Dokumentacja projektu pt.:

„Gra – strzelanka 2D ”

Autorzy:
Tomasz Baran
Bartłomiej Depciuch
Informatyka III
gr. lab I

Spis treści

1. Autorzy projektu 3
2. Cel 3
3. Wykorzystane technologie 3
4. Opis projektu 3
5. Niuanse w kodzie 7

1. Autorzy projektu

Zespół składa się z 2 osób, tj. Tomasz Baran, Bartłomiej Depciuch. Każdy członek zespołu był równomiernie zaangażowany w realizację projektu. Prace nad grą oparte były w większości na anglojęzycznych poradnikach z Unity.

2. Cel

Celem projektu jest stworzenie gry typu strzelanka w systemie operacyjnym Android. Użytkownik po wejściu w aplikacje ma możliwość zagrania w 3-poziomową grę, gdzie z każdym poziomem przeciwników jest coraz więcej. Projekt ten został zrealizowany na potrzeby przedmiotu Programowanie urządzeń mobilnych.

3. Wykorzystane technologie

Lista technologii wykorzystanych w projekcie:

- ✓ Unity 2018.2.14f1
- ✓ System operacyjny Android 6.0.1

Oprócz tego w projekcie zostały użyte darmowe assety z Unity Asset Store.

4. Opis projektu

Cały projekt wraz z filmikiem prezentującym grę został umieszczony pod linkiem:

https://github.com/gtpolak/Gra

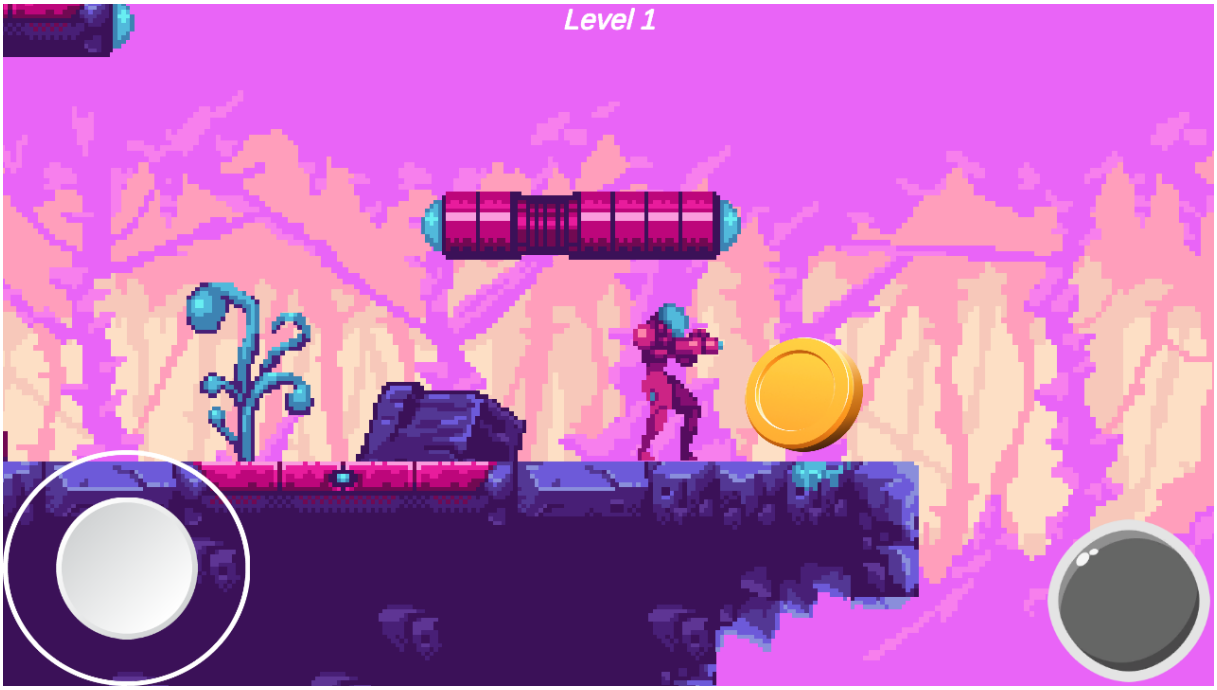
Po wejściu w aplikację użytkownikowi ukazuje się Menu Główne, które pozwala na 3 opcje:

1) Rozpoczęcie gry poprzez kliknięcie przycisku poziomu pierwszego
2) Zresetowanie poziomów gry (po każdym przejściu poziomu jest on zapisywany)
3) Wyjście z aplikacji

Użytkownik po wybraniu przycisku z nr. poziomu przenosi się do rozgrywki.

Aby przejść dany poziom należy pokonać / ominąć przeciwników znajdujących się na mapie gry.

Przejście do następnego poziomu następuje automatycznie wraz z wejściem w obiekt monety, który został umieszczony na końcu każdego poziomu gry.



Po wejściu w monetę użytkownikowi ukazuje się napis „Wygrałeś” co uświadamia go, że przeszedł dany poziom i za chwilę rozpocznie się kolejny.

Przeciwnicy umieszczeni w grze mogą przerwać użytkownikowi grę, gdy ten w nie zwyczajnie wejdzie.

Następuje wtedy zatrzymanie gry z ukazaniem się napisu „Gra skończona”. Użytkownik w tym momencie zostaje przeniesiony do Menu Głównego.

W Menu Głównym gracz ma do wyboru – wybrać poziom, który już odblokował i może go powtórzyć, zresetować poziomy w celu rozpoczęcia gry od nowa lub wyjść z aplikacji kończąc tym samym rozgrywkę.

5. Niuanse w kodzie

Pierwszy kod zastosowany w grze znajduje się w menu startowym. Odpowiada on za przyciski oraz zmianę sceny. Po kliknięciu „RESET POZIOMÓW” nasza gra zostanie przywrócona do stanu pierwotnego i będziemy zmuszeni po raz kolejny przechodzić poziom, żeby odblokować kolejny.

```
4   using UnityEngine.UI;
5   using UnityEngine.SceneManagement;
6
7   public class MainManuControlScript : MonoBehaviour {
8
9       public Button level02Button, level03Button;
10      int levelPassed;
11
12      // Use this for initialization
13      void Start () {
14          levelPassed = PlayerPrefs.GetInt ("LevelPassed");
15          level02Button.interactable = false;
16          level03Button.interactable = false;
17
18          switch (levelPassed) {
19          case 1:
20              level02Button.interactable = true;
21              break;
22          case 2:
23              level02Button.interactable = true;
24              level03Button.interactable = true;
25              break;
26          }
27      }
28
29      public void levelToLoad (int level)
30      {
31          SceneManager.LoadScene (level);
32      }
33
34      public void resetPlayerPrefs()
35      {
36          level02Button.interactable = false;
37          level03Button.interactable = false;
38          PlayerPrefs.DeleteAll ();
39      }
40  }
41
```

Przechodzimy do kolejnych scen, które zawierają rozgrywkę. W niej znajduje się kod odpowiadający praktycznie za wszystko. Jednym z bardziej zaawansowanych kodów był kod odpowiadający za ruch postaci. Kod został zaczerpnięty z kanał youtubera o nazwie „Brackeys”.

```
using UnityEngine;
using UnityEngine.Events;

public class CharacterController2D : MonoBehaviour
{
    [SerializeField] private float m_JumpForce = 400f;                          // Amount of force added when the player jumps.
    [Range(0, 1)] [SerializeField] private float m_CrouchSpeed = .36f;          // Amount of maxSpeed applied to crouching movement. 1 = 100%
    [Range(0, .3f)] [SerializeField] private float m_MovementSmoothing = .05f;  // How much to smooth out the movement
    [SerializeField] private bool m_AirControl = false;                         // Whether or not a player can steer while jumping;
    [SerializeField] private LayerMask m_WhatIsGround;                          // A mask determining what is ground to the character
    [SerializeField] private Transform m_GroundCheck;                           // A position marking where to check if the player is grounded.
    [SerializeField] private Transform m_CeilingCheck;                          // A position marking where to check for ceilings
    [SerializeField] private Collider2D m_CrouchDisableCollider;                // A collider that will be disabled when crouching

    const float k_GroundedRadius = .2f; // Radius of the overlap circle to determine if grounded
    private bool m_Grounded;            // Whether or not the player is grounded.
    const float k_CeilingRadius = .2f; // Radius of the overlap circle to determine if the player can stand up
    private Rigidbody2D m_Rigidbody2D;
    private bool m_FacingRight = true;  // For determining which way the player is currently facing.
    private Vector3 m_Velocity = Vector3.zero;

    [Header("Events")]
    [Space]

    public UnityEvent OnLandEvent;

    [System.Serializable]
    public class BoolEvent : UnityEvent<bool> { }

    public BoolEvent OnCrouchEvent;
    private bool m_wasCrouching = false;

    private void Awake()
    {
        m_Rigidbody2D = GetComponent<Rigidbody2D>();

        if (OnLandEvent == null)
            OnLandEvent = new UnityEvent();

        if (OnCrouchEvent == null)
            OnCrouchEvent = new BoolEvent();
    }

    private void FixedUpdate()
    {
        bool wasGrounded = m_Grounded;
        m_Grounded = false;

        // The player is grounded if a circlecast to the groundcheck position hits anything designated as ground
        // This can be done using layers instead but Sample Assets will not overwrite your project settings.
        Collider2D[] colliders = Physics2D.OverlapCircleAll(m_GroundCheck.position, k_GroundedRadius, m_WhatIsGround);
        for (int i = 0; i < colliders.Length; i++)
        {
            if (colliders[i].gameObject != gameObject)
            {
                m_Grounded = true;
                if (!wasGrounded)
                    OnLandEvent.Invoke();
            }
        }
    }


    public void Move(float move, bool crouch, bool jump)
    {
        // If crouching, check to see if the character can stand up
        if (!crouch)
        {
            // If the character has a ceiling preventing them from standing up, keep them crouching
            if (Physics2D.OverlapCircle(m_CeilingCheck.position, k_CeilingRadius, m_WhatIsGround))
            {
                crouch = true;
            }
        }

        //only control the player if grounded or airControl is turned on
        if (m_Grounded || m_AirControl)
        {

            // If crouching
            if (crouch)
            {
                if (!m_wasCrouching)
                {
                    m_wasCrouching = true;
                    OnCrouchEvent.Invoke(true);
                }

                // Reduce the speed by the crouchSpeed multiplier
                move *= m_CrouchSpeed;

                // Disable one of the colliders when crouching
                if (m_CrouchDisableCollider != null)
                    m_CrouchDisableCollider.enabled = false;
            } else
            {
                // Enable the collider when not crouching
                if (m_CrouchDisableCollider != null)
                    m_CrouchDisableCollider.enabled = true;

                if (m_wasCrouching)
                {
                    m_wasCrouching = false;
                    OnCrouchEvent.Invoke(false);
                }
```

```
            }
        }

        // Move the character by finding the target velocity
        Vector3 targetVelocity = new Vector2(move * 10f, m_Rigidbody2D.velocity.y);
        // And then smoothing it out and applying it to the character
        m_Rigidbody2D.velocity = Vector3.SmoothDamp(m_Rigidbody2D.velocity, targetVelocity, ref m_Velocity, m_MovementSmoothing);

        // If the input is moving the player right and the player is facing left...
        if (move > 0 && !m_FacingRight)
        {
            // ... flip the player.
            Flip();
        }
        // Otherwise if the input is moving the player left and the player is facing right...
        else if (move < 0 && m_FacingRight)
        {
            // ... flip the player.
            Flip();
        }
    }
    // If the player should jump...
    if (m_Grounded && jump)
    {
        // Add a vertical force to the player.
        m_Grounded = false;
        m_Rigidbody2D.AddForce(new Vector2(0f, m_JumpForce));
    }
}


private void Flip()
{
    // Switch the way the player is labelled as facing.
    m_FacingRight = !m_FacingRight;

    transform.Rotate(0f, 180f, 0f);
}
}

```

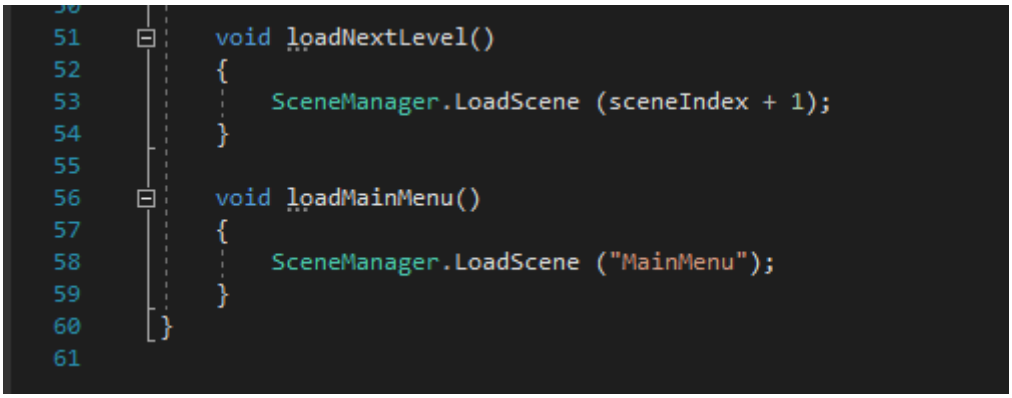
Poza samym kodem odpowiadającym za ruch, skok, oraz strzał postaci, potrzebowaliśmy również przypisać odpowiednią siłę oraz ruchy do joysticka. Kod również został zaczerpnięty z w.w. kanału. Wszelki ruch zostaje przypisany do ruchów joysticka. W zależności od przeciągnięcia kulki postać ma odpowiednio reagować.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour {

    public CharacterController2D controller;
    public Animator animator;

    public Joystick joystick;


    public float runSpeed = 40f;

    float horizontalMove = 0f;
    bool jump = false;
    bool crouch = false;

    // Update is called once per frame
    void Update () {

        if (joystick.Horizontal >= .2f)
        {
            horizontalMove = runSpeed;

        } else if (joystick.Horizontal <= -.2f)
        {
            horizontalMove = -runSpeed;
        }
        else
        {

            horizontalMove = 0f;
        }

        float verticalMove = joystick.Vertical;

        animator.SetFloat("Speed", Mathf.Abs(horizontalMove));

        if (verticalMove >= .5f)
        {
            jump = true;
            animator.SetBool("IsJumping", true);
        }

        if (verticalMove <= -.5f)
        {
            crouch = true;
        } else
        {
            crouch = false;
        }

```

```
55
56      public void OnLanding ()
57      {
58          animator.SetBool("IsJumping", false);
59      }
60
61      public void OnCrouching (bool isCrouching)
62      {
63          animator.SetBool("IsCrouching", isCrouching);
64      }
65
66      void FixedUpdate ()
67      {
68          // Move our character
69          controller.Move(horizontalMove * Time.fixedDeltaTime, crouch, jump);
70          jump = false;
71      }
72  }
73
```

Kolejny skrypt zawiera kilka ważnych fukcji. Odpowiada on za resetowanie poziom, załadowywanie kolejnej sceny, w razie zwycięstwa oraz wyświetla wszelkie informacje jak wygrana, w momencie dotknięcia monety na końcu mapy, lub przegrana, gdy dotkniemy przeciwnika.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.UI;
5   using UnityEngine.SceneManagement;
6
7   public class LevelControlScript : MonoBehaviour {
8
9       public static LevelControlScript instance = null;
10      GameObject levelSign, gameOverText, youWinText;
11      int sceneIndex, levelPassed;
12
13      // Use this for initialization
14      void Start () {
15
16          if (instance == null)
17              instance = this;
18          else if (instance != this)
19              Destroy (gameObject);
20
21          levelSign = GameObject.Find ("LevelNumber");
22          gameOverText = GameObject.Find ("GameOverText");
23          youWinText = GameObject.Find ("YouWinText");
24          gameOverText.gameObject.SetActive (false);
25          youWinText.gameObject.SetActive (false);
26
27          sceneIndex = SceneManager.GetActiveScene ().buildIndex;
28          levelPassed = PlayerPrefs.GetInt ("LevelPassed");
29      }
30
31      public void youWin()
32      {
33          if (sceneIndex == 3)
34              Invoke ("loadMainMenu", 1f);
35          else {
36              if (levelPassed < sceneIndex)
37                  PlayerPrefs.SetInt ("LevelPassed", sceneIndex);
38              levelSign.gameObject.SetActive (false);
39              youWinText.gameObject.SetActive (true);
40              Invoke ("loadNextLevel", 1f);
41          }
42      }
43
44      public void youLose()
45      {
46          levelSign.gameObject.SetActive (false);
47          gameOverText.gameObject.SetActive (true);
48          Invoke ("loadMainMenu", 1f);
49      }
50
```



```
51      void loadNextLevel()
52      {
53          SceneManager.LoadScene (sceneIndex + 1);
54      }
55
56      void loadMainMenu()
57      {
58          SceneManager.LoadScene ("MainMenu");
59      }
60  }
61
```

Są to ciekawsze kody zastosowane w naszej aplikacji. Resztę kodu można znaleźć w folderze Scripts znajdującym się w folderze Assets. Są to zazwyczaj krótkie kody odpowiadające za pojedyńcze czynności.