Beszmi’s game engine

Basic details:

written currently in c++17 and SDL3. OOP-first with performance in mind. Every current object/texture in the engine is just a proof of concept that features work and debugging! All png and jpg images are put in the assets folder and its subfolders are automatically loaded and can be used to make objects.

Main:

deals with starting the game and running the game loop and the timings.

Game files and classes:

Game class stores everything a game needs, its window, renderer, textures, objects and tilemaps (will be integrated better later), screen size and cam.

init function currently used to set up everything in the game will be later made more dynamic and interactable. new objects for the game can currently be added here

handle events function deals with inputs currently only keyboard mouse will be added shortly.

update called all the objects update functions

render function deals with the rendering of all objects and displays any accumulated errors during the current cycle.

Clean cleans up the memory of the game engine.

Input

Handled in game::handleEvents, can detect WASD and left mouse button, extended later

Game obj files and classes:

Currently houses all the classes of all object types that the engine can use, will be split up when enough object types are added.

GameObject class: every object class is derived from this base class; it has a lot of functions to make derived classes function.
It has a transform component (stored in the camera files) that calculated local and

global positions for all objects but mostly only useful objects that have a cluster class parent.
Has all needed SDL components, and a Boolean to make the object render or not, a scale for the size of the object when it renders.
has 4 kind of constructors based on if you want to set an object in a specific case or not and if you want the object to have a parent object (note that only currently only game object clusters can be set as parents). There are warnings about uninitialized variables in visual studio, those can be ignored as they are only uninitialized if the texture look up fails in which case the program handles it with quitting.
Non-getter/setter functions are all virtual and are usually overwritten by derived classes.
It has an integer for layering lower means further back higher means further forward

Game object container class

stores all kinds of derived object classes and the base class's objects, with templated functions of spawn as and get.

Spawn as: make a unique smart pointer with the constructor of the object's classes, taking in its details. (See it used in game.cpp's init function)

Get returns a pointer to and Game object derived class as either const or non-const.

Render all renders all objects in the cached order based on what layer they are on. and applies all relevant render changes to objects

Update all just runs all the contained object's functions of the same name with the passed in arguments.

Game object cluster

Contains multiple objects in a vector, it can add objects to itself based on if we want to add it based on it current world position or locally with an offset from the parent object.
It has on overridden update function which currently only slows down the object so you can tell it apart from regular objects.
It's render function loops thru all the objects contained in the vectors and calls their render function.

Derived objects

Button

It's a derived game object that has an extra boolean to keep track of its state, currently it just stays in 1 place but it will be clickable for an action later.

stretched bg obj

A static class that stores 1 sprite component and renders it out, stretched to the screen's dimensions. It needs to be initialized with no texture (set its texture to anything that starts with the character "-") with the init function that needs a texture and screen details.

Sprite object

A dynamic class that is made up of a vector of sprite components, has a current element index and a time "t" that keeps track of time for the objects next update. Most importantly it has a state that tells it what to do on the next update:

sprite States (examples):

- 0 static
- 1/-1 animate (forward/backward)
- 2/-2 animate once on click (forward/back)
- 3/-3 animate continuously on click (forward/back)
- 4/-4 animate continuously on hover (forward/back)

It renders out the object currently picked in the vector by the current element index.

Texture manager

Works automatically after calling and loads in all textures from the asset folder and its subfolders. The texture's name will be its relative path from assets path + the file extension. (if its in the main assets folder then it doesn't need a path) At the end of the program's runtime, it also cleans up the memory occupied by the textures.

Sprites

sprite component

Stores 1 texture, and nothing else. it can be created with or without a texture. if without it needs its texture set with the set_tex function. For rendering it needs all relevant details passed through can be rendered with or without scaling.

stretched background:

Derived from sprite component with a static class that stores its own details, and the screens details to be able to be rendered out to the whole screen size. Can be created empty or already filled with details, if empty it need to be initialized with set_screen and set_tex.

tilemap

A static class that can be used to tile the whole screen with a texture or just a predefined grid.

Create one with making a unique smart pointer and addig it into the game objects maps vector

It needs to be set up with a with its details grid/map which can be done with:

- fill_grid: in params, use rows count, col count, tile index (usually 0)
- fill_background: in params, use 0, 0, tile index (usually 0)

It automatically culls tilemap cells and only renders visible parts