Gra platformowa z wykorzystaniem Unity i języka programowania C#

Tomasz Niedźwiedź

Wyższa Szkoła Ekonomii i Informatyki w Krakowie, 2018

tomasz12niedzwiedz@gmail.com

Streszczenie:

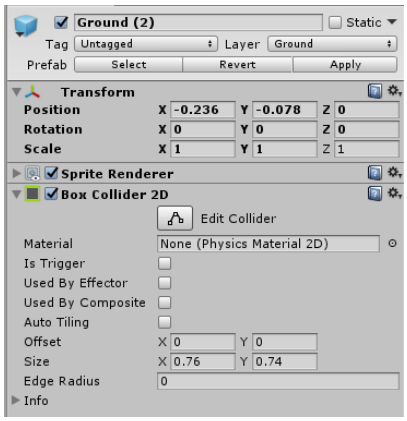
Prosta gra platformowa oparta o język programowania C# i silnik Unity, mechanicznie gotowa do dalszego rozwoju.

1. WSTĘP

C# jest językiem programowania zaprojektowany w latach 1998 – 2001 przez zespół Andersa Hejlsberga dla firmy Microsoft. Łączy sobie użyteczność innych języków programowania takich jak Object Pascal, Delphi, C++ czy Java. Unity jest zintegrowanym środowiskiem do tworzenia gier trójwymiarowych i dwuwymiarowych, lub innych materiałów interaktywnych, takich jak wizualizacje czy animacje. Unity obsługuje języki programowania C# i UnityScript (podobny do JavaScript w składni), ja wybrałem C# ze względu na moją naukę w tym kierunku.

2. PROJEKT

Na początku stworzyłem płaszczyznę, po której gracz będzie mógł się poruszać. Prefab *Ground* otrzymał Box Collider 2D i został umieszczony w warstwie *Ground*.



Drugim krokiem było stworzenie gracza, skupiałem się głównie na mechanice i kodzie, więc nasz gracz przyjmuje postać niebieskiego kwadratu i został zaopatrzony w Box Collider 2D, Rigidbody 2D i skrypt Player Controller.
Skrypt Player Controller został zaopatrzony w kilka zmiennych niezbędnych do poprawnego poruszania się graczem.

```
public class PlayerController : MonoBehaviour {

    public float moveSpeed;
    public float jumpHeight;
    public Transform groundCheck;
    public float groundCheckRadius;
    public LayerMask whatIsGroud;
    private bool grounded;
    private bool doubleJump;
    private float moveVelocity;
```

Zmienne kolejno odpowiadają za prędkość poruszania się, wysokość skoku, sprawdzanie czy znajdujemy się na płaszczyźnie Ground, promień w jakim groundCheck ma działać, określenie, że płaszczyzna Ground to ziemia, boolean sprawdzający czy znajdujemy się w powietrzu czy nie, boolean odpowiadający za podwójny skok, prędkość poruszania się naszego gracza.

Funkcja FixedUpdate wywołująca się kilka razy na klatkę gry odpowiada w tym wypadku za ciągłe sprawdzanie czy znajdujemy się na płaszczyźnie *Ground*.

```
void FixedUpdate()
{
    grounded = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, whatIsGroud);
}
```

W funkcji Update, która uruchamia się co każdą klatkę gry został umieszczony kod odpowiadający za poruszanie się w lewo i w prawo i za skok.
Skrypt skoku wygląda następująco:

```
public void Jump()
{
    GetComponent<Rigidbody2D>().velocity = new Vector2(GetComponent<Rigidbody2D>().velocity.x, jumpHeight);
}
```

Zmuszamy fizykę ciała 2D, żeby zwiększyło na ułamek sekundy prędkość na osi x o wartość podaną w zmiennej *jumpHeight*.

Podobnie jest podczas ruchu w lewo lub w prawo, wtedy wymuszamy ciągły ruch po osi y.

```
moveVelocity = 0f;

if (Input.GetKey(KeyCode.D))
{
    //GetComponent<Rigidbody2D>().velocity = new Vector2(moveSpeed, GetComponent<Rigidbody2D>().velocity.y);
    moveVelocity = moveSpeed;
}

if (Input.GetKey(KeyCode.A))
{
    //GetComponent<Rigidbody2D>().velocity = new Vector2(-moveSpeed, GetComponent<Rigidbody2D>().velocity.y);
    moveVelocity = -moveSpeed;
}

GetComponent<Rigidbody2D>().velocity = new Vector2(moveVelocity, GetComponent<Rigidbody2D>().velocity.y);
#endregion
```

Aby wykorzystać podwójny skok musimy sprawdzić, czy postać znajduje się na płaszczyźnie *Ground*, tylko jeśli boolean będzie wynosił 0 i skok był wykonany jedynie raz, to funkcja zostanie wykonana.

```
if (grounded)
{
    doubleJump = false;
}

if (Input.GetKeyDown(KeyCode.Space) && grounded)
{
    Jump();
}

if (Input.GetKeyDown(KeyCode.Space) && !doubleJump && !grounded)
{
    Jump();
    doubleJump = true;
}
```

Aby nasza postać mogła zginąć i powrócić do zabawy, został utworzony skrypt Level Manager. W funkcji *RespawnPlayerCo()* tworzy dla nas element *deathParticle*, który jest wizualnym efektem podczas umierania, następnie wyłącza możliwość ruchu naszym graczem, aby ten pojawił się w ostatnio odwiedzonym punkcie kontrolnym, również ze stosownym efektem wizualnym.

```
public IEnumerator RespawnPlayerCo()
{
    Instantiate(deathParticle, player.transform.position, player.transform.rotation);
    player.enabled = false;
    player.GetComponent<Renderer>().enabled = false;
    gravityStore = player.GetComponent<Rigidbody2D>().gravityScale;
    player.GetComponent<Rigidbody2D>().gravityScale = 0f;
    player.GetComponent<Rigidbody2D>().velocity = Vector2.zero;
    Debug.Log("You lost da wae, I will show u da wae");
    yield return new WaitForSeconds(respawnDelay);
    player.GetComponent<Rigidbody2D>().gravityScale = gravityStore;
    player.transform.position = currentCheckpoint.transform.position;
    player.enabled = true;
    player.GetComponent<Renderer>().enabled = true;
    Instantiate(respawnParticle, currentCheckpoint.transform.position, currentCheckpoint.transform.rotation);
}
```

Wspomniany efekt wizualny jest efektem cząsteczkowy, który dość mocno może wadzić w optymalizację naszej gry, jeśli gracz będzie umierał dużo razy pod rząd. Particle są przechowywane w pamięci, warto jest usunąć, pomaga w tym ten prosty kod.

```
void Start()
{
    levelManager = FindObjectOfType<LevelManager>();
}

void Update () {

}

private void OnTriggerEnter2D(Collider2D other)
{
    if (other.name == "Player")
    {
        levelManager.currentCheckpoint = gameObject;
        Debug.Log("Activate de wae = " + transform.position);
    }
}
```

Wspomniany punkt kontrolny, jest miejsce, gdzie nasz gracz wraca po śmierci. Za każdym razem, gdy przejdzie przez obiekt oznaczony jako *Checkpoint*, będzie pojawiał się w nim po śmierci.
Skrypt „Checkpoint” na samym początku odwołuje się do skryptu *LevelManager* w którym definiujemy, jak wygląda śmierć i odnowienie się gracza. W funkcji OnTriggerEnter2D(), znajdującej się w skrypcie „Checkpoint” definiujemy aktualny punkt do którego wrócimy po śmierci za pomocą wykrycia kolizji z graczem.

Aby obiekt typu „Checkpoint” był przeźroczysty, zastosowałem prosty kod, który oddziałuje na RGBA i zmienia kanał alpha na połowę wartości.

```
void Update () {
    spawnPoint.color = new Color(1f, 1f, 1f, .5f);
}
```

Czym jest gra platformowa bez przeciwników, zaimplementowałem prostego przeciwnika, który patroluje teren poruszając się od jednej ściany do drugiej.

```
public float moveSpeed;
public bool moveRight;
public Transform wallCheck;
public float wallCheckRadius;
public LayerMask whatIsWall;
private bool wall;
```

Kolejno: prędkość poruszania się, ruch w prawo, sprawdzenie czy znajdujemy się obok ściany, promienień działania wallCheck, określenie co jest ścianą, bool odpowiadający na pytanie czy spotkaliśmy ścianę.

Sam skrypt wygląda następująco:

```
void Update ()
{
    wall = Physics2D.OverlapCircle(wallCheck.position, wallCheckRadius, whatIsWall);

    if(wall)
    {
        moveRight = !moveRight;
    }

    if (moveRight)
    {
        transform.localScale = new Vector3(-1f, 1f, 1f);
        GetComponent<Rigidbody2D>().velocity = new Vector2(moveSpeed, GetComponent<Rigidbody2D>().velocity.y);
    }
    else
    {
        transform.localScale = new Vector3(1f, 1f, 1f);
        GetComponent<Rigidbody2D>().velocity = new Vector2(-moveSpeed, GetComponent<Rigidbody2D>().velocity.y);
    }
}
```

„Wall” został ustawiony jako *Ground* na naszym przypadku nie robi to znaczenia. Jeśli ściana zostanie wykryta (wykrywana krawędź jest tylko po osi y) wektor poruszania się przeciwnika zostaje odwrócony, co pozwala na efektywne patrolowanie.

Przeciwnik został też zaopatrzony w skrypt *Kill Player*, który pozwala zabić gracza, co cofa go do ostatniego punktu kontrolnego.

```
public LevelManager levelManager;

void Start ()
{
    levelManager = FindObjectOfType<LevelManager>();
}

void Update ()
{

}

private void OnTriggerEnter2D(Collider2D other)
{
    if(other.name == "Player")
    {
        levelManager.RespawnPlayer();
    }
}
```

Tak przygotowany kod jest otwartą drogą do dalszego rozwijania nasze gry, na przykład skrypt *Kill Player* możemy wykorzystać aby stworzyć płaszczyznę, która będzie nas ranić. Efekt przy śmierci może po lekkiej modyfikacji być także wybuchem. Ogranicza nas tylko wyobraźnia.

[i]Bibliografia

---

[i]https://pl.wikipedia.org/wiki/Unity_(silnik_gry)
https://pl.wikipedia.org/wiki/C_Sharp
https://docs.unity3d.com/Manual/index.html
https://www.youtube.com/user/gamesplusjames/