UPHILL RACER GAME

By – andrew steev jacob
Sap - 590028964

USING RAYLIB

WHAT IS RAYLIB ?

- Raylib is a **simple, beginner-friendly game development library** written in C.
  It helps you create **2D and 3D games** easily without needing complicated engines.
- Created by Ramon Santmorie(RAY)
- No Engine, Just a Library Raylib is NOT Unity or Unreal.
- It does NOT:

  Give you an editor

  Handle physics automatically

  Give you a scene system

  You code everything manually — which is why it's great for learning.

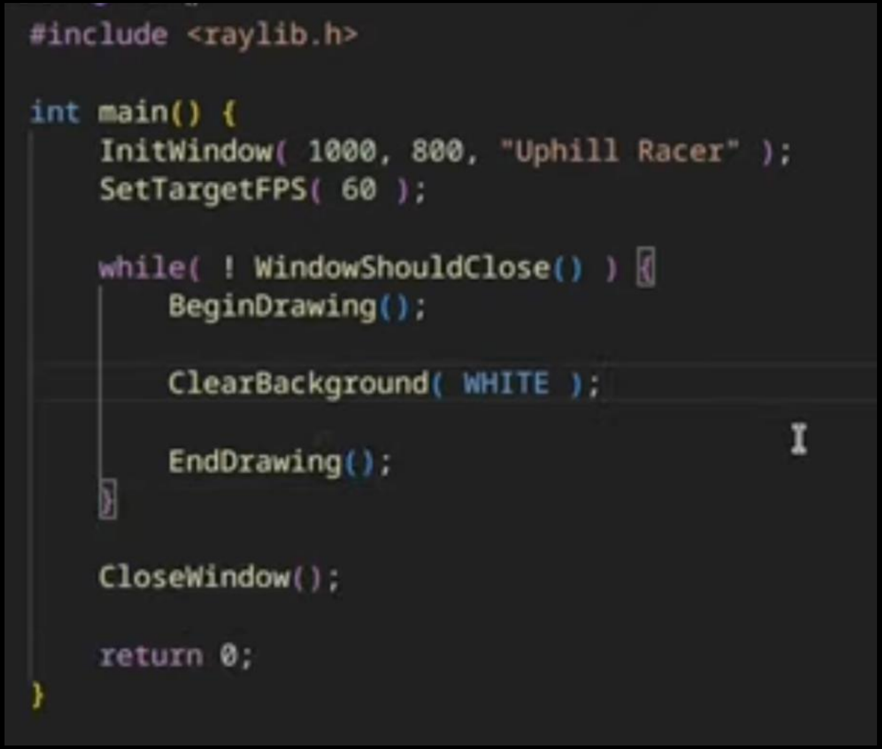
GOALS FOR THIS PROJECT

- Simulate gravity on 2d objects
- Simulate friction on 2d objects
- Simulate terrain collision on 2d objects
- Simulate car suspension
- Generate a randomly generated terrain

PART 1 (BASIC BOILER PLATE AND DRAWING)

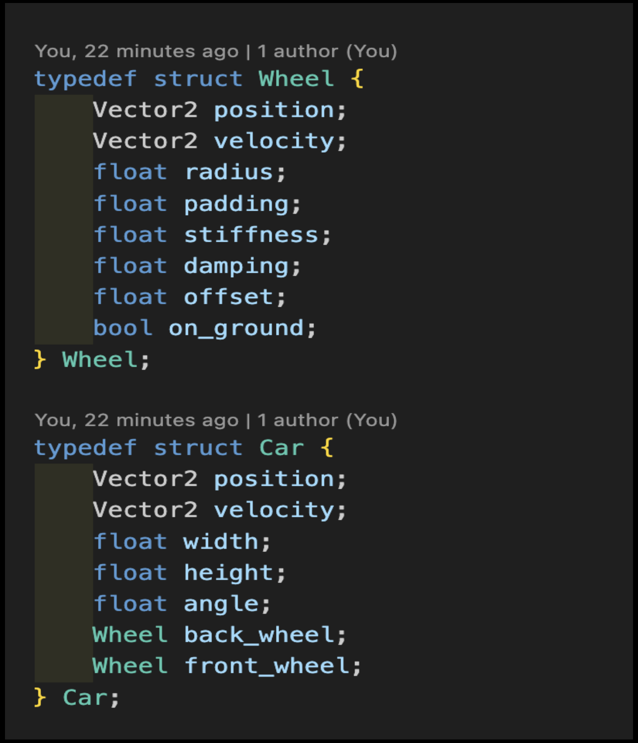
- First we create a basic window

This creates a blank window and now we add the main car



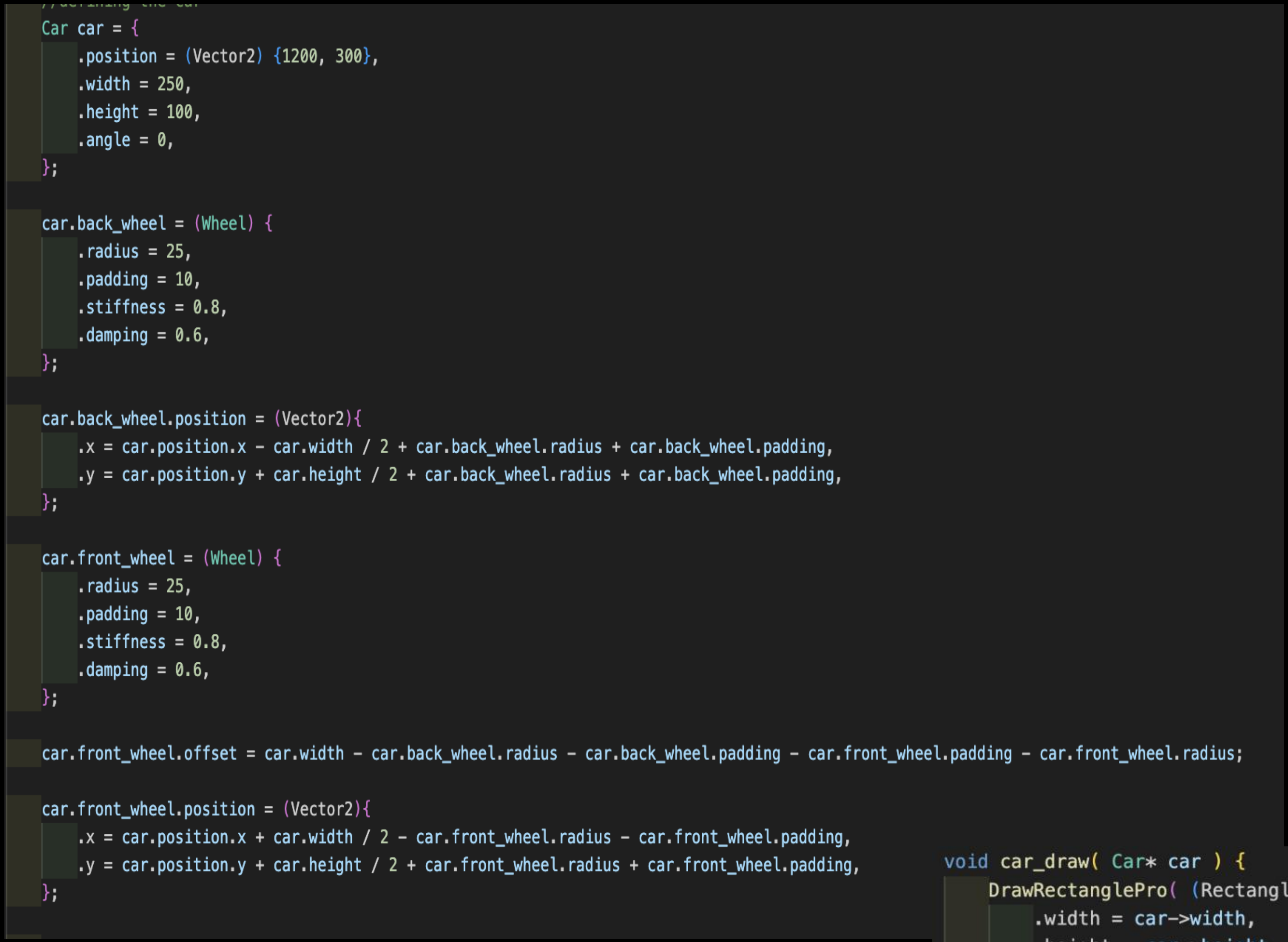
```c
#include <raylib.h>

int main() {
    InitWindow( 1000, 800, "Uphill Racer" );
    SetTargetFPS( 60 );

    while( ! WindowShouldClose() ) {
        BeginDrawing();

        ClearBackground( WHITE );

        EndDrawing();
    }

    CloseWindow();

    return 0;
}
```

NOW TO ADD THE CAR

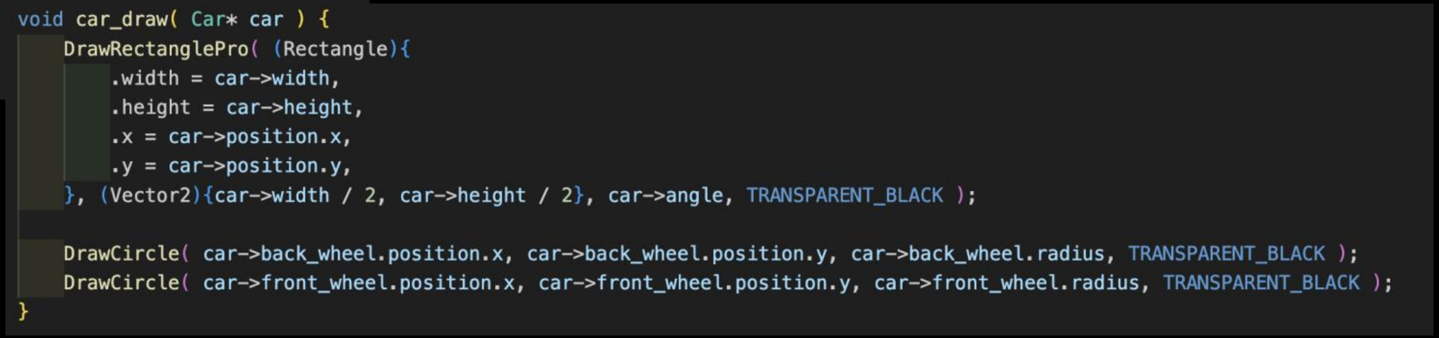


```c
You, 22 minutes ago | 1 author (You)
typedef struct Wheel {
    Vector2 position;
    Vector2 velocity;
    float radius;
    float padding;
    float stiffness;
    float damping;
    float offset;
    bool on_ground;
} Wheel;

You, 22 minutes ago | 1 author (You)
typedef struct Car {
    Vector2 position;
    Vector2 velocity;
    float width;
    float height;
    float angle;
    Wheel back_wheel;
    Wheel front_wheel;
} Car;
```

DRAWING THE CAR



```c
Car car = {
    .position = (Vector2) {1200, 300},
    .width = 250,
    .height = 100,
    .angle = 0,
};

car.back_wheel = (Wheel) {
    .radius = 25,
    .padding = 10,
    .stiffness = 0.8,
    .damping = 0.6,
};

car.back_wheel.position = (Vector2){
    .x = car.position.x - car.width / 2 + car.back_wheel.radius + car.back_wheel.padding,
    .y = car.position.y + car.height / 2 + car.back_wheel.radius + car.back_wheel.padding,
};

car.front_wheel = (Wheel) {
    .radius = 25,
    .padding = 10,
    .stiffness = 0.8,
    .damping = 0.6,
};

car.front_wheel.offset = car.width - car.back_wheel.radius - car.back_wheel.padding - car.front_wheel.padding - car.front_wheel.radius;

car.front_wheel.position = (Vector2){
    .x = car.position.x + car.width / 2 - car.front_wheel.radius - car.front_wheel.padding,
    .y = car.position.y + car.height / 2 + car.front_wheel.radius + car.front_wheel.padding,
};
```



```c
void car_draw( Car* car ) {
    DrawRectanglePro( (Rectangle){
        .width = car->width,
        .height = car->height,
        .x = car->position.x,
        .y = car->position.y,
    }, (Vector2){car->width / 2, car->height / 2}, car->angle, TRANSPARENT_BLACK );

    DrawCircle( car->back_wheel.position.x, car->back_wheel.position.y, car->back_wheel.radius, TRANSPARENT_BLACK );
    DrawCircle( car->front_wheel.position.x, car->front_wheel.position.y, car->front_wheel.radius, TRANSPARENT_BLACK );
}
```

Using struct we were able to assign values to the car for example the values of HEIGHT,WIDTH and everything else that we might need in the code in future

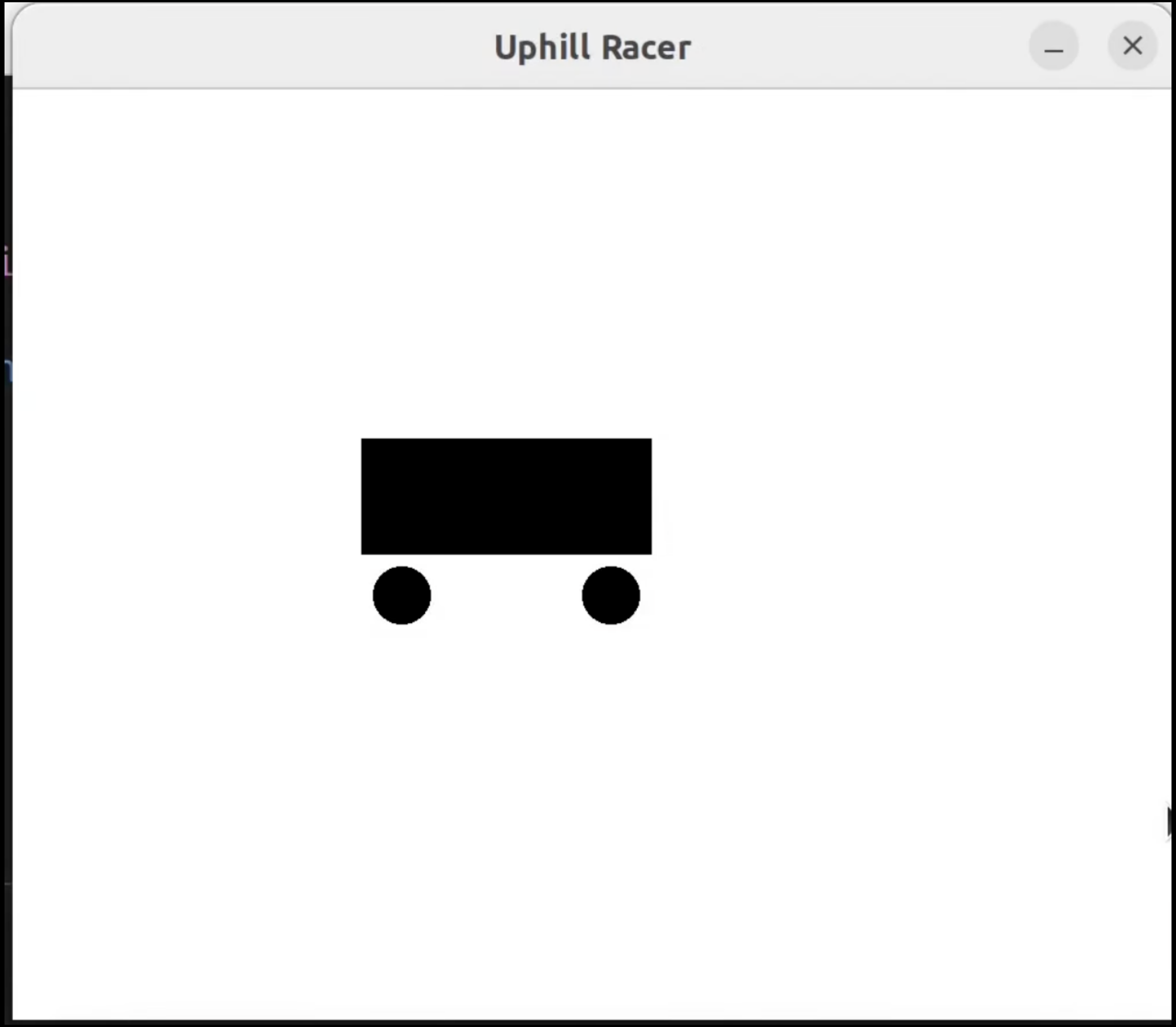
- Using a basic formula I was able to assign values of both the front and back wheel with respect to the car , we need to write the coordinate of wheels with respect to car both in x and y axis

backwheel for example x coordinate is written as

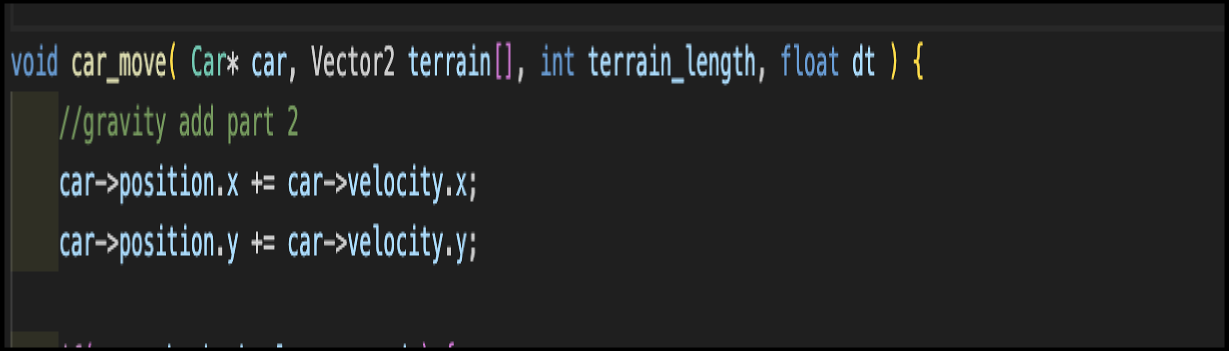
.x = car position – car width /2 + car backwheel radius + backwheel padding

Padding is the extra space in between the wheels

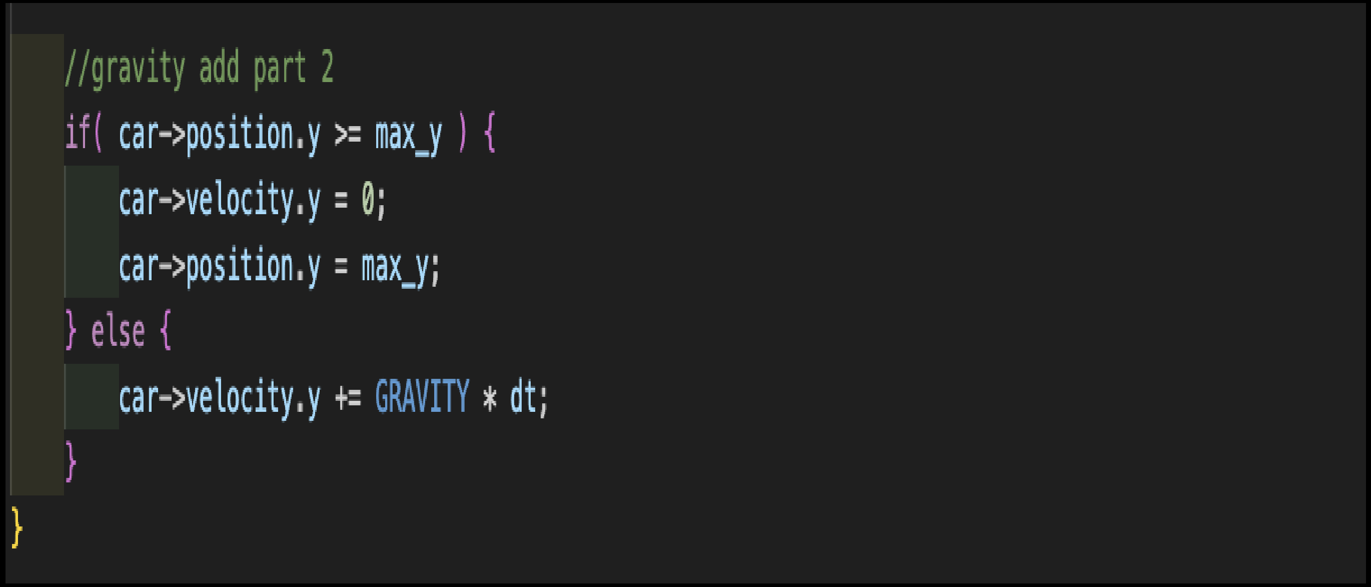
FINAL OUTPUT



PART2 – GRAVITY



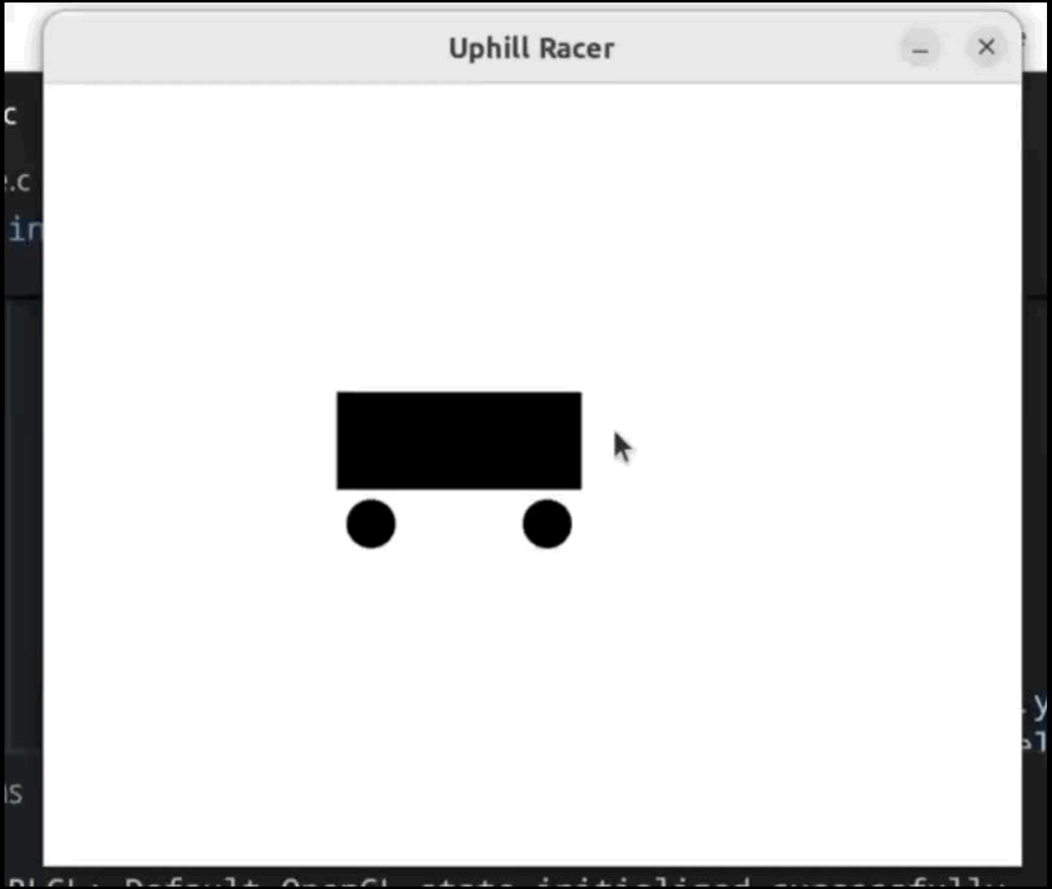
```
void car_move( Car* car, Vector2 terrain[], int terrain_length, float dt ) {
    //gravity add part 2
    car->position.x += car->velocity.x;
    car->position.y += car->velocity.y;
```



```
    //gravity add part 2
    if( car->position.y >= max_y ) {
        car->velocity.y = 0;
        car->position.y = max_y;
    } else {
        car->velocity.y += GRAVITY * dt;
    }
}
```

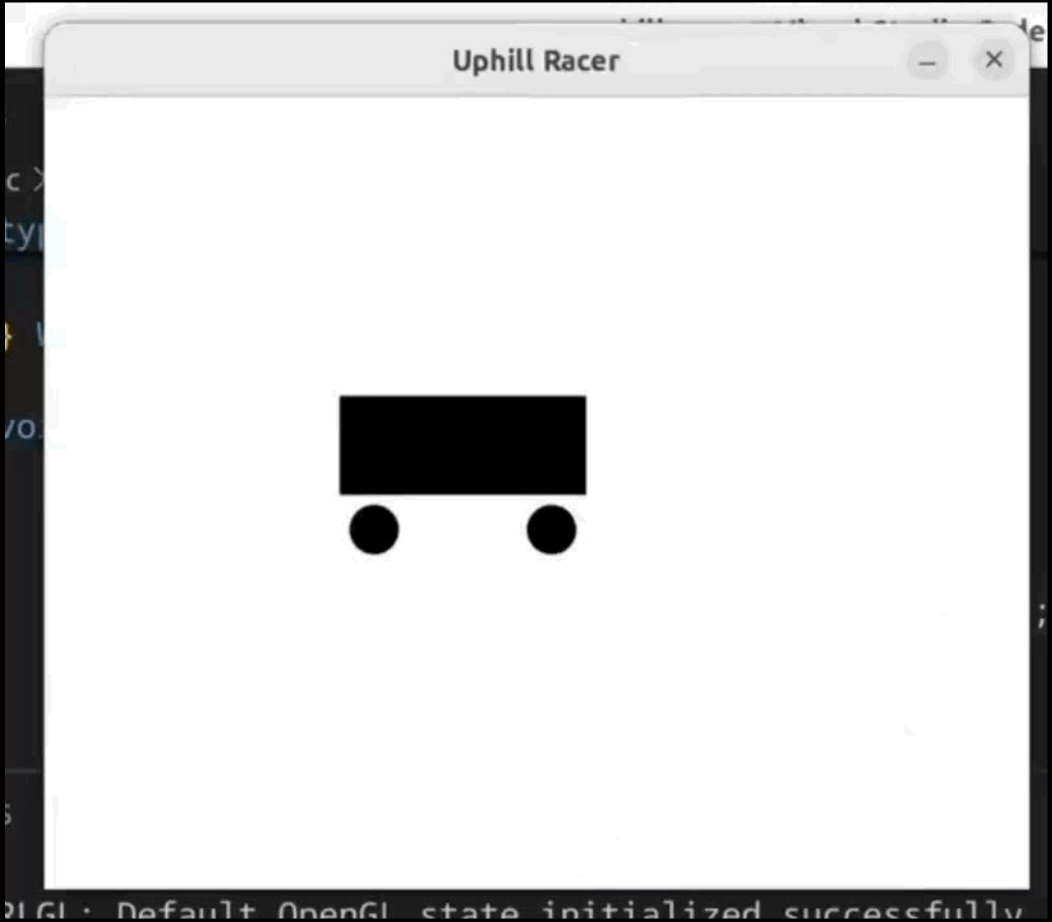
- Using a basic GetFrametime() function that is inbuild in raylib we are able to get per second rate of movement on screen
- Defining gravity at start in terms of pixels
- By defining a dt = GetFrametime() we can just multiply the dt with the GRAVITY and add it to the position of the car , moslty acts in y axis but for future changes added changes in x axis also .
- Used the if statement so that when the car reaches the bottom on the defined screen , it stops moving

WITH NO IF CONDITION.



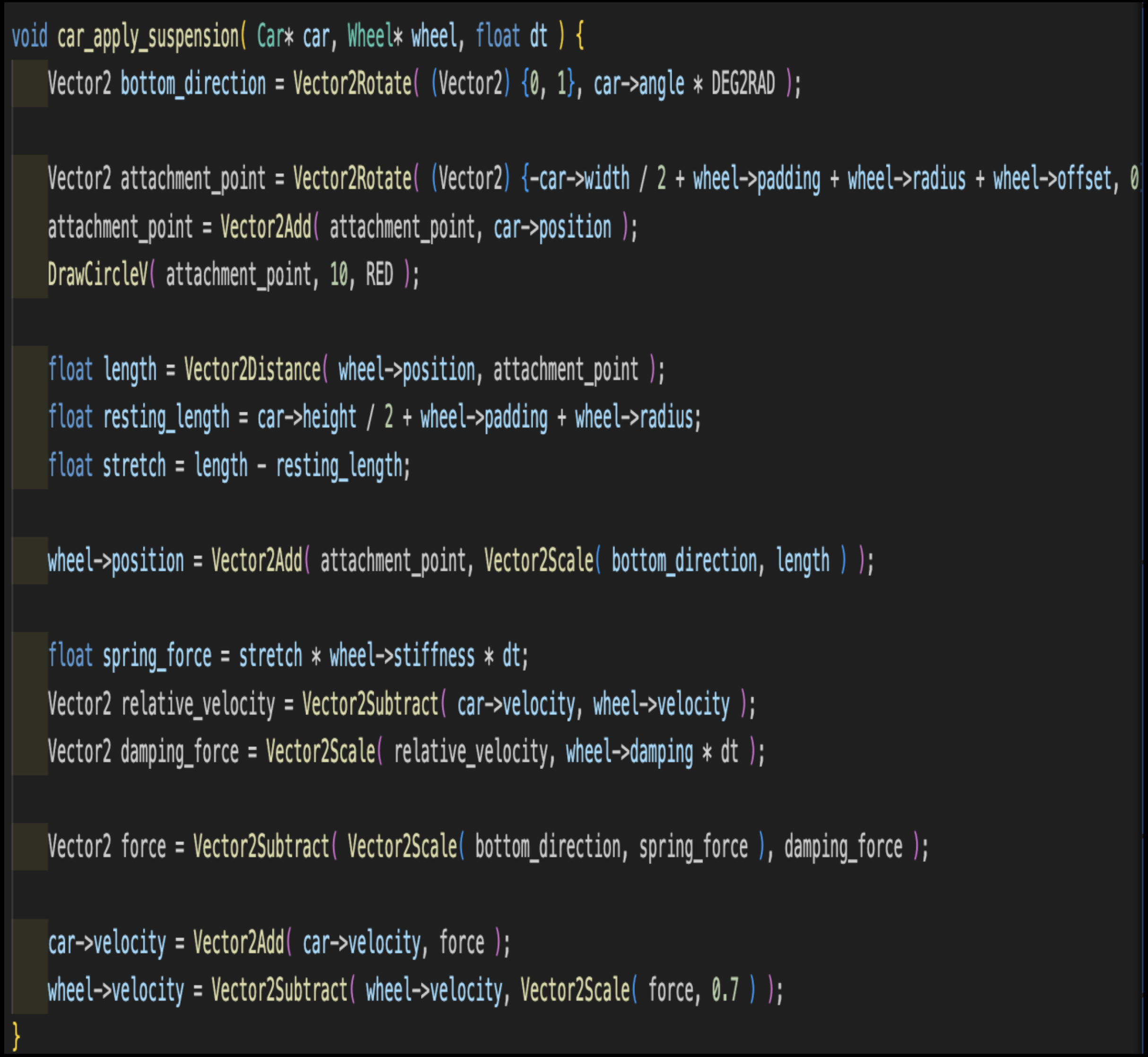
```
RLGL: Default OpenGL state initialized successfully
```

WITH IF CONDITION.



```
RLGL: Default OpenGL state initialized successfully
```

PART3 – CAR SUSPENSION



```c
void car_apply_suspension( Car* car, Wheel* wheel, float dt ) {
    Vector2 bottom_direction = Vector2Rotate( (Vector2) {0, 1}, car->angle * DEG2RAD );

    Vector2 attachment_point = Vector2Rotate( (Vector2) {-car->width / 2 + wheel->padding + wheel->radius + wheel->offset, 0
    attachment_point = Vector2Add( attachment_point, car->position );
    DrawCircleV( attachment_point, 10, RED );

    float length = Vector2Distance( wheel->position, attachment_point );
    float resting_length = car->height / 2 + wheel->padding + wheel->radius;
    float stretch = length - resting_length;

    wheel->position = Vector2Add( attachment_point, Vector2Scale( bottom_direction, length ) );

    float spring_force = stretch * wheel->stiffness * dt;
    Vector2 relative_velocity = Vector2Subtract( car->velocity, wheel->velocity );
    Vector2 damping_force = Vector2Scale( relative_velocity, wheel->damping * dt );

    Vector2 force = Vector2Subtract( Vector2Scale( bottom_direction, spring_force ), damping_force );

    car->velocity = Vector2Add( car->velocity, force );
    wheel->velocity = Vector2Subtract( wheel->velocity, Vector2Scale( force, 0.7 ) );
}
```

Car supsension work on the basic idea of hooke's law , finding out the attachment point the resting lenth and the actual length

Calculating stretch = length – resting length

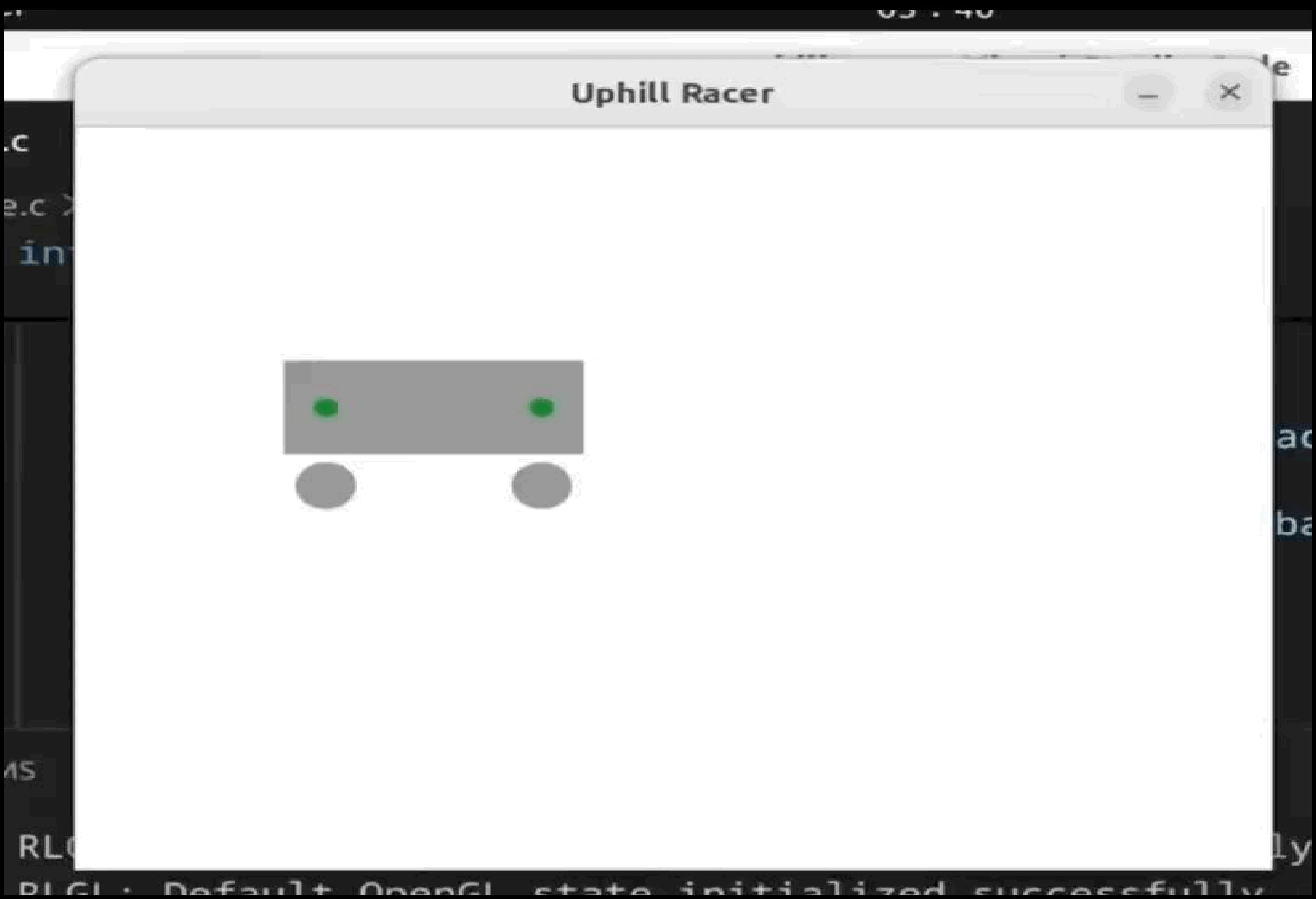
Calclulating the spring force = stretch * wheel stifness (more the stiffness more the continous bounce)

WRITING THE CODE JUST TILL HERE MAKES AN INFINITE LOOP

To slow down the total bouce of car we introduce DAMPING that decreases the bounce with gradual bounces

Updating the velocity and position of wheels according to the calculated values

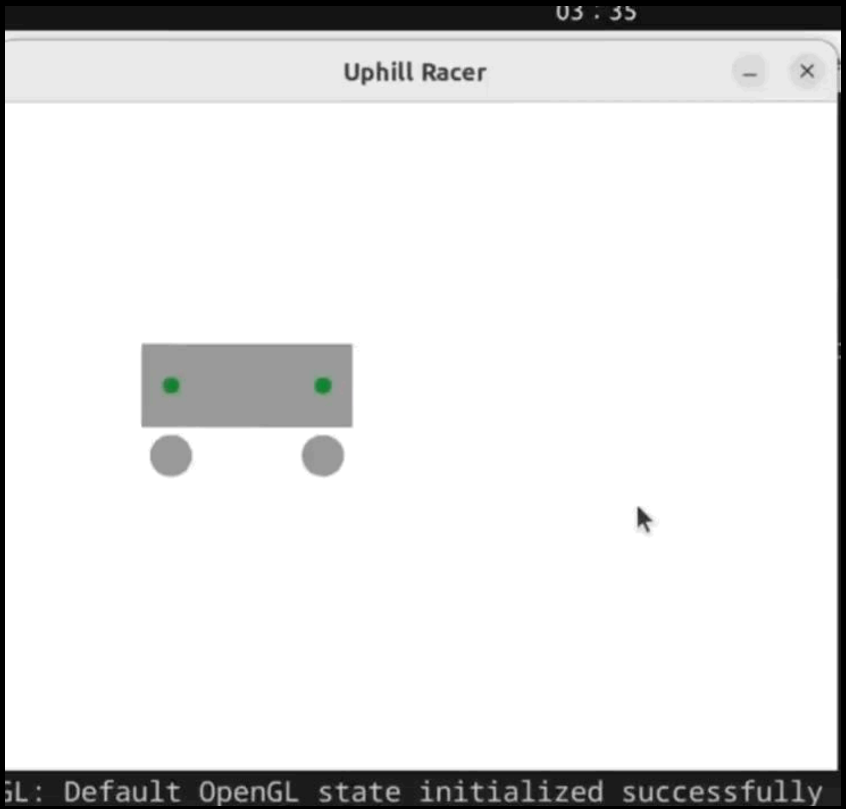
ENOUGH OF THEORY- FINAL OUTPUT (CAR SUSPENSION)



```
RLGL: Default OpenGL state initialized successfully
```

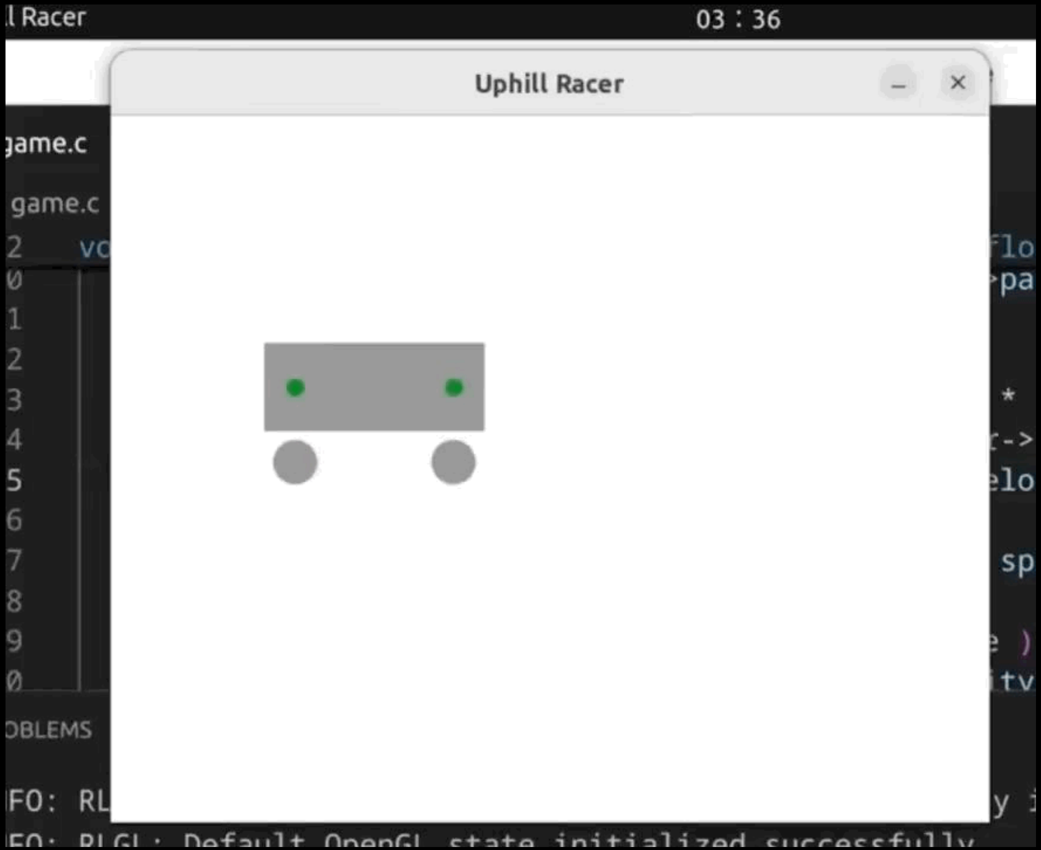
SOME FAILS

WEIRD WOBBLE



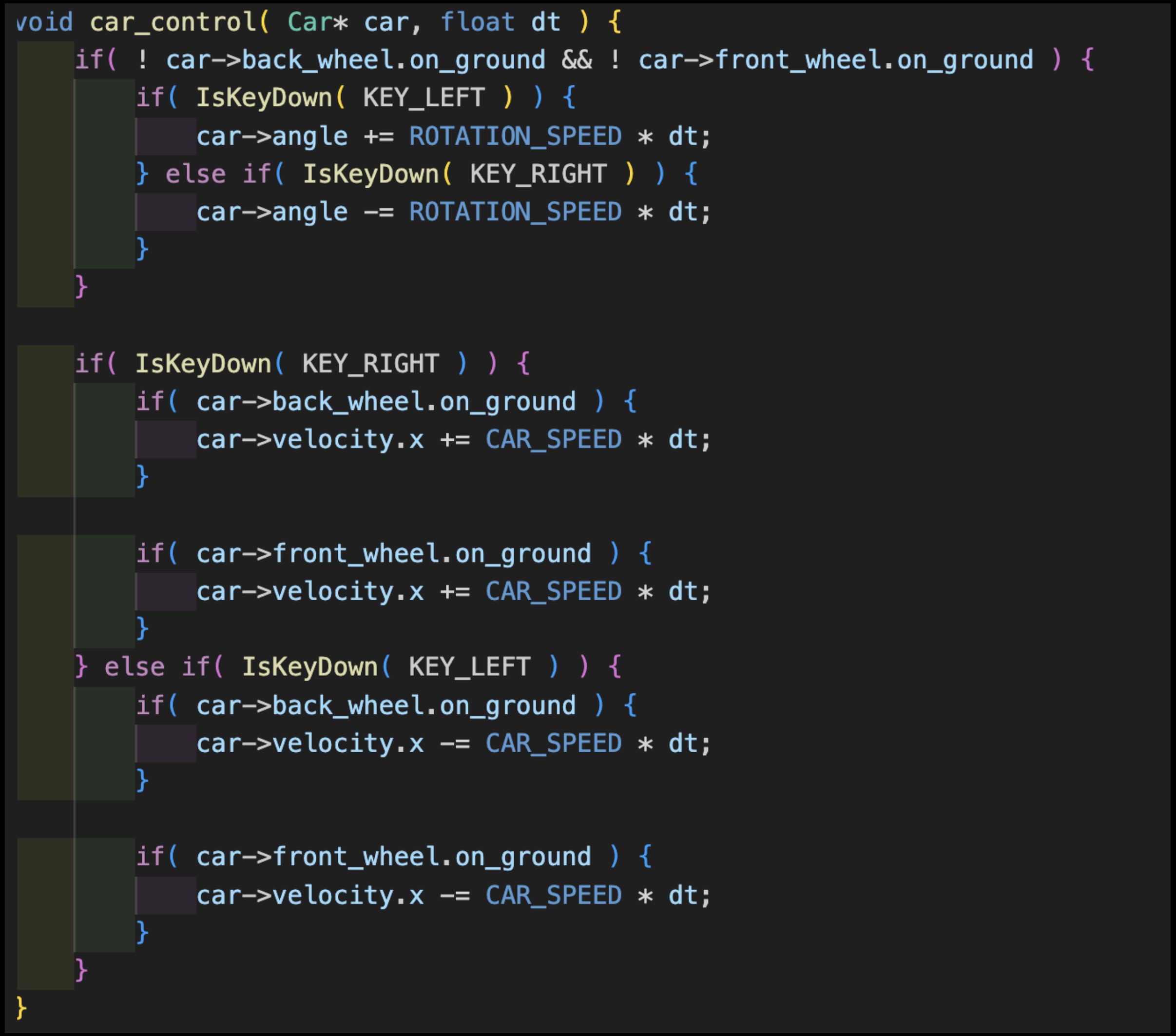
```
GL: Default OpenGL state initialized successfully
```

INFINITE BOUNCE



```
FO: RL
FO: RLGL: Default OpenGL state initialized successfully
```

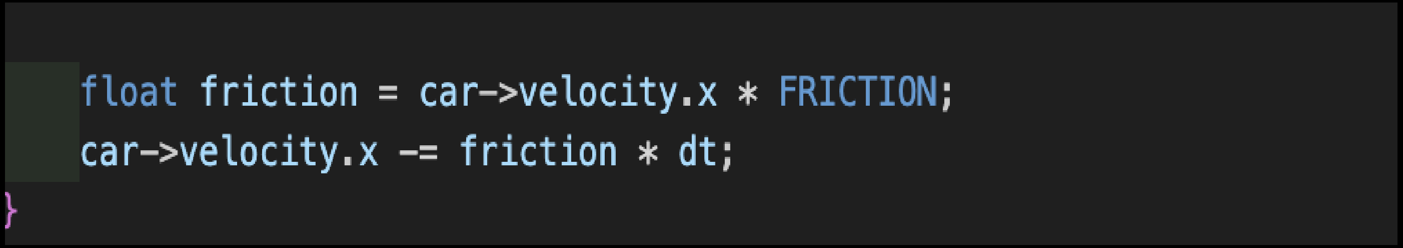
ASSIGNING BUTTONS FOR CAR TO MOVE



```c
void car_control( Car* car, float dt ) {
    if( ! car->back_wheel.on_ground && ! car->front_wheel.on_ground ) {
        if( IsKeyDown( KEY_LEFT ) ) {
            car->angle += ROTATION_SPEED * dt;
        } else if( IsKeyDown( KEY_RIGHT ) ) {
            car->angle -= ROTATION_SPEED * dt;
        }
    }

    if( IsKeyDown( KEY_RIGHT ) ) {
        if( car->back_wheel.on_ground ) {
            car->velocity.x += CAR_SPEED * dt;
        }

        if( car->front_wheel.on_ground ) {
            car->velocity.x += CAR_SPEED * dt;
        }
    } else if( IsKeyDown( KEY_LEFT ) ) {
        if( car->back_wheel.on_ground ) {
            car->velocity.x -= CAR_SPEED * dt;
        }

        if( car->front_wheel.on_ground ) {
            car->velocity.x -= CAR_SPEED * dt;
        }
    }
}
```

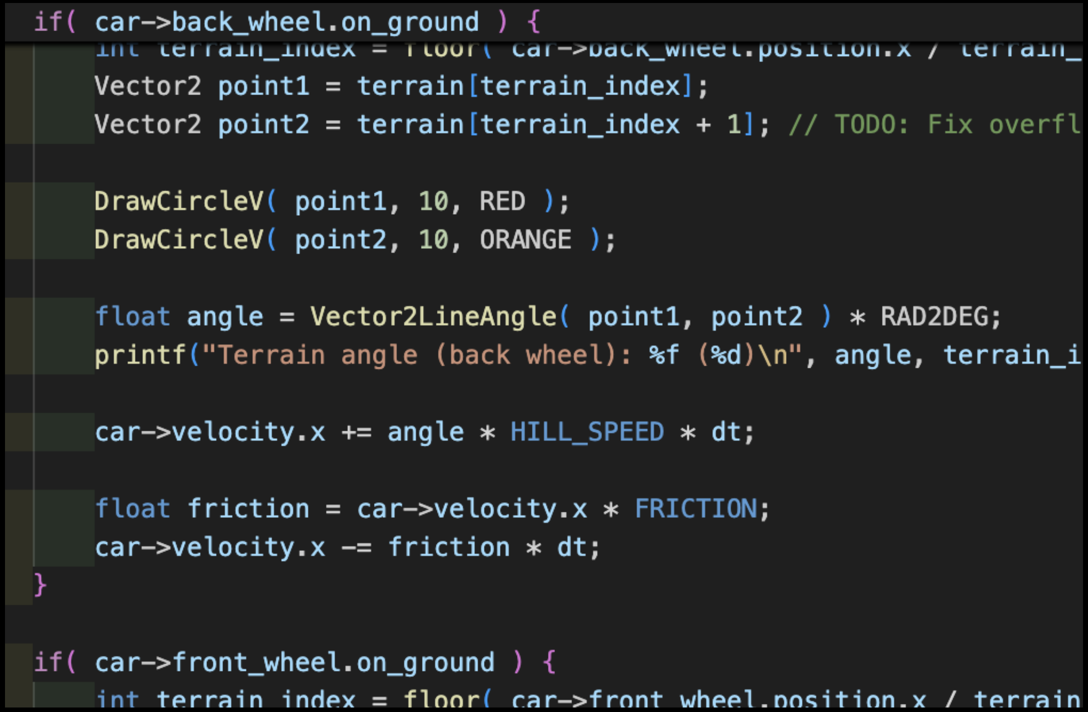
ADDING FRICTION

Used If - condition for both front and back wheel . So that friction only acts when the wheels of the car on the ground or detect a collision



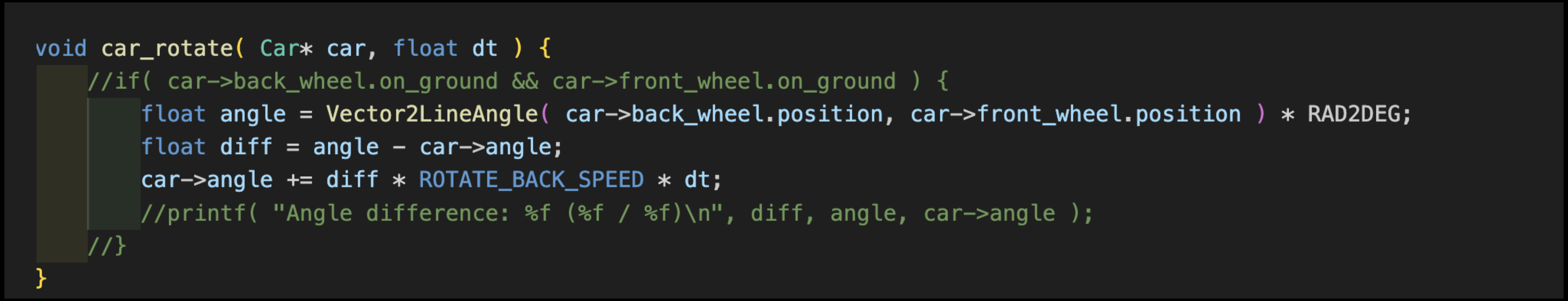
```
    float friction = car->velocity.x * FRICTION;
    car->velocity.x -= friction * dt;
}
```

Friction only works in x axis so , code to change the velocity in x axis with respect to friction



```
if( car->back_wheel.on_ground ) {
    int terrain_index = floor( car->back_wheel.position.x / terrain_
    Vector2 point1 = terrain[terrain_index];
    Vector2 point2 = terrain[terrain_index + 1]; // TODO: Fix overfl

    DrawCircleV( point1, 10, RED );
    DrawCircleV( point2, 10, ORANGE );

    float angle = Vector2LineAngle( point1, point2 ) * RAD2DEG;
    printf("Terrain angle (back wheel): %f (%d)\n", angle, terrain_i

    car->velocity.x += angle * HILL_SPEED * dt;

    float friction = car->velocity.x * FRICTION;
    car->velocity.x -= friction * dt;
}

if( car->front_wheel.on_ground ) {
    int terrain_index = floor( car->front_wheel.position.x / terrain
```

PART4 – ROTATING THE CAR

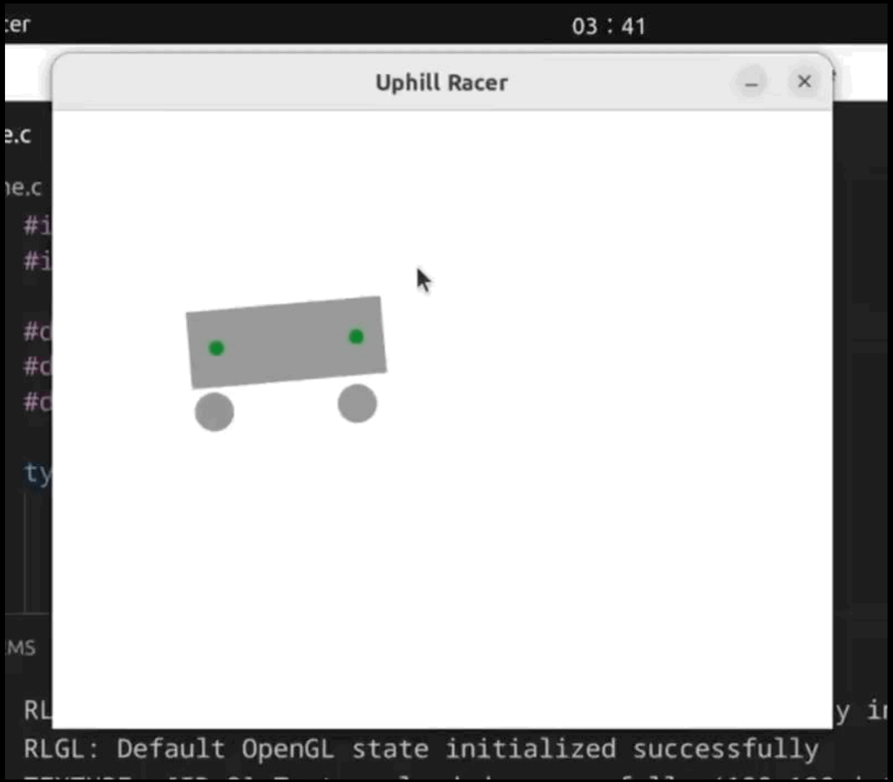


```
void car_rotate( Car* car, float dt ) {
    //if( car->back_wheel.on_ground && car->front_wheel.on_ground ) {
        float angle = Vector2LineAngle( car->back_wheel.position, car->front_wheel.position ) * RAD2DEG;
        float diff = angle - car->angle;
        car->angle += diff * ROTATE_BACK_SPEED * dt;
        //printf( "Angle difference: %f (%f / %f)\n", diff, angle, car->angle );
    //}
}
```

Defining the car angle in car struct and assigning it initial value 0 that is in degree we can use this basic Vectro2LIneAngle() function and since the position of back and front wheel have coordinate (x,y) we can change the new ANGLE in degree by (RAD2DEG)
And can change the rotation angle of the car , by press of some buttons

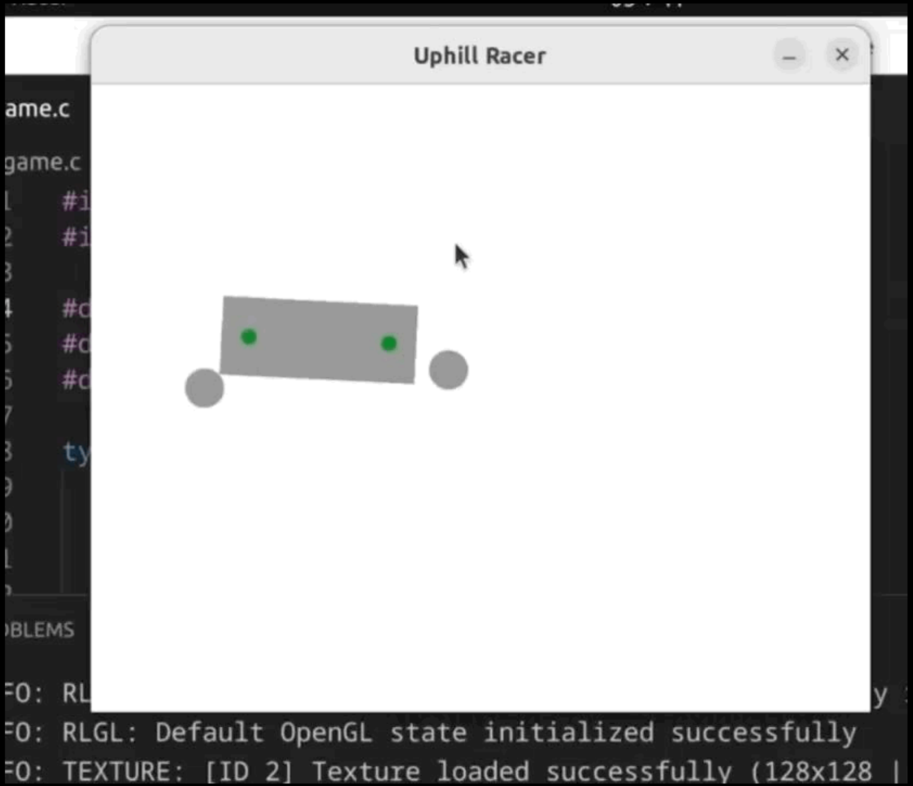
HOW IT LOOKS

- What I got



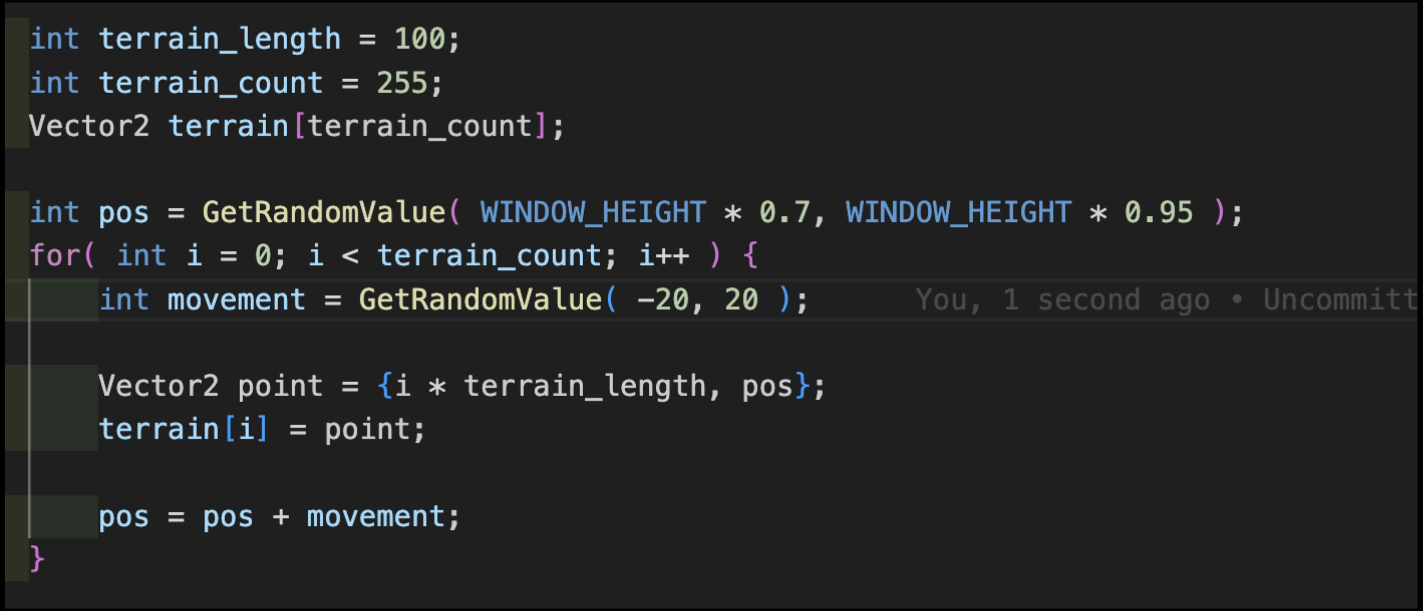
```
RLGL: Default OpenGL state initialized successfully
```

FAILS

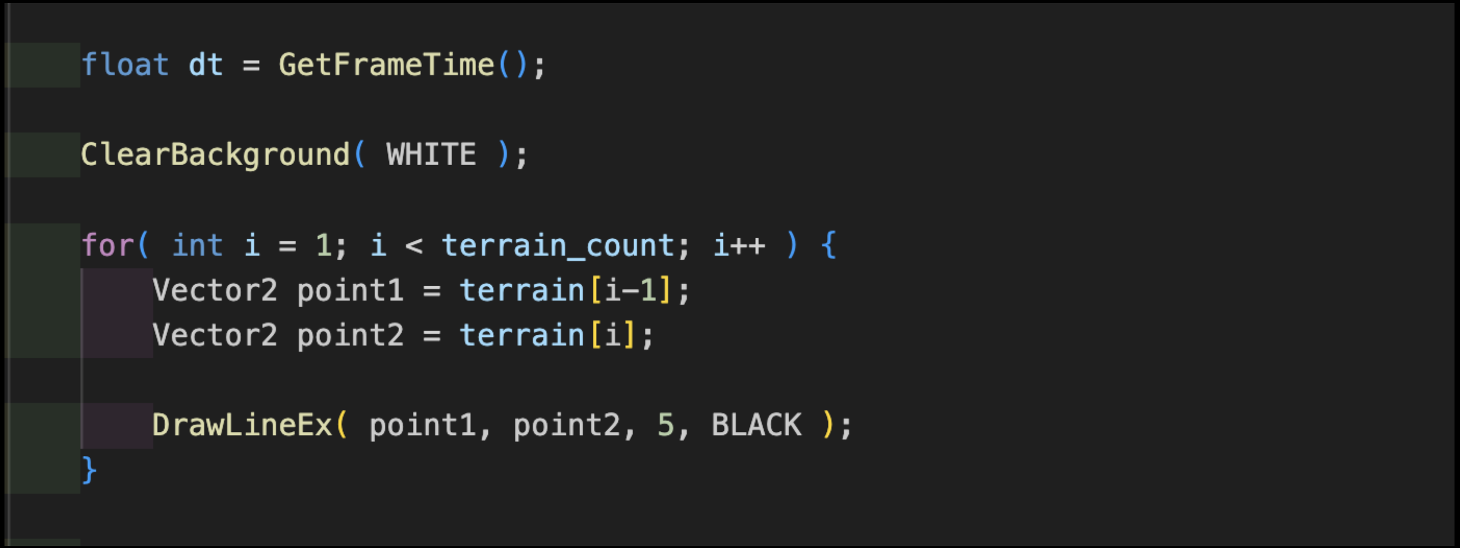


```
FO: RLGL: Default OpenGL state initialized successfully
FO: TEXTURE: [ID 2] Texture loaded successfully (128x128 |
```

MAKING RANDOMLY GENERATED TERRAIN

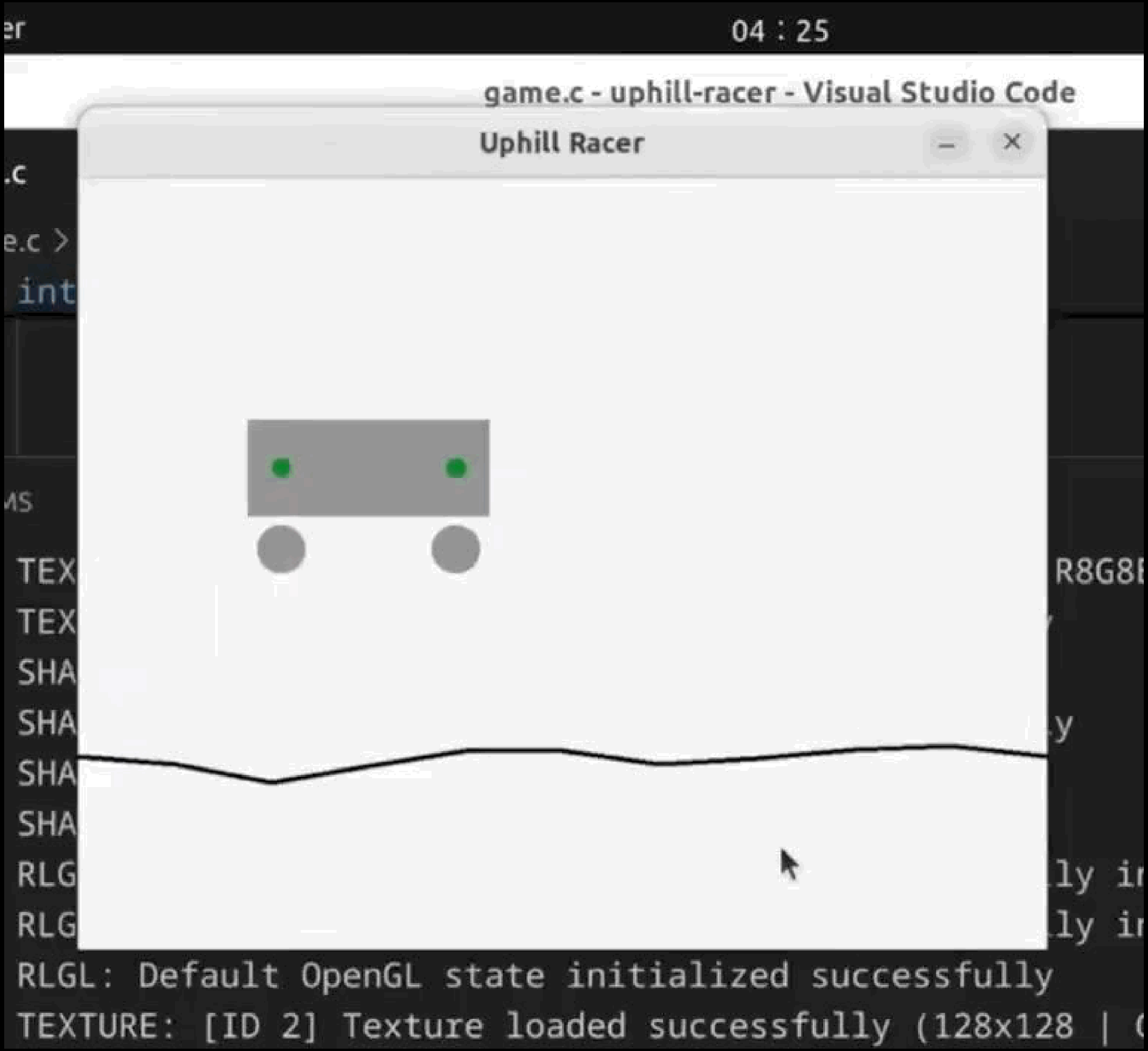


```
int terrain_length = 100;
int terrain_count = 255;
Vector2 terrain[terrain_count];

int pos = GetRandomValue( WINDOW_HEIGHT * 0.7, WINDOW_HEIGHT * 0.95 );
for( int i = 0; i < terrain_count; i++ ) {
    int movement = GetRandomValue( -20, 20 );       You, 1 second ago • Uncommitt

    Vector2 point = {i * terrain_length, pos};
    terrain[i] = point;

    pos = pos + movement;
}
```



```
float dt = GetFrameTime();

ClearBackground( WHITE );

for( int i = 1; i < terrain_count; i++ ) {
    Vector2 point1 = terrain[i-1];
    Vector2 point2 = terrain[i];

    DrawLineEx( point1, point2, 5, BLACK );
}
```

HOW IT LOOKS LIKE



```
TEX                                                    R8G8
TEX
SHA
SHA                                                     y
SHA
SHA
RLG                                                     ly i
RLG                                                     ly i
RLGL: Default OpenGL state initialized successfully
TEXTURE: [ID 2] Texture loaded successfully (128x128 | 
```

WASN'T ABLE TO SIMULATE COLLISION ON RANDOMLY GENERATED TERRAIN

- Used gpt and was able to fingure some things out
- Just to figure out what is happening , I plotted points on the terrain

FINAL OUTPUT

- Let u9999s also play around with some values

WHAT I WAS ABLE TO DO

- Simulate gravity on 2d objects ✅
- Simulate friction on 2d objects ✅
- Simulate terrain collision on 2d objects ✅
- Simulate car suspension ✅
- Generate a randomly generated terrain ✅
- Make good game graphics ❌ 🥹

CHALLENGES I FACED

- Car suspension – not even chatgpt could help me with this
- Collision detection - didn’t understand what to do , found a code online (pasted it)
- Making the wheels move with the car while rotating

THANK YOU!!

- Just the basic prototype

  you can check more of the coming update on my git - andrewsteevjacob