Коновалов В.В.,

Бужинская Н.В.,

Нижнетагильский государственный социально-педагогический институт,
(филиал) ФГБОУ ВО «Российский государственный профессионально-педагогический университет»

Россия, Нижний Тагил

РАЗРАБОТКА 2D-ИГРЫ ПЛАТФОРМЕРА НА UNITY С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА ПРОГРАММИРОВАНИЯ C#

Аннотация

Данная статья подробно исследует ключевые этапы разработки мобильной игры на платформе Unity с использованием тайлов и специализируется на нескольких важных аспектах создания игрового приложения. Рассматриваются процессы проектирования и создания игровой сцены, включая разработку и использование тайловых карт, а также создание анимаций и их переключение. В статье также описывается создание звукового сопровождения для игры и персонажей.

Ключевые слова: разработка мобильных игр, Unity, Тайловые карты, Анимации игры, Звуковые эффекты

DEVELOPMENT OF A 2D PLATFORMER GAME IN UNITY USING C#

Abstract

This article thoroughly examines the key stages of developing a mobile game on the Unity platform utilizing tiles. It specializes in several crucial aspects of creating a gaming application. It covers the processes of designing and building a game scene, including the development and utilization of tilemaps, as well as the creation of animations and their seamless transitions. The article also describes the creation of soundtracks for the game and sound effects for the characters.

Keywords: mobile game development, Unity, Tilemaps, Game animations, Sound effects

Платформеры — это своеобразный, довольно легко узнаваемый жанр компьютерной игры. Он заключается в том, что у персонажа есть задача — пройти по разным уровням, преодолевая препятствия [1].

Перейдем к созданию игры. Игра выполнена в форме платформера, в котором игроку предлагается добраться на каждом из уровней от начальной точки к финишу. Игра включает несколько уровней (рис. 1), (рис. 2). Уровни отличаются сложностью и механиками, которые там применяются.

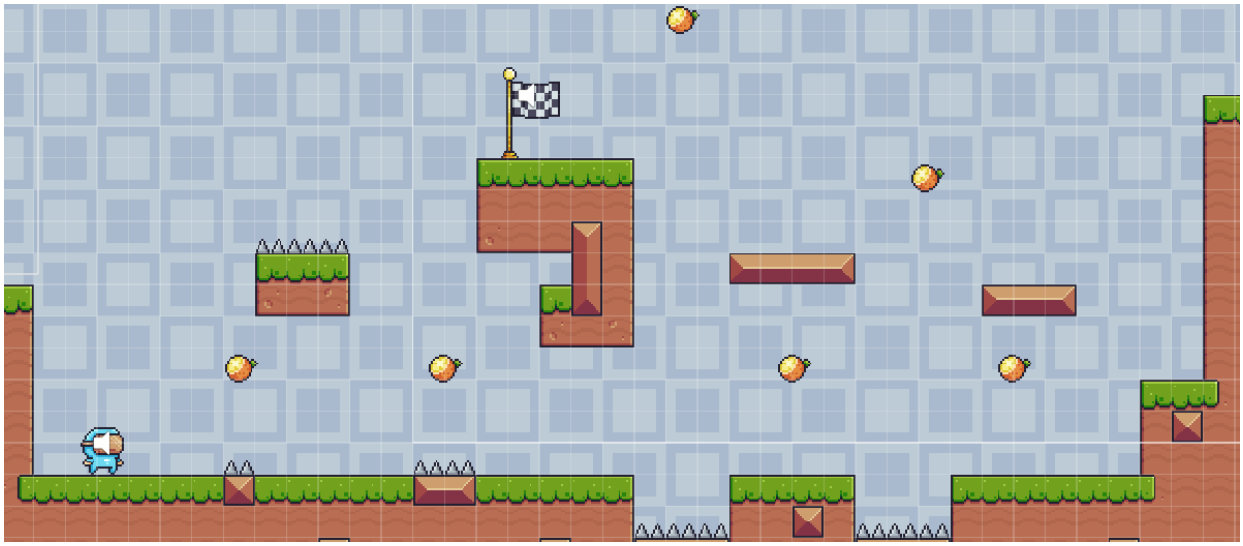

Рис. 1. Первая сцена

Рис. 2. Вторая сцена

Рассмотрим ключевые этапы разработки игры.

Для начала разработки необходимо установить Unity для непосредственной работы над проектом, а также Visual Studio для создания скриптов на языке программирования C#.

Карта мира будет составлена из тайлов, которые представляют собой небольшие фрагменты изображений, собранные в сетку. Эти тайлы в конечном итоге формируют цельное изображение, представляющее уровень игры (рис. 3).

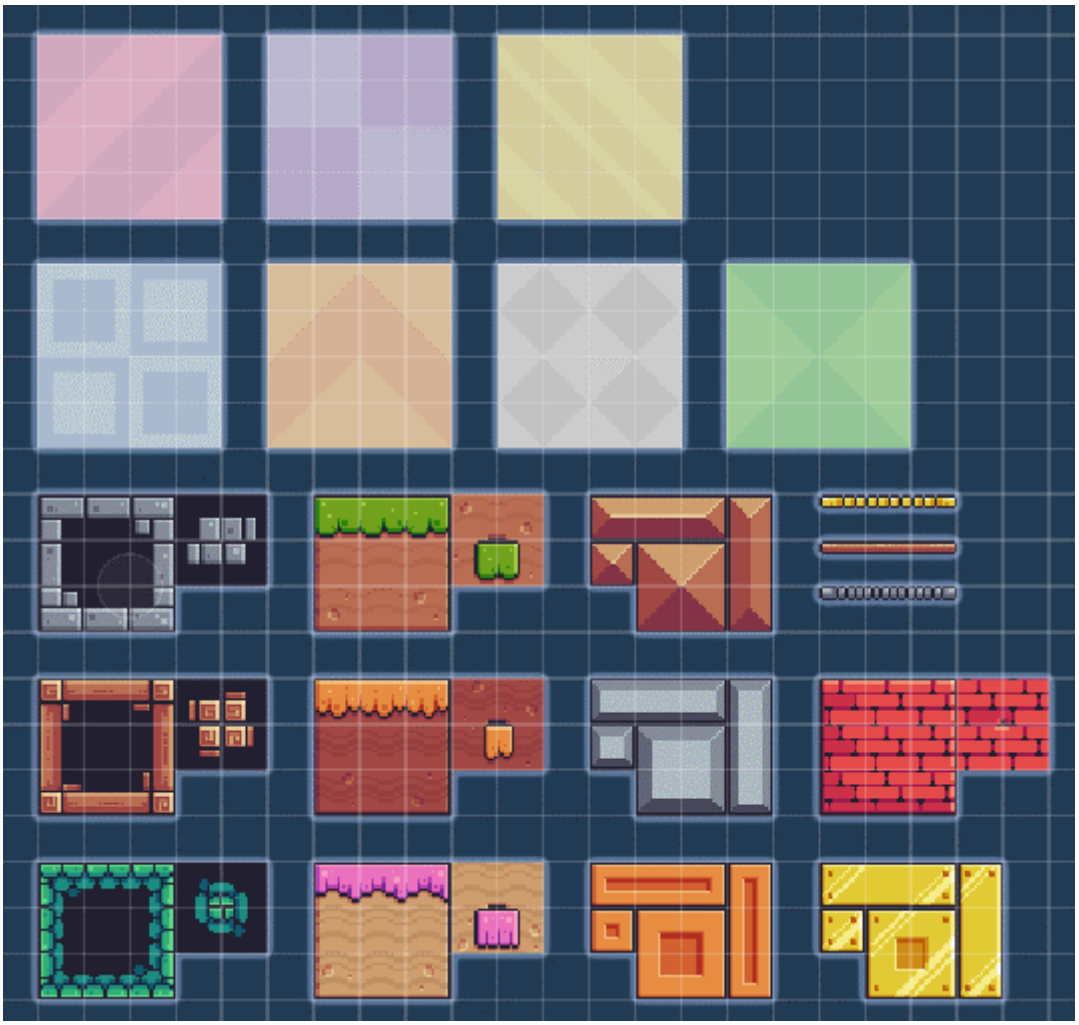

Рис. 3. Карта тайлов проекта

Для создания персонажа, мы начнем с создания простого спрайта. Затем мы присоединим к нему компонент «Rigidbody 2D», который придаст физическую природу персонажу. Далее, для определения границ персонажа в пространстве и его взаимодействия с окружением, мы прикрепим компонент «BoxCollider 2D».

После этого, необходимо написать скрипт для управления движением персонажа в пространстве. Мы реализуем управление с использованием джойстика (листинг 1).

Листинг 1

Скрипт передвижения персонажа

```
[SerializeField] private float moveSpeed = 7f;
[SerializeField] private float jumpForce = 15f;
public Joystick joystick;
private bool isJumpButtonPressed = false;
private void Update()
{
    dirX = joystick.Horizontal;
    if (dirX < 0f)
        dirX = -1f;
    if (dirX > 0f)
        dirX = 1f;
    rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);
    if (isJumpButtonPressed)
    {
        if (IsGrounded())
        {
            rb.velocity = new Vector2(rb.velocity.x, jumpForce);
        }
        isJumpButtonPressed = false;
    }
    UpdateAnimationState();
}
```

Добавим в этот же скрипт функционал для смены анимаций, в зависимости от действий персонажа (листинг 2).

Листинг 2

Скрипт смены анимаций персонажа

```
private enum MovementState { idle, running, jumping, falling }
private void UpdateAnimationState()
{
    MovementState state;
    if (dirX > 0f){
        state = MovementState.running;
        sprite.flipX = false;
    }
    else if (dirX < 0f){
        state = MovementState.running;
        sprite.flipX = true;
    }
    else{
        state = MovementState.idle;
    }
    if (rb.velocity.y > .1f){
        state = MovementState.jumping;
    }
    else if (rb.velocity.y < -.1f){
        state = MovementState.falling;
    }
    anim.SetInteger("state", (int)state);
}
```

Далее, создадим анимацию в файлах проекта и откроем окно анимаций. Затем перенесем спрайты одной анимации в ранее созданную анимацию. Настоим скорость воспроизведения для всех анимаций. Теперь необходимо присоединить компонент «Animator» к персонажу и перейти в него. В открывшемся окне нужно настроить переключение анимаций для персонажа, в зависимости от его действий (рис. 4). Также установим время перехода для каждой анимации на 0 секунд для более отзывчивого воспроизведения.

Создадим две переменные: одну для отслеживания состояния персонажа (целочисленного типа) и вторую для отслеживания состояния смерти персонажа (тип булеан). Добавим переменные состояния в переходы каждой анимации и

свяжем их с значениями, определенными ранее в скрипте персонажа (листинг 2).

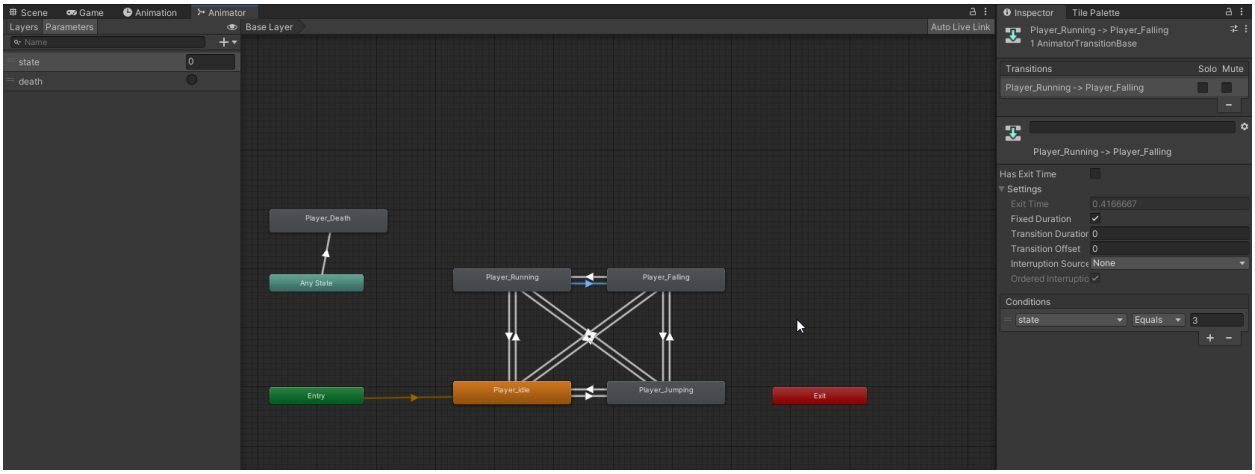

Рис. 4. Аниматор персонажа

Теперь мы перейдем к созданию игрового уровня с использованием тайлов. Для этого нам потребуется создать 2D тайлмап, используя квадратные тайлы. Затем мы откроем окно с палитрой тайлов и загрузим заранее подготовленный тайлмап. Создадим желаемый игровой уровень и присоединим к тайлмапу компоненты: «Tilemap Collider 2D», «Rigidbody 2D» (изменим тип с динамического на статический), «Composite Collider 2D» и «Platform Effector 2D».

Далее мы приступим к добавлению ловушек, начиная с шипов. Мы создадим спрайт и прикрепим к нему компонент «Box Collider 2D». Зададим этому объекту метку (тег) «Trap» и создадим скрипт, который впоследствии будет отвечать за смерть персонажа (Листинг 3).

Листинг 3

Скрипт гибели персонажа

```
private void Start()
{
    rb = GetComponent<Rigidbody2D>();
    anim = GetComponent<Animator>();
}
private void OnCollisionEnter2D(Collision2D collision)
{
```

```
        if (collision.gameObject.CompareTag("Trap"))
        {
            Die();
        }
    }
    private void Die()
    {
        deathSoundEffect.Play();
        rb.bodyType = RigidbodyType2D.Static;
        anim.SetTrigger("death");
    }
    private void RestartLevel()
    {
     SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
```

Добавим цепную пилу как следующую ловушку. Она будет состоять из спрайта, который вращается по кругу. Прикрепляем к этому спрайту компонент «Circle Collider 2D» и создаем скрипт для управления вращением пилы (листинг 4).

Листинг 4

Скрипт вращения цепной пилы

```
    private void Update()
    {
        transform.Rotate(0, 0, 360 * speedRotate *
Time.deltaTime);
    }
```

Теперь перейдем к созданию движущихся платформ. Для этого создадим спрайт и прикрепим к нему компонент «Box Collider 2D» в двух экземплярах. Один из коллайдеров создаем с размерами, соответствующими размеру платформы, а второй создаем над первым коллайдером с высотой один, и установим флажок, который делает этот коллайдер триггером.

Далее создаем два пустых объекта и размещаем их в двух точках, где будут находиться конечные пути движения платформы. После этого напишем

скрипт для перемещения платформы от одной точки к другой (листинг 5). Этот скрипт также позволяет добавлять больше двух точек для движущейся платформы в самом движке Unity.

Листинг 5

Скрипт движущихся платформ

```
    private int currentWaypointIndex = 0;
    [SerializeField] private float speed = 2f;
    private void Update()
    {
        if
(Vector2.Distance(waypoints[currentWaypointIndex].transform.positi
on, transform.position) < .1f)
        {
            currentWaypointIndex++;
            if (currentWaypointIndex >= waypoints.Length)
            {
                currentWaypointIndex = 0;
            }
        }
        transform.position =
Vector2.MoveTowards(transform.position,
waypoints[currentWaypointIndex].transform.position, Time.deltaTime
* speed);
    }
```

Однако, в данное время персонаж не перемещается вместе с платформой. Для этого напишем скрипт, который позволит персонажу начать перемещаться вместе с платформой при соприкосновении с триггером (листинг 6).

Листинг 6

Скрипт для перемещения персонажа вместе с движущейся платформой

```
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            collision.gameObject.transform.SetParent(transform);
```

```
    }
}
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.gameObject.name == "Player")
    {
        collision.gameObject.transform.SetParent(null);
    }
}
```

Теперь перейдем к созданию предметов для сбора, которые будут давать игроку очки. Для этого создадим спрайт и прикрепим к нему компонент «Box Collider 2D», затем установим флажок для этого коллайдера, чтобы он стал триггером. Дадим этому объекту метку (тег) «Orange» и напишем скрипт для сбора апельсинов (листинг 7).

Листинг 7

Скрипт сбора апельсинов

```
private int score = 0;
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Orange"))
    {
        Destroy(collision.gameObject);
        score++;
        orangesText.text = "Score: " + score;
    }
}
```

Далее, приступим к созданию небольшого пользовательского интерфейса, начиная с счетчика очков. Для этого создадим область пользовательского интерфейса, добавив объект «Canvas». В свойствах выберем «Render Mode» и установим «Screen Space – Overlay». Затем добавим объект «Text», разместим его в верхнем левом углу и привяжем его. В скрипте выше (листинг 7) уже содержится код для отображения счетчика очков.

Далее, создадим элемент управления в виде джойстика. Для этого скачаем джойстик из ресурсов Unity Asset Store и импортируем его в наш проект. Скрипт для управления персонажем с использованием джойстика мы уже написали (листинг 1). Разместим этот элемент управления в нижнем левом углу пользовательского интерфейса.

Также создадим кнопку для выполнения прыжка. Для этого создадим спрайт и прикрепим к нему компонент «Button». Выберем функцию, которая будет вызываться при нажатии на эту кнопку, и дополним скрипт управления персонажем этой функцией (листинг 8). Мы уже создали метод для выполнения прыжка ранее, но он еще не вызывался.

Листинг 8

Функция для срабатывания триггера

```
public void OnJumpButtonClicked()
{
    isJumpButtonPressed = true; // Устанавливаем флаг, когда кнопка нажата
}
```

Теперь давайте создадим финишную линию. При достижении этой линии игрок будет перенесен на следующую сцену. Для этого создадим спрайт и прикрепим к нему компонент «Box Collider 2D», затем установим флажок, чтобы этот коллайдер стал триггером. Теперь напишем скрипт для реализации перехода на следующую сцену (листинг 9).

Листинг 9

Скрипт для перехода на следующую сцену

```
private bool LevelCompleted = false;
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.name == "Player" && !LevelCompleted)
    {
        finishSound.Play();
        LevelCompleted = true;
```

```
            Invoke("CompleteLevel", 1.2f);
        }
    }
    private void CompleteLevel()
    {
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildInd
ex + 1);
    }
```

Давайте теперь создадим звуковое сопровождение и звуковые эффекты. Для создания звукового сопровождения, создадим пустой объект и прикрепим к нему компонент «Audio Source». Затем мы прикрепим желаемую музыкальную дорожку и установим флажок «Loop» для воспроизведения музыки в цикле. Аналогично создадим аудио-источник для финиша.

Чтобы добавить звук прыжка персонажа, мы обновим скрипт управления движением персонажа (листинг 10).

Листинг 10

Обновленная часть скрипта передвижения персонажа

```
    [SerializeField] private AudioSource jumpForceEffect;
    private void Update()
    {
        dirX = joystick.Horizontal;
        if (dirX < 0f)
            dirX = -1f;
        if (dirX > 0f)
            dirX = 1f;
        rb.velocity = new Vector2(dirX * moveSpeed,
rb.velocity.y);
        if (isJumpButtonPressed)
        {
            if (IsGrounded())
            {
                jumpForceEffect.Play();
                rb.velocity = new Vector2(rb.velocity.x,
jumpForce);
```

```
            }
            isJumpButtonPressed = false;
        }
        UpdateAnimationState();
    }
```

Аналогично создаем звук для сбора предметов и гибели персонажа.

Далее создаем стартовую сцену, главное меню. Создадим объект кнопку и добавим вызов функции при её нажатии (листинг 11).

Листинг 11

Скрипт для запуска уровней

```
    public void StartGame()
    {
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildInd
ex + 1);
    }
```

Аналогично создаем конечную сцену, где кнопку для выхода создаем аналогично и пишем небольшой скрипт (листинг 12).

Листинг 12

Скрипт завершения игры

```
    public void Quit()
    {
        Application.Quit();
    }
```

Рис. 5. Итоговый вид игры

Получаем итоговой вид игры (рис. 5). В дальнейшем планируется усовершенствование игры, включая улучшение главного меню, добавление возможности выбора уровней и внедрение системы сохранения игрового прогресса.

Список источников информации

1. Жанр компьютерных игр: платформеры. URL: https://gamersgate.ru/reviews/zhanr-kompyuternykh-igr-platfo/ (дата обращения: 01.10.2023).

2. Корнилов, А. В. Unity. Полное руководство / А. В. Корнилов. — Санкт-Петербург : Наука и Техника, 2020. — 432 с. — ISBN 978-5-94387-795-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/175394 (дата обращения: 10.10.2023). — Режим доступа: для авториз. пользователей.

3. Про создание платформера на Unity. URL: https://habr.com/ru/companies/microsoft/articles/236125/ (дата обращения: 06.10.2023).

4. 2D Platformer: Creating a Tilemap using Spritesheets. URL: https://medium.com/@eveciana21/2d-platformer-creating-a-tilemap-using-spritesheets-181543b6ab64 (дата обращения: 08.10.2023).