Московский Авиационный Институт
(Национальный исследовательский университет)

«Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа №4
по курсу «Программирование игр»

Студент: Лазаревич О.А.

Группа: М8О-108М-20

Преподаватель: Аносова Н.П.

Москва, 2022

Лабораторная работа №4 «Astr Circle»

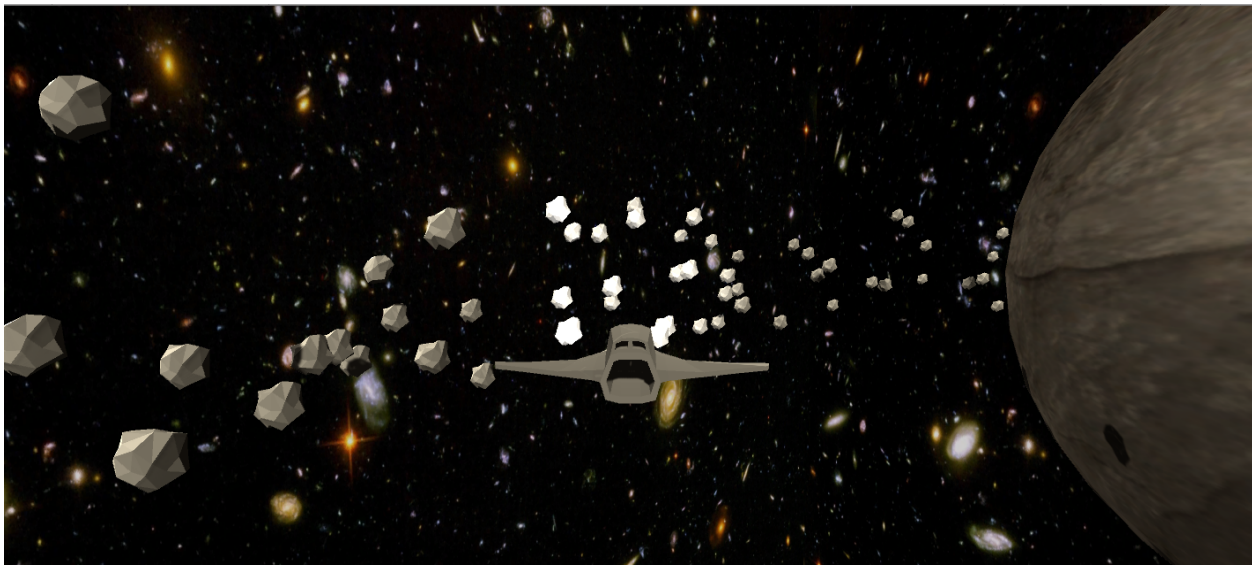
Цель работы

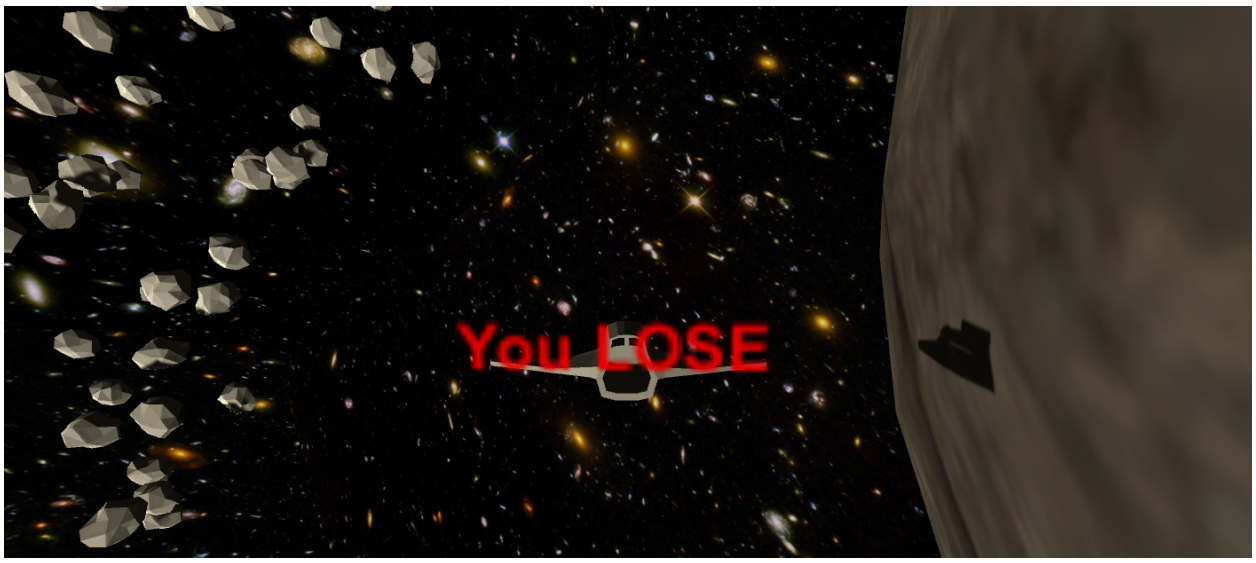
Смоделировать сцену, на которой игрок сможет двигаться по круговой траектории. Ограничить траекторию движения так чтобы движение игрока происходило только внутри этой траектории.

Задание

- Мы летим среди астероидов и уворачиваемся;
- Нужно реализовать движение по кругу с камерой, которая находится за корабликом;
- Желательно делать проверку на столкновение не со всеми астероидами.

Ход работы





Для выполнения работы был выбран движок Unity. Фон неба отличный от того что Unity нам предлагает по умолчанию. Поэтому сперва создадим новый материал, представляющий собой панорамную текстуру неба (skybox). Затем создадим планету вокруг которой будет вращаться пояс стероидов.

Далее добавим на сцену игрока. Игроком является космический корабль. Добавим два прожектор (Spot Light) спереди и два точечных источника света (Point Light). Чтобы при передвижении игрока камера следовала за ним добавим её в качестве наследника корабля.

Сцена готова. Приступим к описанию поведения действующих объектов. Всего на сцене будут принимать участие два типа действующих объектов: астероиды и космический корабль.

Определим два класса для определения поведения космического корабля. Сперва определим класс `ControlBehaviour` который будет отвечать за управление кораблем.

```
public class ControlBehaviour: MonoBehaviour {
 private Rigidbody _rb = null;
 private System.Single _speed = 0.0F;
 public System.Single _angularReaction = 10.0F;
 public System.Single _slowing = 1.5F;
 public System.Single _acceleration = 1.0F;
 public System.Single _maxSpeed = 10.0F;
 public GameObject _sadText;
```

Публичные поля класса видны из редактора Unity и их значения будут задаваться в процессе тестирования для точной настройки. Так поля `_angularReaction` и `_acceleration` задают скорость поворота и ускорения корабля. Поле `_maxSpeed` максимальную скорость которую он может развить. Поле `_slowing` задаёт замедление и является обратной по смыслу величиной поля `_acceleration`. Приватное поле `_speed` хранит текущее значение скорости корабля.

Движение корабля необходимо обрабатывать внутри метода `FixedUpdate` потому что второй скрипт тоже меняет положение корабля в пространстве, а это возможно только внутри метода `FixedUpdate`.

```
void FixedUpdate() {
   turnAround();
   moveForward();
 }
```

Метод `FixedUpdate` в зависимости от ввода пользователя сперва поворачивает корабль, затем двигает его вперёд или назад.

Поворот игрока производится внутри метода `turnAround` при помощи `Rigidbody.MoveRotation`.

```
void turnAround() {
   System.Single inpX = -1.0F * Input.GetAxis("Horizontal");
   if(inpX != 0.0F) {
     Vector3 euler = _rb.rotation.eulerAngles;
     euler.z += -1.0F * Mathf.Sign(inpX) * _angularReaction *
Time.deltaTime;
     _rb.MoveRotation(Quaternion.Euler(euler));
   }
 }
```

Метод `moveForward` двигает корабль вперёд либо назад в зависимости от предпочтений игрока при помощи метода `Rigidbody.MovePosition`.

```
void moveForward() {
   System.Single inpZ = -1.0F * Input.GetAxis("Vertical");

   if(inpZ != 0.0F) {
     _speed += 1.0F * Mathf.Sign(inpZ) * _acceleration * Time.deltaTime;
     _speed = Mathf.Clamp(_speed, -_maxSpeed, _maxSpeed);
```

```
    } else {
      _speed += 1.0F * Mathf.Sign(_speed) * _slowing * Time.deltaTime;
      _speed = Mathf.Clamp(_speed, 0.0F, _maxSpeed);
    }

    if(_speed != 0.0F && _speed < 0.0F) {
      _rb.MovePosition(_rb.position + transform.up * _speed *
Time.deltaTime);
    }
  }
```

В каждом кадре увеличиваем скорость в соответствии с заданным значением ускорения, пропорционально времени между соседними кадрами. Ограничиваем скорость на отрезке (`-_maxSpeed; _maxSpeed`).

Астероиды создаются так же, как и в 3-ей лабораторной работе за исключением, что вращением им задается с помощью метода `RotateAround`.

```
void Update(){
    transform.RotateAround(axle, Vector3.up, _angularVelocity *
Time.deltaTime);
  }
```

Выводы

В результате выполнения лабораторной работы был реализован симулятор космического корабля, который может свободно двигаться по кругу.

Листинг

```
// GlobalBehaviour.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GlobalBehaviour: MonoBehaviour {

  void Start()
  {
    StartCoroutine(create());
  }
```

```
 IEnumerator create()
 {
   while (true)
   {
     Vector3 pos = new Vector3(
       Random.Range(-_width, _width),
       Random.Range(-_height, _height),
       _distance);

     GameObject obj =
       (GameObject)Instantiate(Resources.Load("Rock"), pos,
Quaternion.identity);

     RockBehaviour rock = obj.GetComponent<RockBehaviour>();
     rock._speed = Random.Range(_speedMin, _speedMax);
     rock._rotationSpeed = Random.Range(_rotationSpeedMin,
_rotationSpeedMax);

    yield return new WaitForSeconds(0.15F);
   }
 }

 public System.Single _respawnTime = 0.15F;
 public System.Single _distance = 100.0F;
 public System.Single _speedMin = 10.0F;
 public System.Single _speedMax = 30.0F;
 public System.Single _rotationSpeedMin = 1.0F;
 public System.Single _rotationSpeedMax = 10.0F;
 public System.Single _width = 10.0F;
 public System.Single _height = 10.0F;

}

// ControlBehavour.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ControlBehaviour: MonoBehaviour {
 private Rigidbody _rb = null;
 private System.Single _speed = 0.0F;
 public System.Single _angularReaction = 10.0F;
 public System.Single _slowing = 1.5F;
```

```
 public System.Single _acceleration = 1.0F;
 public System.Single _maxSpeed = 10.0F;
 public GameObject _sadText;
 void Awake() {
   _rb = GetComponent<Rigidbody>();
   _sadText.SetActive(false);
 }

 private void OnCollisionEnter(Collision collision)
 {
   if (collision.gameObject.tag == "Enemy")
   {
     _sadText.SetActive(true);
   }
 }

 void turnAround() {
   System.Single inpX = -1.0F * Input.GetAxis("Horizontal");
   if(inpX != 0.0F) {
     Vector3 euler = _rb.rotation.eulerAngles;
     euler.z += -1.0F * Mathf.Sign(inpX) * _angularReaction *
Time.deltaTime;
     _rb.MoveRotation(Quaternion.Euler(euler));
   }
 }

 void moveForward() {
   System.Single inpZ = -1.0F * Input.GetAxis("Vertical");

   if(inpZ != 0.0F) {
     _speed += 1.0F * Mathf.Sign(inpZ) * _acceleration * Time.deltaTime;
     _speed = Mathf.Clamp(_speed, -_maxSpeed, _maxSpeed);
   } else {
     _speed += 1.0F * Mathf.Sign(_speed) * _slowing * Time.deltaTime;
     _speed = Mathf.Clamp(_speed, 0.0F, _maxSpeed);
   }

   if(_speed != 0.0F && _speed < 0.0F) {
     _rb.MovePosition(_rb.position + transform.up * _speed *
Time.deltaTime);
   }
 }
  void FixedUpdate() {
   turnAround();
   moveForward();
```

```
 }
}

// RockBehavour.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RockBehaviour: MonoBehaviour {

 void Awake() {
   _rigidbody = GetComponent<Rigidbody>();

   _eulerAngles = new Vector3(
     Random.Range(-1.0F, 1.0F),
     Random.Range(-1.0F, 1.0F),
     Random.Range(-1.0F, 1.0F)
   );
 }

 private Vector3 axle = new Vector3(100f, 100f, 60f);

 void FixedUpdate () {
   Vector3 rot = _rigidbody.rotation.eulerAngles;
   rot += _eulerAngles * _rotationSpeed * Time.deltaTime;
   _rigidbody.MoveRotation(Quaternion.Euler(rot));

 }
 void Update(){
   transform.RotateAround(axle, Vector3.up, _angularVelocity *
Time.deltaTime);
 }

 private Rigidbody _rigidbody;

 public System.Single _speed = 1.0F;
 public System.Single _rotationSpeed = 1.0F;
 public System.Single _angularVelocity = 15F;

 public System.Single _deadEnd = -10.0F;

 private Vector3 _eulerAngles;

}
```