Asteroids Game

Autor: Marcin Białecki

1. Wstęp

Pilotujesz statek kosmiczny, który właśnie wleciał w rój asteroid.
Dobrze byłoby z takiej sytuacji wyjść cało. Sprawdź swoje umiejętności i przeleć jak największą odległość.
Pamiętaj, że spotkanie z asteroidą może być bolesne!
Powodzenia!

2. Opis gry

Stworzona przeze mnie gra „Asteroids Game" polega na tym, aby pokonać jak największą odległość unikając losowo pojawiających się asteroid lecąc bojowym statkiem kosmicznym wyposażonym w 8 działek laserowych o różnej mocy, dzięki który można niszczyć pojawiające się przeszkody.

3. Sterowanie

strzałki oraz "a","w","s","d" - poruszanie statkiem
"q", "e" - spowalnianie i przyspieszanie statku
Spacja - strzelanie
"c" - zmiana widoku kamery
"Esc", "p" - pauza

4. Mechanika gry

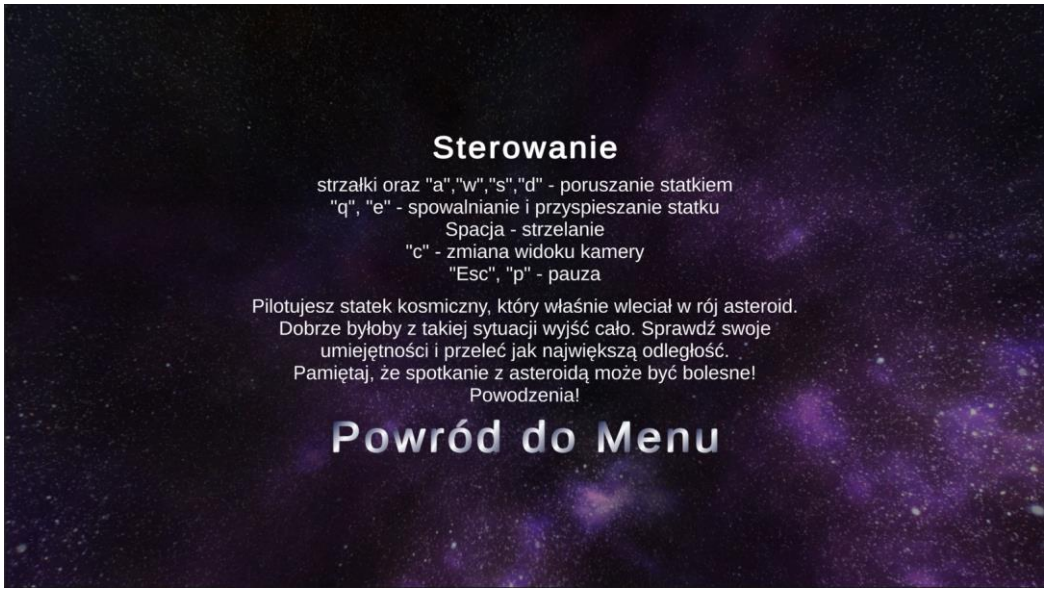
Statek kosmiczny sterowany jest za pomocą dodawania odpowiedniej siły w pionie i poziomie, aż do określonego limitu, który ogranicza szybkość przemieszczania się w tych kierunkach. Skrypt zmniejsza automatycznie prędkość gracza w pionie i poziomie tak, aby wynosiła 0. Statek może przyspieszać bez końca, choć jest wprowadzone spowalnianie statku gracza w momencie spadku klatek poniżej 30 na sekundę, w celu zapobiegnięciu przeciążeniu się gry, co za tym idzie prędkość maksymalna jest zależna od możliwości komputera, na którym jest uruchomiona. Można też spowalniać statek kosmiczny, lecz do poziomu minimalnej prędkości, która przyjmuje określoną wartość na starcie i zwiększa się stopniowo wraz

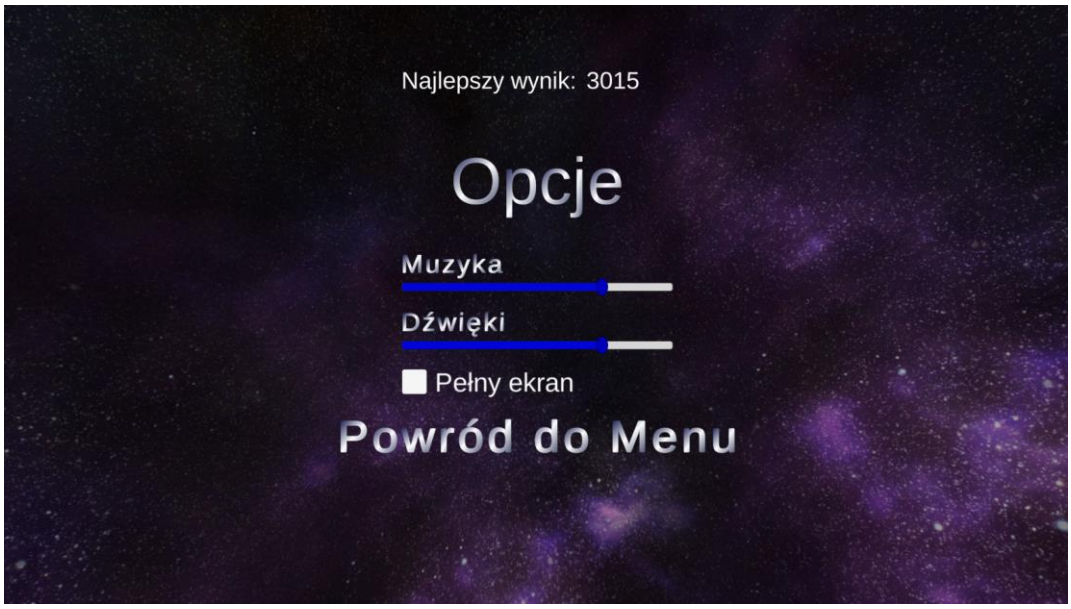
z upływem czasu w grze. Nie wzrasta ona jedynie w przypadku zmniejszenia się klatek w grze poniżej 30 lecąc przy obecnej minimalnej prędkości. W określonej minimalnej odległości przed graczem gracza tworzą się nowe asteroidy. Mają one losową wielkość, masę, szybkość poruszania się i prędkość obrotu oraz określoną maksymalną wartością, którą może przyjąć każda z tych właściwości. Bojowy statek kosmiczny, którym sterujemy wyposażony jest w 8 działek laserowych o 4 różnych wielkościach, których częstotliwość strzelania, moc i wielkość pocisków oraz szybkość ich lotu zależna jest od rozmiarów każdego z działek. Statek ma określoną swoją wytrzymałość i ilość życia w %. W przypadku zderzenia się z innym obiektem traci on określoną ilość życia zależnie od siły jaka oddziaływała na statek przy zderzeniu, przez co z czym większą prędkością lecimy, to mamy coraz mniejsze szanse na wyjście w jednym kawałku z ewentualnej kolizji. Jeśli jednak przeżyjemy bliskie spotkanie z ciałem obcym, to mamy szanse popodziwiać piękne widoki panoramy kosmicznej przełączając się w tryb widoku z pierwszej osoby za pomocą przycisku „c". Przez siły, które zadziałały na statek przy zderzeniu traci on stabilność lotu, lecz próbuje on wystabilizować lot, ale zanim mu się to uda, mija trochę czasu, przez co bardzo łatwo zderzyć się z kolejnym obiektem. Statek eksploduje, gdy jego życie spanie do zera. Efekt eksplozji posiadają także pociski laserowe, które po zderzeniu z innym ciałem uszkadzają go, a same wtedy wybuchają. W grze ukryty jest swojego rodzaju „Easter egg" w postaci 13 rodzajów pojawiających się losowo innych statków kosmicznych lecących z losowymi, często bardzo dużymi prędkościami, w różnych kierunkach. Z racji tego, że nie jest ich zbyt wiele oraz mają stosunkowo małe rozmiary w porównaniu do asteroid, to nie zawsze jest łatwo je zobaczyć. Gracz niewiedzący o tym może się zastanawiać, dlaczego co jakiś czas pojawiają się spore eksplozje jakby coś zderzyło się z asteroidą, a to inny statek lecący z dużą prędkością wpadł na jedną z nich i eksplodował, przypominając graczowi, że zapewne niebawem też czeka go bliskie spotkanie z asteroidą, ponieważ z każdą chwilą maksymalna ilość asteroid wzrasta! Oprócz efektów wizualnych dodane zostały także efekty dźwiękowe do wystrzałów z broni i eksplozji, których głośność również zależy od parametrów obiektów je emitujących oraz odległości w jakiej znajdują się od statku gracza. Dodana została również muzyka. Gra posiada także kilka menu: główne, ustawień oraz pauzy. Ustawienia wprowadzone przez gracza zapisywane są za pomocą PlayerPref, dzięki czemu po uruchomieniu gry w późniejszym czasie ustawienia są zgodne z wcześniej ustawionymi. Tym samym sposobem zapisywana jest największy osiągnięty wynik. Został także

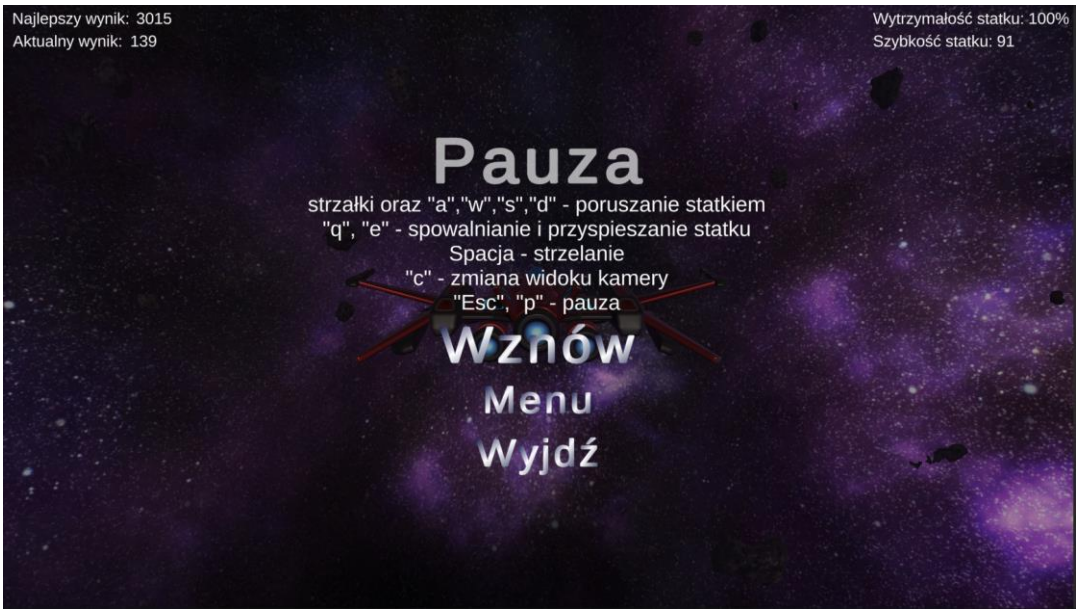
utworzony instalator, dzięki czemu w łatwiejszy sposób można udostępnić grę innym osobą.

Gra została przeze mnie samodzielnie zaprogramowana korzystając z poradników zamieszczonych na YouTube oraz dokumentacji i dyskusji na forach programistycznych, które pomogły mi zaprogramować potrzebne funkcje oraz rozwiązywać napotykane problemy. Gotowe darmowe obiekty i grafiki zostały pobrany z Asset Store.

5. Interfejs użytkownika

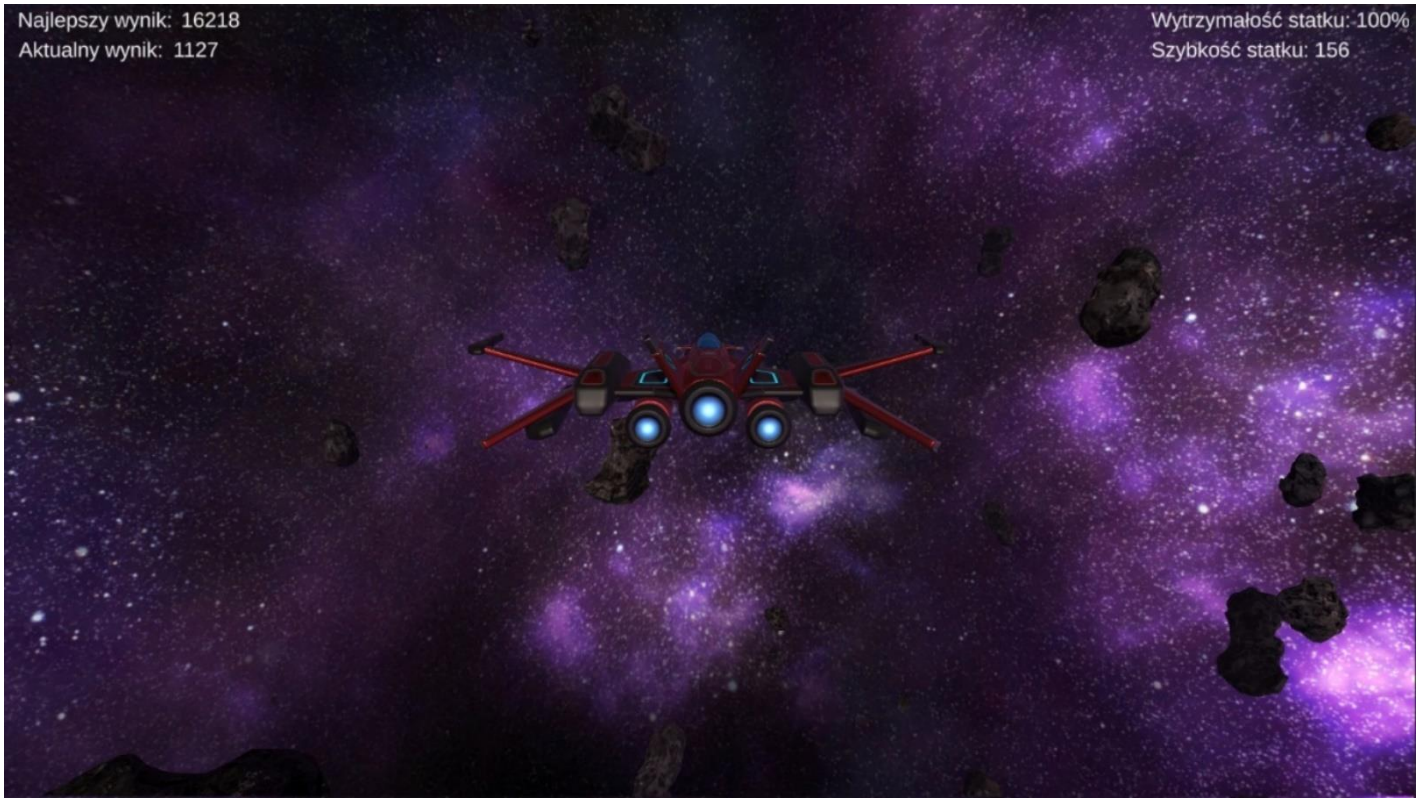


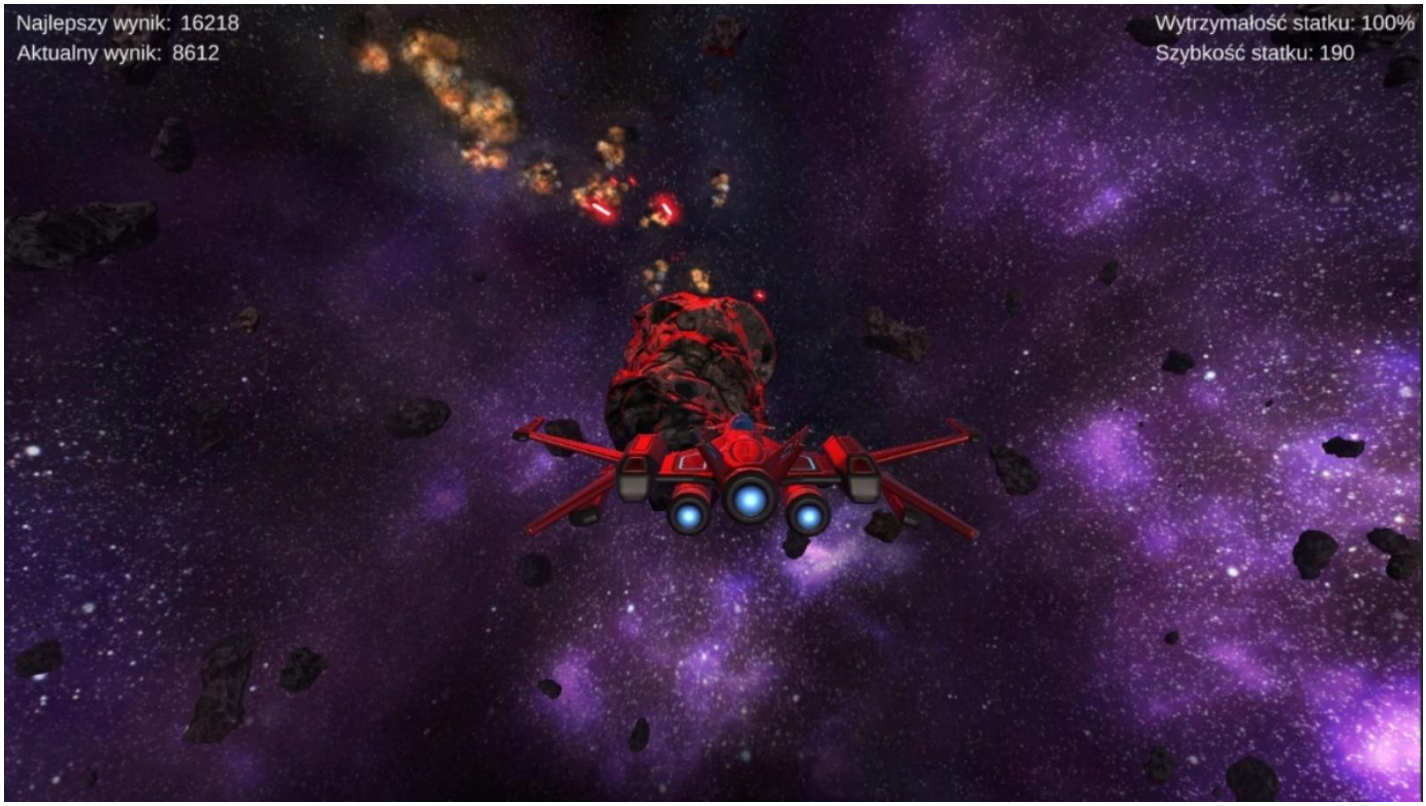

Najlepszy wynik: 3015
Opcje
Muzyka
Dźwięki
Pełny ekran
Powród do Menu

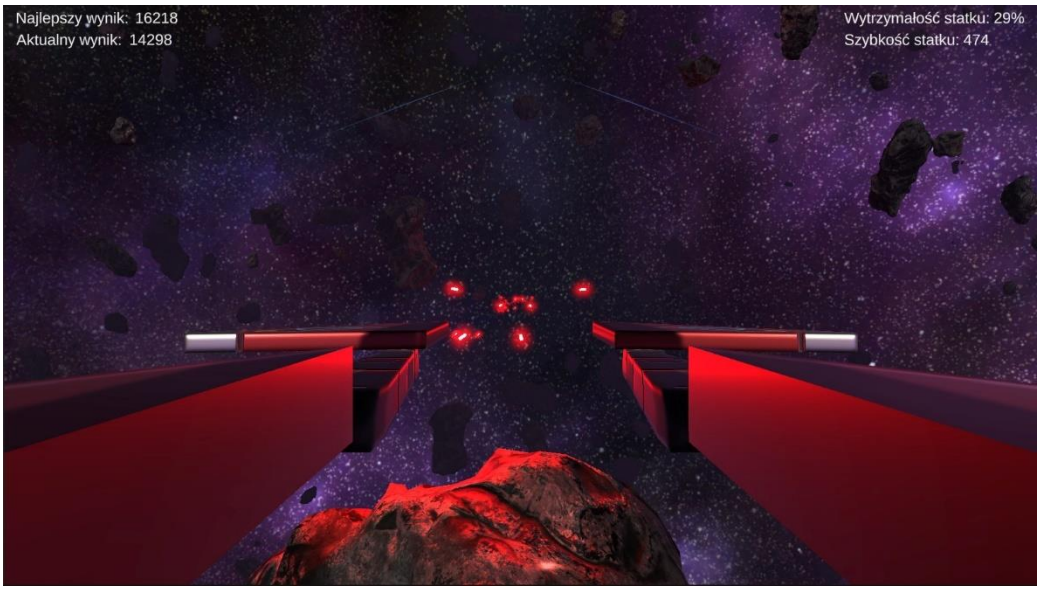

Najlepszy wynik: 3015
Aktualny wynik: 139
Wytrzymałość statku: 100%
Szybkość statku: 91
Pauza
strzałki oraz "a","w","s","d" - poruszanie statkiem
"q", "e" - spowalnianie i przyspieszanie statku
Spacja - strzelanie
"c" - zmiana widoku kamery
"Esc", "p" - pauza
Wznów
Menu
Wyjdź

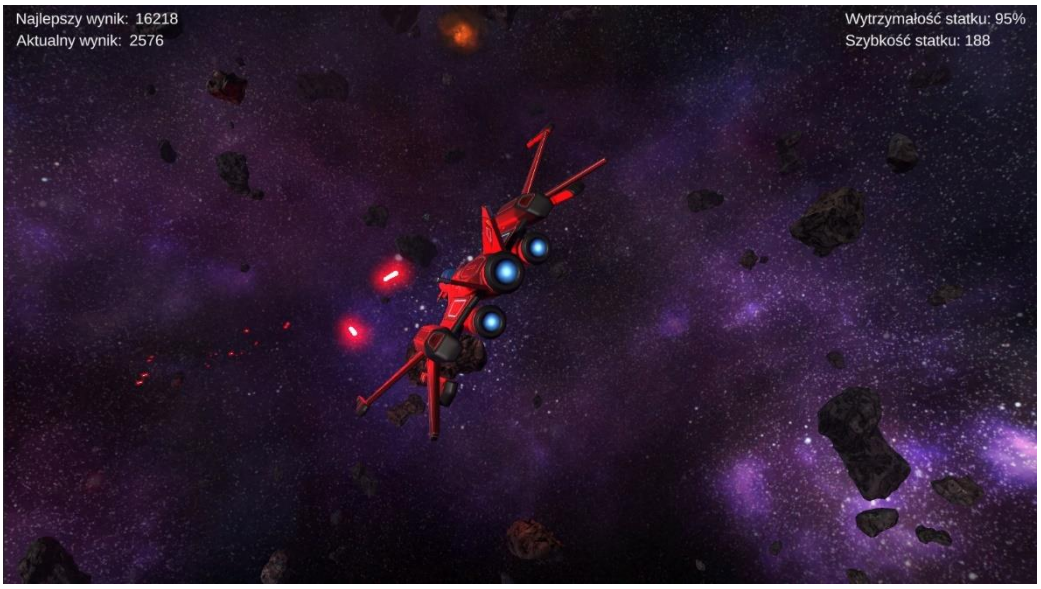

Najlepszy wynik: 3015
Aktualny wynik: 1467
Wytrzymałość statku: 0%
Szybkość statku: 0
Statek Zniszczony
Wynik: 1478
Nowa Gra
Menu
Wyjdź

6. Rozgrywka


Najlepszy wynik: 16218
Aktualny wynik: 1127
Wytrzymałość statku: 100%
Szybkość statku: 156


Najlepszy wynik: 16218
Aktualny wynik: 8612
Wytrzymałość statku: 100%
Szybkość statku: 190


Najlepszy wynik: 16218
Aktualny wynik: 14298
Wytrzymałość statku: 29%
Szybkość statku: 474


Najlepszy wynik: 16218
Aktualny wynik: 2576
Wytrzymałość statku: 95%
Szybkość statku: 188


Najlepszy wynik: 16218
Aktualny wynik: 3668
Wytrzymałość statku: 0%
Szybkość statku: 0
Statek Zniszczony
Wynik: 3669
Nowa Gra
Menu
Wyjdź

7. Skrypty

Skrypty zostały dokładnie opisane za pomocą komentarzy w kodzie oraz stosowaniu nazw zmiennych i funkcji możliwie najlepiej opisujących ich zastosowanie.

Player.cs

```
using UnityEngine;
using TMPro; //biblioteka potrzebna do uzycia TextMeshPro

public class Player : MonoBehaviour
{
    //pobieranie potrzebnych uchwytow obiektow
    public GameObject playerContainer;
    public GameObject Camera1;
    public GameObject Camera2;
    public GameObject obstacleGenerator;
    public GameObject endGamePanel;
    public GameObject explosionPrefab;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI healtText;

    //deklaracja potrzebnych zmiennych dostepnych do edycji w Unity oraz przez
zewnetrzne skrypty
    public float forwardForce = 2000f;
    public float sidewaysForce = 200f;
    public float maxSidewaysSpeed = 150f;
    public float basicMinSpeed = 150;
    public float increasingSpeed = 1;
    public float currentMinSpeed = 0;
    public float spacecraftDurability = 100;
    public int health = 100;

    public float fireRate = 1f;
    public bool fireRateScale = true;
    public GameObject[] weapons;

    private float[] nextTimeToFire;

    Rigidbody rb;

    bool lags = false; //zmienna przechowujaca to czy pomiedzy ostatnimi klatkami
pojawily sie duze opoznienia

    void Start() //funkcja wywolywana w momencie powstania obiektu, wczesniej od
funkcji Update()
    {
        //ustawianie poczatkowych wartosci zmiennych
        health = 100;
        currentMinSpeed = basicMinSpeed;

        rb = GetComponent<Rigidbody>();
        Camera2.SetActive(false); //wylacza kamere widoku statku z przodu

        nextTimeToFire = new float[weapons.Length]; //tworzy odpowiednia ilosc
zmiennych w tablicy odpowiadajacych liczbie dzialek statku kosmicznego
        for (int i = 0; i < weapons.Length; i++)
            nextTimeToFire[i] = 0; //przypisuje wartosc 0 do wszystkich powstałych
nowych zmiennych w tablicy
```

```
    }

    //funckaj FixedUpdate() wywolywana jest przy każdym przeliczaniu fizyki w grze
    //używana jest przy np. dodawaniu siły do Rigidbody
    void FixedUpdate()
    {
        //sterowanie statkiem kosmicznym w gore, dol i boki za pomoca klawiszy
"a","d","w","s" lub strzalek
        //poprzez dodawanie do Rigidbody odpowiednio skierowanej sily oddzialujacej
na statek
        if ((Input.GetKey("a") || Input.GetKey(KeyCode.LeftArrow)) && rb.velocity.x
> -maxSidewaysSpeed)
            rb.AddForce(-sidewaysForce * Time.deltaTime, 0, 0,
ForceMode.VelocityChange); // Time.deltaTime przechowuje czas jaki minal od
ostatniej klatki
        if ((Input.GetKey("d") || Input.GetKey(KeyCode.RightArrow)) && rb.velocity.x
< maxSidewaysSpeed)
            rb.AddForce(sidewaysForce * Time.deltaTime, 0, 0,
ForceMode.VelocityChange);
        if ((Input.GetKey("w") || Input.GetKey(KeyCode.UpArrow)) && rb.velocity.y <
maxSidewaysSpeed)
            rb.AddForce(0, sidewaysForce * Time.deltaTime, 0,
ForceMode.VelocityChange);
        if ((Input.GetKey("s") || Input.GetKey(KeyCode.DownArrow)) && rb.velocity.y
> -maxSidewaysSpeed)
            rb.AddForce(0, -sidewaysForce * Time.deltaTime, 0,
ForceMode.VelocityChange);

        //spowalnianie i przyspieszanie statku klawiszami "q" i "e"
        if (Input.GetKey("q") && rb.velocity.z > currentMinSpeed)
            rb.AddForce(0, 0, -forwardForce * Time.deltaTime * rb.mass);
        if (Input.GetKey("e"))
            rb.AddForce(0, 0, forwardForce * Time.deltaTime * rb.mass);

        //strzelanie po wcisnieciu spacji
        if(Input.GetKey(KeyCode.Space))
        {
            if(health > 0) //jesli gracz "zyje"
            {
                for (int i = 0; i < nextTimeToFire.Length; i++) //wykonuje dzialania
dla wszystkich dzialek statku
                    if (nextTimeToFire[i] <= Time.time) //jesli uplynela juz
odpowiednia ilosc czasu, to pozwala na wystrzelenie z dzialka
                    {
                        weapons[i].GetComponent<Shot>().Shoot(); //wystrzeliwuje
pocisk z danego dzialka

                        //jesli "fireRateScale" ustawione jest na "true", to zmienia
czestotliwosc wystrzeliwania pocisków zaleznie od wielkosci dzialka
                        //w przypadku gdy ustawione jest na "false", wszystkie
dzialka wystrzeliwuja pociski z taka sama czestotliwoscia
                        if (fireRateScale)
                            nextTimeToFire[i] = Time.time + 1f / fireRate *
Mathf.Abs(Mathf.Pow(weapons[i].transform.localScale.x, 3));
                        else
                            nextTimeToFire[i] = Time.time + 1f / fireRate;
                    }
            }
        }

        //zapobieganie przeciazeniu gry poprzez zmniejszanie predkosci gracza
```

```
        //w momencie spadku klatek poniżej 30 na sekunde (odpowiada za to zmienna
lags ustawiana wtedy na true)
        if (lags && rb.velocity.z > 0)
            rb.AddForce(0, 0, -forwardForce * Time.deltaTime * rb.mass);

        //zwiekszanie minimalnej predkosci statku kosmicznego kiedy nie ma znacznego
spadku klatek,
        //lub zwiekszanie minimalnej predkosci mimo spadku klatek kiedy gracz leci
szybciej niz minimalna predkosc
        if (!lags || currentMinSpeed + 20 < rb.velocity.z)
            currentMinSpeed += increasingSpeed * Time.deltaTime;

        //przyspieszanie tatku kiedy leci z mniejsza predkosci niz minimalna
        if (currentMinSpeed > rb.velocity.z)
            rb.AddForce(0, 0, forwardForce * Time.deltaTime * rb.mass);

        //ograniczenie predkosci lotu w pionie i poziomie do okreslonej wartosci
oraz zmiejszanie tych predkosci stopniowo do zera
        rb.AddForce(sidewaysForce * Time.deltaTime * (-rb.velocity.x /
maxSidewaysSpeed), sidewaysForce * Time.deltaTime * (-rb.velocity.y /
maxSidewaysSpeed), 0, ForceMode.VelocityChange);

        //Obraca obiekt w podanym kierunku o okreslona ilosc stopni.
        //Dziala dobrze, ale ustawiajac obiekt nie wyglada jakby dzialaly na niego
jakies sily,
        //tylko po porstu wraca na dana pozycje.
        //transform.rotation = Quaternion.RotateTowards(transform.rotation,
Quaternion.LookRotation(Vector3.forward), 1);

        //plynna stabilizacja obrotu, tak aby statek kosmiczny byl skierowany
zgodnie z wektorem (0, 0, 1)
        Vector3 spaceShipRotationVector = new Vector3(transform.rotation.x,
transform.rotation.y, transform.rotation.z);
        if (spaceShipRotationVector.magnitude > 0.001f)
            rb.angularVelocity -= spaceShipRotationVector / 50;
    }

    private void Update() //funkcja uruchamiajaca sie w kazdej klatce gry
    {
        //przelaczanie sie miedzy widokie z pierwszej i trzeciej osoby
        if (Input.GetKeyDown("c"))
        {
            if (Camera2.activeSelf)
                Camera2.SetActive(false);
            else
                Camera2.SetActive(true);
        }

        //sprawdzanie czy nienastapil spadek klatek ponizej 30 na sekunde
        if (1 / Time.deltaTime < 30)
            lags = true;
        else
            lags = false;
    }

    private void OnCollisionEnter(Collision collision) //wywolywana jest w przypadku
wystapienia kolizji z innym obiektem
    {
        if (health > 0)
        {
            //odjecie zycia graczowi po zderzeniu z innym obiektem, zaleznie od sily
jaka temu towarzyszyla oraz ustawionej wytrzymalosci statku
```

```
            health -= (int)((collision.impulse / Time.fixedDeltaTime).magnitude /
(GetComponent<Rigidbody>().mass * spacecraftDurability));
            if(health <= 0)
            {
                health = 0;
                EndGame(); //wywolanie funkcji konczoncej gre w przypadku spadku
zycia do zerowej lub mniejszej wartosci
            }
            healtText.text = health.ToString() + "%"; //wypisanie aktualnej ilosci
zycia gracza w %
        }
    }

    //funkcja konczaca gre
    private void EndGame ()
    {
        obstacleGenerator.SetActive(false); //zatrzymanie generowania nowych
obiektow
        playerContainer.SetActive(false); //ukrycie statku gracza (nie dezaktywacja
calego obiektu gracza, poniewaz posiada na sobie aktyne kamery)
        GetComponent<Rigidbody>().isKinematic = true; //wylaczenie interakcji gracza
z otoczeniem
        GameObject explosion = Instantiate(explosionPrefab, transform.position,
transform.rotation); //aktywacja eksplozji statku gracza
        Destroy(explosion, 3f); //usuniecie obiektu eksplozji gracza po 3 skundach
        endGamePanel.SetActive(true); //aktywacja menu konca gry
        scoreText.text = transform.position.z.ToString("0"); //wypisanie wyniku
gracza na ekranie

        if (PlayerPrefs.GetFloat("topScore", 0) < transform.position.z) //jesli
wynik byl wiekszy od najlepszego poprzedniego wyniku,
            PlayerPrefs.SetFloat("topScore", transform.position.z); //to zapisz go
na stale jako nowy najlepszy wynik
    }
}
```

ObstacleGenerator.cs

```
using UnityEngine;

public class ObstacleGenerator : MonoBehaviour
{
    //deklaracja potrzebnych zmiennych dostepnych do edycji w Unity oraz przez
zewnetrzne skrypty
    public GameObject player;
    public GameObject[] asteroidPrefabs;
    public GameObject[] spaceShipsPrefabs;

    //deklaracja potrzebnych zmiennych dostepnych do edycji w Unity oraz przez
zewnetrzne skrypty
    public float numberOfAsteroids = 300f;
    public float incresingNumberOfAsteroids = 2f;
    public float renderDistance = 1000f;
    public float renderSpace = 1000f;
    public float removalDistance = 1000f;
    public float asteroidMaxRandomForce = 3000f;
    public float asteroidScale = 50f;
```

```
    public float numberOfSpaceShips = 10f;
    public float spaceShipMinSpeed = 50f;
    public float spaceShipMaxSpeed = 1000f;

    //funckaj FixedUpdate() wywolywana jest przy każdym przeliczaniu fizyki w grze
    //używana jest przy np. dodawaniu siły do Rigidbody
    void FixedUpdate()
    {
        //zwiekszanie maksymalnej ilosci asteroid w danym czasie o ustalona wartosc
na sekunde
        numberOfAsteroids += incresingNumberOfAsteroids * Time.deltaTime;

        GameObject[] asteroids = GameObject.FindGameObjectsWithTag("Asteroid");
//pobieranie tablicy utworzonych asteroid

        //korekcja odleglosci w ktorej tworzone sa obiekty, tak aby nie bylo
        //nagle pojawiajacych sie obiektow obok gracza kiedy osiaga duza szybkosc
        float playerSpeedCorrection =
player.GetComponent<Rigidbody>().velocity.magnitude;

        //usuwanie asteroid bedacych daleko od gracza
        for (int i = 0; i < asteroids.Length; i++)
            if (asteroids[i].transform.position.z - player.transform.position.z < -
300 || (asteroids[i].transform.position - player.transform.position).magnitude >
(renderDistance + removalDistance + playerSpeedCorrection))
                Destroy(asteroids[i]);

        GameObject[] spaceShips = GameObject.FindGameObjectsWithTag("SpaceShip");
//pobieranie tablicy utworzonych statkow kosmicznych
        //usuwanie statkow kosmicznych bedacych daleko od gracza
        for (int j = 0; j < spaceShips.Length; j++)
            if (spaceShips[j].transform.position.z - player.transform.position.z < -
300 || (spaceShips[j].transform.position - player.transform.position).magnitude >
(renderDistance + removalDistance + playerSpeedCorrection))
                Destroy(spaceShips[j]);

        //zwiekszanie limitu ilosci asteroid i statkow tak, aby gestosc pojawiania
się ich byla podobna
        //mimo powiekszania sie przesterzeni ich tworzenia wraz ze wzrostem
predkosci gracza
        float multiplier = 1 + playerSpeedCorrection / renderDistance;

        //tworzenie nowych asteroid w odpowiedniej odleglosci od gracza oraz
losowych wartosciach masy, skali i predkosci
        GameObject asteroid;
        if (asteroids.Length < numberOfAsteroids * multiplier)
        {
            for (int i = (int)(numberOfAsteroids * multiplier) - asteroids.Length; i
> 0; i--)
            {
                asteroid = Instantiate(asteroidPrefabs[Random.Range(0,
asteroidPrefabs.Length)], new Vector3(Random.Range(-renderSpace, renderSpace),
Random.Range(-renderSpace, renderSpace), renderDistance + Random.Range(0, 500)) +
player.GetComponent<Rigidbody>().velocity + player.transform.position,
Quaternion.identity);
                float randomNumber = Random.Range(0, asteroidScale);
                asteroid.GetComponent<Rigidbody>().mass *= Mathf.Pow(randomNumber,
3);
                asteroid.transform.localScale *= randomNumber;
                asteroid.GetComponent<Rigidbody>().AddForce(Random.insideUnitSphere
* Random.Range(0, asteroidMaxRandomForce) *
asteroid.GetComponent<Rigidbody>().mass);
```

```
            }
        }

        //tworzenie nowych statkow kosmicznych w odpowiedniej odleglosci od gracza
oraz losowym zwrotem i predkoscia z okreslonego przedzialu
        GameObject spaceShip;
        if (spaceShips.Length < numberOfSpaceShips * multiplier)
        {
            for (int i = (int)(numberOfSpaceShips * multiplier) - spaceShips.Length;
i > 0; i--)
            {
                spaceShip = Instantiate(spaceShipsPrefabs[Random.Range(0,
spaceShipsPrefabs.Length)], new Vector3(Random.Range(-renderSpace, renderSpace),
Random.Range(-renderSpace, renderSpace), renderDistance + Random.Range(0, 500)) +
player.GetComponent<Rigidbody>().velocity + player.transform.position,
Random.rotation);
                spaceShip.GetComponent<Rigidbody>().velocity =
spaceShip.transform.forward * Random.Range(spaceShipMinSpeed, spaceShipMaxSpeed);
            }
        }
    }
}
```

SettingsApply.cs

```
using System; //biblioteka potrzebna do uzycia funkcji Convert.ToInt32() i
Convert.ToBoolean()
using UnityEngine;
using UnityEngine.Audio;

public class SettingsApply : MonoBehaviour
{
    //pobieranie potrzebnych uchwytow obiektow
    public AudioMixer musicAudioMixer;
    public AudioMixer soundsAudioMixer;

    //domyslne ustawienia gry
    public float defaulMusicVolume = -20f;
    public float defaulSoundsVolume = -20f;
    public bool defaulIsFullScreen = true;
    public bool cursorVisible = true;

    void Start()
    {
        //ustawienie glosnosci muzyki zgodnie z zapisanymi ustawieniami gracza przy
wykorzystaniu "PlayerPrefs"
        float musicVolume = PlayerPrefs.GetFloat("musicVolume", defaulMusicVolume);
        musicAudioMixer.SetFloat("volume", musicVolume);

        //ustawienie paska poziomu glosnosci dzwiekow w grze
        float soundsVolume = PlayerPrefs.GetFloat("soundsVolume",
defaulSoundsVolume);
        soundsAudioMixer.SetFloat("volume", soundsVolume);

        //ustawienie pelnego ekranu gry zgodnie z wczesniejszymi ustawieniami gracza
        Screen.fullScreen = Convert.ToBoolean(PlayerPrefs.GetInt("isFullScreen",
Convert.ToInt32(defaulIsFullScreen)));
```

```
        Cursor.visible = cursorVisible;
    }
}
```

Menu.cs

```
using UnityEngine;
using UnityEngine.SceneManagement; //biblioteka potrzebna do obslugi scen

public class Menu : MonoBehaviour
{
    public void PlayGame()
    {
        //zaladowuje kolejna scene zgodnie z kolejnosci w menadzerze scen, w tym
przypadku scene gry
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    public void QuitGame()
    {
        Application.Quit(); //zamyka gre
    }
}
```

OptionsMenu.cs

```
using System; //biblioteka potrzebna do uzycia funkcji Convert.ToInt32()
using UnityEngine;
using UnityEngine.Audio; //biblioteka odpowiadajaca za obsluge audio
using UnityEngine.UI; //biblioteka potrzebna do obslugi elementow interfejsu
uzytkownika takich jak "Slider" i "Toggle"

public class OptionsMenu : MonoBehaviour
{
    //pobranie potrzebnych uchwytow do obiektow
    public AudioMixer musicAudioMixer;
    public AudioMixer soundsAudioMixer;
    public GameObject sliderMusicVolume;
    public GameObject sliderSoundsVolume;
    public GameObject fullscreenToggle;
    public GameObject SettingsApply;

    private void Start()
    {
        //ustawienie paska poziomu glosnosci muzyki zgodnie z zapisanymi
ustawieniami gracza przy wykorzystaniu "PlayerPrefs"
        //w przypadku nieustawionej wartosci przyjmuje ustawiona domyslna wartosc
        float musicVolume = PlayerPrefs.GetFloat("musicVolume",
SettingsApply.GetComponent<SettingsApply>().defaulMusicVolume);
        sliderMusicVolume.GetComponent<Slider>().value = musicVolume;

        //ustawienie paska poziomu glosnosci dzwiekow w grze
        float soundsVolume = PlayerPrefs.GetFloat("soundsVolume",
SettingsApply.GetComponent<SettingsApply>().defaulSoundsVolume);
```

```
        sliderSoundsVolume.GetComponent<Slider>().value = soundsVolume;

        //ustawienie zaznaczenia przelacznika zgodnie z tym czy gra zajmuje obecnie
pelny ekran
        fullscreenToggle.GetComponent<Toggle>().isOn = Screen.fullScreen;
    }

    public void SetMusicVolume(float volume)
    {
        musicAudioMixer.SetFloat("volume", volume); //ustawia glosnosc muzyki
        PlayerPrefs.SetFloat("musicVolume", volume); //zapisuje ustawienia glosnosc
muzyki za pomoca "PlayerPrefs"
    }

    public void SetSoundsVolume(float volume)
    {
        soundsAudioMixer.SetFloat("volume", volume); //ustawia glosnosc dzwiekow w
grze
        PlayerPrefs.SetFloat("soundsVolume", volume); //zapisuje ustawienia glosnosc
dzwiekow w grze za pomoca "PlayerPrefs"
    }

    public void SetFullScreen(bool isFullScreen)
    {
        Screen.fullScreen = isFullScreen; //wlacza lub wylacza pelny ekran
        PlayerPrefs.SetInt("isFullScreen", Convert.ToInt32(isFullScreen)); //zapis
wyboru pelnego ekranu
    }
}
```

PauseMenu.cs

```
using UnityEngine;
using UnityEngine.SceneManagement; //biblioteka potrzebna do obslugi scen

public class PauseMenu : MonoBehaviour
{
    public static bool gameIsPaused = false;

    public GameObject pauseMenuUI;

    public GameObject player;

    void Update()
    {
        //nasluchuje wcisniecia klawisza "Esc" lub "p" w celu zatrzymania gry,
oprocz przypadku gdzie zycie gracza jest mniejsze lub rowne 0
        if((Input.GetKeyDown(KeyCode.Escape) || Input.GetKeyDown("p")) &&
player.GetComponent<Player>().health > 0)
        {
            if (gameIsPaused)
                Resume();
            else
                Pause();
        }
    }

    //wznawia zatrzymana gre
```

```
    public void Resume ()
    {
        pauseMenuUI.SetActive(false); //ukrywa menu pauzy
        Time.timeScale = 1f; //uruchamia ponownie czas w grze
        gameIsPaused = false; //ustawia pomocnicza zmienna
        Cursor.visible = false; //ukrywa kursor
    }

    public void Pause()
    {
        //jesli obecny wynik gracza jest wiekszy od najlepszego poprzedniego wyniku
        //to wtedy na wszelki wypadek zapisuje sie jego wartosc popzez wykorzystanie
"PlayerPrefs"
        if (PlayerPrefs.GetFloat("topScore", 0) < player.transform.position.z)
            PlayerPrefs.SetFloat("topScore", player.transform.position.z);

        pauseMenuUI.SetActive(true); //pokazanie menu pauzy
        Time.timeScale = 0f; //zatrzymanie czasu gry
        gameIsPaused = true; //ustawienie pomocniczej zmiennej
        Cursor.visible = true; //pokazuje kursor
    }

    public void LoadMenu()
    {
        Time.timeScale = 1f; //uruchamia ponownie czas w grze
        SceneManager.LoadScene("Menu"); //laduje scene z menu glownym
    }

    public void QuitGame()
    {
        Application.Quit(); //zamyka gre
    }
}
```

LaserBullet.cs

```
using UnityEngine;

public class LaserBullet : MonoBehaviour
{
    public float bulletDeleteTime = 3f; //czas po ktorym pocisk ma zniknac zostajac
usunietym

    public GameObject explosionPrefab; //gotowa eksplozja pocisku
    void Start()
    {
        Destroy(gameObject, bulletDeleteTime); //ustawia zniszczenie pocisku po
okreslonym czasie w sekundach
    }
    private void OnCollisionEnter(Collision collision)
    {
        //w zaleznosci od masy napotkanego obiektu usuwa go lub zmienisza jego skale
i mase
        if (collision.rigidbody && collision.gameObject.CompareTag("Asteroid"))
        {
            if (collision.rigidbody.mass < GetComponent<Rigidbody>().mass)
                Destroy(collision.gameObject); //usuniecie obiektu
            else
```

```
            {
                //zmniejszenie odpowiednio skali i masy trafionego obiektu
                collision.transform.localScale *=
Mathf.Pow(((collision.rigidbody.mass - GetComponent<Rigidbody>().mass) /
collision.rigidbody.mass), 3);
                collision.rigidbody.mass -= GetComponent<Rigidbody>().mass;
            }
        }

        //utworzenie wybuchu pocisku o odpowiedniej wielkosci oraz odtworzenie
dzwieku wybuchu z odpowiednia glosnoscia
        GameObject explosion = Instantiate(explosionPrefab, transform.position,
transform.rotation);
        explosion.transform.localScale *= (1 /
((Mathf.Pow(transform.localScale.magnitude, 3) * 100) + 0.1f));
        explosion.GetComponent<AudioSource>().volume =
Mathf.Clamp(Mathf.Pow(transform.localScale.magnitude, 3) * 100, 0, 1);
        Destroy(explosion, 3f); //usuniecie obiektu wybuchu po okreslonej ilosci
sekund

        Destroy(gameObject); //usuniecie pocisku
    }
}
```

Shot.cs

```
using UnityEngine;

public class Shot : MonoBehaviour
{
    //pobieranie potrzebnych uchwytow obiektow
    public GameObject bulletPrefab;
    public Rigidbody playerRigidbody;

    //deklaracja zmiennych mozliwych do edycji w Unity
    public float bulletSpeed = 250f;
    public bool bulletSpeedScale = true;
    public bool playSound = true;

    private void Awake() //funkcja wywolywana jeszcze przed funckcja Start()
    {
        //ustawianie glosnosci wystrzalu w zaleznosci od wielkosci broni
        GetComponent<AudioSource>().volume =
Mathf.Clamp(Mathf.Abs(transform.localScale.x), 0, 2) / 2;
    }

    //funckja wystrzeliwujaca pocisk
    public void Shoot()
    {
        //utworzenie pocisku
        GameObject bullet = Instantiate(bulletPrefab, transform.position +
transform.forward * 2.4f * Mathf.Abs(transform.localScale.x), transform.rotation);
        bullet.transform.localScale *= Mathf.Abs(transform.localScale.x);
//dostosowanie wielkosci pocisku
        bullet.GetComponent<Rigidbody>().mass *=
Mathf.Pow(Mathf.Abs(transform.localScale.x), 3); //dostosowanie masy pocisku
```

```
        //jesli "bulletSpeedScale" ustawione jest na "true", to zmienia predkosc
pocisku zaleznie od jego wielkosci
        //w przypadku gdy ustawione jest na "false", wszystkie pociski leca z ta
sama okreslona predkoscia
        if (bulletSpeedScale)
            bullet.GetComponent<Rigidbody>().velocity =
playerRigidbody.velocity.magnitude * transform.forward + transform.forward *
bulletSpeed * 1 / Mathf.Pow(Mathf.Abs(transform.localScale.x), 3);
        else
            bullet.GetComponent<Rigidbody>().velocity = playerRigidbody.velocity +
transform.forward * bulletSpeed;

        if(playSound)
            GetComponent<AudioSource>().Play(); //jesli dzwiek jest ustawiony na
wlaczony, to odgrywa dzwiek wystrzalu
    }
}
```

SpaceShip.cs

```
using UnityEngine;

public class SpaceShip : MonoBehaviour
{
    public GameObject explosionPrefab; //gotowa eksplozja

    //ustawienia statkow kosmicznych
    public float increasingSpeed = 1;
    public float spacecraftDurability = 100;
    public int health = 100;

    Rigidbody rb;

    // Start is called before the first frame update
    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        //zwieksza predkosc statku o okreslona wartosc na sekunde
        rb.velocity += (transform.forward * increasingSpeed * Time.deltaTime);
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (health > 0)
        {
            //zmniejsza zycie statku po zderzeniu z innym obiektem, zaleznie od sily
jaka temu towarzyszyla oraz ustawionej wytrzymalosci statku
            health -= (int)((collision.impulse / Time.fixedDeltaTime).magnitude /
(GetComponent<Rigidbody>().mass * spacecraftDurability));
            if (health <= 0)
            {
                health = 0;
```

```
                GameObject explosion = Instantiate(explosionPrefab,
transform.position, transform.rotation); //tworzy eksplozje
                Destroy(explosion, 3f); //usuwa obiekt eksplozji po 3 sekundach

                Destroy(gameObject); //usuwa statek kosmiczny
            }
        }
    }
}
```

ScoreUpdate.cs

```
using UnityEngine;
using TMPro; //biblioteka potrzebna do uzycia TextMeshPro

public class ScoreUpdate : MonoBehaviour
{
    //pobieranie potrzebnych uchwytow obiektow
    public GameObject player;
    public TextMeshProUGUI topScoreText;
    public TextMeshProUGUI currentScoreText;

    void Start()
    {
        //wyswietlenie najlepszego dotychczas osiagnietego wyniku
        topScoreText.text = PlayerPrefs.GetFloat("topScore", 0).ToString("0");
    }

    void Update()
    {
        //aktualizacja obecnego wyniku gracza, dopóki jego zycie nie jest mniejsze
lub rowne 0
        if(player.GetComponent<Player>().health > 0)
            currentScoreText.text = player.transform.position.z.ToString("0");
    }
}
```

TopScoreLoad.cs

```
using UnityEngine;
using TMPro; //biblioteka potrzebna do uzycia TextMeshPro

public class TopScoreLoad: MonoBehaviour
{
    public TextMeshProUGUI topScoreText; //pobiera potrzebny uchwyt do obiektu
przechowujacego tekst

    void Start()
    {
        //wypisuje najlepszy dotychczas osiagniety wynik w grze
        topScoreText.text = PlayerPrefs.GetFloat("topScore", 0).ToString("0");
    }
}
```

SpeedValueTextUpdate.cs

```
using UnityEngine;
using TMPro; //biblioteka potrzebna do uzycia TextMeshPro

public class SpeedValueTextUpdate : MonoBehaviour
{
    //pobieranie potrzebnych uchwytow obiektow
    public GameObject player;
    public TextMeshProUGUI speedValueText;

    void Update()
    {
        //jesli zycie gracza jest powyzej 0, to wyswietla aktualna predkosc gracza
        if (player.GetComponent<Player>().health > 0)
            speedValueText.text =
player.GetComponent<Rigidbody>().velocity.z.ToString("0");
        else
            speedValueText.text = "0"; //wyswietla predkosc 0 jesli zycie gracza
jest mniejsze lub rowne 0
    }
}
```

RandomRotation.cs

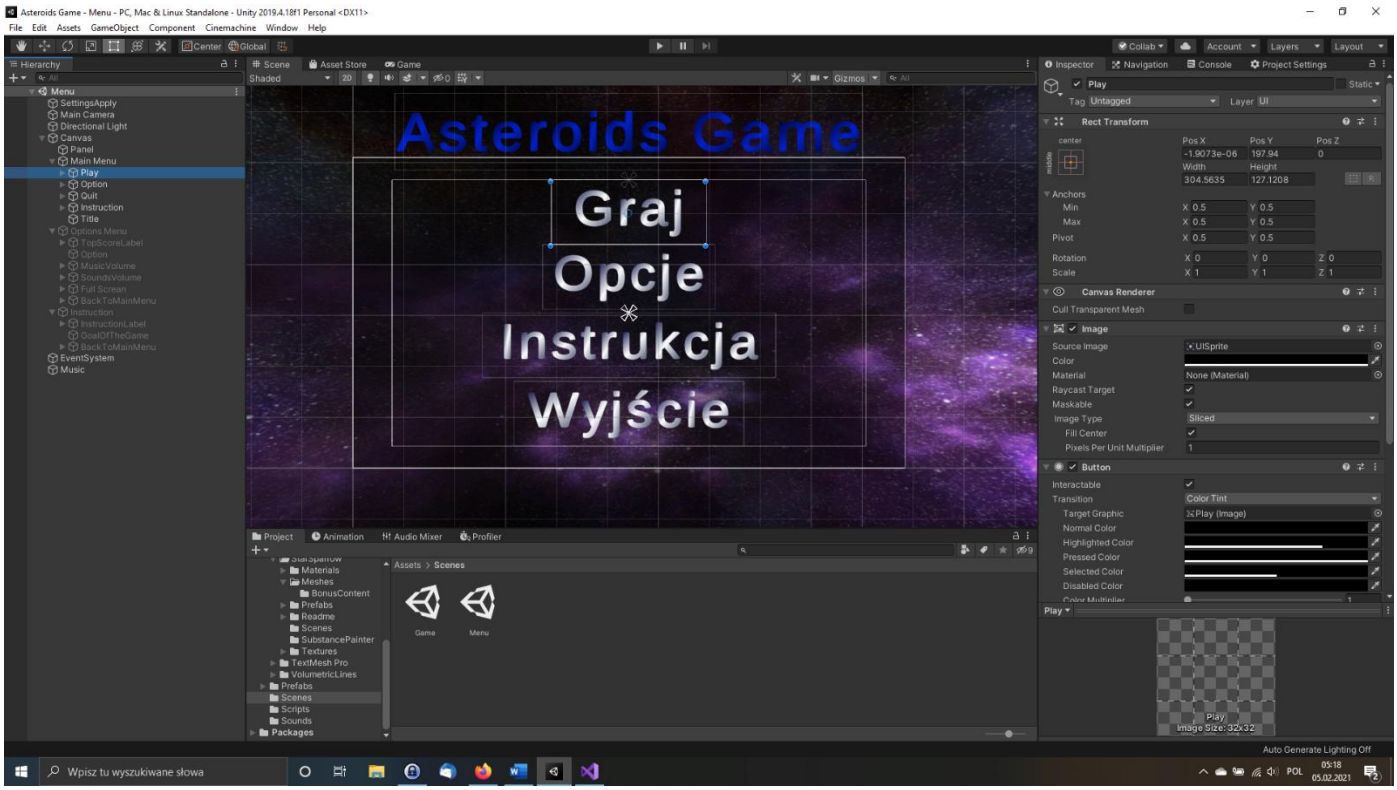
```
using UnityEngine;

public class RandomRotation : MonoBehaviour
{
    [SerializeField] //sprawia, ze zmienna moze byc edytowana w Unity mimo
deklaracji jako private
    private float rotationRangeSpeed = 5f; //maksymalna szybkosc obrotu

    void Start()
    {
        //ustawienie losowego kierunku i szybkosci obracania się obiektu
        GetComponent<Rigidbody>().angularVelocity = Random.insideUnitSphere *
rotationRangeSpeed;
    }
}
```

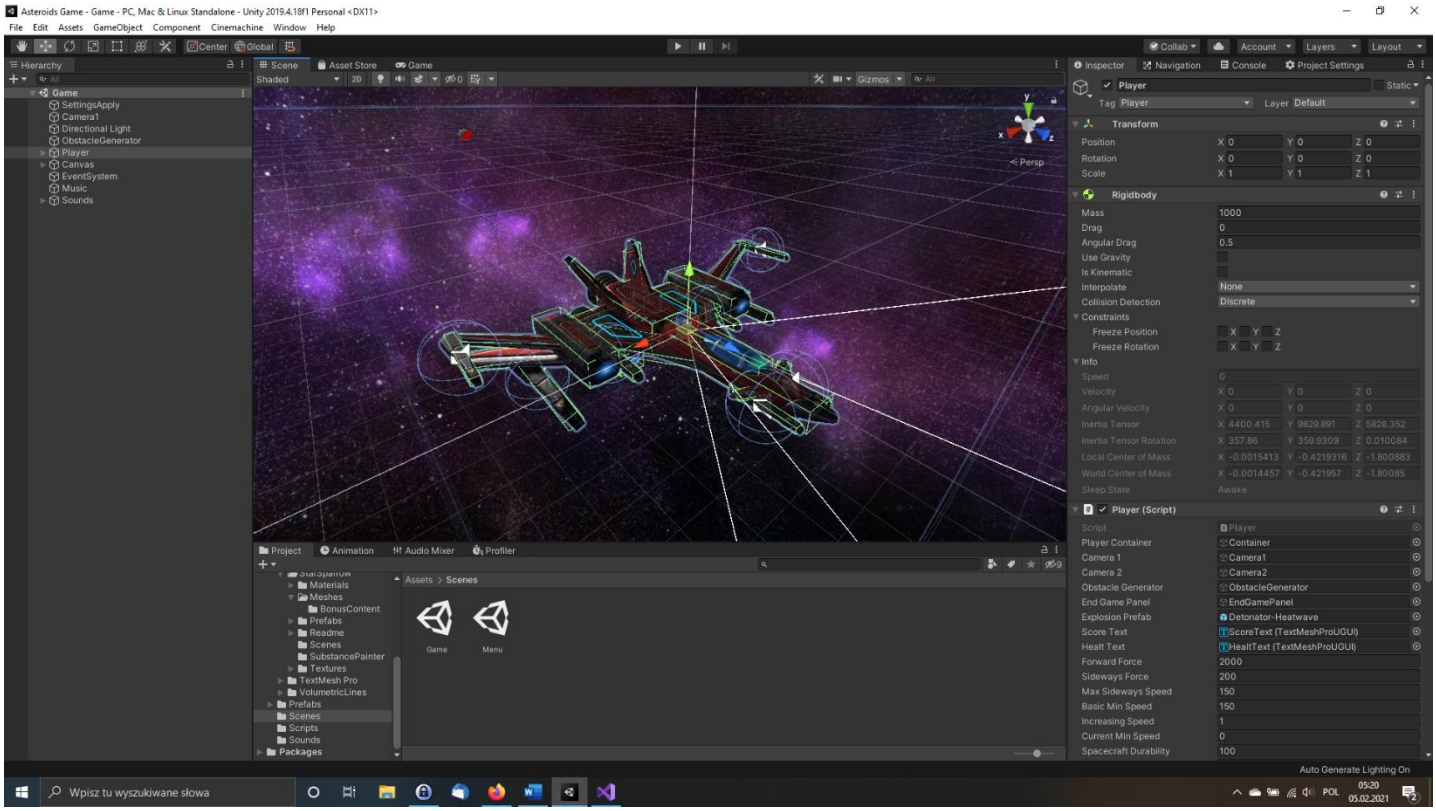
EndGame.cs

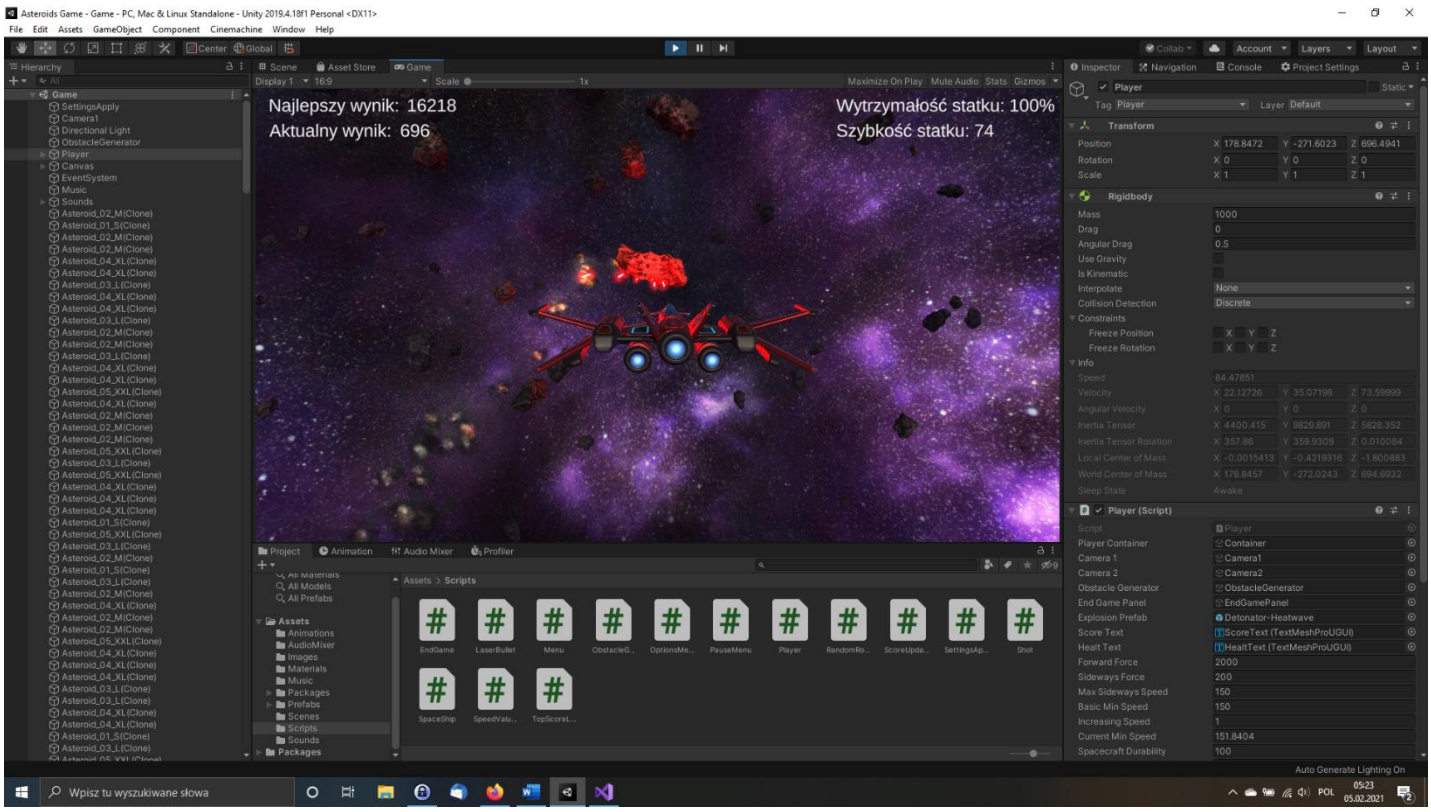
```
using UnityEngine;
using UnityEngine.SceneManagement; //biblioteka potrzebna do obslugi scen

public class EndGame : MonoBehaviour
{
    private void Start()
    {
        Cursor.visible = true; //pokazuje kursor
    }

    void Update()
```

```
    {
        //Zaczyna nowa gre po wcisnieciu kalwisza "Enter"
        if (Input.GetKeyDown(KeyCode.Return))
            NewGame();
    }

    public void NewGame()
    {
        SceneManager.LoadScene(1); //laduje ponownie scene gry, dzieki czemu gra
zaczyna sie na nowo
    }

    public void LoadMenu()
    {
        SceneManager.LoadScene("Menu"); //laduje scene z glownym menu
    }

    public void QuitGame()
    {
        Application.Quit(); //zamyka gre
    }
}
```

8. Edytor Unity




Asteroids Game - Game - PC, Mac & Linux Standalone - Unity 2019.4.18f1 Personal <DX11>
File Edit Assets GameObject Component Cinemachine Window Help
Hierarchy
Game
SettingsApply
Camera1
Directional Light
ObstacleGenerator
Player
Canvas
EventSystem
Music
Sounds
Scene
Asset Store
Game
Persp
Inspector
Navigation
Console
Project Settings
Player
Tag Player
Layer Default
Transform
Position
Rotation
Scale
Rigidbody
Mass 1000
Drag 0
Angular Drag 0.5
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position
Freeze Rotation
Info
Player (Script)
Player Container Container
Camera 1 Camera1
Camera 2 Camera2
Obstacle Generator ObstacleGenerator
End Game Panel EndGamePanel
Explosion Prefab Detonator-Heatwave
Score Text ScoreText (TextMeshProUGUI)
Healt Text HealtText (TextMeshProUGUI)
Forward Force 2000
Sideways Force 200
Max Sideways Speed 150
Basic Min Speed 150
Increasing Speed 1
Current Min Speed 0
Spacecraft Durability 100
Project
Animation
Audio Mixer
Profiler
Assets > Scenes
Game
Menu
Auto Generate Lighting On
Wpisz tu wyszukiwane słowa
POL
05-20
05.02.2021


Asteroids Game - Game - PC, Mac & Linux Standalone - Unity 2019.4.18f1 Personal <DX11>
File Edit Assets GameObject Component Cinemachine Window Help
Najlepszy wynik: 16218
Aktualny wynik: 696
Wytrzymałość statku: 100%
Szybkość statku: 74
Assets > Scripts
Auto Generate Lighting On
Wpisz tu wyszukiwane słowa
POL
05-23
05.02.2021