Московский Авиационный Институт
(Национальный исследовательский университет)

«Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа №1
по курсу «Программирование игр»

Студент: Лазаревич О.А.

Группа: М8О-108М-20

Преподаватель: Аносова Н.П.

Москва, 2022

Лабораторная работа №1 «Bouncing Balls»

Цель работы

Реализовать создание шариков и их упругое столкновение и трение в 2D.

Задание

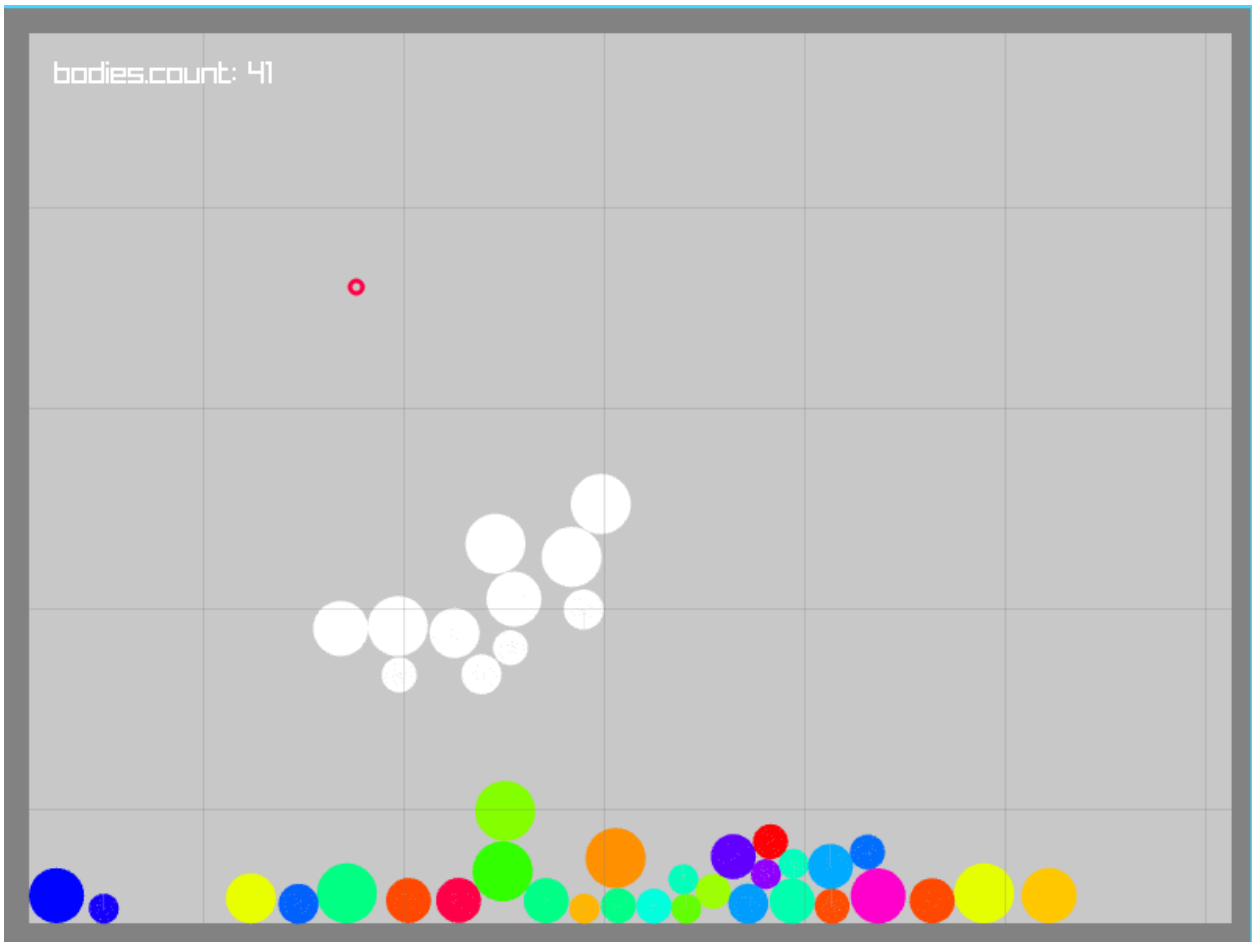
ЛКМ по свободному месту -> шарик;

ЛКМ по шарику -> цвет шарика красный;

Красные шарики собираются на ПКМ в свободное место;

Реализовать упругое столкновение и трение.

Ход работы



Для выполнения работы был выбран язык C, библиотека для создания игр RayLib и библиотека для физики Ferox.

В основнйо функции **main()** я задаю значение FPS:

```
SetTargetFPS( TARGET_FPS );
```

Инициализирую объект окна:

```
InitWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "Bouncing Balls");
```

Создаю объект игрового мира:

```
frWorld *world = InitGame();
```

Обновляю и отрисовываю его, пока окно не открыто:

```
while (!WindowShouldClose()) {
   UpdateGame(world);
   DrawGame(world);
 }
```

В конце разрушаю все игровые объекты и объект окна:

```
frReleaseWorldBodies(world);
CloseWindow();
```

Теперь по подробнее о каждой функции. **InitGame()** создает: сам игровой мир **frCreateWorld()** и ограничительные стены из библиотеки Ferox, затем добавляет их мир.

```
// Create world
 frWorld *world = frCreateWorld(
   frVec2ScalarMultiply(FR_WORLD_DEFAULT_GRAVITY, 0.00001f),
   WORLD_RECTANGLE
 );
```

```
// Create boundaries
frBody *boundaryDown = CreateBoundaryDown(GRAY);
frBody *boundaryUp = CreateBoundaryUp(GRAY);
frBody *boundaryRight = CreateBoundaryLeft(GRAY);
frBody *boundaryLeft = CreateBoundaryRight(GRAY);
 // add boundaries to world
frAddToWorld(world, boundaryUp);
frAddToWorld(world, boundaryDown);
frAddToWorld(world, boundaryRight);
frAddToWorld(world, boundaryLeft);
```

Функция **UpdateGame**(frWorld *world) обрабатывает нажатия ЛКМ, ПКМ. Здесь я создаем шарик белого цвета, если нажимаем на уже созданный шарик, то окрашиваем его в белый. Если мы уже окрасили его в белы и повторно нажали не него, то возвращаем ему его цвет.

```
if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON)) {
   frShape *targetShape = GetTargetShape(world, GetMousePosition());
   bool collisionTrue = false;
   if (targetShape != NULL) collisionTrue = true;

   if(!collisionTrue)
   {
     frBody *circle = CreateColoredCircle(frGetRandomColor());
     frAddToWorld(world, circle);
   }
   else if(frGetShapeType(targetShape) == FR_SHAPE_CIRCLE &&
           ColorToInt(frGetShapeColor(targetShape)) !=
ColorToInt(WHITE))
   {
     frSetShapeColor(targetShape, WHITE);
   }
   else if(frGetShapeType(targetShape) == FR_SHAPE_CIRCLE &&
```

```
          ColorToInt(frGetShapeColor(targetShape)) ==
ColorToInt(WHITE))
   {
     frSetShapeColor(targetShape, frGetShapeInitColor(targetShape));
   }
 }
```

Здесь на нажатие на ПКМ добавляется сила к каждому белому шарику, вектор которой направлен на точку на которую мы нажали. Когда расстояние становится небольшим, возвращаем шарикам из первоначальный цвет.

```
 if (IsMouseButtonDown(MOUSE_BUTTON_RIGHT)) {
     for (int i = 0; i < frGetWorldBodyCount(world); i++) {
       frBody *body = frGetWorldBody(world, i);
       frShape *shape = frGetBodyShape(body);
     if(frGetShapeType(shape) == FR_SHAPE_CIRCLE &&
         ColorToInt(frGetShapeColor(shape)) == ColorToInt(WHITE))
     {
       float distance = Vector2Distance(
                           frVec2PixelsToMeters(GetMousePosition()),
                           frGetBodyPosition(body)
                                       );

       if (distance <= 5.0f)
         frSetShapeColor(shape, frGetShapeInitColor(shape));

       float ds = 1.5f / distance;
       Vector2 targetVec = Vector2Lerp(
                               frGetBodyPosition(body),
                               frVec2PixelsToMeters(GetMousePosition()),
                               ds
                                       );
       frSetBodyPosition(body, targetVec);
     }
```

```
    }
  }
```

В случае, если объект вылетел за границы игрового поля, он разрушается.

```
 // remove bodies out of screen
  for (int i = 0; i < frGetWorldBodyCount(world); i++) {
    frBody *body = frGetWorldBody(world, i);

    if (!CheckCollisionRecs(frGetBodyAABB(body), WORLD_RECTANGLE))
      frRemoveFromWorld(world, body);
  }
```

Чтобы шарикам можно было вернуть их цвет пришлось немного доработать библиотеку Ferox и добавить туда несколько функций и свойство цвета его объектам.

```
 typedef struct frShape {
    frShapeType type;
    frMaterial material;
    float area;
    Color initColor;
    Color color;
    union {
        struct {
            float radius;
        } circle;
        struct {
            struct {
                Vector2 data[FR_GEOMETRY_MAX_VERTEX_COUNT];
                int count;
            } vertices;
            struct {
                Vector2 data[FR_GEOMETRY_MAX_VERTEX_COUNT];
```

```
            int count;
        } normals;
    } polygon;
};
} frShape;

frShape *frCreateCircle(frMaterial material, float radius, Color
initColor) {
    frShape *result = frCreateShape();

    result->type = FR_SHAPE_CIRCLE;
    result->area = PI * (radius * radius);
    result->material = material;
    result->initColor = initColor;
    result->color = initColor;

    result->circle.radius = radius;

    return result;
}

frShape *frCreateRectangle(frMaterial material, Vector2 p1, Vector2 p2,
Color initColor) {
    frShape *result = frCreateShape();

    result->type = FR_SHAPE_POLYGON;
    result->material = material;
    result->area = -FLT_MAX;
    result->initColor = initColor;
    result->color = initColor;

    Vector2 vertices[4] = {
        p1,
        (Vector2){p1.x, p2.y},
```

```
        p2,
        (Vector2){p2.x, p1.y}
    };

    frSetPolygonVertices(result, vertices, 4);

    return result;
}

frShape *frCreatePolygon(frMaterial material, Vector2 *vertices, int
count, Color initColor) {
    frShape *result = frCreateShape();

    if (vertices == NULL || count < 2 || count >
FR_GEOMETRY_MAX_VERTEX_COUNT)
        return NULL;

    result->type = FR_SHAPE_POLYGON;
    result->material = material;
    result->area = -FLT_MAX;
    result->initColor = initColor;
    result->color = initColor;

    frSetPolygonVertices(result, vertices, count);

    return result;
}

Color frGetShapeInitColor(frShape *s) {
    return (s != NULL) ? s->initColor : WHITE;
}

Color frGetShapeColor(frShape *s) {
    return (s != NULL) ? s->color : WHITE;
```

```
}

void frSetShapeInitColor(frShape *s, Color color) {
   if (s != NULL) s->initColor = color;
}

void frSetShapeColor(frShape *s, Color color) {
   if (s != NULL) s->color = color;
}
```

Функция отрисовки `DrawGame(frWorld *world)` задает цвет фона:

```
ClearBackground(LIGHTGRAY);
```

Отрисовывает каждое тело в игре:

```
for (int i = 0; i < frGetWorldBodyCount(world); i++){
   if (frGetBodyType)
     frDrawBody(frGetWorldBody(world, i));
 }
```

Отрисовывает текст в углу экрана:

```
DrawTextEx(
   GetFontDefault(),
   TextFormat(
     "bodies.count: %d\n",
     frGetWorldBodyCount(world) - 4
   ),
   (Vector2) { 32, 32 },
   20,
   1,
   WHITE
 );
```

Выводы

В результате выполнения лабораторной работы была реализована простая 2D игра на C. У меня получилось изучить базовые элементы движка RayLib и взаимодействие с объектами.

Листинг

```
#include "ferox.h"
#include "raylib.h"
#include "raymath.h"

#define TARGET_FPS 60

#define SCREEN_WIDTH 800
#define SCREEN_HEIGHT 600

#define SCREEN_WIDTH_IN_METERS (SCREEN_WIDTH /
FR_GLOBAL_PIXELS_PER_METER)
#define SCREEN_HEIGHT_IN_METERS (SCREEN_HEIGHT /
FR_GLOBAL_PIXELS_PER_METER)

#define WORLD_RECTANGLE ((Rectangle) { 0, 0, SCREEN_WIDTH_IN_METERS,
SCREEN_HEIGHT_IN_METERS })

#define BODY_MATERIAL ((frMaterial) { 20.0f, 0.0f,  100.0f, 100.0f })
// #define BODY_MATERIAL ((frMaterial) { 100.0f, 0.0f,  0.0f, 0.0f })
#define BOUNDARY_MATERIAL FR_DYNAMICS_DEFAULT_MATERIAL

static frWorld *InitGame(void);
static void UpdateGame(frWorld *);
static void DrawGame(frWorld *);
static frBody *CreateBoundaryDown(Color);
static frBody *CreateBoundaryUp(Color);
static frBody *CreateBoundaryLeft(Color);
static frBody *CreateBoundaryRight(Color);
static frBody *CreateColoredCircle(Color);
static frShape *GetTargetShape(frWorld *, Vector2);

int main(void) {
 SetConfigFlags(FLAG_MSAA_4X_HINT);
```

```
 SetTargetFPS(TARGET_FPS);
  InitWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "Bouncing Balls");

 frWorld *world = InitGame();

 while (!WindowShouldClose()) {
   UpdateGame(world);
   DrawGame(world);
 }
  frReleaseWorldBodies(world);
 CloseWindow();
 return 0;
}


frWorld *InitGame(){
 // Create world
 frWorld *world = frCreateWorld(
   frVec2ScalarMultiply(FR_WORLD_DEFAULT_GRAVITY, 0.00001f),
   WORLD_RECTANGLE
 );

 // Create boundaries
 frBody *boundaryDown = CreateBoundaryDown(GRAY);
 frBody *boundaryUp = CreateBoundaryUp(GRAY);
 frBody *boundaryRight = CreateBoundaryLeft(GRAY);
 frBody *boundaryLeft = CreateBoundaryRight(GRAY);
  // add boundaries to world
 frAddToWorld(world, boundaryUp);
 frAddToWorld(world, boundaryDown);
 frAddToWorld(world, boundaryRight);
 frAddToWorld(world, boundaryLeft);

 return world;
}


void UpdateGame(frWorld *world){

 if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON)) {
   frShape *targetShape = GetTargetShape(world, GetMousePosition());
   bool collisionTrue = false;
   if (targetShape != NULL) collisionTrue = true;

   if(!collisionTrue)
```

```
   {
     frBody *circle = CreateColoredCircle(frGetRandomColor());
     frAddToWorld(world, circle);
   }
   else if(frGetShapeType(targetShape) == FR_SHAPE_CIRCLE &&
           ColorToInt(frGetShapeColor(targetShape)) !=
ColorToInt(WHITE))
   {
     frSetShapeColor(targetShape, WHITE);
   }
   else if(frGetShapeType(targetShape) == FR_SHAPE_CIRCLE &&
           ColorToInt(frGetShapeColor(targetShape)) ==
ColorToInt(WHITE))
   {
     frSetShapeColor(targetShape, frGetShapeInitColor(targetShape));
   }
 }


 if (IsMouseButtonDown(MOUSE_BUTTON_RIGHT)) {
     for (int i = 0; i < frGetWorldBodyCount(world); i++) {
       frBody *body = frGetWorldBody(world, i);
       frShape *shape = frGetBodyShape(body);
     if(frGetShapeType(shape) == FR_SHAPE_CIRCLE &&
         ColorToInt(frGetShapeColor(shape)) == ColorToInt(WHITE))
     {
       float distance = Vector2Distance(
                           frVec2PixelsToMeters(GetMousePosition()),
                           frGetBodyPosition(body)
                                        );

       if (distance <= 5.0f)
         frSetShapeColor(shape, frGetShapeInitColor(shape));

       float ds = 1.5f / distance;
       Vector2 targetVec = Vector2Lerp(
                               frGetBodyPosition(body),
                               frVec2PixelsToMeters(GetMousePosition()),
                               ds
                                        );
       frSetBodyPosition(body, targetVec);
     }
   }
 }
```

```
 // remove bodies out of screen
 for (int i = 0; i < frGetWorldBodyCount(world); i++) {
   frBody *body = frGetWorldBody(world, i);

   if (!CheckCollisionRecs(frGetBodyAABB(body), WORLD_RECTANGLE))
     frRemoveFromWorld(world, body);
 }
}

void DrawGame(frWorld *world){
 BeginDrawing();

 ClearBackground(LIGHTGRAY);
  for (int i = 0; i < frGetWorldBodyCount(world); i++){
   if (frGetBodyType)
     frDrawBody(frGetWorldBody(world, i));
 }
  frDrawSpatialHash(frGetWorldSpatialHash(world));

 frSimulateWorld(world, (1.0f / 60.0f) * 100);
  DrawTextEx(
   GetFontDefault(),
   TextFormat(
     "bodies.count: %d\n",
     frGetWorldBodyCount(world) - 4
   ),
   (Vector2) { 32, 32 },
   20,
   1,
   WHITE
 );

 EndDrawing();
}

frBody *CreateBoundaryDown(Color color){
 frBody *body = frCreateBodyFromShape(
       FR_BODY_KINEMATIC,
       (Vector2) { SCREEN_WIDTH_IN_METERS * 0.5f,
SCREEN_HEIGHT_IN_METERS},
       frCreateRectangle(
         BOUNDARY_MATERIAL,
         (Vector2) {SCREEN_WIDTH_IN_METERS, 2.0f },
         (Vector2) { 0, 0 },
         color
```

```
        )
 );
 return body;
}
 frBody *CreateBoundaryUp(Color color){
 frBody *body = frCreateBodyFromShape(
        FR_BODY_KINEMATIC,
       (Vector2) { SCREEN_WIDTH_IN_METERS * 0.5f, 0},
       frCreateRectangle(
         BOUNDARY_MATERIAL,
         (Vector2) { SCREEN_WIDTH_IN_METERS , 2.0f },
         (Vector2) { 0, 0 },
         color
       )
 );
 return body;
}
 frBody *CreateBoundaryLeft(Color color){
 frBody *body = frCreateBodyFromShape(
       FR_BODY_KINEMATIC,
       (Vector2) { SCREEN_WIDTH_IN_METERS, SCREEN_HEIGHT_IN_METERS *
0.5f},
       frCreateRectangle(
         BOUNDARY_MATERIAL,
         (Vector2) {2.0f, SCREEN_HEIGHT_IN_METERS  - 2.0f},
         (Vector2) { 0, 0 },
         color
       )
 );
 return body;
}
 frBody *CreateBoundaryRight(Color color){
 frBody *body = frCreateBodyFromShape(
       FR_BODY_KINEMATIC,
       (Vector2) { 0, SCREEN_HEIGHT_IN_METERS * 0.5f},
       frCreateRectangle(
         BOUNDARY_MATERIAL,
         (Vector2) {2.0f, SCREEN_HEIGHT_IN_METERS  - 2.0f},
         (Vector2) { 0, 0 },
         color
       )
 );
 return body;
}
```

```
frBody *CreateColoredCircle(Color color){
 frBody  *body = frCreateBodyFromShape(
     FR_BODY_DYNAMIC,
     frVec2PixelsToMeters(GetMousePosition()),
     frCreateCircle(
       BODY_MATERIAL,
       0.1f * GetRandomValue(6, 12),
       color
     )
 );

 return body;
}

frShape *GetTargetShape(frWorld *world, Vector2 mousePosition){
 frShape *targetShape = NULL;
 for (int i = 0; i < frGetWorldBodyCount(world); i++) {
   frBody *body = frGetWorldBody(world, i);
   frShape *shape = frGetBodyShape(body);

   if(CheckCollisionPointRec(frVec2PixelsToMeters(mousePosition),
frGetBodyAABB(body)))
   {
     targetShape = shape;
     break;
   }
 }
 return targetShape;
}
```