<table>
<tr><td colspan="3">Politechnika Świętokrzyska w Kielcach
Wydział Elektroniki, Automatyki i Informatyki</td></tr>
<tr><td colspan="3">Laboratorium: Komunikacja Człowiek - Komputer</td></tr>
<tr><td>Numer laboratorium:</td><td rowspan="2">Temat:
Dokumentacja techniczna</td><td>Grupa: 3ID14A
Karol Piekarski</td></tr>
<tr><td>Ocena:</td><td>Data laboratorium:</td></tr>
</table>

1. Opis tematyki projektu

Gra polega na kierowaniu pojazdem kosmicznym i unikaniu zderzenia z asteroidami oraz statkami przeciwników. Strzelając do nich, można uzyskać punkty oraz monety, które można wykorzystywać do ulepszania borni i tarczy statku oraz do kupna nowych statków. Gra nie ma z góry określonego czasu, użytkownik gra dopóki jego statek nie zostanie zniszczony.
Po przegranej pojawia sie ekran zakończenia gry, użytkownik może zobaczyć ilość zdobytych punktów, monet, zapisać swój wynik, wrócić do menu głównego oraz ponownie uruchomić gre.
W menu głównym gracz ma możliwość zobaczenia zapisanych wyników, kupna nowych statków oraz zmiany ustawień gry.

2. Użyty język programowania, biblioteki, IDE, OS itp.

Do wykonania projektu wykorzystałem język C# oraz silnik Unity.

3. Zrzuty ekranu z przykładowym zadziałaniem.

Menu główne:

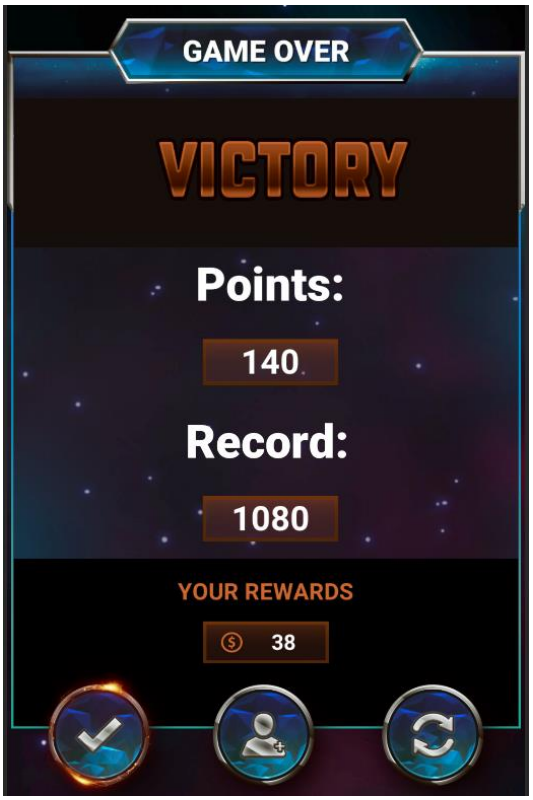
Gra:

Sklep:

Ranking punktowy:

Ulepszanie broni i tarczy:

Ekran zakończenia gry:



4. Opis użytych algorytmów i najważniejszych fragmentów implementacji.

Skrypt odpowiedzialny za poruszanie statku:

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


public class Movement_Ship : MonoBehaviour
{
       [SerializeField]
       Vector2 Area; //Wektor określający zakres poruszania statku

    [SerializeField]
       Camera Camera; //Kamera

    Rigidbody2D rgBody; // Komponent Rigidbody2D

    public Vector2 targetPosition; // Pozycja klikniecia

    // Use this for initialization
    void Start () {
              Camera = FindObjectOfType<Camera> (); //Przypianie komponentu kamery
        rgBody = GetComponent<Rigidbody2D>(); // Przypisanie komponentu Rigidbody2D
       }
```

```
    //Metoda odpowiedzialna za wizualne przedstawienie zakresu ruchu statku
       private void OnDrawGizmos()
       {
              Gizmos.DrawWireCube (Vector3.zero, Area*2f);
       }

    void Update () {

        targetPosition = (Vector2)Camera.ScreenToWorldPoint(Input.mousePosition); //
Odczyt pozycji klikniecia w ekran dotykowy

        targetPosition.x = Mathf.Clamp(targetPosition.x, -Area.x, Area.x);
//Przycinanie pozycji, jezeli gracz wyszedł za wektor x, określający zakres poruszania
statku
        targetPosition.y = Mathf.Clamp(targetPosition.y, -Area.y, Area.y) + 0.7f;
//Przycinanie pozycji, jezeli gracz wyszedł za wektor y, określający zakres poruszania
statku

        transform.position = Vector3.Lerp(transform.position, (Vector2)targetPosition,
Time.deltaTime * 7f); //Wyliczani i przypisanie pozycji statku


    }
}
```

Skrypt odpowiedzialny za wykrywanie kolizji:

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

//Automatyczne dodanie komponentów
[RequireComponent(typeof(PolygonCollider2D))]
[RequireComponent(typeof(Rigidbody2D))]
[RequireComponent(typeof(SpriteRenderer))]

public class Shipe : MonoBehaviour {

    public event System.Action ShipDestroyed; //Zdarzenie "ShipDestroyed"

    //Metoda odpowiedzialna za kolizje gracza z asteroidą oraz pociskiem przeciwnika
    private void OnTriggerEnter2D(Collider2D other)
       {
        if(other.gameObject.GetComponent<Asteroid>())
        {
            var asteroid = other.gameObject.GetComponent<Asteroid>();

            if (asteroid == null)
            {
                return;
            }
            Destroy(other.gameObject);

            if (ShipDestroyed != null)
            {
                ShipDestroyed.Invoke();
            }

        }
```

```
        if(other.gameObject.tag=="Bullet")
        {
            var bullet = other.gameObject.GetComponent<Bullet>();

            if (bullet == null)
            {
                return;
            }
            Destroy(bullet.gameObject);

            if (ShipDestroyed != null)
            {
                ShipDestroyed.Invoke();
            }

        }

    }
}
```

Skrypt odpowiedzialny za generowanie asteroid:

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

public class GenerateAsteroids : MonoBehaviour {

    [SerializeField]
    GameObject[] typeAsteroid; //Tablica obiektów(asteroid)

    [SerializeField]
    public float SpawnTime=2f; //Zmienna przechowujaca informacje co ile maja sie
pojawiać asteroidy

  [SerializeField]
    public bool Spawning=true; //Zmienna przechowujaca informacje czy generowanie
asteroid jest właczone/wyłączone

    public int AsteroidLevel{ get; set; }
    public int AsteroidRange{ get; set; }


    // Use this for initialization
    void Start () {
        AsteroidLevel = 0;
        AsteroidRange = 4;
        StartCoroutine (SpawnCoroutine ());
    }

    //Metoda odpowiedzialna za generowanie asteroid co okreslony czas
    IEnumerator SpawnCoroutine()
    {
        while (true)
        {
            while (Spawning) {
                SpawnAsteroids ();

                yield return new WaitForSeconds (SpawnTime);
            }
```

```
                        yield return new WaitForEndOfFrame ();
                }

        }

        //Metoda wybierajaca losowo z tablicy generowana asteroide
        private int RandomType()
        {
                var index = AsteroidLevel +  Random.Range (-AsteroidRange,
AsteroidRange);
                index = Mathf.Clamp (index, 0, typeAsteroid.Length-1);
         Debug.Log(index);
         return index;
        }

        //Metoda tworzaca na scenie wylosowana asteroide
        private void SpawnAsteroids()
        {
         var asteroidType = RandomType();
         var obj=Instantiate (typeAsteroid[asteroidType], transform.position,
Quaternion.identity);
                obj.transform.position += Vector3.right * Random.Range (-2f, 2f);
        }
}
```

Skrypt opisujacy asteroide:

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


//Automatyczne przypisanie komponentów
[RequireComponent(typeof(Rigidbody2D))]
[RequireComponent(typeof(SpriteRenderer))]


public class Asteroid : MonoBehaviour {

        [SerializeField]
        private float Strenght = 6f; //Zycie asteroidy

    [SerializeField]
    private int Money = 10; //Monety za zniszczoną asteroide

    [SerializeField]
    private int Points; //Punkty za zniszczoną asteroide

        [SerializeField]
        GameObject DestroyingParticles; //System czasteczek po trafieniu asteroidy

        [SerializeField]
        GameObject DestroyParticles; //System czasteczek po zniszczeniu asteroidy

        private SpriteRenderer spriteRenderer; //Grafika asteroidy


        // Use this for initialization
        void Awake () {
                spriteRenderer = GetComponent<SpriteRenderer> ();
                SpeedAsteroid ();
        }
```

```
        // Metoda odpwoiedzialna zalosowe genrowanie szybkosci porusznia sie astroidy
        private void SpeedAsteroid()
        {
         var targetSpeed = Random.Range(2f,4f);
               GetComponent<Rigidbody2D> ().velocity = Vector3.down * targetSpeed;
        }

        //Metoda odpowiedzialna za wykrywanie kolizji z pociskiem
        private void OnTriggerEnter2D(Collider2D other)
        {
               var obj=other.gameObject;
               var bullet=obj.GetComponent<Bullet>();

               if (bullet != null && bullet.tag!="Bullet") {
                      GenerateParticle (DestroyingParticles, other.transform.position);

                      Strenght-=bullet.Power;
                      Destroy (obj);

                      if (Strenght <= 0) {
                             GenerateParticle (DestroyingParticles,
other.transform.position);
                             FindObjectOfType<GameManager> ().Money += Money;
                    FindObjectOfType<GameManager>().Points += Points;
                    //FindObjectOfType<QuestCounter>().CounterDestroyedAsteroids += 1;
                             Destroy (gameObject);
                      }
               }
        }

        //Metoda odpowiedzialna za generowannie systemu czasteczek na scenie
        private void GenerateParticle(GameObject prefab, Vector3 position)
        {
               var particles=Instantiate (prefab, position, Quaternion.identity);
               particles.GetComponent<ParticleSystemRenderer> ().material.mainTexture =
spriteRenderer.sprite.texture;

        }
}
```

Skrypt opisujacy pocisk:

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

//Typ numeryczny opisujacy rodzja pocisku
public enum CannonType{Single, Double,Triple,Boss}


//Klasa opisujava pocisk
[System.Serializable]
public class BulletType
{
    public CannonType CannonType; //Typ pocisku
    public Sprite Image; //Grafika pociksu
       public float ShootDuration; //Czestotliwość strzału
       public float Speed = 20f; //Szybkosc pocisku
       public float Power=1f; // Moc pocisku
}
```

```
//Automatyczne dodanie komponentów
[RequireComponent(typeof(Rigidbody2D))]
[RequireComponent(typeof(SpriteRenderer))]
[RequireComponent(typeof(PolygonCollider2D))]

public class Bullet : MonoBehaviour {

    [SerializeField]
    public float Power = 1f;

    Rigidbody2D rgbody2D;


    //Metoda odpowiedzialna za konfigurowanie pocisku
    public void ConfigureBullet(BulletType bulletType)
    {
        Power = bulletType.Power;
        GetComponent<SpriteRenderer>().sprite = bulletType.Image;
        GetComponent<Rigidbody2D>().velocity = transform.rotation * Vector3.up *
bulletType.Speed;
    }
}
```

Skrypt opisujacy system strzelnia:

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

//Typ numeryczny okreslajacy typ statku
public enum ShipType { Red, Green, Blue, Yellow,RedUpgrade, GreenUpgrade, BlueUpgrade,
YellowUpgrade };

[RequireComponent(typeof(AudioSource))]
public class Ship_Gun : MonoBehaviour, iUpgradable
{

       [SerializeField]
       GameObject Bullet; //Poocisk

    [SerializeField]
    AudioClip ShootClip; //Dźwięk strzału

    [SerializeField]
    ShipType shipType; //Typ statku

    [SerializeField]
    BulletType[] BulletTypes; //Tablica typów pocisków

    float LastShootTime=0f; //Czas ostatniego strzału

    private AudioSource AdSource; //Komponent dźwiekowy


    //Implementacja interfesju iUpgradable
    #region iUpgradable

    public int MaxLevel
    {
        get { return BulletTypes.Length - 1; }
```

```
}

public int CurrentLevel { get; private set; }

public int UpgradeCost
{
    get { return CurrentLevel* 50 + 25; }
}

public void Upgrade()
{
    CurrentLevel += 1;
}

#endregion



   BulletType BulletType
{
    get
    {
        return BulletTypes[CurrentLevel];
    }
}


// Use this for initialization
void Start () {
    AdSource = GetComponent<AudioSource>();
    AdSource.clip = ShootClip;

}

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

          if (!Input.GetMouseButton(0))
                 return;

          if (!CanShoot ())
                 return;

                 ShotBullets ();
          LastShootTime = Time.timeSinceLevelLoad;

   }
//Metoda odpowiedzialna za strzelanie
private void ShotBullets()
{
    if (shipType == ShipType.Red)
    {
        if (BulletType.CannonType == CannonType.Single)
        {
            ShootBullet(Vector3.zero + Vector3.up * 0.6f, Vector3.zero);

            Debug.Log(Vector3.up);
        }
        else if (BulletType.CannonType == CannonType.Double)
        {
            ShootBullet(Vector3.left * 0.5f, Vector3.back * 1.7f);
            ShootBullet(Vector3.right * 0.53f, Vector3.forward * 1.7f);
```

```
            }
            /*
            else if (BulletType.CannonType == CannonType.Triple)
            {
                ShootBullet(Vector3.zero + Vector3.up * 0.6f, Vector3.zero);
                ShootBullet(Vector3.left * 0.53f, Vector3.forward * 5f);
                ShootBullet(Vector3.right * 0.53f, Vector3.back * 5f);
            }
            */
        }
        if (shipType == ShipType.Green)
        {
            if (BulletType.CannonType == CannonType.Single)
            {
                ShootBullet(Vector3.zero + Vector3.up * 0.3f, Vector3.zero);
            }
            else if (BulletType.CannonType == CannonType.Double)
            {
                ShootBullet(Vector3.zero + Vector3.left *0.07f + Vector3.up * 0.6f,
Vector3.zero);
                ShootBullet(Vector3.zero + Vector3.right *0.07f + Vector3.up * 0.6f,
Vector3.zero);
            }
            /*
            else if (BulletType.CannonType == CannonType.Triple)
            {
                ShootBullet(Vector3.zero + Vector3.up * 0.6f, Vector3.zero);
                ShootBullet(Vector3.left * 0.53f, Vector3.forward * 5f);
                ShootBullet(Vector3.right * 0.53f, Vector3.back * 5f);
            }
            */
        }

        if (shipType == ShipType.Blue)
        {
            if (BulletType.CannonType == CannonType.Double)
            {
                ShootBullet(Vector3.zero + Vector3.left * 0.21f + Vector3.up * 0.6f,
Vector3.zero);
                ShootBullet(Vector3.zero + Vector3.right * 0.21f + Vector3.up * 0.6f,
Vector3.zero);
            }
        }

        if (shipType == ShipType.Yellow)
        {
            if (BulletType.CannonType == CannonType.Single)
            {
                ShootBullet(Vector3.zero + Vector3.up * 0.4f, Vector3.zero);
            }
        }

        if (shipType == ShipType.RedUpgrade)
        {

            if (BulletType.CannonType == CannonType.Double)
            {
                ShootBullet(Vector3.zero + Vector3.left * 0.22f + Vector3.up * 0.6f,
Vector3.zero);
                ShootBullet(Vector3.zero + Vector3.right * 0.22f + Vector3.up * 0.6f,
Vector3.zero);
            }
```

```
            /*
            else if (BulletType.CannonType == CannonType.Triple)
            {
                ShootBullet(Vector3.zero + Vector3.up * 0.6f, Vector3.zero);
                ShootBullet(Vector3.left * 0.53f, Vector3.forward * 5f);
                ShootBullet(Vector3.right * 0.53f, Vector3.back * 5f);
            }
            */
        }

        if (shipType == ShipType.GreenUpgrade)
        {

            if (BulletType.CannonType == CannonType.Double)
            {
                ShootBullet(Vector3.zero + Vector3.left * 0.3f + Vector3.up * 0.6f,
Vector3.zero);
                ShootBullet(Vector3.zero + Vector3.right * 0.3f + Vector3.up * 0.6f,
Vector3.zero);
            }
            /*
            else if (BulletType.CannonType == CannonType.Triple)
            {
                ShootBullet(Vector3.zero + Vector3.up * 0.6f, Vector3.zero);
                ShootBullet(Vector3.left * 0.53f, Vector3.forward * 5f);
                ShootBullet(Vector3.right * 0.53f, Vector3.back * 5f);
            }
            */
        }

        if (shipType == ShipType.BlueUpgrade)
        {

            if (BulletType.CannonType == CannonType.Double)
            {
                ShootBullet(Vector3.zero + Vector3.left * 0.27f + Vector3.up * 0.62f,
Vector3.zero);
                ShootBullet(Vector3.zero + Vector3.right * 0.27f + Vector3.up * 0.62f,
Vector3.zero);
            }
            /*
            else if (BulletType.CannonType == CannonType.Triple)
            {
                ShootBullet(Vector3.zero + Vector3.up * 0.6f, Vector3.zero);
                ShootBullet(Vector3.left * 0.53f, Vector3.forward * 5f);
                ShootBullet(Vector3.right * 0.53f, Vector3.back * 5f);
            }
            */
        }

        if (shipType == ShipType.YellowUpgrade)
        {

            if (BulletType.CannonType == CannonType.Triple)
            {
                ShootBullet(Vector3.zero + Vector3.up * 0.65f, Vector3.zero);
                ShootBullet(Vector3.zero + Vector3.left * 0.65f + Vector3.up * 0.1f,
Vector3.zero);
                ShootBullet(Vector3.zero + Vector3.right * 0.65f + Vector3.up * 0.1f,
Vector3.zero);
            }
            /*
```

```
            else if (BulletType.CannonType == CannonType.Triple)
            {
                ShootBullet(Vector3.zero + Vector3.up * 0.6f, Vector3.zero);
                ShootBullet(Vector3.left * 0.53f, Vector3.forward * 5f);
                ShootBullet(Vector3.right * 0.53f, Vector3.back * 5f);
            }
            */
        }

        AdSource.Play();
       }

    //Metoda odpowiedzialna za tworzenie pocisku na scenie
       private void ShootBullet(Vector3 postion, Vector3 rotation)
       {
             var bullet = Instantiate (Bullet, transform.position + postion+Vector3.up
* 0.1f, Quaternion.Euler (rotation));
        Debug.Log("test" + transform.position );
             bullet.GetComponent<Bullet> ().ConfigureBullet (BulletType);
       }

    //Metoda odpowidzialna za odstep czasowy miedzy strzałami
       public bool CanShoot()
       {
             return (Time.timeSinceLevelLoad - LastShootTime >=
BulletType.ShootDuration);
       }
}
```

5. Podsumowanie

Po przeprowadzeniu testów przez niezależnych użytkowników, zostało znalezionych kilka błedów, które zostały naprawione.

Listy kontrolne oceny ergonomicznej interfejsu użytkownika wypadły bardzo dobrze. Uwagi zostały uwzględnione i wprowadzone do projektu, dzięki czemu udało się uzyskać jeszcze lepszy interfejs użytkownika.