Colegiul Național „ Vasile Lucaciu“

Baia Mare

Atestat la informatică

ROLI’S ADVENTURE

Îndrumător:

Prof. Filip Adela

Prof. Diaconescu Adriana

Autor:

Bozîntan Carla

CUPRINS

INTRODUCERE

ROLI'S ADVENTURE este un joc 2D adaptat pe baza faimosului Angry Birds, o franciză media finlandeză bazată pe acțiunea creată de Rovio Entertainment. Seria de jocuri se concentrează pe mai multe păsări la care se face referire cu același nume care încearcă să-și salveze ouăle de porcii de culoare verde.

În această adaptare, protagonistul, pasărea Roli, se află într-o misiune ce presupune distrugerea tuturor inamicilor, monștrii de culoare mov, care au pătruns în Universul său. Jocul conține opt nivele, fiecare monstru valorând zece puncte, care vor fi acordate jucătorului la finalul fiecărui nivel.

Jocul este realizat cu ajutorul aplicațiilor: Unity, versiunea 2020.3.0 și cu ajutorul mediului de programare Visual Studio Community 2019.

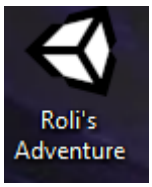
Unity este un motor de joc multiplatform, smartphone, computer, console video și jocuri web. Software-ul are particularitatea de a folosi codul (C #) pe „ NET " , platforma cu Mono implementare. Editorul său se baza anterior pe MonoDevelop, prin MonoDevelop-Unity, dar începând cu versiunea 2018.1 se bazează pe Visual Studio Community .

Visual Studio Community este un mediu de dezvoltare, bogat în aplicații, utilizat pentru a dezvolta programe de calculator pentru sisteme de operare, site-uri web, aplicații, servicii și aplicații mobile.

Aceste limbaje permit beneficierea de caracteristicile .NET Framework care oferă acces la tehnologii cheie care simplifică dezvoltarea de aplicații web ASP și XML Web Services cu Visual Web Developer.

UTILIZAREA JOCULUI

Pentru lansarea jocului este nevoie de existența celor două aplicații pe calculatorul respectiv: Unity și Visual Studio Community.

Se face click pe iconița 

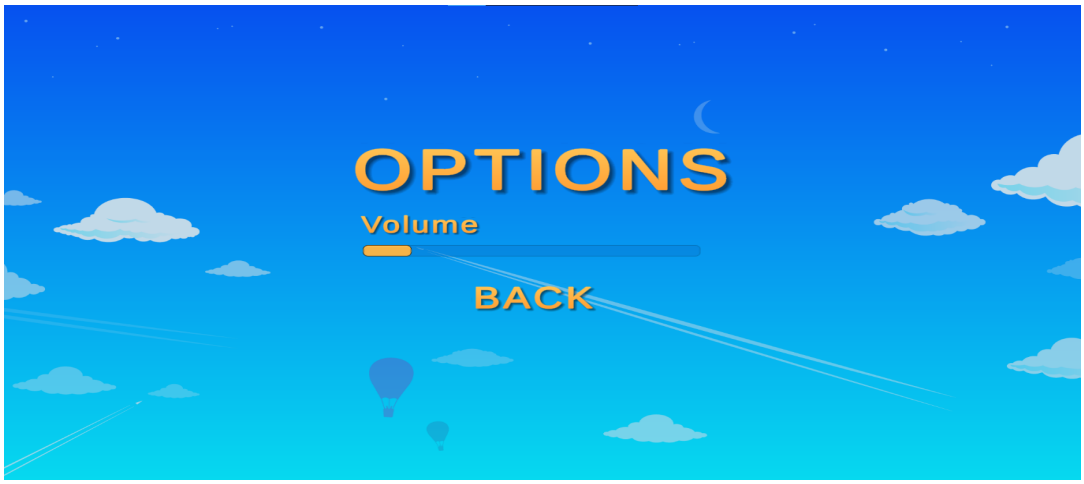
existentă pe desktop.

Meniul principal

După lansare va apărea pagina de start a proiectului care conține butoanele:

- PLAY- direcționarea spre primul nivel
- QUIT- ieșirea din aplicație

- OPTIONS- va deschide o nouă pagină care va permite reglarea volumului sunetului jocului, iar butonul BACK va face redirectionarea spre meniul principal



Parcurgerea nivelelor

În urma selectării butonului PLAY se va face redirecționarea spre primul nivel.

1. Deplasarea păsării

Pentru mișcarea păsării se plasează cursorul mouse-ului pe pasăre și se menține apăsat în timpul tragerii spre stânga, în momentul pregătirii pentru lansare.

După găsirea poziției dorite pentru lansare se eliberează pasărea și se urmărește impactul pe care îl are asupra obstacolelor și asupra monștrilor.

2. Distrugerea monștrilor

Distrugerea oricărui monstru se poate face prin următoarele trei modalități:

- Prin intermediul impactului cauzat de lovitura păsării;
- Prin intermediul loviturii cauzate de unul dintre obstacole;
- Prin intermediul ciocnirii cu unul dintre ceilalți monștrii.

3. Scopul fiecărui nivel

Scopul existent valabil pentru fiecare nivel este de a obține scorul maxim care variază în funcție de nivel. Acest scor se calculează în funcție de numărul de monștrii distruși, astfel se acordă zece punce pentru fiecare monstru. În momentul în care toți monștrii au fost distruși, adică scorul va fi maxim, se va trece automat la nivelul următor.

În cazul din imagine nu se va trece la nivelul următor.

4. Resetarea nivelului

În momentul în care se dorește resetarea nivelului din cauza pozițiilor neconvenabile a monștrilor sau a cutiilor, se face click pe pasăre și se lansează în afara limitelor impuse jocului:

- Deplasarea exagerată în partea de sus a ecranului;
- Deplasarea exagerată în partea dreaptă a ecranului.

Finalul jocului

În urma parcurgerii tuturor celor opt nivele existente va apărea meniul de Game Over, care conține două butoane:

- Restart- butonul va permite redirecționatrea spre primul nivel al jocului;
- Main Menu- butonul va permite redirecționarea spre meniul principal al jocului.

COD SURSĂ

❖ Cod sursă asociat păsării

1. Declararea variabilelor necesare

```
[SerializeField] float _fortaLansare = 300;//introducem un câmp in meniul unity pentru a putea modifica forța de lansare;
[SerializeField] float _maxDistantaTragere = 5;//introducem un câmp in meniul unity care va reține valoarea maxima a distanței de lansare prin tragere a pasarii;
Vector2 _pozitiaStart;//declarăm variabila în care reținem poziția inițială a păsării
    Rigidbody2D _rigidbody2D;//declarăm variabila în care vom reține Rigidbody2D pentru a da eficiență algoritmului
SpriteRenderer _spriteRenderer;//declaram variabila in care vom reține SpriteRenderer pentru a da eficiență  algoritmului
```

2. Funcția Awake

Prin intermediul acestei funcții se apelează toate componentele declarate, exact in momentul începerii jocului.

```
 void Awake()
{
   _pozitiaStart = transform.position;//salvăm poziția inițială  a păsării
   _rigidbody2D = GetComponent<Rigidbody2D>();//salvăm pentru a da eficiență algoritmului
```

```
    _spriteRenderer = GetComponent<SpriteRenderer>();//salvăm
pentru a da eficiență algoritmului
  }
```

3. Setarea limitelor cadrului jocului

Pentru această acțiune se va folosi funcția Update care v-a verifica poziția păsării în reperul cartezian xOy. Dacă pasărea se află în afara coordonatelor stabilite, se va reseta scena curentă.

```
Void  Update()
  {
    if(transform.position.y > 10)//dacă a întrecut limitele pe axa y
    {
      string numeScenaCurenta =
SceneManager.GetActiveScene().name;//declarăm numele scenei
curente într-o variabila
      SceneManager.LoadScene(numeScenaCurenta);//preluăm
numele scenei curente și o reîncărcăm
    }

    if (transform.position.x > 30)//dacă a întrecut limitele pe axa x
    {
      string numeScenaCurenta =
SceneManager.GetActiveScene().name;//declarăm numele scenei
curente într-o variabila
      SceneManager.LoadScene(numeScenaCurenta);//preluăm
numele scenei curente și o reîncărcăm
    }

  }
```

4. Funcția Start

Prin intermediul acestei funcții se va calcula poziția inițială a păsării în momentul inițierii jocului, stabilindu-se totodată controlul doar prin cod asupra păsării, nu și prin obiecte fizice.

```
void Start()
  {
    _pozitiaStart = _rigidbody2D.position;//îi dăm o valoare poziției inițiale în care se află pasărea
    _rigidbody2D.isKinematic = true;//controlăm doar prin cod,  nu prin obiecte fizice
  }
```

5. Schimabarea culorii păsării in momentul selectării

Se va folosi funcția OnMouseDown care se apelează in momentul selectării păsării.

```
void OnMouseDown()
{
    _spriteRenderer.color = Color.red;
}
```

6. Stabilirea unei direcții și calculul forței de lansare

Se folosește metoda OnMouseUp, apelată în momentul selectării și mișcării păsării.

```
void OnMouseUp()
```

```
{
    Vector2 pozitiaCurenta = _rigidbody2D.position;//salvăm poziția curentă
    Vector2 directia = _pozitiaStart - pozitiaCurenta;//scădem poziția de start din poziția curentă pentru a le pune într-o direcție
    directia.Normalize();//returnăm valoarea normală a vectorului
    _rigidbody2D.isKinematic = false;//dezactivăm controlul doar prin cod pentru a putea adăuga forța de lansare a păsării
    _rigidbody2D.AddForce(directia * _fortaLansare);//adăugăm puterea de lansare a păsării
    _spriteRenderer.color = Color.white;//schimbarea culorii în alb la momentul deselectării păsării
}
```

7. Lansarea păsării

Se va utiliza funcția OnMouseDrag în momentul tragerii cursorului după selectarea păsării. De asemenea, se setează un cadru în reperul cartezian xOy care va împiedica tragerea cursorului într-o altă direcție decât în partea stângă.

```
void OnMouseDrag()
{
    Vector3 mousePosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);//redarea
```

poziției curosurului de la mouse în spațiul lumii create din camera principală

```
Vector2 pozitiaDorita = mousePosition;//implementăm vectorul unidimensional, poziția dorită, care va reține doar coordonatele x si y;

float distance = Vector2.Distance(pozitiaDorita, _pozitiaStart);//redă valoarea în metrii a distanței dintre cei doi parametrii

if(distance>_maxDistantaTragere)//verificăm dacă distanța este prea mare

{

Vector2 direction = pozitiaDorita - _pozitiaStart;//stabilim direcția de tragere pentru lansarea păsării

direction.Normalize();

pozitiaDorita = _pozitiaStart + (direction * _maxDistantaTragere);//schimbăm poziția dorită ca să fie punctul aflat la distanța maximă de tragere pentru lansare

}

if (pozitiaDorita.x > _pozitiaStart.x)//verificăm dacă suntem pozitionați la dreapta poziției de start

pozitiaDorita.x = _pozitiaStart.x;

_rigidbody2D.position = pozitiaDorita;//setăm poziția

}
```

8. Detectarea lovirii unui obiect

Se va folosi funcția OnCollisionEnter2D pentru detectarea momentului lovirii unui obiect, iar în interiorul acesteia se va apela funcția de numărare a intervalului de timp după care pasărea să se reseteze la poziția inițială.

```
void OnCollisionEnter2D(Collision2D collision)
  {
    StartCoroutine(ResetareDupaIntarziere());//resetarea păsării la poziția inițială după un anumit interval de timp
  }
```

9. Funcția de numărare a intervalului de timp

```
 IEnumerator ResetareDupaIntarziere()
  {
    yield return new WaitForSeconds(3);//asteaptă trei secunde iar apoi rulează următoarele linii de cod
    _rigidbody2D.position = _pozitiaStart;//va reseta poziția inițială a păsării
    _rigidbody2D.isKinematic = true;//setăm pe kinematic ca sistemul să nu controleze pasărea automat, însa avem nevoie să-i setăm viteza la 0 pentru a nu se reseta la nesfârșit
    _rigidbody2D.velocity = Vector2.zero;//va opri păsarea
  }
```

❖ Cod sursă asociat monstrului

1. Declararea variabilelor necesare

```
public static event Action OnPlayerDeath;//funcție care nu va lua parametrii, creeăm un eveniment (ecranul de game over)

[SerializeField] Sprite _spriteMoarte;//vom putea adăuga animații pentru momentul distrugerii monstrului
[SerializeField] ParticleSystem _partcileSystem;//vom putea controla animațiile adăugate în momentul distrugerii monstrului
bool _aMurit;//declarăm variabila prin care verificăm daca a murit monstrul
```

2. Distrugerea monstrului

Se va utiliza OnCollisionEnter2D, prin care se va verifica distrugerea și de asemenea se va apela funcția care efectuează distrugerea în momentul impactului.

```
void OnCollisionEnter2D(Collision2D collision)
{
  if(TrebuieSaMoaraDeLaImpact(collision))//verificăm dacă monstrul moare de la impact
  {
    StartCoroutine(Moare());//apelăm IEnumerator
  }
}
```

```
bool TrebuieSaMoaraDeLaImpact(Collision2D collision)
  {
    if (_aMurit)//verificăm dacă deja a murit
    {
      return false;//nu a murit, continuă până ce moare de la un anumit impact
    }
    Pasare pasare = collision.gameObject.GetComponent<Pasare>();
    //verificăm dacă pasărea lovește
    if (pasare != null)//daca pasarea exista
      return true;//monstrul va muri de la impact
    if (collision.contacts[0].normal.y < -0.5)//verificăm dacă obiectul cade de deasupra monstrului
      return true;
    return false;
  }
```

3. Funcția de verificare a distrugerii tuturor monștrilor

Se va verifica dacă toți monștrii sunt distruși, se va număra scorul și de asemenea se va aplica animația corespunzătoare distrugerii unui monstru.

```
IEnumerator Moare()
  {
    _aMurit = true;
    Scor.scorValoare += 10;//creștem valoarea scorului după ce a fost distrus
    GetComponent<SpriteRenderer>().sprite = _spriteMoarte;//aplicăm efectul în momentul distrugerii monstrului
    _partcileSystem.Play();//vor aparea animațiile in momentul distrugerii monstrului
    yield return new WaitForSeconds(1);//se va aștepta un interval de o secundă după care se vor efectua următoarele linii de cod
    gameObject.SetActive(false);//facem ca monstrii să fie inactivi
    OnPlayerDeath?.Invoke();//invocăm evenimentul (ecranul de game over) creeat, prin ? stim ca nu este nul
  }
```

❖ Cod sursă corespunzător controlării nivelelor

1. Declararea variabilelor necesare

```
[SerializeField] string _numeNivelUrmator;//declarăm variabila în care vom reține nivelul următor
Monstru[] _monstrii;//reținem colecția de monstrii într-un vector
```

2. Reținerea numărului de monștrii corespunzător fiecărui nivel

```
void OnEnable()
{
    _monstrii = FindObjectsOfType<Monstru>();
}
```

3. Funcția de trecere la nivelul următor

```
void MergiNivelUrmator()
{
    Debug.Log("Mergi la " + _numeNivelUrmator);//trecem la nivelul următor
    SceneManager.LoadScene(_numeNivelUrmator);//încărcăm scena
}
```

4. Funcția de verificare a distrugerii tuturor monștrilor

```
bool MonstriiSuntMorti()
{
```

```
    foreach (var monstru in _monstrii)//parcurgem toți monștrii
    {
        if (monstru.gameObject.activeSelf)//verificăm dacă monștrii sunt activi
            return false;//returnăm fals, deoarece sunt distruși
    }
    return true;
  }
```

5. Funcția Update

În interiorul acestei funcții se vor apela cele doua funcții de mai sus, pentru a se face trecerea la nivelul următor.

```
void Update()
  {
    if (MonstriiSuntMorti())//verificăm dacă am distrus toți monstrii
        MergiNivelUrmator();//trece la următorul nivel
  }
```

❖ Cod sursă pentru menținerea scorului

1. *Declararea variabilelor necesare*

```
public static int scorValoare;//variabila în care se reține valoarea scorului

public Text scor;//variabila în care se reține textul scorului
```

2. Funcția Start

Se va face inițializarea variabilei în care se reține valoarea scorului cu zero și se va creea o referință scorului.

```
void Start()
  {
    scorValoare = 0;//inițializăm valoarea variabilei în care reținem scorul cu 0 de fiecare dată când începe jocul
    scor = GetComponent<Text>();//îi creeăm o referință scorului
  }
```

3. Funcția Update

```
void Update()
  {
    scor.text = "SCORE: " + scorValoare;//îi dăm valoarea scorului
  }
```

❖ Cod sursă pentru meniul principal

1. Funcția corspunzătoare butonului de PLAY

Se va utiliza o funcție declarata public, care se va apela de fiecare dată când butonul PLAY este apăsat și va direcționa spre primul nivel.

```
public void PlayGame()
  {
SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);//încărcăm nivelul următor luând indexul nivelului curent încărcat
  }
```

2. Funcția corespunzătoare butonului de QUIT

Se va utiliza o funcție declarata public, care se va apela de fiecare dată când butonul QUIT este apăsat și va ieși din joc.

```
public void QuitGame()
  {
    Application.Quit();//se va închide programul
  }
```

❖ Cod sursă corespunzător ecranului de Game Over

1. Declararea variabilelor necesare

```
public GameObject gameOverMenu;//facem o referință pentru meniu
```

2. Funcția de activare

```
private void OnEnable()//functie de activare
{
        Monstru.OnPlayerDeath += ActivareGameOverMenu;//apelăm enevimentul creat în codul pentru monstru și activăm meniul
}
public void ActivareGameOverMenu()
{
        gameOverMenu.SetActive(true);//îl activăm prin funcția SetActive
}
```

3. Funcția de dezactivare

```
    private void OnDisable()
    {
        Monstru.OnPlayerDeath -= ActivareGameOverMenu;//apelăm evenimentul creat în codul pentru monstru și dezactivam meniul
    }
```

4. Funcție corespunzătoare butonului de restart

```
    public  void RestartGame()
    {
        SceneManager.LoadScene("Nivelul1");//reîncărcăm jocul la nivelul 1
    }
```

5. Funcție corespunzătoare butonului care trimite spre meniul principal

```
    public void MergiLaMainMenu()
    {
        SceneManager.LoadScene("MainMenu");//facem trecerea spre meniul principal
    }
```

BIBLIOGRAFIE

https://pressstart.vip/assets

https://www.youtube.com/watch?v=zc8ac_qUXQY

https://www.youtube.com/watch?v=Lu76c85LhGY&t=544s

https://game.courses/birds/

https://opengameart.org/

https://www.youtube.com/watch?v=zc8ac_qUXQY&feature=youtu.be

https://www.youtube.com/watch?v=ZfRbuOCAeE8&feature=youtu.be

https://www.youtube.com/watch?v=QbqnDbexrCw&feature=youtu.be