Ubisoft NEXT 2023-2024

Engine Core

Entity

An `Entity` is a struct with an ID, tag, position, and optional pointers to 'components' like Rigidbodies, Colliders and Sprites.

I used the Builder pattern to allow for easy method chaining to configure Entities with ease.

A static field 'id' increments to ensure that an Entity can be uniquely identified.

An Entity's tag is just an enum that can be one of `DEFAULT`, `PLAYER`, `ENVIRONMENT`, `ENEMY`, and `PROJECTILE`.

Special care was taken while writing the move and copy constructors for the Entity, because incorrect management of memory here could be disasterous.

Resource

A `Resource` is currently a typedef-d `std::tuple` that can store at most a single instance of a predefined list of 'resources'.

`ResourceMut` and `Resource` are defined to provide mutable and const access, respectively.

System

A `System` is struct that stores a function and some state about itself. It can run once, every frame, or at constant frame intervals. They can also be individually enabled and disabled.

The underlying container used to store Entities can be set using a flag at compile time, `USE_LIST_ENTITIES` uses the `std::list` container instead of the default `std::vector`.

World

A `World` contains all the Entities, Systems and Resources used in the game.

A system can be set to either be a render or update system.

The actual Systems are run on the invocation of `RunUpdateSystems` or `RunRenderSystems`.

Graphics

Camera

unused

Color

Defines a 4-component RGBA color, with utilities to create colors from a 0-255 range and a 32 bit integer.

Also defines static constants of a few common colors.

Font

Defines font utilities, including functions to get the approximate dimensions of a string, and fetch font data from a table.

Debug

Provides debug drawing capabilities.

Includes `DrawRect`, `DrawPaddedRect` and `DrawCircle`, as well as the overloaded `DrawInfo` to debug some common data types.

An optional, templated `DrawInfo` can be enabled to print numerical primitives using `std::to_string()`.

Math

Vec2, Vec3, Vec4

2, 3, and 4-component single-precision vectors.

IVec2

2 component integer vector.

Quaternions

Fancy 4D rotations. (not used)

Rect

Utility structure that contains the bounds of a box.

Utils

Generic `clamp()` function.

Physics

Collider

A `Collider` is essentially a tagged union with a pointer to the `Entity` that owns it.

It can have a type of either `Circle` or `AABB`.

Collisions between two colliders, can be checked.

Collision resolution has not been implemented.

Rigidbody

2D Rigidbody with mass, velocity and force. Uses an implicit Euler calculation.

Rigidbodies can be set to be kinematic or use gravity using a bitflag.

Resources

Time

Time defines a simple interface for accessing time, including a time-scaled version of the delta between frames by default.

Manager (PhysicsManager)

Handles and stores the Rigidbodies in the game.

Menu

The pause menu, and also serves as a global state manager of sorts.

Includes toggles for audio (unused), larger text for accessibility, toggling off/on debug info and showing/hiding the controls to the game.

GameTest.cpp

I worked off the provided codebase and the Entity System system (sorry, Components) made iterating through the development process a breeze.

Due to time constraints, the majority of Systems are simple functions that perform a singular objective.

The functions are:

`debug_info_entities` - position info about all entities

`spawn_enemy` - spawn a new enemy at a fixed frame interval
`create_menu` - set up the menu

`update_player` - move and shoot as the player
`update_sprite` - update sprite position and animation state
`update_menu` - handle menu interaction
`update_rigidbodies` - rigidbody physics
`update_enemies` - enemy logic

`draw_sprites` - draw all sprites
`draw_colliders` - draw all colliders
`draw_menu` - draw the menu
`draw_projectiles` - draw all projectiles
`draw_enemies` - draw all enemies

`show_controls` - draw game controls to the screen

`kill_floor` - kill anything that goes out of bounds
`kill_unused_rigidbodies` - clean up rigidbodies that do not have a set entity