Московский Авиационный Институт
(Национальный исследовательский университет)

«Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа №3
по курсу «Программирование игр»

Студент: Лазаревич О.А.

Группа: М8О-108М-20

Преподаватель: Аносова Н.П.

Москва, 2022

Лабораторная работа №3 «Astr»

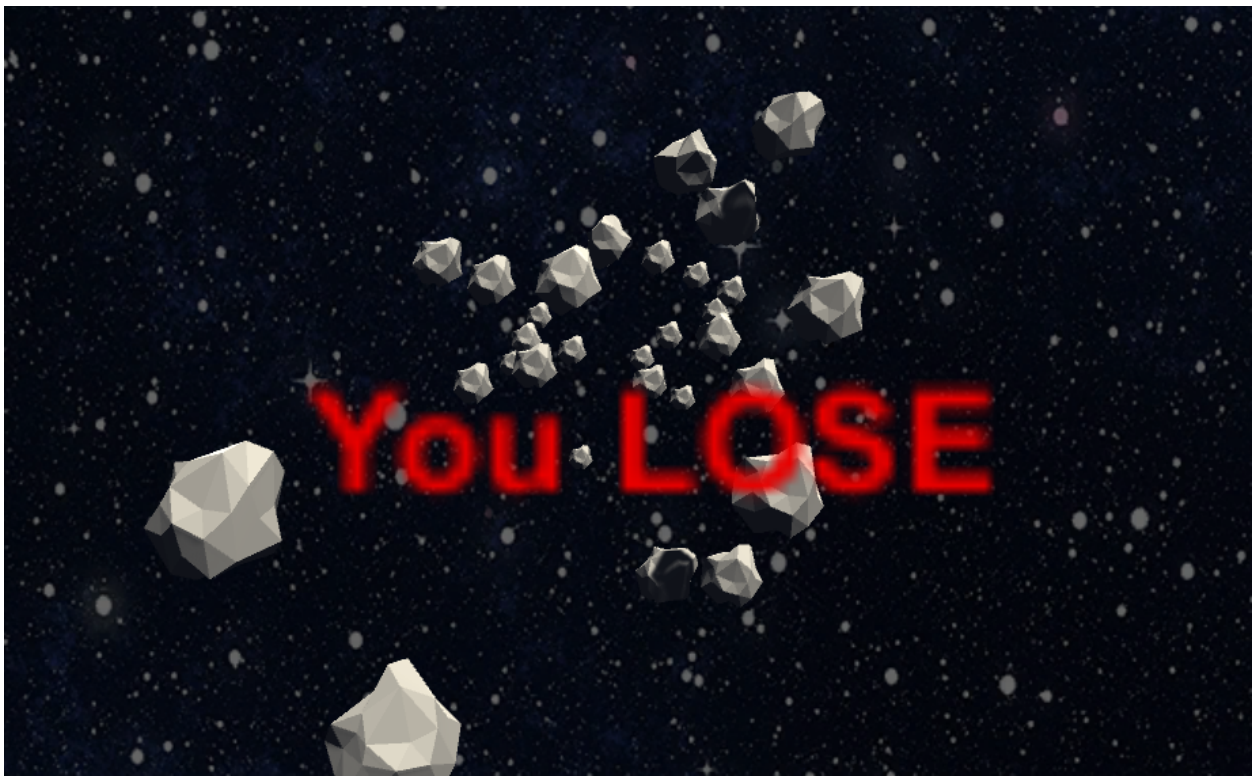
Цель работы

Изучить трёхмерное программирование игр. Изучить принципы реализации игр по типу бесконечного движения. Динамически создавать препятствия. Рационально использовать ресурсы.

Задание

- На нас летят астероиды и мы уворачиваемся;
- Поле с астероидами движется на нас.

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).

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

Сперва создадим префабы астероидов после чего опишем их поведение. Ради разнообразия были созданы три различные модели астероидов. Поэтому создадим три префаба которые не отличаются ничем кроме модели астероида. Добавим компоненту `Rigidbody` чтобы управлять движением астероидов и `Box Collider` чтобы обрабатывать события столкновений.

Опишем класс `RockBehaviour` и добавим его каждому префабу астероида в качестве компонента.

```
public class RockBehaviour: MonoBehaviour {
 private Rigidbody _rigidbody;
 public System.Single _speed = 1.0F;
 public System.Single _rotationSpeed = 1.0F;
 public System.Single _deadEnd = -10.0F;
 private Vector3 _eulerAngles;
```

Поле `_speed` задаёт скорость движения астероида по встречному направлению. Поле `_deadEnd` задает границу, при достижении которой астероид уничтожается. Этой границей является невидимая плоскость, перпендикулярная направлению движения астероидов и находящаяся немного позади камеры. Таким образом будет решена проблема утечки памяти. Поле `_rotationSpeed` задает максимальную скорость вращения астероида вокруг собственных осей.

В методе `Awake` в первую очередь сохраним компонент твердого тела и зададим вращение астероида в случайном порядке.

```
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)
   );
 }
```

Состояние астероида будем обновлять в методе `FixedUpdate`. Движение задано как прямолинейное и без ускорения.

```
void FixedUpdate () {

   Vector3 pos = _rigidbody.position;
   pos.z -= _speed * Time.deltaTime;
   _rigidbody.MovePosition(pos);

   Vector3 rot = _rigidbody.rotation.eulerAngles;
```

```
    rot += _eulerAngles * _rotationSpeed * Time.deltaTime;
    _rigidbody.MoveRotation(Quaternion.Euler(rot));

    if(pos.z < _deadEnd)
      Object.Destroy(gameObject);
  }
```

Каждый кадр обновляем положение и вращение астероида. При достижении граничного условия `_deadEnd` астероид самоуничтожается вызовом метода `Object.Destroy`.

На этом описание поведения астероида завершено. Теперь можем приступить к описанию самого сложного объекта — игрока.

Игровой объект представляет собой модель корабля, созданная в Blender. Игрок может управлять летающей тарелкой направляя её влево или право тем самым избегая столкновений с надвигающимися астероидами. В качестве устройства ввода выступает клавиатура.

Определим класс `UFOInputBehaviour` как класс для управления летающей тарелкой.

```
public class UFOInputBehaviour: MonoBehaviour {
 public System.Single _speed = 1.0F;
 public System.Single _amplitude = 1.0F;
 private Rigidbody _rb;
 public GameObject _sadText;
```

Поля `_speed` и `_amplitude` задают скорость передвижения и предельное отклонение тарелки от центральной точки. Поле `_rb` определяется как компонент твердого тела.

В методе `Awake` производим инициализацию полей. Получаем компонент твёрдого тела игрового объекта, а также поведения камеры.

```
void Awake() {
   _rb = GetComponent<Rigidbody>();
   _sadText.SetActive(false);
 }
```

Далее в методе `FixedUpdate` будем передвигать летающую тарелку в зависимости от ввода пользователя.

```
void FixedUpdate() {
   System.Single inpX = Input.GetAxis("Horizontal");
   System.Single inpY = Input.GetAxis("Vertical");

   Vector3 pos = _rb.position;

   pos.x += inpX * _speed * Time.deltaTime;
   pos.y += inpY * _speed * Time.deltaTime;

   pos.x = Mathf.Clamp(pos.x, -_amplitude, _amplitude);
   pos.y = Mathf.Clamp(pos.y, -_amplitude*0.6F, _amplitude*0.4F);

   _rb.MovePosition(pos);
 }
```

Здесь в зависимости от наличия ввода по горизонтальной или вертикальной осям происходит вычисление нового положения тарелки в пространстве с учетом заданной скоростной характеристики. Методом `Matf.Clamp` ограничиваем движение тарелки.

Последняя функция, которая реализует функционал летающего аппарата является `OnCollisionEnter`. Она реализует условие проигрыша. В случае если тарелка наткнулась на встречный метеорит на экране высвечивается надпись о том, что игра окончена.

```
private void OnCollisionEnter(Collision collision)
 {
```

```
    if (collision.gameObject.tag == "Enemy")
    {
      _sadText.SetActive(true);
    }
  }
```

Поле `_sadText` хранит ссылку на игровой объект представляющий собой текстовое сообщение на экране. В начале игры этот объект деактивируется чтобы игрок его не видел раньше времени.

В методе `OnCollisionEnter`, который срабатывает по событию столкновения двух объектов (в нашем случае это корабль и астероид), происходит активация текстового объекта. Активация происходит только в том случае если второй объект имел тег «Enemy» (который имеют все префабы астероидов).

Последний в данной работе класс `GlobalBahaviour` реализует глобальную логику игры. Данный компонент применяется к пустому, невидимому в игровом мире объекту. Его задача заключается в генерации астероидов в дали от игрока. Поведение астероидов определено так что они сами движутся в сторону игрока. В силу относительности движения, однако создаётся впечатление что это корабль движется навстречу астероидам.
По истечении определённого промежутка времени `_timeout` происходит создание очередного астероида. На некотором удалении `_distance` от плоскости движения корабля имеется некоторая другая воображаемая плоскость. На этой плоскости выбирается случайная точка pos. В этой точке методом - генератором `create` создаётся новый астероид на основе одного из трёх префабов. Только что созданному астероиду задаётся случайная скорость и вращение. Таким образом достигается достаточная вариативность астероидов. Далее астероид живёт своей жизнью.

Выводы

Был рассмотрен процесс создания трехмерной игры Astr. Была рассмотрена техника создания бесконечного движения с динамически создаваемыми препятствиями и их рационального использования.

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 = 1.0F;
  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;

}

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

public class RockBehaviour: MonoBehaviour {
  private Rigidbody _rigidbody;
  public System.Single _speed = 1.0F;
  public System.Single _rotationSpeed = 1.0F;
  public System.Single _deadEnd = -10.0F;
  private Vector3 _eulerAngles;

  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)
    );
  }

  void FixedUpdate () {

    Vector3 pos = _rigidbody.position;
    pos.z -= _speed * Time.deltaTime;
    _rigidbody.MovePosition(pos);

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

```
    if(pos.z < _deadEnd)
      Object.Destroy(gameObject);
  }


}

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

public class UFOInputBehaviour: MonoBehaviour {
 public System.Single _speed = 1.0F;
 public System.Single _amplitude = 1.0F;
 private Rigidbody _rb;
 public GameObject _sadText;

 void Awake() {
   _rb = GetComponent<Rigidbody>();
   _sadText.SetActive(false);
 }
  void FixedUpdate() {
   System.Single inpX = Input.GetAxis("Horizontal");
   System.Single inpY = Input.GetAxis("Vertical");

   Vector3 pos = _rb.position;

   pos.x += inpX * _speed * Time.deltaTime;
   pos.y += inpY * _speed * Time.deltaTime;

   pos.x = Mathf.Clamp(pos.x, -_amplitude, _amplitude);
   pos.y = Mathf.Clamp(pos.y, -_amplitude*0.6F, _amplitude*0.4F);

   _rb.MovePosition(pos);
 }

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

```
}
```