МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
**Высшая школа электроники и компьютерных наук**
**Кафедра системного программирования**

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

# Разработка компьютерной игры в жанре «Top-Down Puzzle» для платформы Windows

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-327.ВКР

Научный руководитель,
доцент кафедры СП, к.т.н.
_____ Н.Ю. Долганина

Автор работы,
студент группы КЭ-402
_____ М.Д. Панфиленко

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет**
**(национальный исследовательский университет)»**
**Высшая школа электроники и компьютерных наук**
**Кафедра системного программирования**

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

**ЗАДАНИЕ**
**на выполнение выпускной квалификационной работы бакалавра**
студенту группы КЭ-402
Панфиленко Максиму Дмитриевичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка компьютерной игры в жанре «Top-Down Puzzle» для платформы Windows.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Godot docks. [Электронный ресурс] URL: https://devdocs.io/godot (дата обращения: 05.02.2024 г.).

3.2. Godot Engine 4.0 documentation. [Электронный ресурс] URL: https://docs.godotengine.org/en/stable (дата обращения: 05.02.2024 г.).

3.3. Itch.io – Библиотека игровых ассетов. [Электронный ресурс] URL: https://itch.io/game-assets/free/tag-tileset (дата обращения: 07.04.2024 г.)

3.4. Роллингз Э., Моррис Д. Проектирование и архитектура игр. // Вильямс, 2006. – 1035 с.

4. **Перечень подлежащих разработке вопросов**

4.1. Проанализировать предметную область.

4.2. Спроектировать архитектуру игрового приложения.

4.3. Реализовать игровое приложение.

4.4. Протестировать игровое приложение.

**5. Дата выдачи задания:** 29.01.2024 г.

**Научный руководитель,**
доцент кафедры СП, к.т.н. Н.Ю. Долганина

**Задание принял к исполнению** М.Д. Панфиленко

**ОГЛАВЛЕНИЕ**

# ВВЕДЕНИЕ

## Актуальность

На сегодняшний момент продукты игровой индустрии представляют самый комплексный вид медиа произведений, аккумулирующий в себе музыку, изобразительное искусство, анимацию, работу дизайнеров, сценаристов, программистов, математические модели и еще множество направлений человеческого труда, которые разнятся от проекта к проекту учитывая обилие жанров и амбиций разработчиков. Хоть данный сплав искусств сам по себе вызывает трепет, такой сложный и многомерный вид деятельности просто бы не прижился, если бы не приносил хорошую прибыль. И с финансовыми показателя этого сегмента индустрии развлечений все прекрасно. Доходы игровой индустрии продолжают стабильно расти практически каждый год как численно, так и в доли от общего рынка. График доходности [1] индустрий развлечений на 2020 г. приведен на рисунке 1.

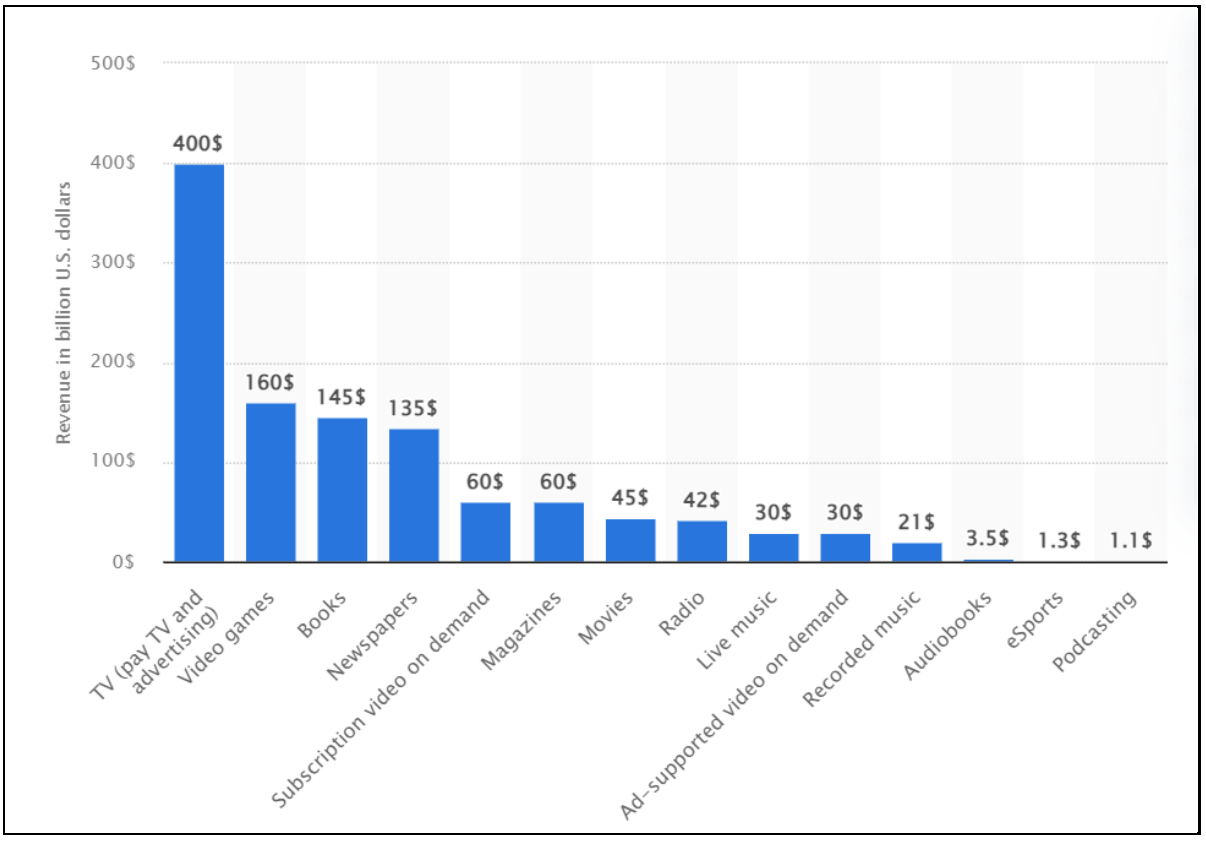


Рисунок 1 – График доходности индустрий развлечений на 2020 год

**Постановка задачи**

Целью выпускной квалификационной работы является разработка компьютерной игры в жанре «Top-Down Puzzle» для платформы Windows. Для достижения поставленной цели необходимо решить следующие задачи:

1) проанализировать предметную область;
2) спроектировать архитектуру игрового приложения;
3) реализовать игровое приложение;
4) протестировать игровое приложение.

**Структура и содержание работы**

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 40 страниц, объем списка литературы – 15 источников.

В первой главе описывается предметная область проекта, а также рассмотрены схожие проекты.

Вторая глава посвящена проектированию системы. В ней приведено более детально описание ключевой игровой механики, определены варианты использования системы и составлены макеты интерфейса.

В третьей главе описана реализация разных аспектов игры и средств разработки.

Четвертая глава описывает результаты тестирования игрового приложения.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1. Описание предметной области

Top-down игра – это вид видеоигр, в котором камера расположена сверху, обычно направлена вниз на игровое поле или сцену [2]. В таких играх игрок обычно управляет персонажем или объектом с точки зрения, находящейся выше их, что позволяет видеть большую часть игрового мира сразу.

Тем не менее, top-down-игры – довольно обширная категория и могут кардинально отличаться по игровому процессу. Одни могут ставить во главу угла боевую систему и динамичные сражения с противниками, а другие фокусироваться на размеренных головоломках и пазлах, которые игроку нужно решить для дальнейшего продвижения. Хотя ничего не мешает разработчику совместить оба решения в одном проекте.

В разрабатываемом игровом приложении будет сделан акцент на решении головоломок в двумерном пространстве.

## 1.2. Анализ аналогичных проектов

**«Sokoban»**

В качестве примера классической игры головоломки с видом сверху можно привести Sokoban. Sokoban (сокобан) [2] – это классическая японская головоломка, в которой игрок управляет персонажем, чья цель состоит в том, чтобы переместить ящики по лабиринту таким образом, чтобы каждый ящик занял свое предназначенное место, обычно обозначенное на полу. Цель игры – перемещаясь по лабиринту, толкать ящики так, чтобы они не заблокировали другие ящики или пути, и чтобы все ящики оказались на своих местах.

Игра состоит из различных уровней, каждый из которых представляет собой лабиринт с ящиками и точками назначения для ящиков. Уровни могут быть разной сложности, и решение некоторых из них требует серьезного стратегического мышления и планирования. Изображение игрового процесса «Sokoban» представлено на рисунке 2.

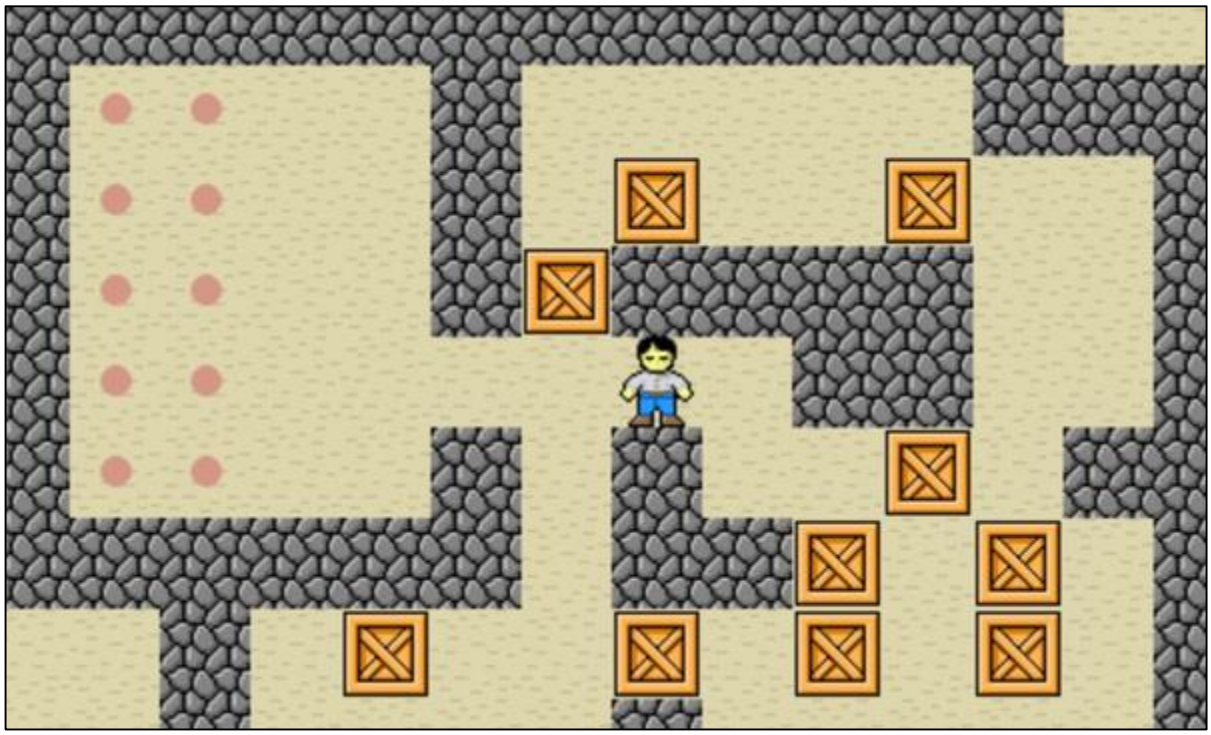

Рисунок 2 – Игровой экран «Sokoban»

**«Огонь и Вода»**

В качестве проекта со схожей игровой механикой можно привести головоломку для двоих игроков «Огонь и Вода». Известная многим браузерная игра для двоих «Огонь и Вода» является одним из лучших примеров [3]. Игроки управляют двумя персонажами – Огнем и Водой, которые обладают уникальными способностями и ограничениями, вынуждающими их работать вместе для преодоления различных препятствий и достижения общей цели. Из-за местных игровых условностей часть игровой локации недоступна для одного из подконтрольных персонажей, а другая для второго (например, персонаж «Огонь» не имеет возможности переплыть затопленное углубление в полу). Несмотря на то, что данная игра относится к жанру платформеров, ее основной концепт хорошо перекликается с разрабатываемым игровым приложением. Изображение игрового процесса «Огонь и Вода» представлено на рисунке 3.

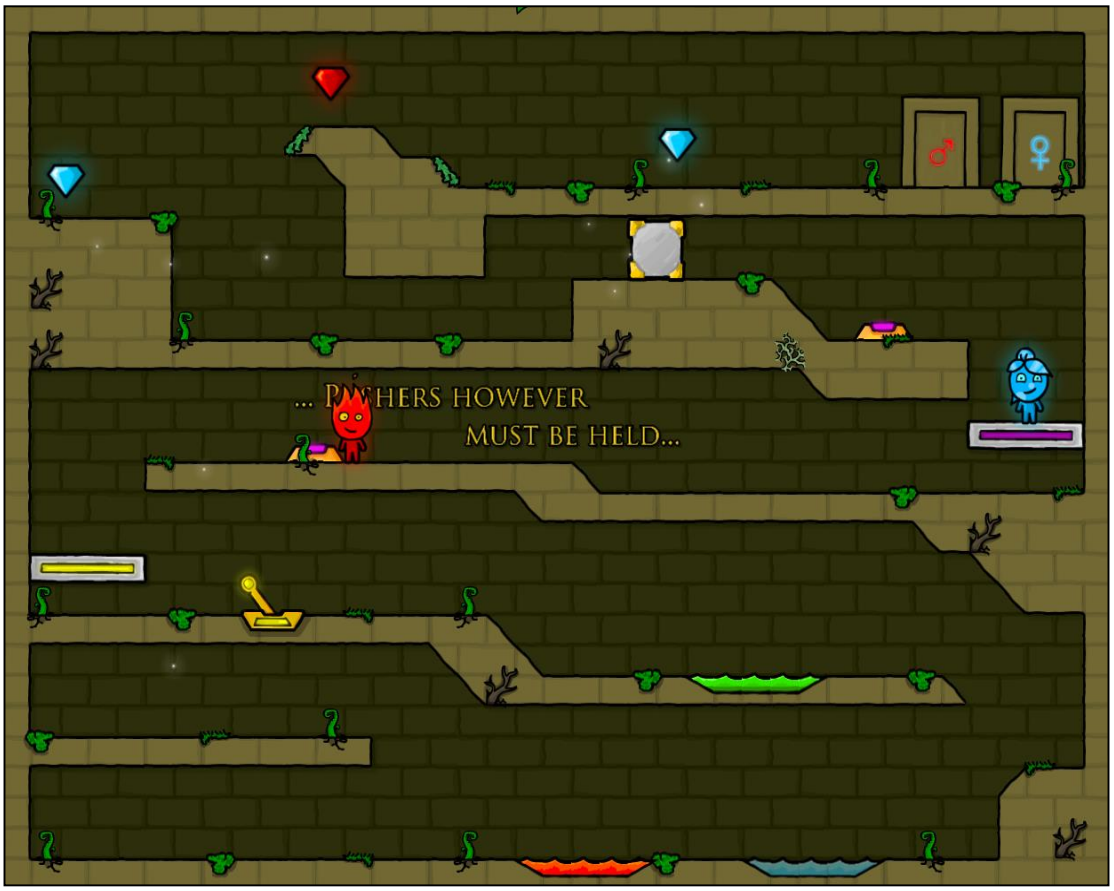


Рисунок 3 – Игровой экран «Огонь и Вода»

В соответствии с целью работы в первой главе был проведен обзор аналогов реализуемого приложения. Игровые механики представленных приложений послужат хорошей базой для разрабатываемого игрового приложения.

# 2. ПРОЕКТИРОВАНИЕ

## 2.1. Концепция игры

Основной игровой механикой является система клонирования персонажа. Каждый игровой уровень будет представлять собой локацию с препятствиями, которые будет невозможно преодолеть в одиночку, но игрок имеет возможность «записать» действия персонажа, создать его клона, и запустить новый цикл, с началом которого уровень обнуляется, но игрок начинает прохождение комнаты вместе со своим двойником, выполняющим действия из предыдущего цикла. Для прохождения комнаты может понадобится создать несколько клонов, количество которых ограничено, и грамотно выверять действия каждого клона во времени относительно других.

## 2.2. Макеты пользовательского интерфейса

Были созданы макеты пользовательского интерфейса [4]. Они представляют собой примерное представление главного меню и внутриигрового интерфейса.

При запуске игры игрок видит главное меню, содержащее название игры и 3 кнопки: «Выбрать уровень», «Помощь» и «Выход». Макет главного меню представлен на рисунке 4.

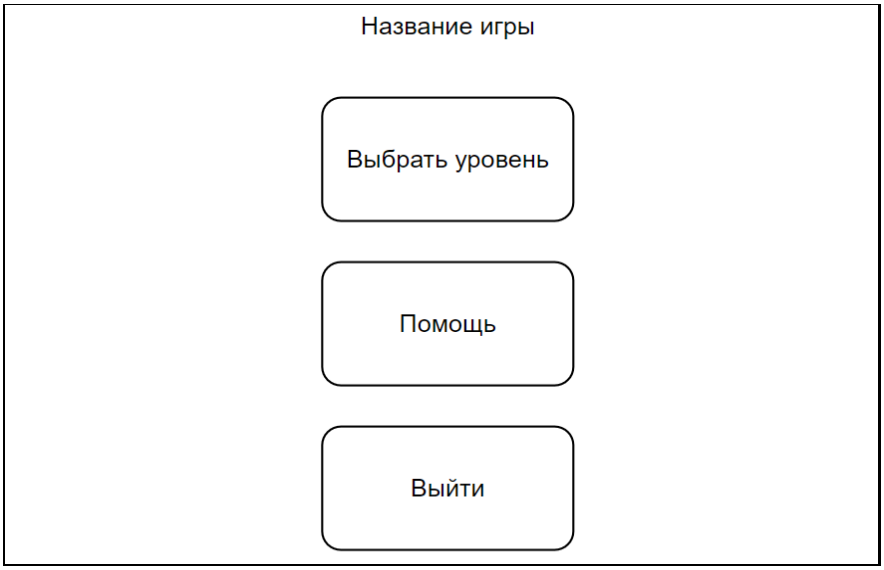


Рисунок 4 – Макет главного меню

При переходе на экран выбора уровней игрок видит меню, содержащее кнопку возврата на главный экран «В главное меню» и сетку из кнопок для перехода на интересующую игрока локацию. Макет экрана выбора уровней представлен на рисунке 5.

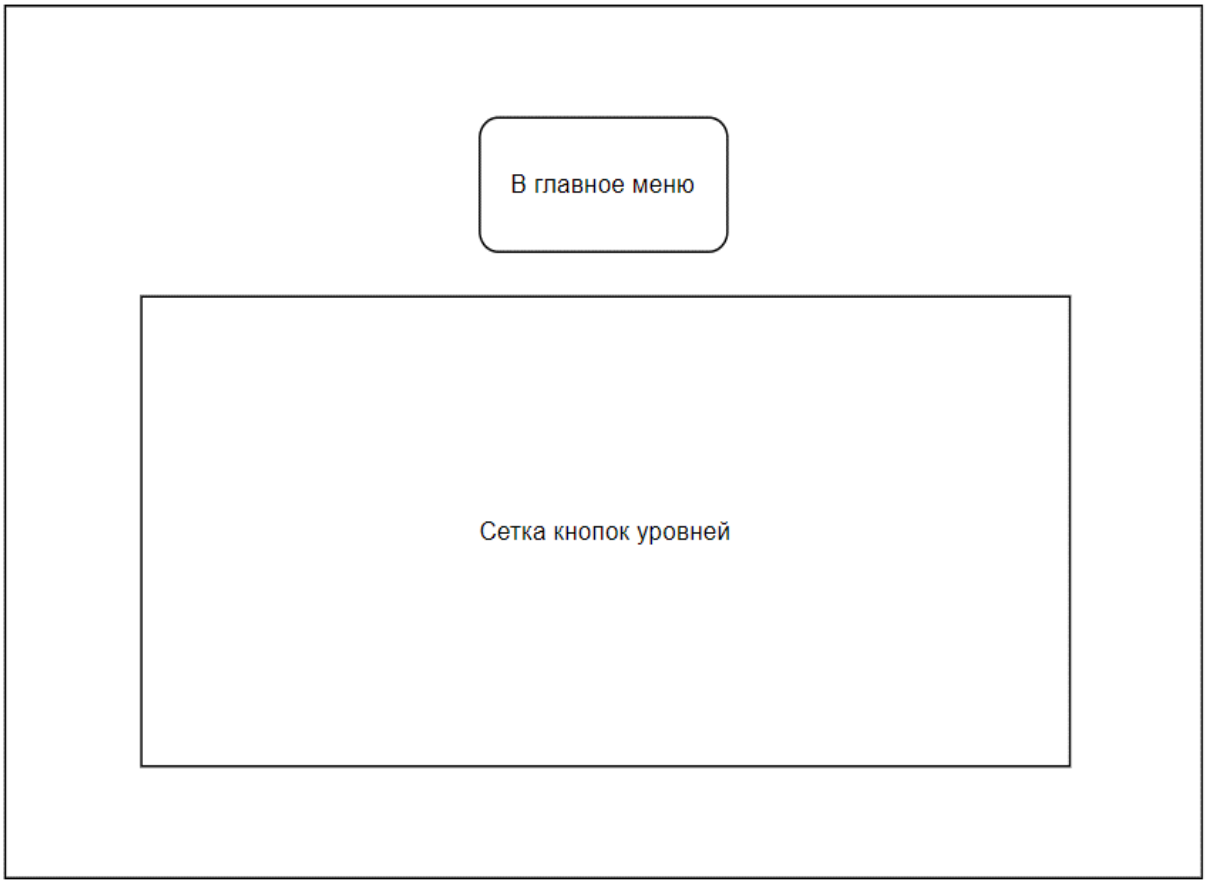


Рисунок 5 – Макет экрана выбора уровней

Во время прохождения игровых уровней пользователь должен иметь возможность видеть таймер и количество доступных клонов текущей игровой локации. Это основная информация, которая нужна игроку. Остальное пространство будет занято отображением игровой локации, ее объектов, подконтрольного персонажа и его клонов. Макет экрана уровня представлен на рисунке 6.

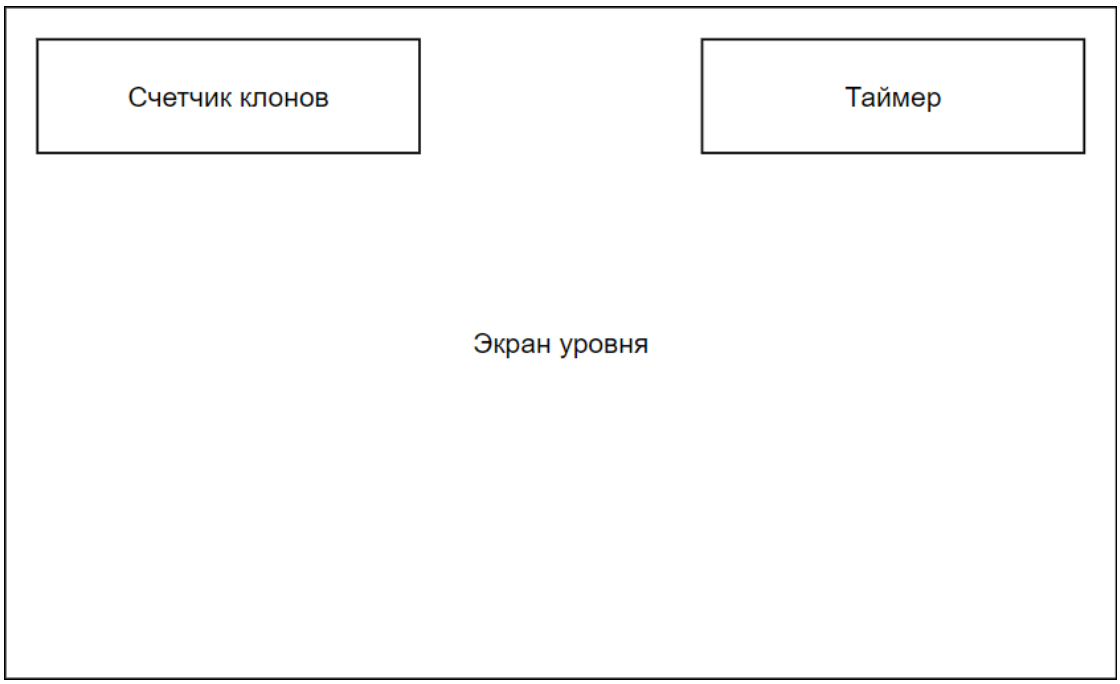


Рисунок 6 – Макет экрана уровня

### 2.3. Функциональные и нефункциональные требования

Функциональные требования [5] – это требования, которые определяют, какие функции или возможности должна предоставлять система, чтобы соответствовать потребностям пользователей, без учета ограничений, связанных с ее реализацией.

Функциональные требования к проектируемому игровому приложению следующие:

1) возможность осуществлять перемещение по горизонтальной и вертикальной оси с учетом столкновений с препятствиями;

2) возможность осуществлять толчок избранных объектов и от объектов;

3) должна быть реализована система циклов с возможностью создания игроком клонов персонажа, повторяющих действия оного в пределах одной игровой локации;

4) должна быть реализована возможность выбора игровых уровней;

5) должна быть реализована функция сброса игровой локации и системы циклов.

Нефункциональные требования – это требования, которые определяют качество, производительность, безопасность и другие свойства системы, которые не относятся к ее функциональности.

Нефункциональные требования к проектируемому игровому приложению следующие:

1) система должна работать на операционной системе Windows 10;

2) система должна быть разработана с использованием Godot Engine [6];

3) система должна быть реализована на скриптовом языке GDScript [7].

## 2.4. Диаграмма вариантов использования

Была разработана диаграмма вариантов использования [8], в которой представлены игровые возможности. Разработанная диаграмма изображена на рисунке 7.

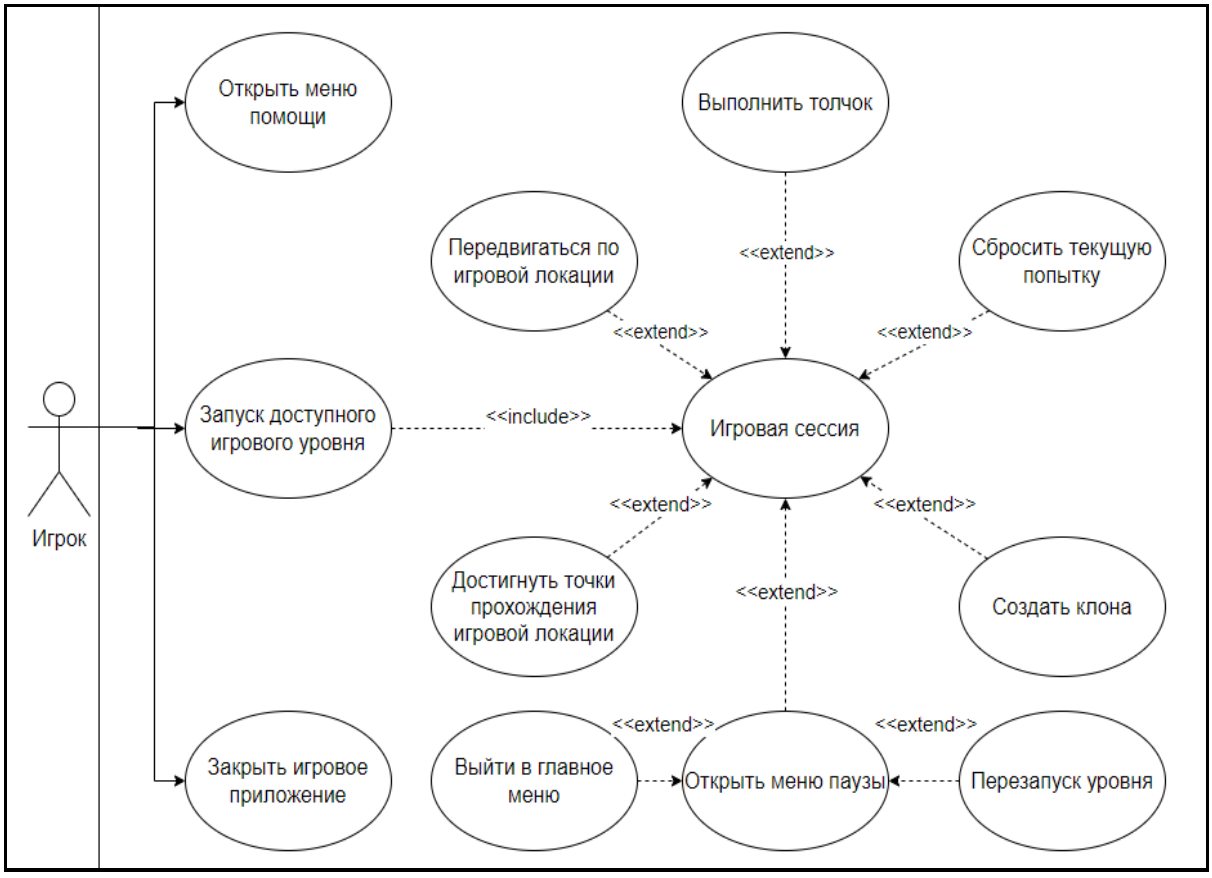


Рисунок 7 – Диаграмма вариантов использования

Описание вариантов использования представлено ниже.

1. Запуск доступного игрового уровня – игрок выбирает доступный уровень из перечня и запускает его.

2. Игровая сессия – игрок находится в процессе управления персонажем и прохождения игрового уровня.

3. Передвигаться по локации – игрок может контролировать поведение персонажа путем перемещения его позиции по горизонтали, осуществлением прыжка и перехода в состояние приседа.

4. Создать клона – игрок начинает прохождение локации со стартовой точки вместе с клоном/клонами из предыдущего цикла.

5. Сбросить текущую попытку – игрок возвращает локацию к исходной точке и перемещает персонажа и его клона/клонов на стартовую позицию.

6. Перезапуск уровня – игрок возвращает локацию к изначальному состоянию и удаляет всех клонов и сбрасывает их счетчик.

7. Достигнуть точки прохождения игровой локации – игрок выполняет условие прохождение игрового уровня.

8. Открыть меню паузы – игрок приостанавливает игру и открывает меню.

9. Выйти в главное меню – игрок прекращает игровую сессию и возвращается в главное меню.

10. Открыть меню помощи – игрок переходит в справочное окно, содержащее информацию об управлении игровым персонажем.

11. Закрыть игровое приложение – игрок закрывает игровое приложение.

12. Выполнить толчок – игрок совершает толчок клона или толчок от стены.

## 2.5. Диаграмма компонентов системы

На рисунке 8 представлена диаграмма компонентов [9] игрового приложения, которая показывает разбиение системы на структурные компоненты и связи между ними.

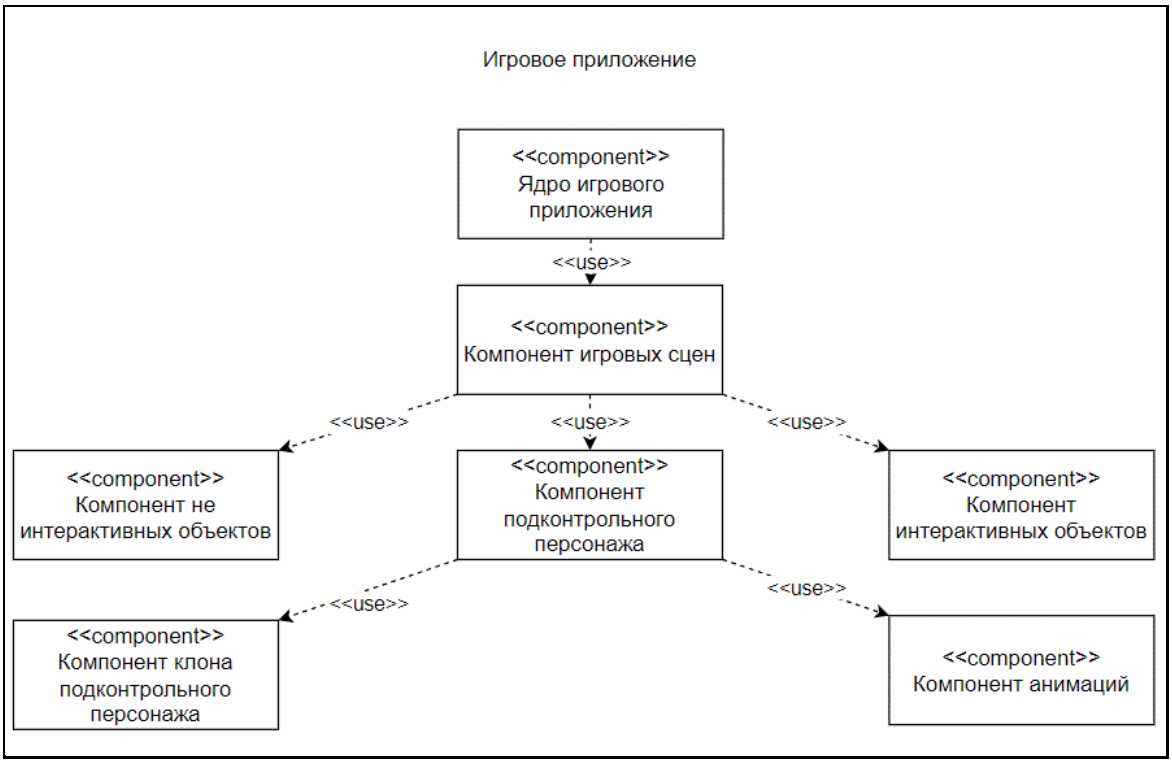


Рисунок 8 – Диаграмма компонентов системы

Описание компонентов, показанных на рисунке 8, представлено ниже.

1. Компонент ядра игрового приложения – реализует базовую функциональность Godot и отвечает за переключения игровых сцен и связей между ними.

2. Компонент игровых сцен – содержит набор классов, реализующих главные экраны меню, меню паузы, основной игровой сценой.

3. Компонент подконтрольного персонажа – содержит набор классов, ответственных за реализацию механизмов управления, отображения и взаимодействия с окружением подконтрольного персонажа.

4. Компонент интерактивных объектов – содержит набор классов, реализующих сущности, с которыми может взаимодействовать подконтрольный персонаж или его клон.

5. Компонент не интерактивных объектов – содержит набор объектов, с которыми подконтрольный персонаж не может взаимодействовать.

6. Компонент клона подконтрольного персонажа – содержит набор классов и методов, ответственных за реализацию механизмов автономного передвижения клонов, отображения и взаимодействия с окружением игровой сцены.

7. Компонент анимаций – стандартный узел Godot, содержащий список предварительно настроенных анимаций.

# 3. РЕАЛИЗАЦИЯ

## 3.1. Программные средства реализации

В качестве игрового движка для создания двумерного платформера хорошим вариантом станет Godot 4. Несмотря на то, что в сравнении с конкурентами это еще достаточно молодой инструмент, он уже обрел хорошую базу активных пользователей, готовых помочь советом, и стремительно приобретает новые возможности. Его достоинства перечислены ниже.

1. Бесплатность и открытый исходный код.
2. Поддержка множества платформ, таких как Windows, macOS, Linux, Android, iOS, Web.
3. Наличие множества инструментов. Godot имеет множество инструментов, которые облегчают создание игр, включая редактор уровней, редактор частиц, редактор шейдеров и многие другие.

Каждый игровой движок основывается на абстракциях, которые используют для сборки приложений. В Godot игра – это дерево узлов, сгруппированных в сцены, которые могут обмениваться информацией с помощью сигналов.

Есть четыре основные концепции, которые будут задействованы при создании игрового приложения на Godot – это узлы, сцены, деревья сцен и сигналы [10].

## 3.2. Краткий обзор концепциий

**Сцена**

В Godot игровое приложение разбивается на многократно используемые сцены. Сцена может представлять собой совершенно различные сущности, к примеру игрового персонажа, объект окружения или элемент интерфейса. На рисунке 9 представлена сцена двери игрового приложения.

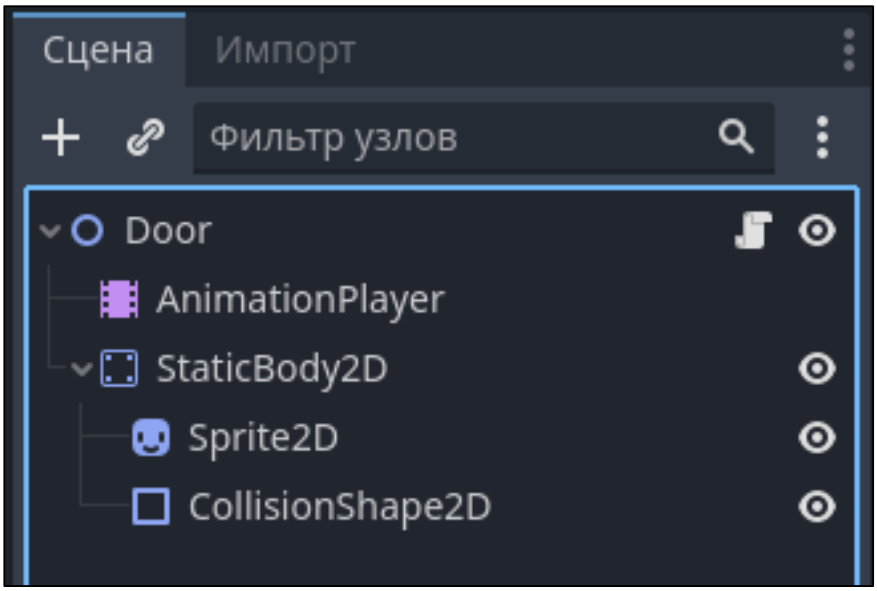


Рисунок 9 – Сцена двери игрового приложения

**Узлы**

Сцена состоит их одного или большего числа узлов. Узлы – самые маленькие элементы игрового приложения, которые выстраиваются в деревья. На рисунке 10 представлен узел `StaticBody2D` из сцены `Door`.

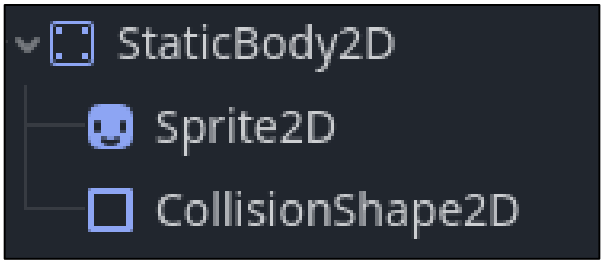


Рисунок 10 – Узел `StaticBody2D`

Godot предоставляет обширную библиотеку базовых типов узлов, которые можно совмещать и расширять для создания более функциональных.

**Дерево сцены**

Все сцены игрового приложения собираются вместе в дереве сцены. Так как сцены – деревья узлов, дерево сцены также является деревом узлов, но проще воспринимать приложение с точки зрения сцен, поскольку они могут представлять персонажей, оружие, двери или пользовательский интерфейс. На рисунке 11 представлено дерево сцены `Level2`.

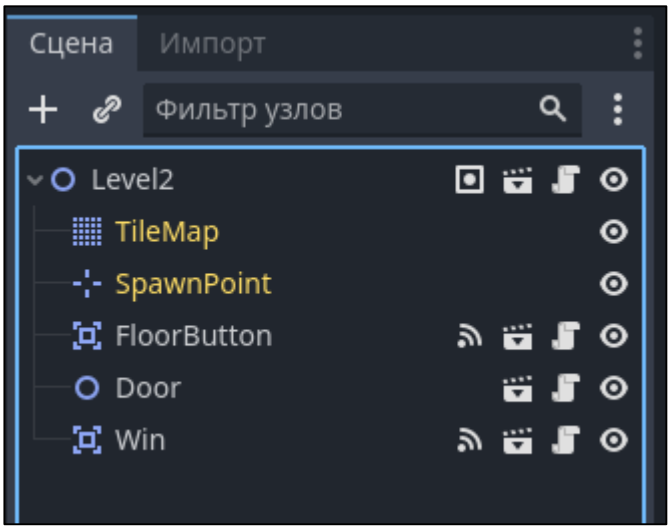


Рисунок 11 – Дерево сцены `Level2`

**Сигналы**

Узлы транслируют сигнал, когда происходит какое-то событие. Эта возможность позволяет задать взаимодействие узлов без жесткого указания их в коде, что позволяет получить большую гибкость в структурировании сцен. На рисунке 12 представлены сигналы узла `Control`.

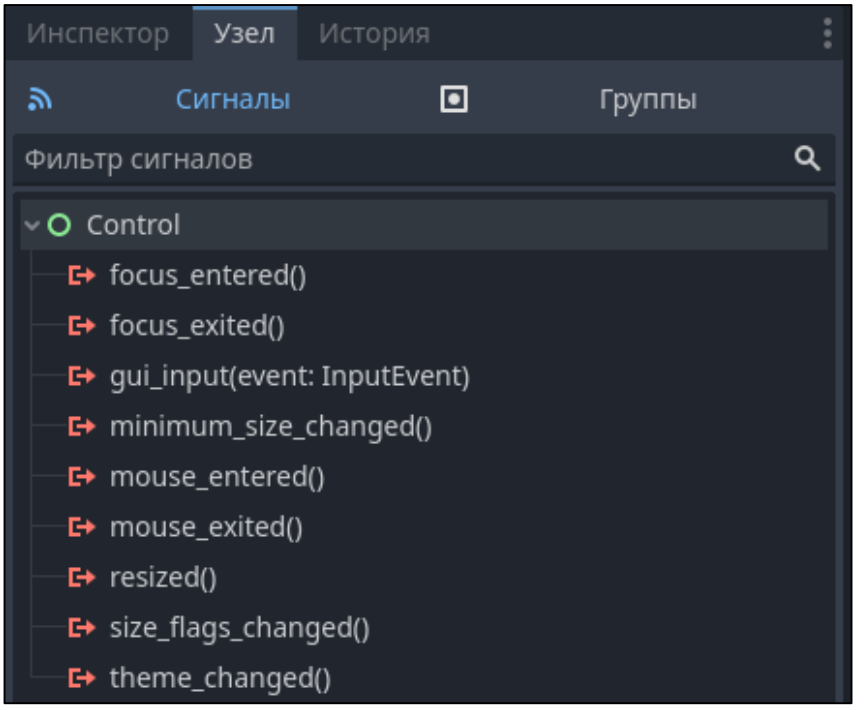


Рисунок 12 – Сигналы узла `Control`

### 3.3. Реализация интерфейса

Главные экраны игрового приложения состоят из трех сцен – главного меню, меню выбора уровней и экрана помощи, содержащего справочную информацию об управлении игровым персонажем.

Главное меню содержит название игрового приложения и три кнопки – «Выбрать уровень», «Помощь» и «Выход». Изображение итогового вида главного меню представлено на рисунке 13.

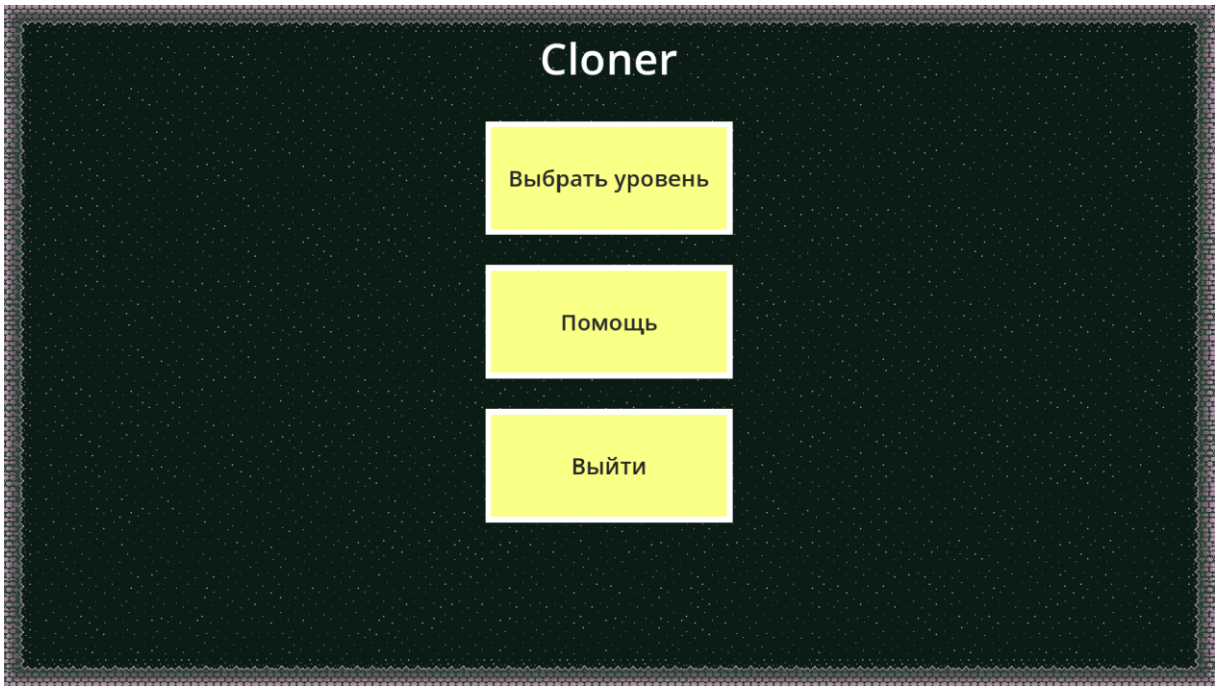


Рисунок 13 – Главное меню

К каждой из кнопок подключен скрипт `SceneButton` с функцией `_on_pressed`, которая меняет текущую игровую сцену при получении сигнала нажатия. Путь к сцене указывается в инспекторе свойств узла кнопки и передается в экспортируемую переменную `scene_path`. Реализация скрипта `SceneButton` продемонстрирована в листинге 1.

Листинг 1 – Реализация скрипта `SceneButton`

```
@export_file var scene_path
func _on_pressed() -> void:
	if scene_path == null:
		return
	get_tree().change_scene_to_file(scene_path)
```

Меню выбора уровней содержит кнопку «В главный экран» и сетку кнопок выбора уровней. Изображение итогового вида главного меню представлено на рисунке 14.


Рисунок 14 – Меню выбора уровней

К сцене подключен скрипт `LevelSelectScreen`, который проходится по списку сцен игровых уровней из указанной в экспортируемой переменной папки и создает именованные кнопки с путями к этим сценам. Сами кнопки автоматически размещаются в сетке `GridContainer`. Реализация скрипта `LevelSelectScreen` продемонстрирована в листинге 2.

Листинг 2 – Реализация скрипта `LevelSelectScreen`

```
extends Control
const LEVEL_BTN = preload("res://ui/buttons/LevelButton.tscn")
@export_dir var dir_path

@onready var grid = $MarginContainer/VBoxContainer/GridContainer

func _ready() -> void:
	get_levels(dir_path)

func get_levels(path):
	var dir =  DirAccess.open(path)

	if dir:
		dir.list_dir_begin()
		var file_name = dir.get_next()
```

```
            while file_name != "":
                  create_level_btn('%s/%s' % [dir.get_current_dir(),
file_name], file_name)
                  file_name = dir.get_next()

            dir.list_dir_end()
      else:
            print("An error occurred when trying to access the path.")

func create_level_btn(lvl_path, lvl_name):
      var btn = LEVEL_BTN.instantiate()
      btn.text = lvl_name.trim_suffix('.tscn').replace("_", " ")
      btn.level_path = lvl_path
      grid.add_child(btn)
```

Меню помощи содержит справочную информацию об управлении во время игрового процесса и кнопку «В главное меню». Изображение итогового вида меню помощи представлено на рисунке 15.

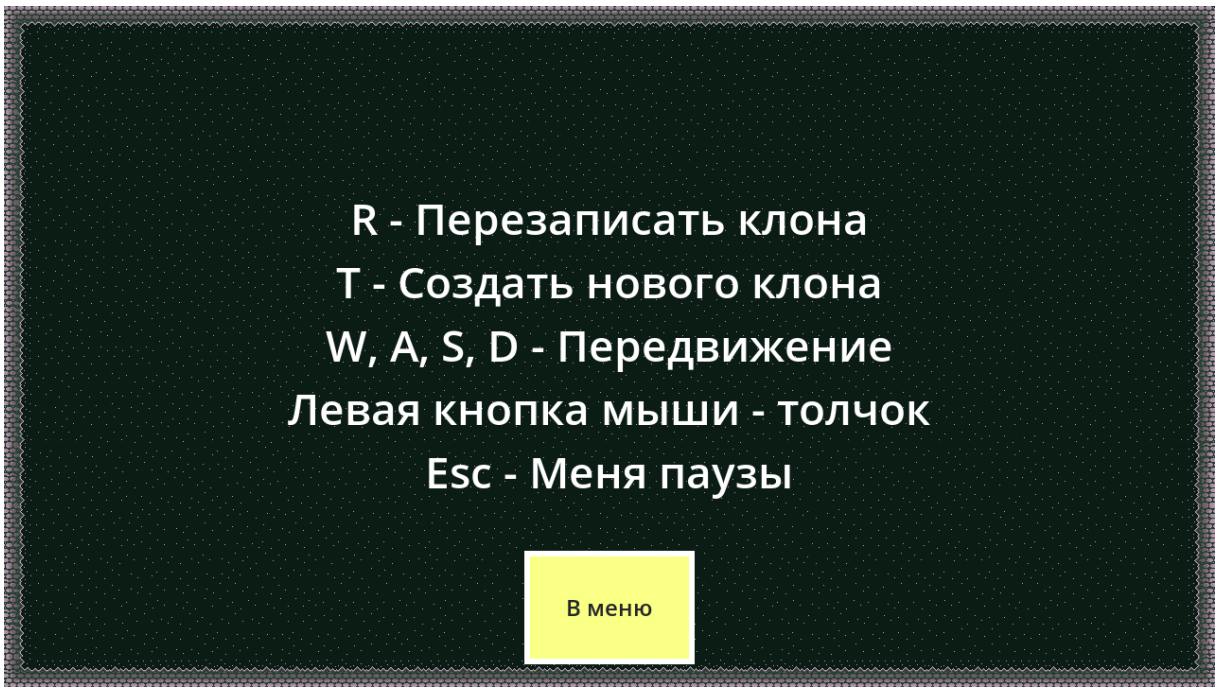


Рисунок 15 – Меню помощи

За отображение количества доступных клонов во время игрового процесса отвечает узел `CloneCountLabel`, в который передается значение переменной `clone_count`, изменяющаяся в скрипте `Game`. Через относительный путь `$CanvasLayer/ThemeWrapper/UI/CloneCountLabel` мы можем получить доступ к `CloneCountLabel`, который является дочерним узлом `Game`. Функция `_process` срабатывает каждый игровой тик и при нажатии

клавиши создания клона уменьшает счетчик доступных на единицу. Базовое значение доступных клонов берется из переменной подключенного к узлу игрового уровня. Каждый игровой уровень имеет собственное значение допустимого количества клонов. Реализация методов скрипта `Game`, задействованных для работы счетчика продемонстрирована в листинге 3.

Листинг 3 – Методы скрипта `Game`, задействованные для реализации счетчика

```
@onready var clone_count_label: Label = $CanvasLayer/ThemeWrap-
per/UI/CloneCountLabel

var init_clone_count: int = 1337
var clone_count: int = 1337:
	set(value):
		clone_count_label.text = "Clones: " + str(value)
		clone_count = value
	get:
		return clone_count

func _ready():
	randomize()
	var level_path = level_manager.level_path
	var level_scene = ResourceLoader.load(level_path)
	var level = level_scene.instantiate()
	add_child(level)

	spawn_point = level_manager.get_spawn_point()
	clone_count = level.clone_count
	init_clone_count = clone_count

	set_game()
	clone_manager.overwrite_clone()

func _process(_delta):
	if Input.is_action_just_pressed("next_clone"):
		# if clone count not bigger max clones
		if clone_count > 0:
			clone_manager.new_clone()
			set_game()
			clone_count -= 1
		else:
			pass

func _on_restart_pressed():
	clone_manager.hard_reset_clones()
	clone_count = init_clone_count
	set_game()
	pause_menu_toggle()
```

Изображение итогового вида счетчика клонов представлено на рисунке 16.

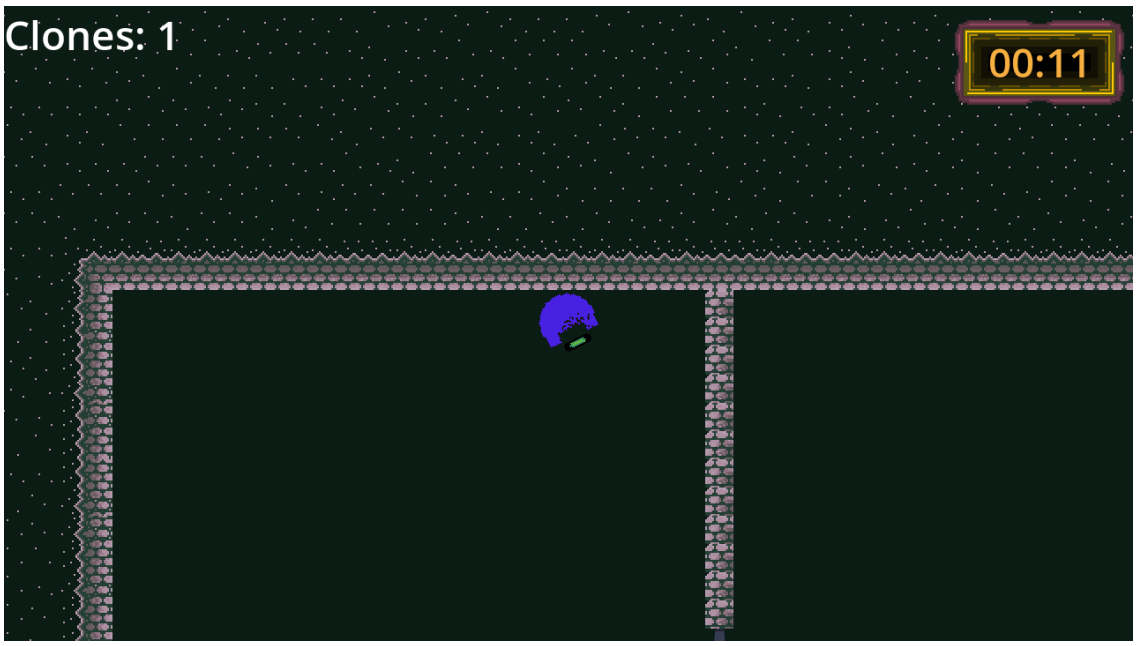


Рисунок 16 – Счетчик клонов

### 3.4. Реализация подконтрольного персонажа

Передвижение персонажа осуществляется через встроенную функцию обработчик физики `_physics_process` и ее свойство `velocity`, представляющее собой двумерный вектор из класса `CharacterBody2D`. Реализация скрипта `Player` продемонстрирована в листинге 4.

Листинг 4 – Реализация скрипта `Player`

```
func _physics_process(delta):
	lookAtFunc()
	if not push_cooldown:
		if get_mouse_left_click_pressed():
			uppdatePushState(pushPlatfState.READY)
		elif get_mouse_left_click_released():
			push_body()
	if not is_pushing:
		directionX = get_move_directionX()
		directionY = get_move_directionY()
		if not is_in_abyss:
			if directionX:
				velocity.x = directionX * speed
			else:
				velocity.x = move_toward(velocity.x, 0, speed)
			if directionY:
				velocity.y = directionY * speed
			else:
				velocity.y = move_toward(velocity.y, 0, speed)
		elif not is_falling:
			velocity.x = 0
			velocity.y = 0
			is_falling = true
			falling_timer.start()
```

```
                cshape.disabled = true
                pushCShape.disabled = true
        move_and_slide()
        apply_animations()

func get_mouse_left_click_released():
        return Input.is_action_just_released("mouse_left_click")
func get_mouse_left_click_pressed():
        return Input.is_action_just_pressed("mouse_left_click")
func get_move_directionX():
        return Input.get_axis("left", "right")
func get_move_directionY():
        return Input.get_axis("up", "down")
```

Анимации подконтрольного персонажа и его клонов были реализованы с помощью стандартного узла `AnimationPlayer`. Узел позволяет создавать набор именованных анимаций, у каждой из которой можно гибко настроить время наложения кадров между переходами и в последствии обращаться к каждой через скрипт. Часть ассетов была взята с бесплатных сторонних ресурсов [15]. Рисунок 17 содержит окно настройки анимаций.

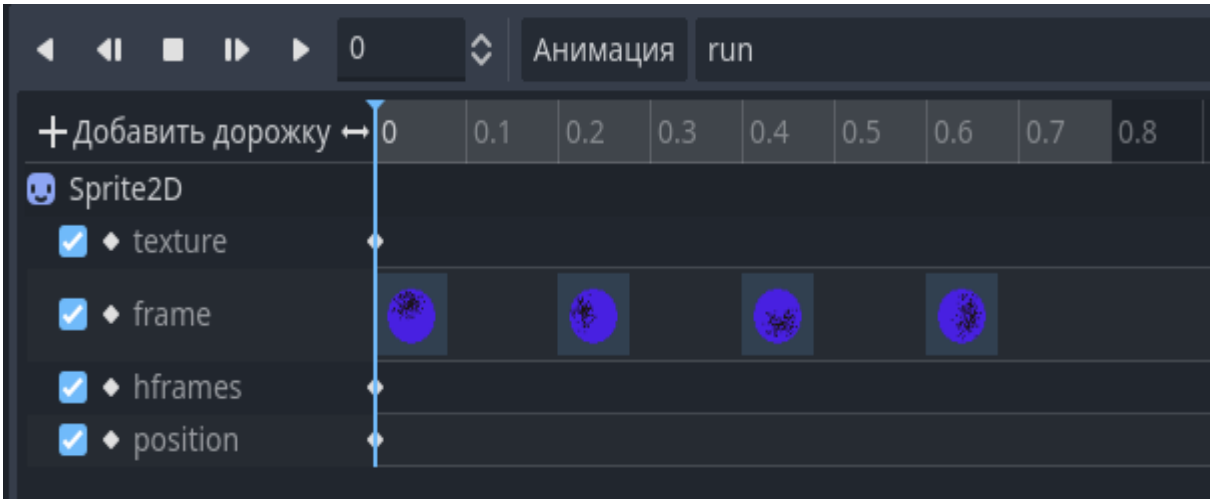


Рисунок 17 – Окно настройки анимаций

Подконтрольный персонаж имеет возможность толкать собственных клонов и отталкиваться от препятствий. Достигается это путем проверки типа объекта коллайдера препятствия и области коллайдера, закрепленной за персонажем. Визуализация коллайдеров представлена на рисунке 18. Реализация функции толчка представлена в листинге 5.

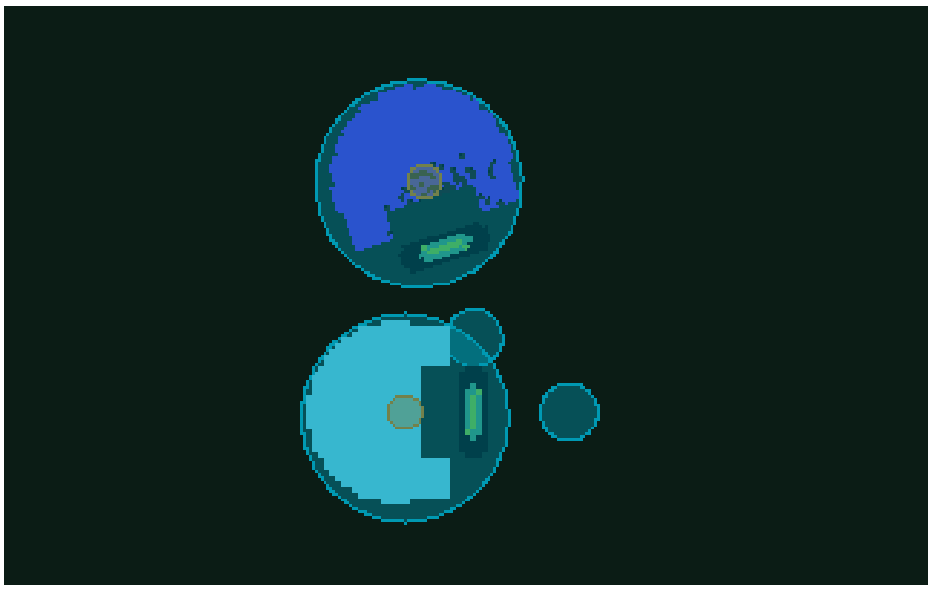

Рисунок 18 – Визуализация коллайдеров

Листинг 5 – Реализация функции толчка

```
func push_body():
	uppdatePushState(pushPlatfState.PUSH)
	push_cooldown_timer.start()
	push_cooldown = true

	for body in push_area.get_overlapping_bodies():
		var direction_to_body = (body.global_position - global_posi-
tion).normalized()

		if body is CharacterBody2D:
			body.velocity = direction_to_body * push_force * 2
			body.is_pushing = true
			body.push_timer.start()

		elif body is TileMap:
			direction_to_body = Vector2(cos(rotation), sin(rotation))
			velocity = direction_to_body * push_force * -2
			is_pushing = true
			push_timer.start()
```

### 3.5. Реализация клонов персонажа

Во время игровой сессии приложение производит запись значения ввода с клавиатуры, продолжительности нажатия и типа ввода (нажатие/отжатие) клавиши для дальнейшего использования этой последовательности клоном. Для хранения ввода клавиши в приложении используется класс `CustomInput`. Реализация класса `CustomInput` продемонстрирована в листинге 6.

Листинг 6 – Реализация класса `CustomInput`

```
enum InputType {
	NONE,
	LEFT,
	RIGHT,
	UP,
	DOWN,
	INTERACT,
	MOUSE_CLICK
}
enum InputValue {
	NONE,
	PRESSED,
	RELEASE,
	CLICK
}
var type: InputType = InputType.NONE
var value: InputValue = InputValue.NONE
var timestamp: float = 0.0
var mouse_click_position: Vector2 = Vector2.ZERO

func set_mouse_click_position(position: Vector2):
	mouse_click_position = position
func get_mouse_click_position() -> Vector2:
	return mouse_click_position
func is_pressed():
	return value == InputValue.PRESSED
func is_released():
	return value == InputValue.RELEASE
```

За считывание, сохранение и управление массивами ввода отвечает класс `CloneManager`. Реализация класса `CloneManager` представлена в листинге 7.

Листинг 7 – Реализация класса `CloneManager`

```
extends Node2D
class_name CloneManager
#@export var max_clone_count: int = 2
const CustomInput = preload("res://shared/CustomInput.gd")
#var clone_count: int = max_clone_count
var saved_histories: Array[Array] = []
var current_history: Array[CustomInput] = []
var record_start_time: float = Time.get_ticks_usec()
func history_start():
	return current_history[0]
func history_end():
	return current_history[current_history.size()-1]
func _input(event):{…}
func start_record():
	current_history = []
	record_start_time = Time.get_ticks_usec()
func record_input(input_type: CustomInput.InputType,
	input_value: CustomInput.InputValue, timestamp: float) -> void:
	var input = CustomInput.new()
	input.type = input_type
	input.value = input_value
	input.timestamp = timestamp - record_start_time
```

```
      current_history.append(input)
func end_record():
      saved_histories.append(current_history)
func hard_reset_clones():
      saved_histories = []
      start_record()
func new_clone():
      end_record()
      start_record()

func overwrite_clone():
      start_record()
```

За сохранение значений ввода игрока отвечает функция `_input`. Реализация функции представлена в листинге 8.

Листинг 8 – Реализация функции `_input`

```
func _input(event):
      if event is InputEventKey:
            var time = Time.get_ticks_usec()
            var input_type: CustomInput.InputType = CustomInput.Input-
Type.NONE
            var input_value: CustomInput.InputValue = CustomInput.Input-
Value.NONE
            match event.keycode:
                  KEY_A:
                        input_type = CustomInput.InputType.LEFT
                  KEY_D:
                        input_type = CustomInput.InputType.RIGHT
                  KEY_W:
                        input_type = CustomInput.InputType.UP
                  KEY_S:
                        input_type = CustomInput.InputType.DOWN
                  KEY_E:
                        input_type = CustomInput.InputType.INTERACT
                  _:
                        input_type = CustomInput.InputType.NONE
            if event.pressed:
                  input_value = CustomInput.InputValue.PRESSED
            else:
                  input_value = CustomInput.InputValue.RELEASE

            if input_type != CustomInput.InputType.NONE:
                  record_input(input_type, input_value, time)

      if event is InputEventMouseButton:
            var timeM = Time.get_ticks_usec()
            var input_type: CustomInput.InputType = CustomInput.Input-
Type.NONE
            var input_value: CustomInput.InputValue = CustomInput.Input-
Value.NONE
            if event.button_index == MOUSE_BUTTON_LEFT:
                  input_type = CustomInput.InputType.MOUSE_CLICK
                  if event.pressed:
                        input_value = CustomInput.InputValue.PRESSED
                  else:
                        input_value = CustomInput.InputValue.RELEASE
            if input_type != CustomInput.InputType.NONE:
                  record_input_mouse(input_type, input_value,
get_global_mouse_position(), timeM)
```

За обработку массивов ввода и приведение клонов в движение отвечает класс `Clone`, наследник класса `Player`, переопределяющий часть его методов. Реализация класса `Clone` представлена в листинге 9.

Листинг 9 – Реализация класса `Clone`

```
extends Player
class_name Clone
@export var color: Color = Color.SKY_BLUE
var input_history: Array[CustomInput] = []

func _ready():
	(sprite.material as ShaderMaterial).set_shader_parameter("color_over-
ride", color)
	speed = 200
	start_time = Time.get_ticks_usec()
func _process(_delta):

	replay_inputs()
func replay_inputs() -> void:
	for input in input_history:
		if Time.get_ticks_usec() >= start_time + input.timestamp:
			apply_input(input)
			input_history.erase(input)
	if input_history.is_empty():
		cancel_move()

func apply_input(input: CustomInput) -> void:
	match input.type:
		CustomInput.InputType.LEFT:
			if input.is_pressed():
				left = true
			elif input.is_released():
				left = false
		CustomInput.InputType.RIGHT:…
		CustomInput.InputType.UP:…
		CustomInput.InputType.DOWN:…
		CustomInput.InputType.MOUSE_CLICK:
			if input.is_pressed():
				mouse_position = input.mouse_click_position
				look_at(mouse_position)
				pushPlatAnimState = pushPlatfState.READY
			elif input.is_released():
				mouse_position = input.mouse_click_position
				look_at(mouse_position)
				left_cliclk = true
				$leftClickTimer.start()
				pushPlatAnimState = pushPlatfState.PUSH
```

За создание клонов на игровом уровне отвечают функции `create_clone` и `create_clones`, которые берут информацию о клоне и последовательности его действий из `clone_manager`. Также каждому клону присваивается цвет из массива для их визуальной дифференциации. Реализация методов создания клонов представлена в листинге 10.

Листинг 10 – Реализация методов `create_clone` и `create_clones`

```
func create_clones():
	for i in clone_manager.saved_histories.size():
		var history = clone_manager.saved_histories[i]
		var color = clone_colors[i]
		var clone = create_clone(history, color)
		clones.append(clone)

func create_clone(history: Array[CustomInput], clone_color = Color.YELLOW):
	var clone: Clone = clone_scene.instantiate()
	clone.color = clone_color
	clone.input_history = history.duplicate()
	add_child(clone)
	clone.position = spawn_point
	return clone
```

Изображение игрового окна с игровым персонажем и клонами представлено на рисунке 19.

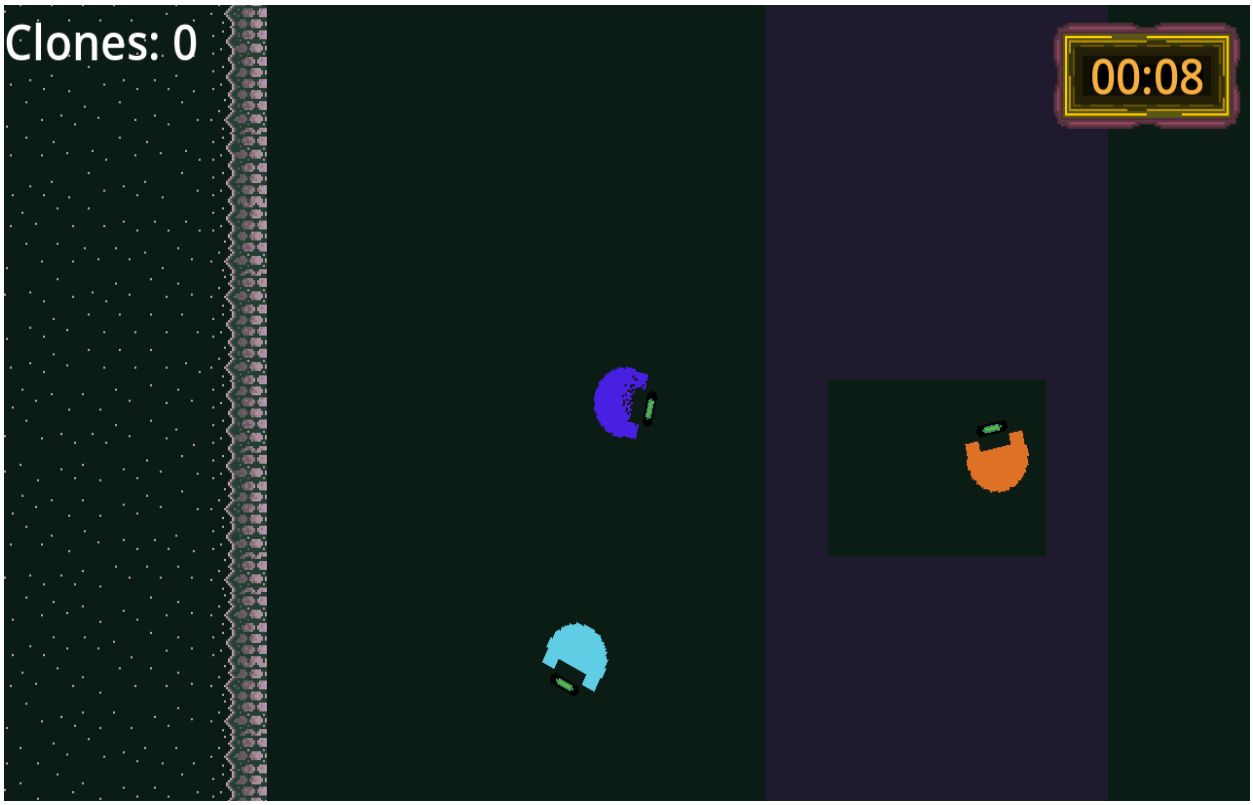


Рисунок 19 – Изображения игрового окна с персонажем и клонами

### 3.6. Интерактивные объекты

**Объект «Победа»**

Цель каждого игрового уровня заключается во взаимодействии игрового персонажа с объектом «Победа», который как правило находится в

труднодоступной части игровой локации. При соприкосновении игрового персонажа с объектом «Победа» уровень будет считаться пройденным и произойдет смена на экран выбора уровня [12]. Функция `_on_body_entered` отвечает за регистрацию соприкосновения с игровым персонажем. Функция `win` отвечает за сброс записанных клонов и смену сцены. Реализация методов объекта «Победа» представлена в листинге 11.

Листинг 11 – Реализация методов объекта «Победа»

```
func _ready():
	$AnimatedSprite2D.play("default")

func _on_body_entered(body):
	if body is Player:
		$WinTimer.start()
		grow_btn(grow_size, 2)
func win():
	clone_manger.hard_reset_clones()
	get_tree().change_scene_to_file("res://scenes/LevelSelect-
Screen.tscn")

func _on_win_timer_timeout():
	win()

func grow_btn(end_size: Vector2, duration: float) -> void:
	var tween := create_tween().set_trans(Tween.TRANS_LIN-
EAR).set_ease(Tween.EASE_IN_OUT)
	tween.tween_property(self, 'scale', end_size, duration)


func grow_btn(end_size: Vector2, duration: float) -> void:
	var tween := create_tween().set_trans(Tween.TRANS_LIN-
EAR).set_ease(Tween.EASE_IN_OUT)
	tween.tween_property(self, 'scale', end_size, duration)
```

Изображение объекта «Победа» и игрового персонажа представлено на рисунке 20.

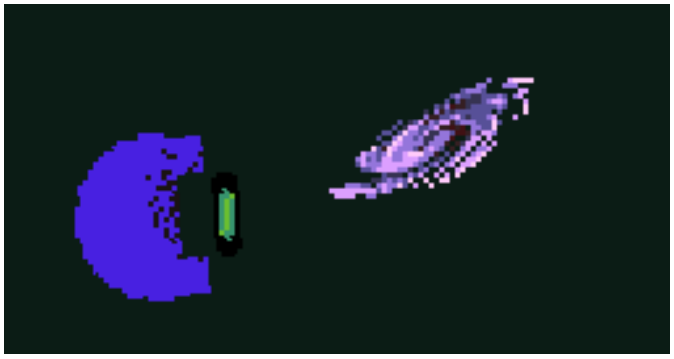

Рисунок 20 – Игровой персонаж и объект «Победа»

**Двери и кнопки**

Двери и кнопки в представленном игровом приложении являются взаимосвязанными объектами. Каждая дверь связана с открывающей ее кнопкой. Анимации открытия и закрытия двери реализованы в узле `AnimationPlayer` через изменение размеров и позиции объекта. Реализация класса `Door` представлена в листинге 12.

Листинг 12 – Реализация класса `Door`

```
extends Switch
class_name Door
@onready var ap: AnimationPlayer = $AnimationPlayer
var opened: bool = false

func on():
	opened = true
	ap.play("activation")

func off():
	opened = false
	ap.play("deactivation")

func reset():
	ap.play("idle")
	opened = false
```

Кнопки в свою очередь имеют экспортируемую в инспектор Godot переменную `interactable_path` которая и служит для связи объекта кнопки с конкретным объектом двери. Объект кнопки так же использует узел `AnimationPlayer` для изменения внешнего вида при взаимодействии с игровым персонажем и его клонами. Реализация класса `FloorButton` представлена в листинге 13.

Листинг 13 – Реализация класса `FloorButton`

```
extends Interactable
class_name FloorButton
@onready var ap: AnimationPlayer = $AnimationPlayer
@export var interactable_path: NodePath
var activated: bool = false

func interact():
	activate(true)
func reset():
	activate(false)
func activate(value: bool):
	activated = value
	if activated == true:
		ap.play("activated")
		(get_node(interactable_path) as Switch).on()
```

```
	else:
		(get_node(interactable_path) as Switch).off()
		ap.play("idle")

var body_count = 0

func _on_body_entered(body: Node2D):
	if body is Player or body is Clone:
		body_count += 1
		if body_count == 1:
			activate(true)
func _on_body_exited(body):
	if body is Player or body is Clone:
		body_count -= 1
		if body_count == 0:
			activate(false)
```

Изображение интерактивных объектов «Дверь» и «Кнопка» представлено на рисунке 21.

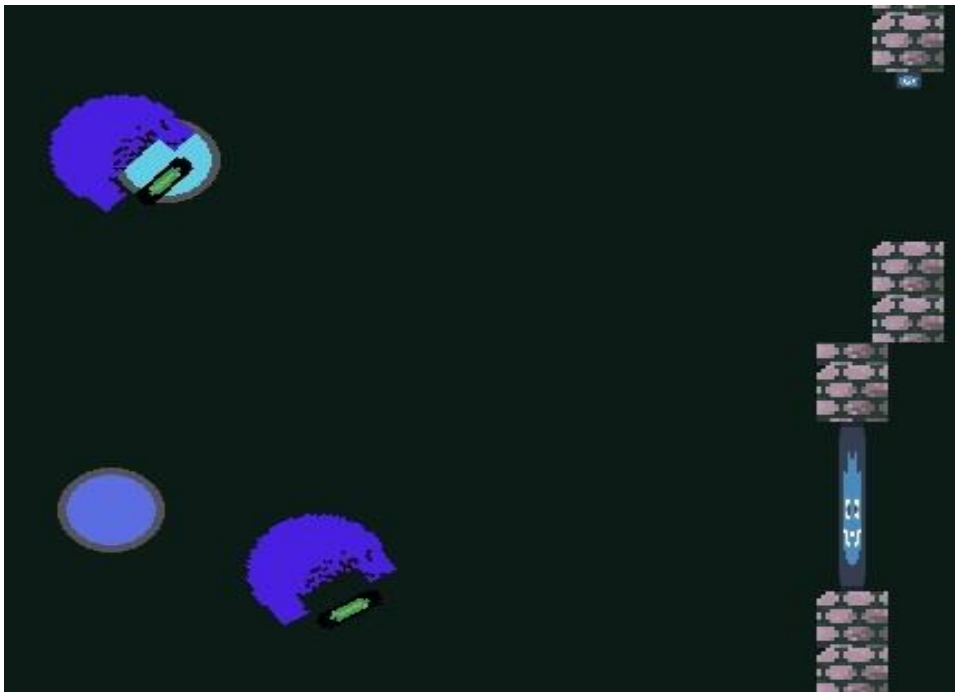

Рисунок 21 – Изображение интерактивных объектов «Дверь» и «Кнопка»

**Объект «Пропасть»**

На игровых локация присутствует область, недоступная игроку и его клонам – «Пропасть». При соприкосновении подконтрольного персонажа с пропастью активируется анимация падения и по истечению двухсекундного таймера игровая локация перезапускается, сохраняя всех записанных клонов. Клоны подконтрольного персонажа при соприкосновении с пропастью исчезают с локации по истечению двухсекундного таймера. Преодоление

пропасти клоном или подконтрольным персонажем возможно только в состоянии толчка. Изображение объекта «Пропасть» и подконтрольного персонажа представлено на рисунке 22.

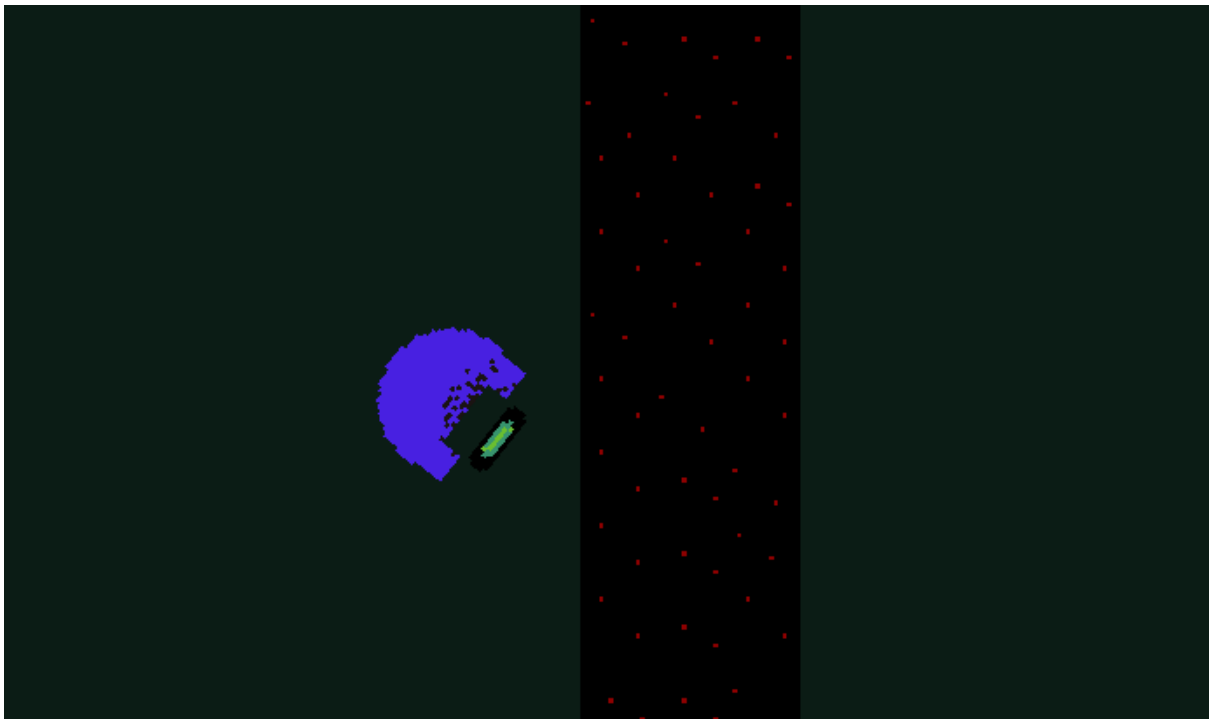

Рисунок 22 – Объект «Пропасть»

## 4. ТЕСТИРОВАНИЕ

В ходе данного тестирования [13] проверялось реализованное приложение на соответствие предъявленным функциональным [14] требованиям, а также особенностям, описанным на этапе проектирования. В таблице 1 представлены результаты функционального тестирования реализованного игрового приложения.

Таблица 1 – Функциональное тестирование

| № | Название теста | Действия | Ожидаемый результат | Тест пройден? |
|---|---|---|---|---|
| 1 | Запуск приложения | Запустить приложение | Игровое приложение успешно запущено | Да |
| 2 | Закрытие приложения | 1. Открыть приложение.<br>2. Нажать кнопку «Выйти». | Игровое приложение успешно закрыто | Да |
| 3 | Открытие экрана помощи | 1. Открыть приложение.<br>2. Нажать кнопку «Помощь». | Экран помощи успешно открыт | Да |
| 4 | Открытие экрана выбора уровня | 1. Открыть приложение.<br>2. Нажать кнопку «Выбрать уровень». | Экран выбора уровня успешно открыт | Да |
| 5 | Запуск уровня | 1. Открыть приложение.<br>2. Нажать кнопку «Выбрать уровень».<br>3. Нажать на одну из нумерованных кнопок выбора уровня. | Игровой уровень успешно запущен | Да |
| 6 | Передвижение подконтрольного персонажа по горизонтальной и вертикальной оси | 1. Запустить один из игровых уровней.<br>2. Зажать клавишу «A», «D», «W» или «S». | Персонаж перемещается влево при зажатии клавиши «A» и вправо при зажатии клавиши «D», вверх при зажатии клавиши «W» и вниз при зажатии клавиши «S» | Да |

Продолжение таблицы 1

| № | Название теста | Действия | Ожидаемый результат | Тест пройден? |
|---|---|---|---|---|
| 7 | Создание клона игрового персонажа | 1. Запустить один из игровых уровней.<br>2. Изменить местоположение подконтрольного персонажа в произвольном направлении.<br>3. Нажать клавишу «T». | Персонаж перемещается на стартовую точку игровой локации.<br>Клон повторяет патерн передвижения подконтрольного персонажа из предыдущего цикла | Да |
| 8 | Открытие двери подконтрольным персонажем посредством напольной кнопки | 1. Запустить игровой уровень №2 или №3.<br>2. Переместить игрового персонажа на напольную кнопку, находящуюся в доступной части локации. | Происходит открытие одной из дверей в видимой части локации | Да |
| 9 | Открытие двери клоном посредством напольной кнопки | 1. Запустить игровой уровень №2 или №3.<br>2. Переместить игрового персонажа на напольную кнопку, находящуюся в доступной части локации.<br>3. Нажать клавишу «T». | Персонаж перемещается на стартовую точку игровой локации.<br>Клон, повторяя патерн передвижения подконтрольного персонажа из предыдущего цикла, нажимает на напольную кнопку.<br>Происходит открытие одной из дверей в видимой части локации. | Да |
| 10 | Прохождение игрового уровня | 1. Запустить один из игровых уровней.<br>2. Используя доступные игровые возможности довести подконтрольного персонажа к объекту «Победа».<br>3. Коснуться игровым персонажем объекта «Победа» | Происходит закрытие игрового уровня. | Да |
| 11 | Выполнение толчка игровым персонажем | 1. Запустить один из игровых уровней.<br>2. Зажать и отпустить левую кнопку мыши. | Подконтрольный персонаж игрока воспроизводит анимацию толчка | Да |

Окончание таблицы 1

| № | Название теста | Действия | Ожидаемый результат | Тест пройден? |
|---|---|---|---|---|
| 12 | Выполнение толчка игровым персонажем от препятствия | 1. Запустить один из игровых уровней.<br>2. Подойти персонажем к ближайшей стене.<br>3.Зажать и отпустить левую кнопку мыши | Подконтрольный персонаж игрока отталкивается от стены. | Да |
| 13 | Выполнение толчка клона игровым персонажем | 1. Запустить один из игровых уровней.<br>2. Создать клона.<br>3. Подойти к клону и направить курсор в его сторону<br>4. Зажать и отпустить левую кнопку мыши | Подконтрольный персонаж игрока отталкивает клона | Да |
| 14 | Падение в пропасть | 1. Запустить один из игровых уровней.<br>2. Подойти персонажем к пропасти и соприкоснуться с ней | Происходит анимация падения и перезагрузка локации | Да |
| 15 | Преодоление пропасти толчком | 1. Запустить один из игровых уровней.<br>2. Подойти персонажем к стене, находящейся возле пропасти.<br>3. Навести курсор на стену и скорректировать угол полета<br>4. Зажать и отпустить левую кнопку мыши | Подконтрольный персонаж игрока отталкивается от стены и преодолевает пропасть. | Да |

## ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана компьютерная игра в жанре «Top-Down Puzzle» для платформы Windows. Разработка велась с использованием игрового движка Godot Engine.

В процессе выполнения работы были решены следующие задачи:

1) проанализирована предметная область;
2) спроектирована архитектура игрового приложения;
3) реализовано игровое приложение;
4) протестировано игровое приложение.

Дальнейшее развитие представленного игрового приложения может включать в себя:

1) добавление новых локаций и игровых ситуаций, основывающихся на разработанных механиках;
2) разработка новых интерактивных объектов для увеличения разнообразия игровых ситуаций;
3) добавление звукового сопровождения;
4) улучшение качества игровых спрайтов;
5) разработка новых анимаций, что добавить динамики и улучшит восприятие игровых действий.

Таким образом, данная работа демонстрирует возможности разработки игровых приложений на базе Godot Engine и открывает перспективы для дальнейшего совершенствования и развития созданного продукта.

**ЛИТЕРАТУРА**

1. Statista. [Электронный ресурс] URL: https://www.statista.com/statistics/1132706/media-revenue-worldwide (дата обращения: 10.02.2024 г.).

2. Gamicus – Top-down perspective video games. [Электронный ресурс] URL: https://gamicus.fandom.com/wiki/Top-down_perspective_video_games. (дата обращения: 10.02.2024 г.).

3. Sokoban. [Электронный ресурс] URL: https://www.sokobanonline.com/ (дата обращения: 07.04.2024 г.).

4. Game forge. [Электронный ресурс] URL: https://gameforge.com/en-US/littlegames/fire-and-water-games/ (дата обращения: 09.02.2024 г.).

5. Draw.io. – Программное обеспечение для рисования графиков. [Электронный ресурс] URL: https://app.diagrams.net/ (дата обращения: 07.04.2024 г.).

6. Бубнов А., Бубнов С., Майков К. Разработка и анализ требований к программному обеспечению. –КУРС, 2023. – 176 с.

7. Godotengine GDScript Basics. [Электронный ресурс] URL: https://docs.godotengine.org/ru/4.x/tutorials/scripting/gdscript/gdscript_basics.html (дата обращения: 05.03.2024 г.)

8. Godotengine – Официальный сайт Godot. [Электронный ресурс] URL: https://godotengine.org/ (дата обращения: 05.03.2024 г.).

9. Visual paradigm – Use case diagram. [Электронный ресурс] URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата обращения: 07.03.2024 г.).

10. Visual paradigm – Component diagram. [Электронный ресурс] URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/ (дата обращения: 07.03.2024 г.).

11. Godotengine – Godot's key concepts. [Электронный ресурс] URL: https://docs.godotengine.org/en/stable/getting_started/introduction/key_concepts_overview.html (дата обращения: 09.03.2024 г.).

12. Pixel-planet-generator. [Электронный ресурс] URL: https://deep-fold.itch.io/pixel-planet-generator (дата обращения: 01.04.2024 г.).

13. Как проводить юзабилити-тестирование. [Электронный ресурс] URL: https://media.contented.ru/opyt/instrukcii/kak-provodit-yuzabilititestiro-vanie/ (дата обращения: 10.04.2024 г.).

14. Функциональное тестирование ПО. [Электронный ресурс] URL: https://unetway.com/tutorial/funkcionalnoe-testirovanie (дата обращения: 22.05.2024 г.).

15. Itch.io – Библиотека игровых ассетов. [Электронный ресурс] URL: https://itch.io/game-assets/free/tag-tileset (дата обращения: 07.04.2024 г.).