Nate Sanchez
CSC 363 Final Project

Description

A basic 2d physics engine. The application contains collision detection algorithms, collision resolution, and linear projection for position correction.

Demonstration

https://youtu.be/iYbfNKsob1I

Dependencies

Notan
https://github.com/Nazariglez/notan
https://lib.rs/crates/notan

Discussion of Project

For my project I wanted to explore 2d physics and creating a basic 2d physics engine. To accomplish this task, I needed a way to display things in real time and for this I am building my project on top of the Notan library. I decided to go with Notan as I didn't need a full-fledged game engine and I didn't need something as low level as Vulkan or WGPU. This project is divided into three parts, graphics, collision detection and collision resolution.

Thankfully, Notan allows me to avoid dealing with any direct OpenGL, but I still decided to organize my project and how textures are handled by entities. One common optimization I employed here was to have entities share textures rather than each entity having a unique texture. Fortunately, I could pass most of the modifying data I needed at runtime without having to modify the texture itself, such as location and size. I decided to use reference counting here to allow shared ownership of the textures between entities with a texture manager to provide easy access to textures.

Collision detection is the first half of the equation for actual physics. My project only consists of circles and non-rotating rectangles to keep the scale of this project from getting out of hand. Since I'll need to detect collision differently for each shape, I am using axis-aligned bounding boxes (AABB) to surround any non-rectangular shapes. An AABB is simply a rectangle with its sides aligned to the axes. Collision detection here is straightforward, and I can essentially check that the corners are outside of any other AABB. If there is collision between AABBs, I check for their child collider and see if those collide, otherwise, I treat it as a collision between two rectangles. Since the type of collider is not known by the AABB, I used the visitor pattern to determine the type of collision algorithm I needed.

Once collisions have been detected, I need to resolve that collision. This part of the project easily provided the most challenge, especially with Rust's rules on ownership. For example, I need to compare every entity with every other entity for collision and then apply the resolution. However, I can't borrow my entity vector twice while allowing mutability. To overcome this issue, I created a new vector to store my resolution velocity and data for position correction and once I am finished checking for collisions, I apply the data to entities. To get the data I needed I was able to build off the aforementioned visitor pattern.

Key Components

- Controls - Click to spawn a new physics entity (currently up to 20, alternating between squares and circles)
- Entity – Something in the game space that can be rendered.
    - Physics Entity – A entity that reacts to the environment.
    - Static Entity – A entity that has a collider but does not move.
- Collider – Used for Collision detection.
    - AABB – A bounding box aligned with the axes, used to improve collision detection
    - Circle Collider – A circle shaped collider
    - Check_collision() – A method to determine the correct collision detection algorithm
    - Check_collision_aabb() – A method to detect collision with an AABB
    - Check_collision_circle() – A method to detect collision with a circle collider
- Textures – Shared data that defines how things are drawn to the screen.

Known Issues

- Circle physic entities can sink into rectangular physics entities.