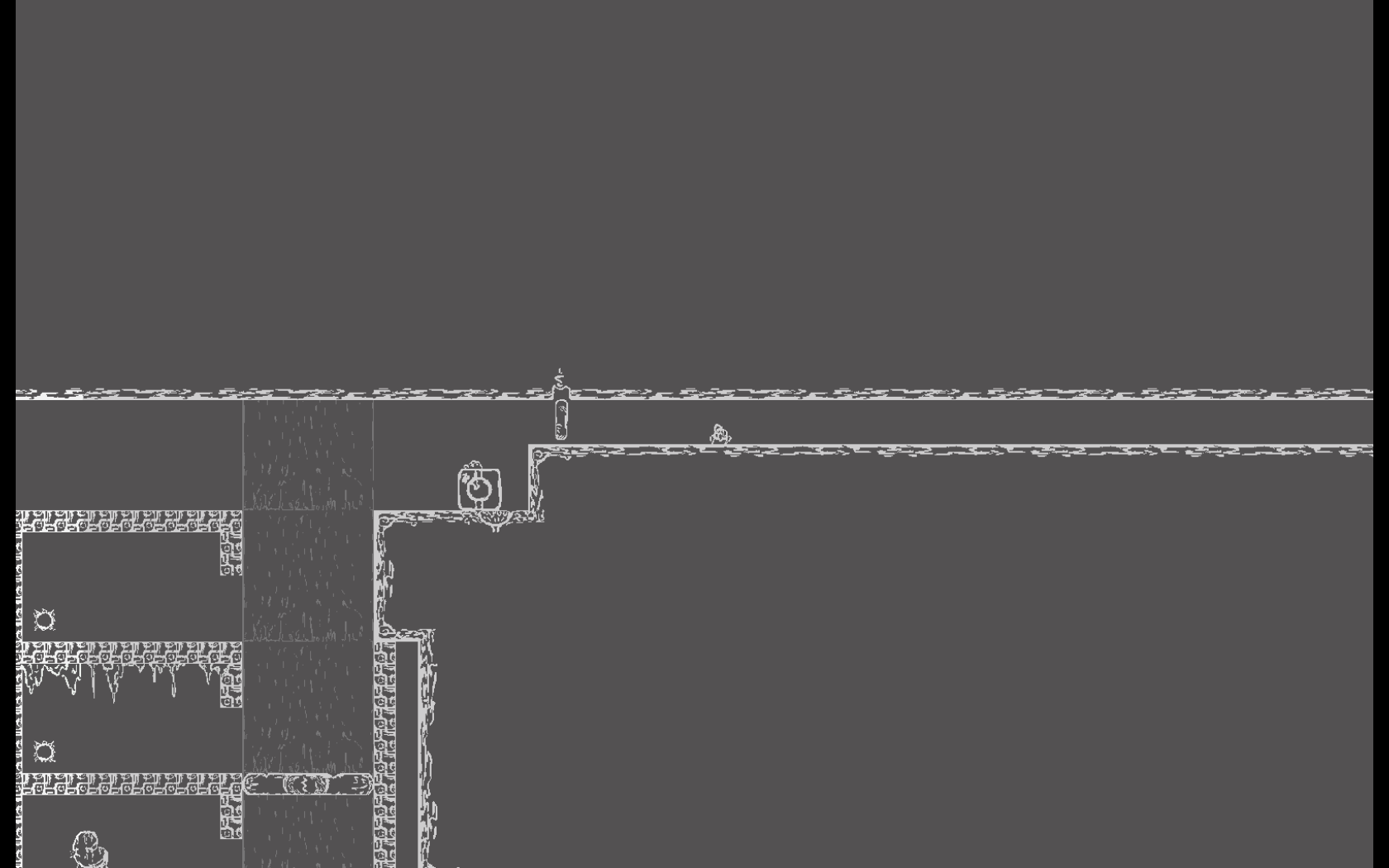


Рис. 20 Концовка игры

* 1. Минимальные системные требования.

Требования программы к работе: операционная система Windows XP и выше (32-разрядная система) или Linux, Процессор Pentium III, 2048 Mb RAM, Видеокарта 250 Mb, 100Mb свободного пространства на жёстком диске.

# Заключение

В данном проекте были сделаны основы физической модели игры, механика игры, игра (а-версия), игровой мир и объекты, уровни, а также сам персонаж.

В качестве развития я выделяю несколько тем: уровни, графику, механику игры.

Использование данного продукта находится исключительно в досуге.

Разработка данной программы принесла немало опыта в сфере разработки игр на Unity.

При разработке программы на Unity использовался C#.

# Список используемой литературы

1. «Искусство создания сценариев в Unity» - Алан Торн
2. «Unity в действии. Мультиплатформенная разработка на C#» - Джозеф Хокинг
3. [https://docs.unity3d.com](https://docs.unity3d.com/)
4. <http://gamesisart.ru/game_dev_create.html#Game_Create_3>

# Приложение 1. Листинг игрового персонажа (Player.cs)

//Игровой персонаж

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Player : MonoBehaviour {

[SerializeField] private float speed; //Скорость бега

[SerializeField] private float jumpForse; //Сила прыжка

[SerializeField] private float jumpTime;

//Находящийся рядом объект

public GameObject neighbor;

public int coinCount;

public bool grounded, rStop, lStop;

public LayerMask groundMask;

public Vector2 save;

private Rigidbody2D rb2D;

private SpriteRenderer sprite;

public Animator animator;

public float Speed

{

set

{

speed = value;

}

get

{

return speed;

}

}

public float JumpForse

{

set

{

jumpForse = value;

}

get

{

return jumpForse;

}

}

//Передвижение по горизонтали

public void Move()

{

rStop = RightStop();

lStop = LeftStop();

if ((Input.GetAxis("Horizontal") > 0 && !rStop)

|| (Input.GetAxis("Horizontal") < 0 && !lStop)

|| (Input.GetAxisRaw("Horizontal") == 0))

{

if (Input.GetAxisRaw("Horizontal") != 0)

{

rb2D.velocity = new Vector2(Speed

\* Input.GetAxis("Horizontal"), rb2D.velocity.y);

animator.SetFloat("Speed", speed);

}

else

{

rb2D.velocity = new Vector2(0, rb2D.velocity.y);

animator.SetFloat("Speed", 0);

}

}

}

public void MoveBox()

{

if (neighbor != null)

{

if (neighbor.gameObject.tag == "Box")

{

int boxCount = 0;

int side = 0;

//Проверка коробки справа

Collider2D[] stopObjs = Physics2D.OverlapCircleAll

(new Vector2((transform.position.x

+ transform.GetComponent<BoxCollider2D>().size.x

/ 2 \* transform.localScale.x), transform.position.y)

, 0.1f, 1024);

if (stopObjs.Length > 0 && Input.GetKey(KeyCode.LeftControl))

{

//Передвигается в два раза медленней если стоит рядом с коробкой и зажат Ctrl

rb2D.velocity = new Vector2(Speed / 2

\* Input.GetAxis("Horizontal"), rb2D.velocity.y);

animator.SetFloat("Speed", speed);

if (Input.GetAxis("Horizontal") > 0)

side = 1;

if (Input.GetAxis("Horizontal") < 0)

side = -1;

animator.SetInteger("PutBox", side);

boxCount++;

}

//Проверка коробки слева

stopObjs = Physics2D.OverlapCircleAll

(new Vector2((transform.position.x

- transform.GetComponent<BoxCollider2D>().size.x

/ 2 \* transform.localScale.x), transform.position.y)

, 0.1f, 1024);

if (stopObjs.Length > 0 && Input.GetKey(KeyCode.LeftControl))

{

rb2D.velocity = new Vector2(Speed / 2

\* Input.GetAxis("Horizontal"), rb2D.velocity.y);

animator.SetFloat("Speed", speed);

if (Input.GetAxis("Horizontal") > 0)

side = -1;

if (Input.GetAxis("Horizontal") < 0)

side = 1;

animator.SetInteger("PutBox", side);

boxCount++;

}

if (Input.GetAxisRaw("Horizontal") == 0)

{

rb2D.velocity = new Vector2(0, rb2D.velocity.y);

animator.SetFloat("Speed", 0);

}

if (boxCount == 0)

animator.SetInteger("PutBox", 0);

}

}

else

animator.SetInteger("PutBox", 0);

}

//Метод для прыжка вверх

public void Jump()

{

grounded = CheckGround();

if (Input.GetButtonDown("Jump") && grounded)

{

rb2D.velocity = new Vector2(rb2D.velocity.x, jumpForse);

jumpTime = 0;

grounded = false;

}

animator.SetBool("Grounded", grounded);

}

//Проверка нахождения на земле

bool CheckGround()

{

//Центральный сенсор

Collider2D[] groundObjs = Physics2D.OverlapCircleAll

(new Vector2(transform.position.x, (transform.position.y

- transform.GetComponent<BoxCollider2D>().size.y

/ 2 \* transform.localScale.y)), 0.01f, groundMask);

if (groundObjs.Length > 0)

{

jumpTime = 0.1f;

return true;

}

//Левый-нижний сенсор

groundObjs = Physics2D.OverlapCircleAll

(new Vector2((transform.position.x

- transform.GetComponent<BoxCollider2D>().size.x

/ 2 \* transform.localScale.x) + 0.08f, (transform.position.y

- transform.GetComponent<BoxCollider2D>().size.y

/ 2 \* transform.localScale.y)), 0.01f, groundMask);

if (groundObjs.Length > 0)

{

jumpTime = 0.1f;

return true;

}

//Правый-нижний сенсор

groundObjs = Physics2D.OverlapCircleAll

(new Vector2((transform.position.x

+ transform.GetComponent<BoxCollider2D>().size.x

/ 2 \* transform.localScale.x) - 0.08f, (transform.position.y

- transform.GetComponent<BoxCollider2D>().size.y

/ 2 \* transform.localScale.y)), 0.01f, groundMask);

if (groundObjs.Length > 0)

{

jumpTime = 0.1f;

return true;

}

if (jumpTime <= 0) return false;

jumpTime -= Time.deltaTime;

return true;

}

//Проверка препятствий справа

public bool RightStop()

{

//Правый сенсор

Collider2D[] stopObjs = Physics2D.OverlapCircleAll

(new Vector2((transform.position.x

+ transform.GetComponent<BoxCollider2D>().size.x / 2

\* transform.localScale.x), transform.position.y)

, 0.01f, groundMask);

if (stopObjs.Length > 0) return true;

return false;

}

//Проверка препятствий слева

public bool LeftStop()

{

//Левый сенсор

Collider2D[] stopObjs = Physics2D.OverlapCircleAll

(new Vector2((transform.position.x

- transform.GetComponent<BoxCollider2D>().size.x / 2

\* transform.localScale.x), transform.position.y)

, 0.01f, groundMask);

if (stopObjs.Length > 0) return true;

return false;

}

//Разворачивает спрайт в сторону движения

public void SprateFlip()

{

if (Input.GetAxis("Horizontal") > 0) sprite.flipX = false;

if (Input.GetAxis("Horizontal") < 0) sprite.flipX = true;

}

void Awake()

{

rb2D = GetComponent<Rigidbody2D>();

sprite = GetComponent<SpriteRenderer>();

animator = GetComponent<Animator> ();

}

private void Start()

{

coinCount = 0;

}

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.tag == "Box")

neighbor = collision.gameObject;

}

private void OnCollisionExit2D(Collision2D collision)

{

if (collision.gameObject.tag == "Box")

{

neighbor = null;

}

}

//Отрисовка сенсоров

private void OnDrawGizmos()

{

//Центральный-нижний

Gizmos.DrawSphere(new Vector3(transform.position.x, (transform.position.y - transform.GetComponent<BoxCollider2D>().size.y / 2 \* transform.localScale.y), 0), 0.01f);

//Левый-нижний

Gizmos.DrawSphere(new Vector3((transform.position.x - transform.GetComponent<BoxCollider2D>().size.x / 2 \* transform.localScale.x) + 0.08f, (transform.position.y - transform.GetComponent<BoxCollider2D>().size.y / 2 \* transform.localScale.y), 0), 0.01f);

//Правый-нижний

Gizmos.DrawSphere(new Vector3((transform.position.x + transform.GetComponent<BoxCollider2D>().size.x / 2 \* transform.localScale.x) - 0.08f, (transform.position.y - transform.GetComponent<BoxCollider2D>().size.y / 2 \* transform.localScale.y), 0), 0.01f);

//Правый

Gizmos.DrawSphere(new Vector3((transform.position.x + transform.GetComponent<BoxCollider2D>().size.x / 2 \* transform.localScale.x), transform.position.y, 0), 0.01f);

//Левый

Gizmos.DrawSphere(new Vector3((transform.position.x - transform.GetComponent<BoxCollider2D>().size.x / 2 \* transform.localScale.x), transform.position.y, 0), 0.01f);

}

}

# Приложение 2. Листинг контроллера камеры

//Главная камера

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraController : MonoBehaviour {

public Transform target;

public float standartSize;

public float learSpeed;

void Start()

{

standartSize = Camera.main.orthographicSize;

transform.position = new Vector3(target.position.x, target.position.y, transform.position.z);

}

void FixedUpdate()

{

transform.position = Vector3.Lerp(transform.position, new Vector3(target.position.x, target.position.y, transform.position.z), learSpeed \* Time.deltaTime);

}

}

# Приложение 3. Листинг главного меню

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour

{

public void StartPressed()

{

SceneManager.LoadScene("Level\_1");

}

public void ExitPressed()

{

Debug.Log("EXIT");

Application.Quit();

}

}