Spike: 11
Title: Collisions

Author: Ryan Chessum, 102564760

Goals / deliverables:

- Code, see /22 – Spike – Collisions /Collisions/
- Spike Report

Technologies, Tools, and Resources used:
List of information needed by someone trying to reproduce this work

- Visual Studio 2019
- C plus plus reference (https://www.cplusplus.com/reference/)
- SDL2

Tasks undertaken:

- Download and install Visual Studio
- Create a new C++ project
- Set up SDL window init
- Create Class for object to move
- Create collision detection system

What we found out:

In this spike I made a basic program that calculates the collisions between circles and circles and rectangles and rectangles.

You can detect if two rectangles are colliding by getting positions for the top, bottom, left and right of the two rectangles. Then you check if the position of the first rectangles bottom is less than the position of the second one's top (top to bottom y cords). Then you check if the first rectangles top is greater than the bottom. Then if the firsts right is less than the second ones left and last if the first ones left is greater than the second ones right. If all of these fail then the objects must be within each other. It also doesn't matter on the order, just as long as they all fail.

```
    if (ab <= bt)
        return false;
    if (at >= bb)
        return false;
    if (ar <= bl)
        return false;
    if (al >= br)
        return false;

    return true;
}
```

For detecting collisions with circles, you get the difference in each coordinate squared and add them together. Then if the result is less than both circles' radiuses squared the two circles are inside each other.

```
int distSquared = (c2_x - c1_x)* (c2_x - c1_x) + (c2_y - c1_y) * (c2_y - c1_y);
int radSquared = (c1_rad + c2_rad) * (c1_rad + c2_rad);
if (distSquared < radSquared)
{
    return true;
}
```

For creating this in working code I made an object that moves around.

```
1   #pragma once
2   #include <SDL.h>
3   #include <SDL_image.h>
4   #include "Modes.h"
5
6   extern SDL_Renderer* renderer;
7
8   class CollidingObject
9   {
10  private:
11      int x_velocity;
12      int y_velocity;
13      int c1 = 0;
14      int c2 = 0;
15      int c3 = 0;
16      SDL_Rect box;
17  public:
18      int x;
19      int y;
20
21      bool isMoving;
22
23      int SIZE = 80;
24      int VELOCITY = 6;
25
26      CollidingObject(bool moving, int start_x, int start_y);
27      ~CollidingObject() {};
28
29      void Move(MODES mode);
30
31      void Render(MODES mode);
32
33      bool operator==(const CollidingObject &co);
34  };
35
36
```

```
void CollidingObject::Move(MODES mode)
{

    if (isMoving)
    {
        x += x_velocity;
        box.x = x;

        if (collisionChecker.CheckForCollision(mode, &x, &y, &SIZE))
        {
            c1 = rand() % 256;
            c2 = rand() % 256;
            c3 = rand() % 256;

            cout << "collision detected" << endl;
            x -= x_velocity;
            x_velocity *= -1;
            box.x = x;
        }

        y += y_velocity;
        box.y = y;

        if (collisionChecker.CheckForCollision(mode, &x, &y, &SIZE))
        {

            c1 = rand() % 256;
            c2 = rand() % 256;
            c3 = rand() % 256;

            cout << "collision detected" << endl;
            y -= y_velocity;
            y_velocity *= -1;
            box.y = y;
        }

        SDL_SetRenderDrawColor(renderer, c1, c2, c3, 255);
    }
}
```

If the object is supposed to be moving it will update its position using a global instance of a collision checker. If it detects a collision the object will be moved back out of the other object and it's velocity in that direction will be reversed.

The shape of the object can be changed with 1 and 2. The collision checker is a state machine. If the collision state is on Square it will use rectangle on rectangle collision detection. If it is on circle it will use circle on circle collision detection. This can be expanded on in the future for more shapes.

```cpp
bool CollisionChecker::CheckForCollision(MODES mode, int *x, int *y, int *size)
{
    if (mode == MODES::SQUARE)
    {
        detectionMode = &r_on_r;
    }
    else if (mode == MODES::CIRCLE)
    {
        detectionMode = &c_on_c;
    }
    return detectionMode->CheckCollision(x, y, size);
}
```

```cpp
bool RectOnRectCollisionDetector::collision(int* x, int* y, int* size)
{
    al = *x;
    ar = *x + *size;
    at = *y;
    ab = *y + *size;

    bl = wall->x;
    br = wall->x + wall->SIZE;
    bt = wall->y;
    bb = wall->y + wall->SIZE;

    if (ab <= bt)
        return false;
    if (at >= bb)
        return false;
    if (ar <= bl)
        return false;
    if (al >= br)
        return false;

    return true;
}

RectOnRectCollisionDetector::RectOnRectCollisionDetector()
{

}

bool RectOnRectCollisionDetector::CheckCollision(int* x, int* y, int* size)
{
    if (*x < 0 || *y < 0 || *x + *size > WIDTH || *y + *size > HEIGHT)
    {
        return true;
    }
    return collision(x, y, size);


    return false;
}
```

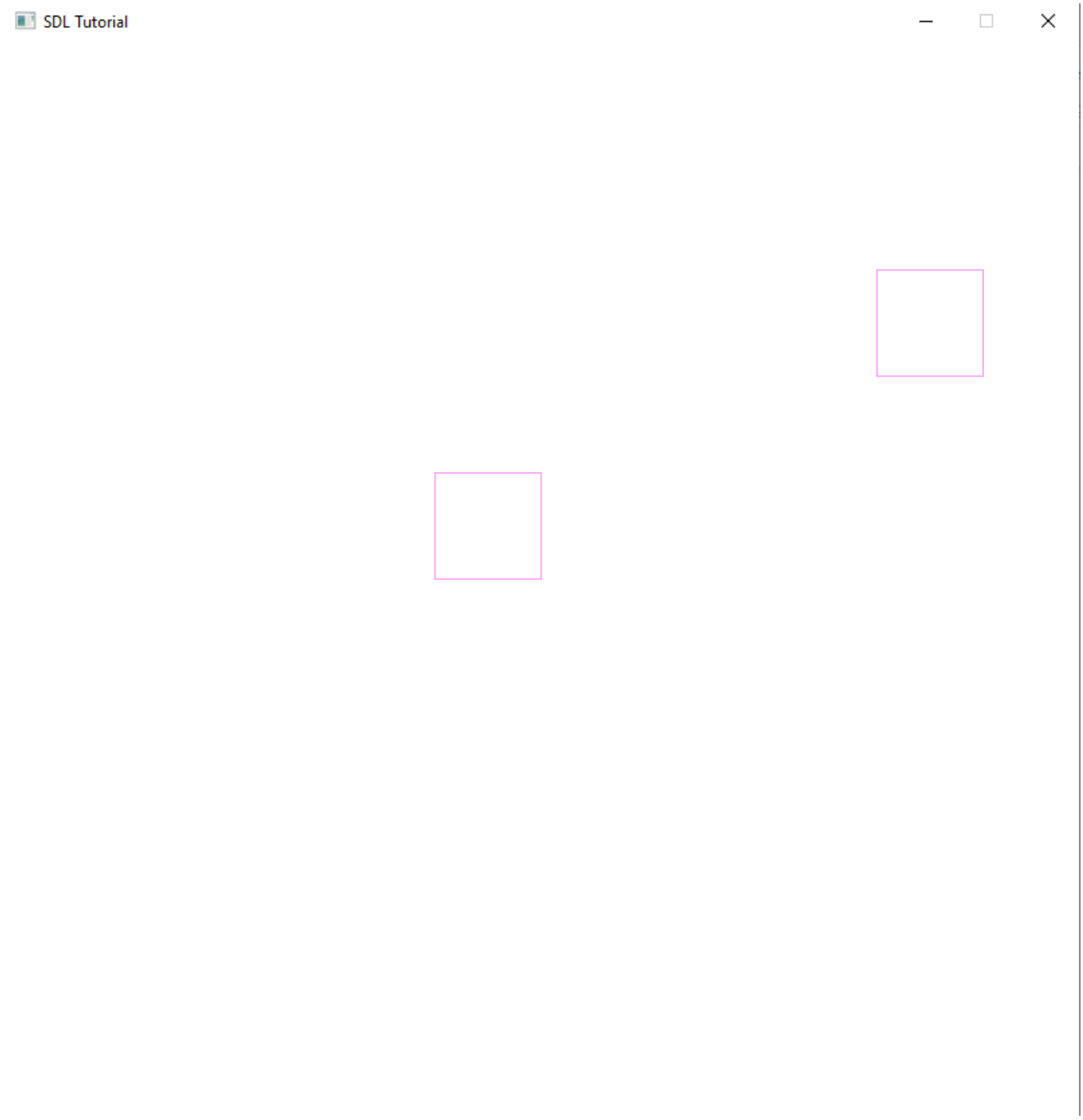
```
bool CircleOnCircleCollisionDetector::CheckCollision(int *x, int *y, int *size)
{
    c2_rad = wall->SIZE / 2;
    c2_x = wall->x + c2_rad;
    c2_y = wall->y + c2_rad;

    c1_rad = *size / 2;
    c1_x = *x + c1_rad;
    c1_y = *y + c1_rad;

    if (*x < 0 || *y < 0 || *x + *size > WIDTH || *y + *size > HEIGHT)
    {
        return true;
    }

    int distSquared = (c2_x - c1_x)* (c2_x - c1_x) + (c2_y - c1_y) * (c2_y - c1_y);
    int radSquared = (c1_rad + c2_rad) * (c1_rad + c2_rad);
    if (distSquared < radSquared)
    {
        return true;
    }

    return false;
}
```

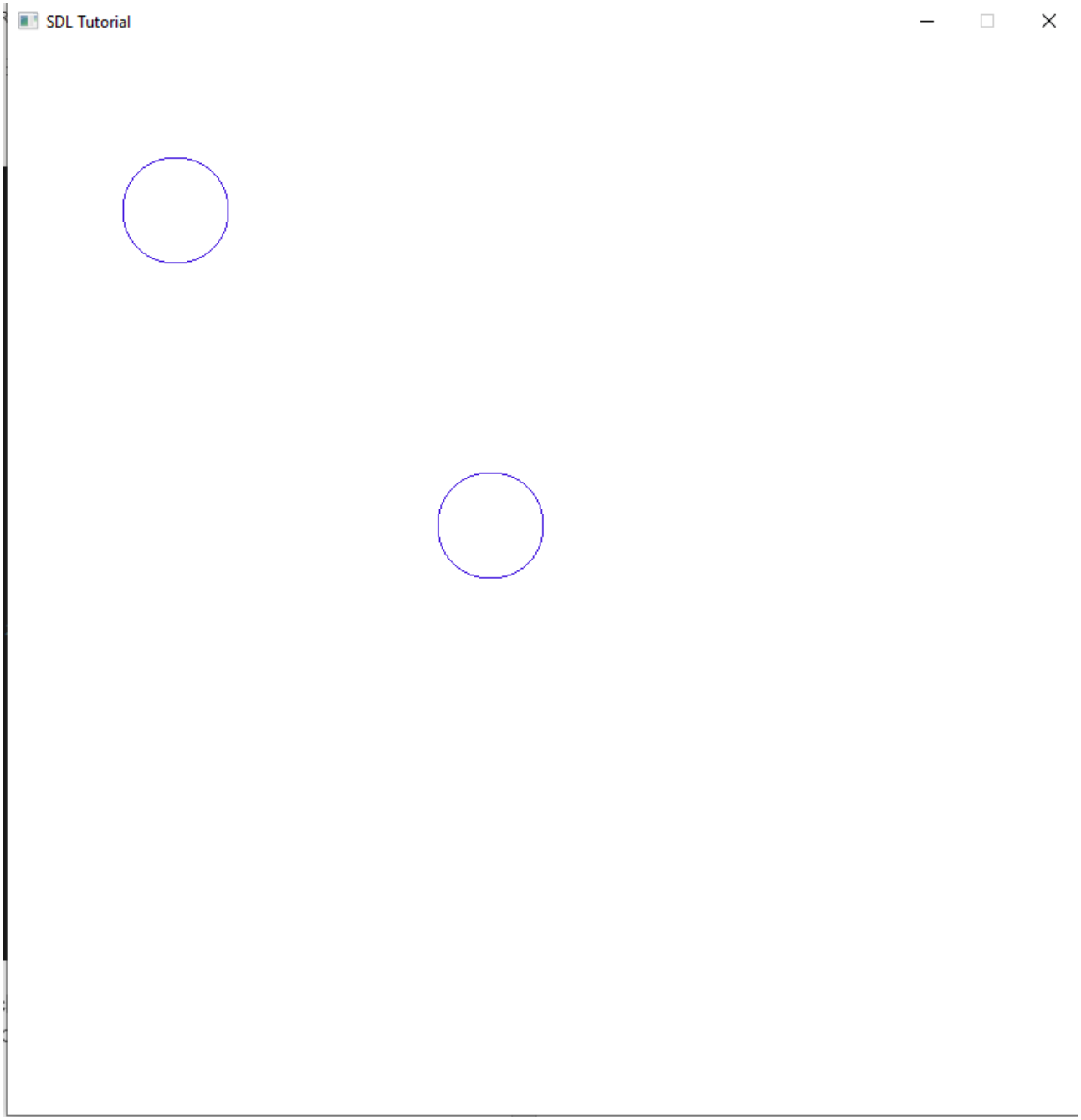
```
void CollidingObject::Render(MODES mode)
{
    if (mode == MODES::SQUARE)
    {
        SDL_RenderDrawRect(renderer, &box);
    }
    else if (mode == MODES::CIRCLE)
    {
        int radius = SIZE / 2;
        DrawCircle(renderer, x + radius, y + radius, radius);
    }
    else
    {
        SDL_RenderDrawRect(renderer, &box);
    }
}
```

The state enum will also decide what render method to call for the object.

```
int main()
{
    SDL_SetMainReady();
    bool success = Init();

    MODES mode = MODES::SQUARE;

    SDL_Event e;

    //While application is running
    while (running && success)
    {
        //Handle events on queue
        while (SDL_PollEvent(&e) != 0)
        {
            //User requests quit
            if (e.type == SDL_QUIT)
            {
                running = false;
            }
            else if (e.type == SDL_KEYDOWN)
            {
                switch (e.key.keysym.sym)
                {
                case SDLK_1:
                    mode = MODES::SQUARE;
                    break;
                case SDLK_2:
                    mode = MODES::CIRCLE;
                    break;
                }
            }

        }
        SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
        SDL_RenderClear(renderer);


        object->Move(mode);

        object->Render(mode);
        wall->Render(mode);

        SDL_RenderPresent(renderer);

    }
    Close();

    return 0;
}
```

In the game loop the screen will first be cleared. Then it will update the moving object. Then it will render each object to the screen.


SDL Tutorial


SDL Tutorial