Raylib: A Powerful Building Block For Old and New

Zane O'Dell

Whether you use powerful tools such as Unreal Engine or are writing your game in VIM, C++ is a powerful language that has the capability to create any number of games, using almost any number of software tools.

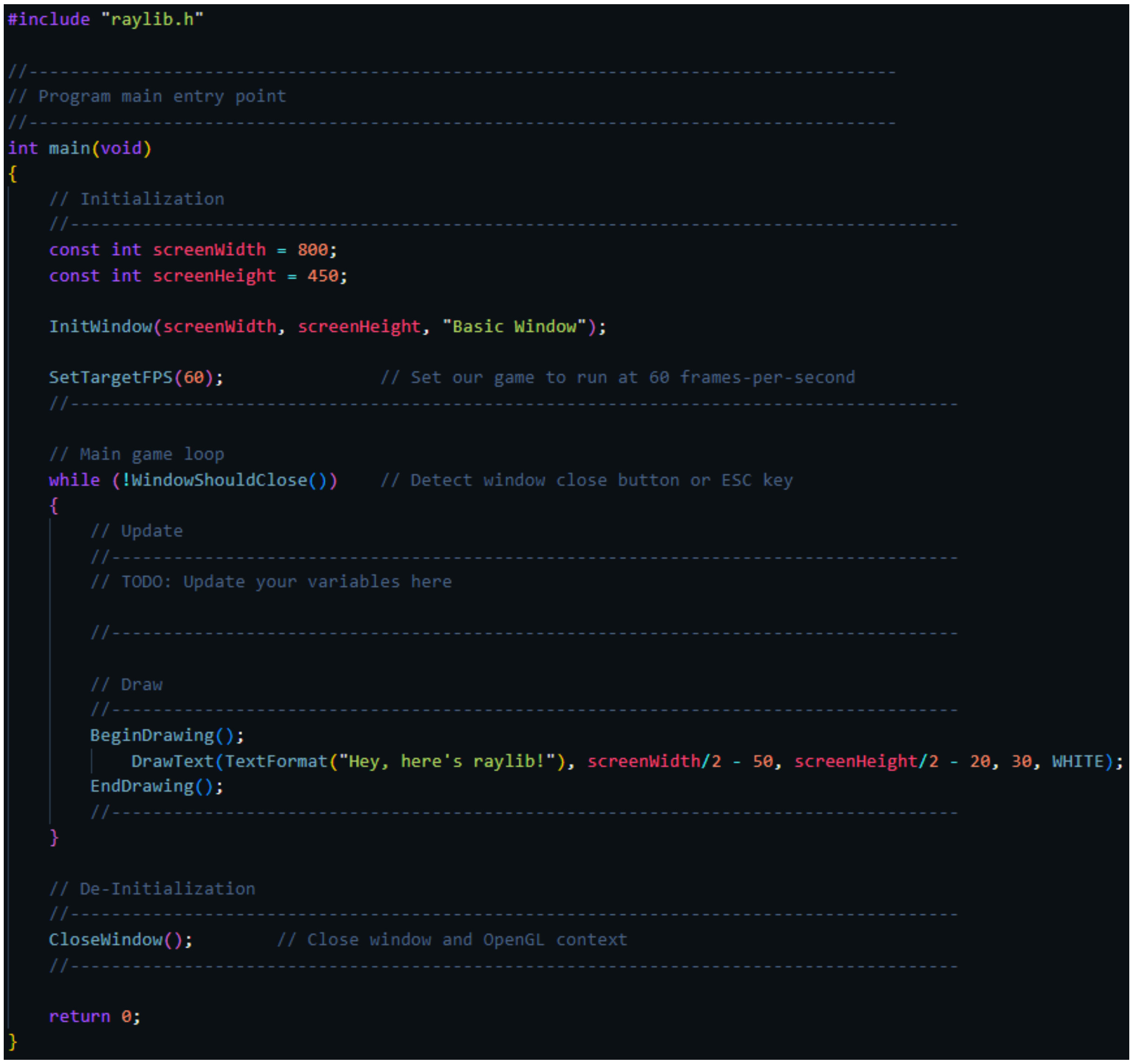
One such tool that can be used is a library written in native C called Raylib. Raylib is a library used for creating games, and has various functions for different aspects of game development. One thing Raylib prides itself on is its lack of a GUI or external tools/editors. Raylib, as they claim on their official website (Raylib), does not provide a large set of API documentation or a plethora of tutorials for programmers to utilize, and encourages its users to dive right into making things with its API and getting comfortable using its library cheat sheet to gain knowledge of its inner workings.

With all of that being said, let's dive right into using Raylib to create a simple game using its tools. The kind of game that will be made is a simple catch game, in which the player will move left and right across the screen catching an item that falls from the top of the screen while avoiding other obstacles that fall from the top of the screen.

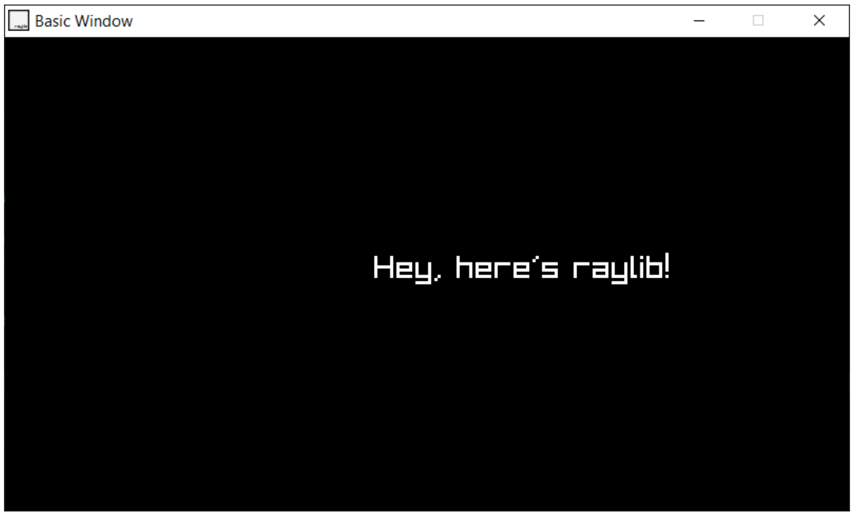
In getting started with Raylib, I followed the install instructions on the website, and using Visual Studio Code as my IDE of preference, we can start creating our simple game.

In order to use Raylib with C++, there are some additional steps to set it up in VSCode. It was a bit difficult to get it set up with my IDE. However, I did find a tutorial to be helpful that is linked here: Raylib VSCode tutorial

First, we can start by creating a simple window and establishing a sort of "game loop", and print a message to the screen.



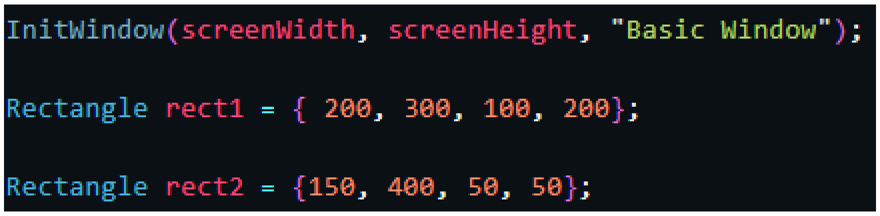
```
#include "raylib.h"

//------------------------------------------------------------------------------------
// Program main entry point
//------------------------------------------------------------------------------------
int main(void)
{
    // Initialization
    //--------------------------------------------------------------------------------------
    const int screenWidth = 800;
    const int screenHeight = 450;

    InitWindow(screenWidth, screenHeight, "Basic Window");

    SetTargetFPS(60);               // Set our game to run at 60 frames-per-second
    //--------------------------------------------------------------------------------------

    // Main game loop
    while (!WindowShouldClose())    // Detect window close button or ESC key
    {
        // Update
        //----------------------------------------------------------------------------------
        // TODO: Update your variables here

        //----------------------------------------------------------------------------------

        // Draw
        //----------------------------------------------------------------------------------
        BeginDrawing();
            DrawText(TextFormat("Hey, here's raylib!"), screenWidth/2 - 50, screenHeight/2 - 20, 30, WHITE);
        EndDrawing();
        //----------------------------------------------------------------------------------
    }

    // De-Initialization
    //--------------------------------------------------------------------------------------
    CloseWindow();        // Close window and OpenGL context
    //--------------------------------------------------------------------------------------

    return 0;
}
```



Hey, here's raylib!

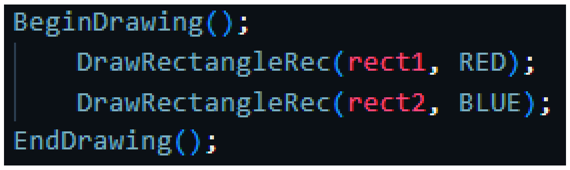
Now that we have a window up and running, let’s try to put some objects on the screen. Raylib makes this very easy in just a few function calls and struct instance creations.

We can create Rectangles to draw on the screen by creating instances of a Rectangle struct like so, with the values from left to right being the rectangle’s x position, y position, width, and height.



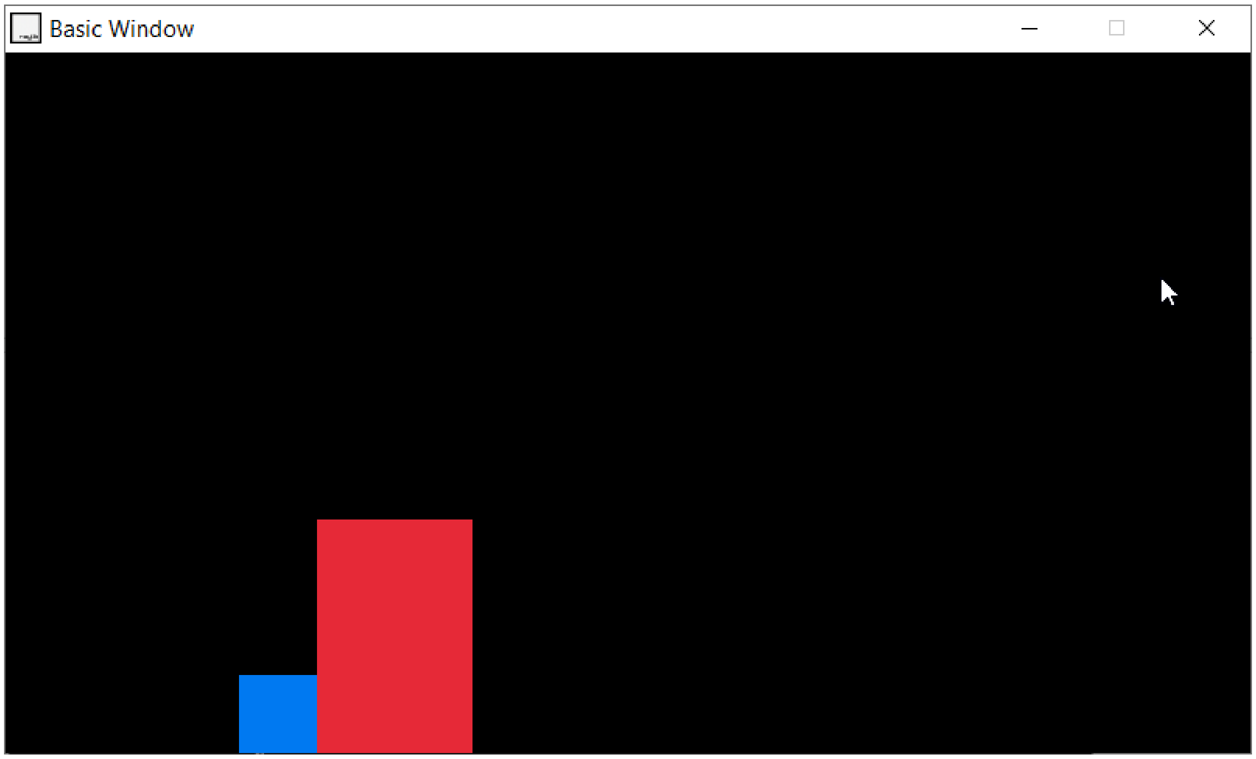
```
InitWindow(screenWidth, screenHeight, "Basic Window");

Rectangle rect1 = { 200, 300, 100, 200};

Rectangle rect2 = {150, 400, 50, 50};
```

Now, to draw them to the screen we can call the DrawRectangleRec function, and pass it the rectangle we want to draw and a color to draw the rectangle in.(Side note, if you want to draw a rectangle without creating one beforehand, there is a DrawRectangle function that takes the four parameters to create a rectangle as well as a color to draw it in.



```
BeginDrawing();
    DrawRectangleRec(rect1, RED);
    DrawRectangleRec(rect2, BLUE);
EndDrawing();
```

The result of the previous lines of code is:



While getting rectangles to display on the screen is nice, we should have our rectangles move. One should act as our item to catch and the other the item we are controlling to catch the falling item.

We can save the positions of our rectangles as variables and using Raylib's GetFrameTime function, we can replicate a behavior similar to Unity's Update and move the rectangles. And, we can also detect collisions using the CheckCollisionRecs function.

Continuing on, we can add textures using simple functions to draw textures on the screen given a specified width and height.

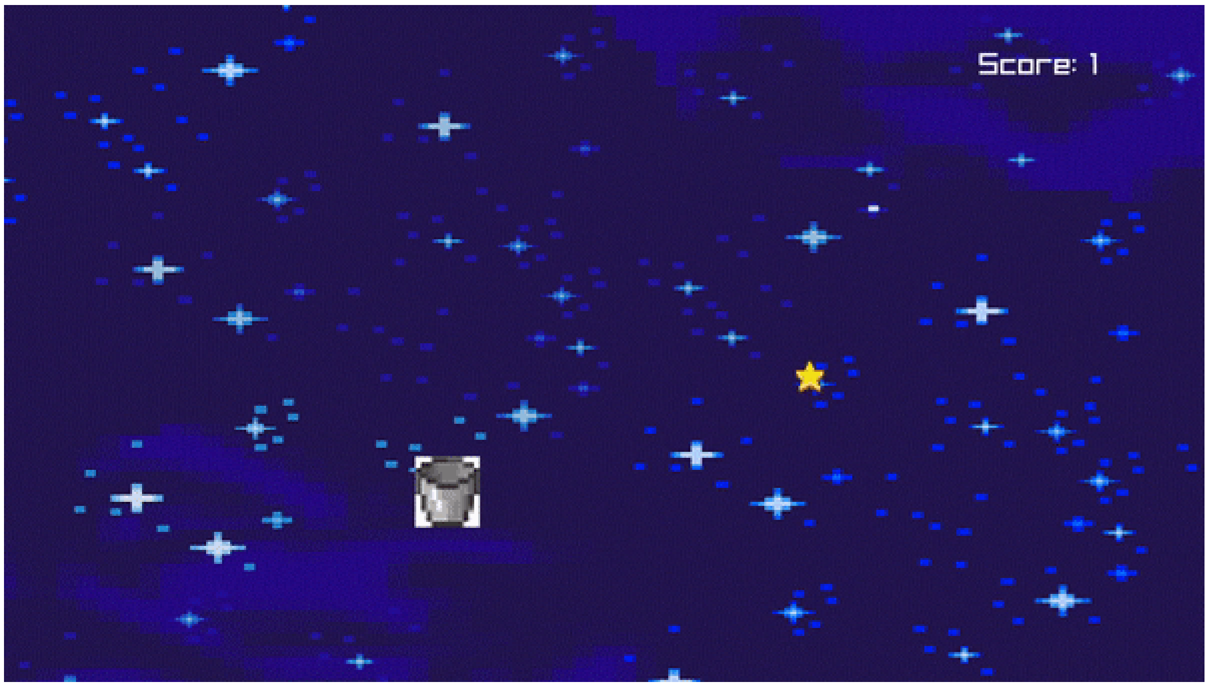
```
/*****************************************************************************
**************
*
*   raylib [core] example - Basic window
*
*   Welcome to raylib!
```

```
*
*   To test examples, just press F6 and execute raylib_compile_execute script
*   Note that compiled executable is placed in the same folder as .c file
*
*   You can find all basic examples on C:\raylib\raylib\examples folder or
*   raylib official webpage: www.raylib.com
*
*   Enjoy using raylib. :)
*
*   Example originally created with raylib 1.0, last time updated with raylib 1.0
*
*   Example licensed under an unmodified zlib/libpng license, which is an OSI-certified,
*   BSD-like license that allows static linking with closed source software
*
*   Copyright (c) 2013-2023 Ramon Santamaria (@raysan5)
*
**************************************************************************************
****************/

#include "raylib.h"


//------------------------------------------------------------------------------------
// Program main entry point
//------------------------------------------------------------------------------------
int main(void)
{
    // Initialization
    //--------------------------------------------------------------------------------------
    const int screenWidth = 800;
    const int screenHeight = 450;

    InitWindow(screenWidth, screenHeight, "Star Catcher");

    float bucketPosX = 400;

    int starPosX = GetRandomValue(100, 700);
```

```
float starPosY = 100;

Texture2D starTexture;

Texture2D bucketTexture;

Texture2D backgroundTexture;

starTexture = LoadTexture("src/singlestar.png");

bucketTexture = LoadTexture("src/minecraftbucket.png");

backgroundTexture = LoadTexture("src/starrysky.png");

Rectangle bucketRect = { bucketPosX, 300, 41, 46};

Rectangle starRect = {static_cast<float>(starPosX), starPosY, 20, 22};

bool collision;

int score = 0;


SetTargetFPS(60);               // Set our game to run at 60 frames-per-second
//--------------------------------------------------------------------------------------

// Main game loop
while (!WindowShouldClose())    // Detect window close button or ESC key
{
    // Update
    //----------------------------------------------------------------------------------
    // TODO: Update your variables here

    if (IsKeyDown(KEY_RIGHT)){
        bucketPosX += GetFrameTime() * 150;
        bucketRect.x += GetFrameTime() * 150;
    }
```

```
        if (IsKeyDown(KEY_LEFT)){
            bucketPosX -= GetFrameTime() * 150;
            bucketRect.x -= GetFrameTime() * 150;
        }

        starPosY += GetFrameTime() * 75;
        starRect.y += GetFrameTime() * 75;

        collision = CheckCollisionRecs(bucketRect, starRect);

        if (starPosY >= screenHeight + 10 || collision){
            starPosY = 100;
            starPosX = GetRandomValue(100, 700);
            starRect.y = starPosY;
            starRect.x = starPosX;
            if (collision){
                score += 1;
            }
            else{
                score = 0;
            }
        }
        //----------------------------------------------------------------------------------

        // Draw
        //----------------------------------------------------------------------------------
        BeginDrawing();
            DrawTexture(backgroundTexture, 0, 0, WHITE);
            DrawTexture(starTexture, starPosX, starPosY, WHITE);
            DrawTexture(bucketTexture, bucketPosX, 300, WHITE);
            DrawText(TextFormat("Score: %d", score), 650, 30, 20, WHITE);
        EndDrawing();
        //----------------------------------------------------------------------------------
    }

    // De-Initialization
    //--------------------------------------------------------------------------------------
    CloseWindow();        // Close window and OpenGL context
    //----------------------------------------------------------------------------------


    return 0;
}
```

Our final result is this:



Overall, Raylib is extremely easy to use and learn. While the official website does not directly have a lot of tutorials to boast, there are plenty of external resources outside of the documentation that programmers can utilize in their game creation journey. Overall, I would give Raylib a more than satisfactory rating and would recommend it to any and all C/C++ programmers looking to make their own video games.