Московский Авиационный Институт
(Национальный исследовательский университет)

«Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа №2
по курсу «Программирование игр»

Студент: Лазаревич О.А.

Группа: М8О-108М-20

Преподаватель: Аносова Н.П.

Москва, 2022

Лабораторная работа №2 «Balls»

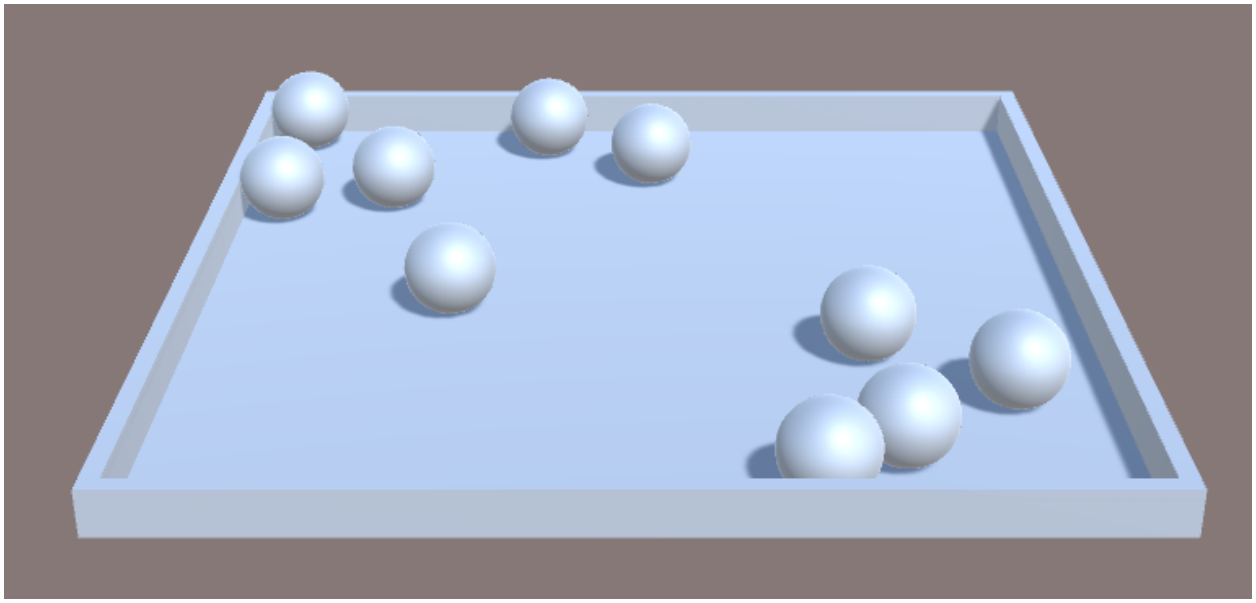
Цель работы

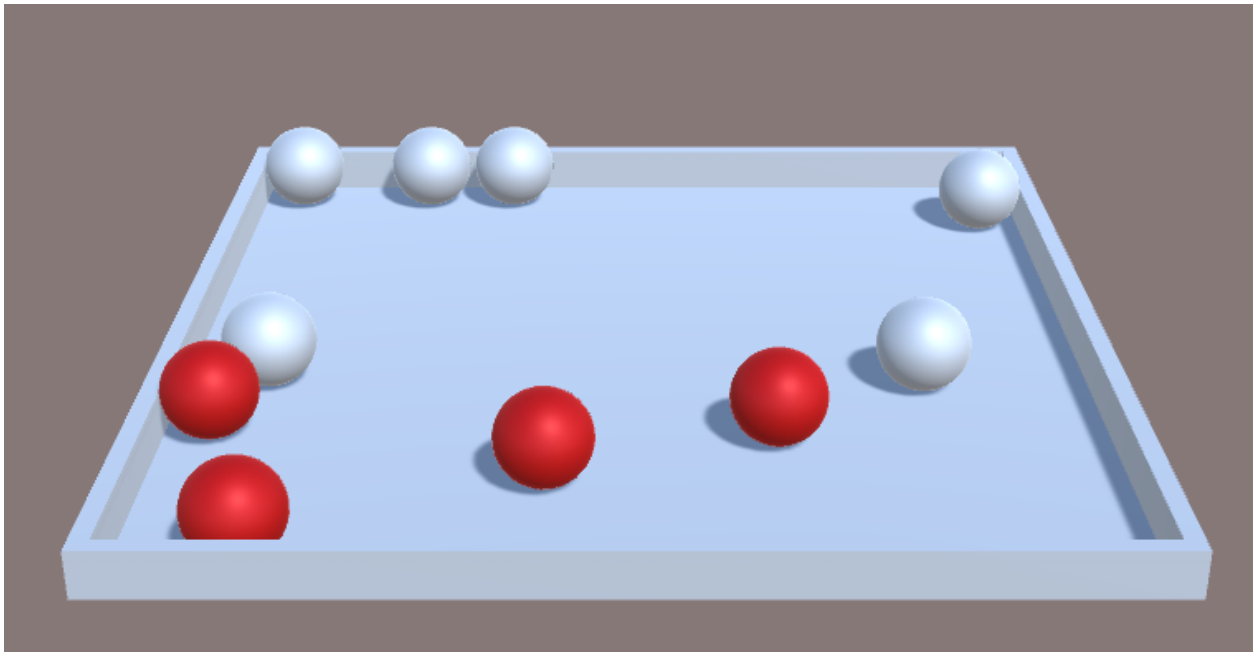
Познакомиться с трехмерными принципами программирования игр. Реализовать трёхмерную обработку столкновений. Реализовать управление трехмерными объектами с помощью мыши.

Задание

- Все то же самое, как в 1 лабораторной работе, только в 3D;
- Возможность точно выбрать шарик;
- Можно реализовать плотную среду, гравитацию, броуновское движение, вращение куба.

Ход работы





В данной работе будут 2 действующих объекта: трехмерная коробка с границами в виде бортиков, несколько шаров. Камера будет статической. Поведение трёхмерного шара абсолютно аналогично поведению двухмерного шара за исключением того, что выделение выбранного шара производится средствами компонента MeshRenderer. В этой работе применяется полноценная трассировка лучей.

```
RaycastHit hit;
     Ray ray = _cam.ScreenPointToRay(Input.mousePosition);

     if (Physics.Raycast(ray.origin, ray.direction * 25.0F, out hit) ==
true)
     { ... }
```

Сперва сопоставляем положение курсора на экране с воображаемой точкой на объективе активной камеры, а потом вызываем метод `Physics.Raycast` чтобы направить луч в сторону сцены. Второй параметр задаёт направление луча, а в третий параметр метод возвращает значение.

Также в данной работе участвует шар только одного типа поэтому можем сразу обращаться к префабу шара «Ball».

Определим класс BallBehaviour и добавим компонент этого класса каждому префабу шара. Этот класс позволит визуально отделять выбранные шары на фоне остальных. Дадим классу два поля: _isSelected и _material.

```
public class BallBehaviour: MonoBehaviour {
 private System.Boolean _isSelected;
 private Material _material;
```

Булево поле _isSelected будет определять выбран шар в данный момент или нет. По умолчанию зададим ему значение false.

Метод Awake вызывается один раз, когда компонент скрипта загружен поэтому здесь проведем базовую инициализацию.

```
void Awake() {
   _material = GetComponent<MeshRenderer>().material;
   IsSelected = false;
 }
```

Здесь просто получим ссылку на объект `Material` посредством вызова метода `GetComponent`.

Добавим свойство IsSelected которое будет задавать значение полю _isSelected и выделять красным цветом выбранный шар. Свойство будет иметь два метода доступа: get и set.

```
public System.Boolean IsSelected {
   get {
     return _isSelected;
   }

   set {
     _material.color = (value == true) ? Color.red : Color.white;
     _isSelected = value;
   }
 }
```

Осталось реализовать управление. Определим класс `GlobalBehaviour` .

```
public class GlobalBehaviour: MonoBehaviour {
 public Camera _cam = null;
```

```
public System.Single _horsepower = 1.0F;
private List<GameObject> _selectedBalls;
```

Поле `_selectedBalls` объявлено как имеющее тип `List<GameObject>` и будет хранить список всех выбранных в данный момент шаров. Поле `_horsepower` определяет скорость движения шара.

Для получения воспользуемся техникой под названием raycast. Данная техника заключается в том, чтобы перпендикулярно поверхности экрана направить луч в сторону сцены и проверить с каким объектом на сцене он столкнулся. Так как ранее мы дали каждому префабу шара тег «PlayableBall», то достаточно лишь проверить тег объекта.

Состояние кнопок мыши поместим в массив `mouseDown`. `mouseDown[0]` определяет нажатие ЛКМ, а `mouseDown[1]` — ПКМ.
Поведение при нажатии ПКМ не сложное — это либо сбросить текущее выделение шаров если мышь указывает на пустое пространство, либо удалить шар если мышь указывает на конкретный шар.

```
if(_selectedBalls.Count > 0)
            {
              for (System.Int32 i = 0; i < _selectedBalls.Count; ++i)
              {
                BallBehaviour behaviour =
_selectedBalls[i].GetComponent<BallBehaviour>();
                behaviour.IsSelected = false;
              }
              _selectedBalls.Clear();
            }

if(_selectedBalls.Count <= 0)
              Object.Destroy(hitObject);
```

По нажатию ЛКМ есть три сценария. Если попали по шару, то добавляем к списку выбранных шаров. Также свойство шара IsSelected задаём равным true что приводит к окрашиванию шара красным цветом.

```
BallBehaviour behaviour = hitObject.GetComponent<BallBehaviour>();
            behaviour.IsSelected = true;
            _selectedBalls.Add(hitObject);
```

Второй сценарий это не попали по шару и нет ранее выбранных шаров. В этом случае создаём новый шар.

```
Vector3 position = hit.point;
              position.y += 2;
              Instantiate(Resources.Load("Ball"), position,
Quaternion.identity);
```

Чтобы создать новый шар вызываем метод `Instantiate` первым аргументом которого указываем название префаба шара.

И наконец третий сценарий это не попали по шару и одновременно некоторые шары уже были выбраны. В этом случае приводим шары в движение.

```
if(_selectedBalls.Count > 0)
            {
              for (System.Int32 i = 0; i < _selectedBalls.Count; ++i)
              {
                Vector3 force = hit.point -
_selectedBalls[i].transform.position;
                force.Normalize();
                force *= _horsepower;
                Rigidbody body =
_selectedBalls[i].GetComponent<Rigidbody>();
                body.AddForce(force);
              }
            }
```

Здесь просто воспользуемся подсистемой Unity для моделирования физических процессов. Для каждого шара в списке выбранных шаров посчитаем единичный вектор направленности в сторону указателя мыши и приложим с помощью метода `AddForce` к каждому шару силу в сторону этого направления.

Выводы

Был рассмотрен трёхмерный вариант лабораторной работы №2. Был рассмотрен трёхмерный вариант использования трассировки лучей для выбора объектов на сцене, рассмотрены принципы обработки столкновений, использована Unity для моделирования физических процессов.

Листинг

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

public class BallBehaviour: MonoBehaviour {
 private System.Boolean _isSelected;
 private Material _material;

 void Awake() {
   _material = GetComponent<MeshRenderer>().material;
   IsSelected = false;
 }

 public System.Boolean IsSelected {
   get {
     return _isSelected;
   }

   set {
     _material.color = (value == true) ? Color.red : Color.white;
     _isSelected = value;
   }
 }
```

```
}

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

public class GlobalBehaviour: MonoBehaviour {
 public Camera _cam = null;
 public System.Single _horsepower = 1.0F;
 private List<GameObject> _selectedBalls;
 void Awake() {
   _selectedBalls = new List<GameObject>();
 }

 void LateUpdate() {
   bool[] mouseUp = { Input.GetMouseButtonUp(0),
Input.GetMouseButtonUp(1) };
   bool[] mouseDown = { Input.GetMouseButtonDown(0),
Input.GetMouseButtonDown(1) };
   bool[] mouseHold = { Input.GetMouseButton(0), Input.GetMouseButton(1)
};


   if (mouseDown[1])
   {
     Cursor.visible = false;
   }
   else if (mouseUp[1])
   {
     Cursor.visible = true;
   }

   if (mouseDown[0] || mouseDown[1])
   {
     RaycastHit hit;
     Ray ray = _cam.ScreenPointToRay(Input.mousePosition);

     if (Physics.Raycast(ray.origin, ray.direction * 25.0F, out hit) ==
true)
     {
       GameObject hitObject = hit.collider.gameObject;
       switch (hitObject.tag)
       {
         case "PlayableBall":
```

```
            if(mouseDown[0])
            {
              BallBehaviour behaviour =
hitObject.GetComponent<BallBehaviour>();
              behaviour.IsSelected = true;
              _selectedBalls.Add(hitObject);
            }
            else if(mouseDown[1])
            {
              if(_selectedBalls.Count <= 0)
                Object.Destroy(hitObject);
            }
            break;

          case "FirmGround":
            if(mouseDown[0])
            {
              if(_selectedBalls.Count > 0)
              {
                for (System.Int32 i = 0; i < _selectedBalls.Count; ++i)
                {
                  Vector3 force = hit.point -
_selectedBalls[i].transform.position;
                  force.Normalize();
                  force *= _horsepower;
                  Rigidbody body =
_selectedBalls[i].GetComponent<Rigidbody>();
                  body.AddForce(force);
                }
              }
              else
              {
                Vector3 position = hit.point;
                position.y += 2;
                Instantiate(Resources.Load("Ball"), position,
Quaternion.identity);
              }
            }
            else if(mouseDown[1])
            {
              if(_selectedBalls.Count > 0)
              {
                for (System.Int32 i = 0; i < _selectedBalls.Count; ++i)
                {
                  BallBehaviour behaviour =
```

```
_selectedBalls[i].GetComponent<BallBehaviour>();
                behaviour.IsSelected = false;
              }
              _selectedBalls.Clear();
            }
          }
          break;

        default:
          break;
      }
    }
  }
}
```