Documenation Ubisoft Next

March 2023

Features

- Entity Component System
- Basic Sound/UI
- Third person camera
- 2d Physics
- Basic 3d Physics

Setup

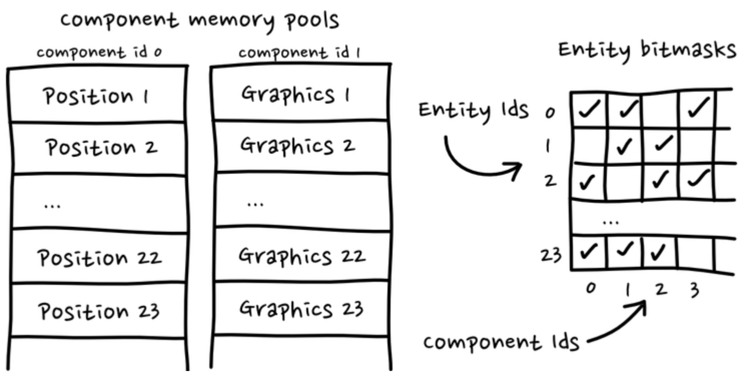
Tested on Visual Studio 2019

Overview

The controls of the game is WASD to move. Space to place bomb. E to throw grenade. The player wins the game by killing every enemy.

Entity Component System

The core part of the code base is the Entity Component System. Entities are simple bit masks which stores that stores which ever components they own. Components are simple struct that contain the data the game needs to operate on. Each component type is stored in their own component pool for cache friendly access.
The scene object will serve as a database of components and help us access and manage it. We can create new entities, which returns an EntityID, just a number. And then assign components to each entity



Example of a component (can be any simple data struct):

```
struct Bomb
{
        double BombCrossSize = 5;
        double Time = 2.5;
};
```

Example of adding component to a entity:

```
Scene scene;
EntityID triangle = scene.NewEntity();
Transform* pTransform = scene.Assign<Transform>(triangle);
```

All System in the code base is a subclass of the System class located under Systems/Systems.h. Systems have two methods they must define Update and Render which will run in the Update and Render phases in our game respectively. To iterate over selected components a custom iterator called the SceneIterator class is defined for us.

Example of subclassing a System:

```
class MouseSystem : public virtual System
{
public:
        MouseSystem(Scene& scene, Mouse* mouse) :
                System(scene), Mouse(mouse) {}
        void Update(float deltaTime) override;
        void Render() override {}
        ~MouseSystem() override = default;

private:
        Mouse* Mouse{ nullptr };
};
```

Example of Iterating over components

```
void SoundSystem::Update(float deltaTime)
{
        int collisonTransform[] = { GetId<SoundEffect>() };
        const auto soundIterator =
                SceneIterator(SystemScene, collisonTransform, 1);
        for (const EntityId soundID : soundIterator)
        {
                auto soundEffect =
               SystemScene.Get<SoundEffect>(soundID);
                App::PlaySound(soundEffect->Soundfile);
                SystemScene.DestroyEntity(soundID);
        }
}
```

In the above class we pass in a list of component indexs via GetId to create our custom Iterator the Iterator then allows us to traverse the entire Scene to find all entities with the SoundEffect component

Once you defined and Instantiate a System you must register the System with the allSystem object the allSystem objects will iterate through all Systems and call their Render and Update function respectively. The order matters if you have a System which depended on another Systems calculation you must register the dependent system after.

Physics

2d and 3d physics are handle by the Physics2D and Physics3D system respectively.
I had originally intended to implement a full 3d rigid body simulation. However due to time constraints It would be feasible to finished in time. So I decided to integrate my already existing 3d physics code that I wrote into the grenade feature.
Had I had more time the next step would have been to implement narrow phase collision using GJK algorithmn and then implement collision response.

References

Online resources that was helpful during the coding of this project

- https://www.youtube.com/watch?v=ih20l3pJoeU
- https://www.david-colson.com/2020/02/09/making-a-simple-ecs.html
- https://blog.winter.dev/2020/designing-a-physics-engine/
- http://www.cs.cmu.edu/ baraff/sigcourse/index.html