Modules in engine:

1. Core

Eventhandler:

- For now implemented only for Android.
- Allow access fingers properties touched screen and pressed button "back".
- Recognize changes in Android app lifecycle(go to background, foreground) and screen orientation changes.

GameLoop:

- Game objects updates, physics simulation, load events, draw, play sound commands starts here.
- GameLoop->run() calls from main() function after all preparations.

Layer:

- Base class for all layers in game.
- Contains and handle all GameObjects in layer.
- Your game should inherit from this class and implement some layers (GUI layer, 3D scene layer, debug layer, …)

LayerStack:

- Contains layers. Contained and accessed via GameState.
- You will first create object of GameState then push your layers to it.

GameState:

- Contains and updates LayerStack.
- Can represents game states (main menu, pause, play,…)

GameStateMachine:

- Handle GameStates.
- Is used in GameLoop.

GameLoop:

- Game objects updates, physics simulation, load events, draw, play sound commands starts here.
- Handle FPS
- GameLoop->run() calls from main() function after all preparations.

GameObject:

- Base class for all objects in game.
- Has subclasses for GUI objects and 3D scene objects (you can inherit from them)

RandomGenerator:

- Helps generate random numbers

SoundsManager:

- Contains and manage sounds and music

Timer:

- Helps count time between certains events.

TimeStep:

- Contains time passed from app start.
- Keep time step of previous frame.

Window:

- Base class for all windows on different platforms.
- And API for unified access to windows on different platforms.

Log:

- Represents log system

2. DataBase

Not implemented.

3. GameObjects

SceneObject:

- Main class for all objects in 3D scene.
- Inherited from Core::GameObject.
- Can contains specific data for 3D scene objects.

SimpleObject:

- Simple 3D object loaded from Collada or FBX formats.

`CollidingSimpleObject:`

- Simple 3D object + collision mesh loaded from Collada or FBX formats.
- Participates in physics simulation.

`AnimatedObject:`

- Skeletal animated(mesh + bones) 3D object loaded from Collada or FBX formats.

`CollidingAnimatedObject:`

- Skeletal animated(mesh + bones) 3D object + collision mesh loaded from Collada or FBX formats.
- Participates in physics simulation.

Above objects contain empty bechavior and standart graphics. You can inherit from them and introduce your own objects for 3D scene with your behavior/grapsics/sounds.

4. GUI

GUIObject:

- Main class for all GUI objects.
- Inherited from Core::GameObject.

Button, CheckBox, Text, DrawAnyFunction:

- Objects for GUI.

5. Network

Not implemented.

6. Physics

Physics:

- Represents physics world for simulation.
- Contains and manage collision meshes of objects on 3D scene
- Keeps collisions after simulation
- Allow cast rays

7. Platform

- Contains platform(Android, Windows, IOS) specyfic stuff (Window, Shader, Buffers, Textures, GUI container, Vertex array).
- For now implemented only for Android.

8. Renderer

Camera:

- Stuf for access camera on 3D scene.

Buffer:

- Base classes for vertex buffers and index buffers on different platfoms.
- Platform specyfic implementation should create buffers from loaded data and prepare them for GPU rendering.

Shader:

- Base class for shaders on different platfoms.

Texture:

- Base class for textures on different platfoms.

VertexArray:

- Base class for vertex arrays on different platfoms.
- Contains all buffers(vertex + index) and draw them.

Renderer:

- Class for create different buffers, vertex array, shaders, textures.
- Should know about platforms and create objects according current platform.

9. Utils

File:

- Helps work with files.

Matrix:

- Some operations on matrices.

Quaternion:

- Some operations on quaternions.

Libraries included in engine

Assimp: load 3D models + animation

Bullet: physics simulation

Glm: math operations

ImGUI: base library for all GUI elements

SDL2: window, input handler, time,…

SDL2_image: load and handle images to textures

SDL2_mixer: sounds and music

Spdlog: logs

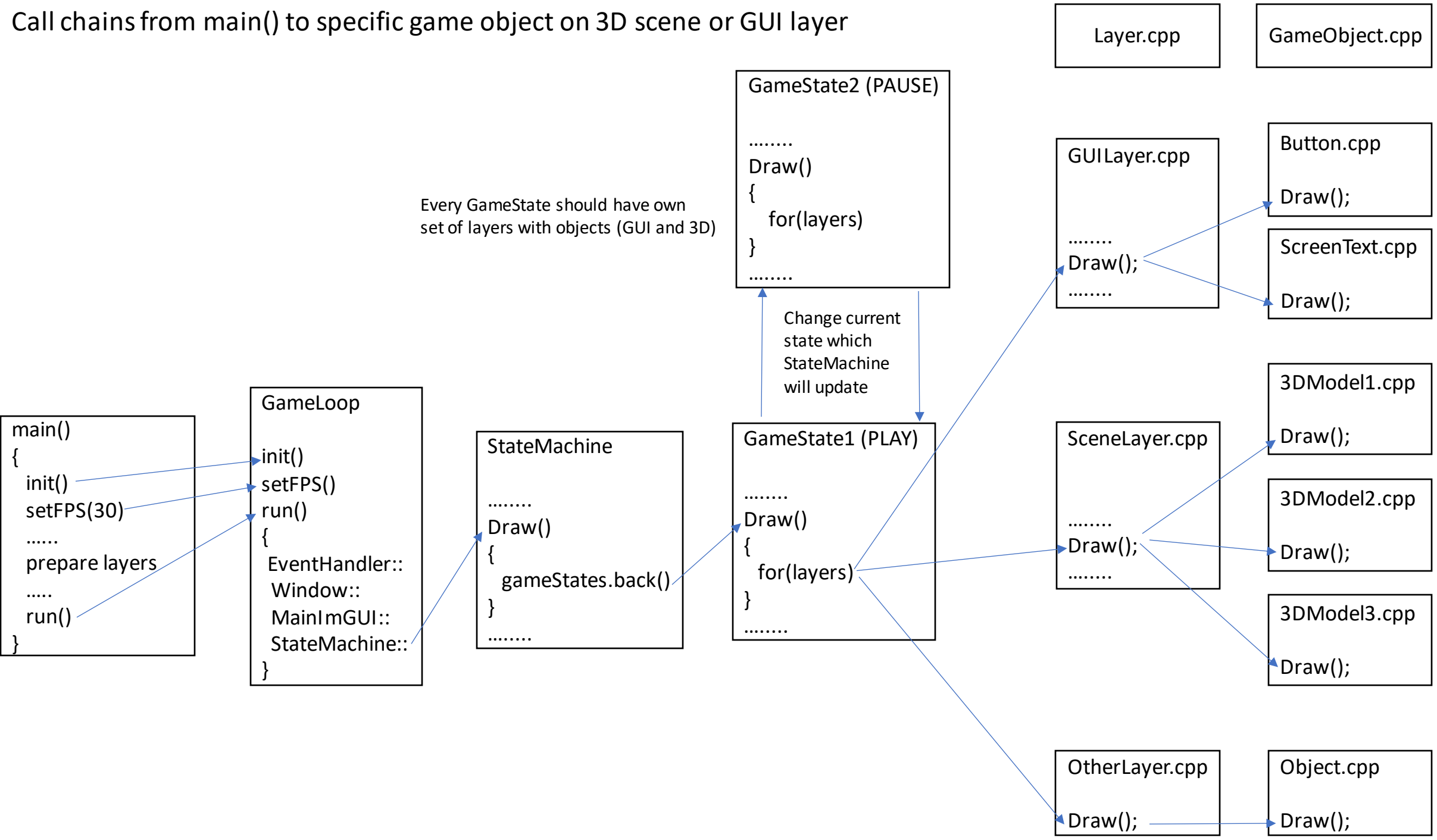
Sqlite: database

Initialize engine and run app:

```
#include "EngineHeaders.h"

// app files
#include "GUILayer.h"
#include "SceneLayer.h"

int main(int argc, char* argv[])
{
    Beryll::GameLoop::create(Beryll::Platform::ANDROID_GLES); // must be called first
    Beryll::GameLoop::setMaxFPS(10);

    // GUI elements properties
    Beryll::MainImGUI::setButtonsFont("fonts/arial-italic.ttf", 5);
    Beryll::MainImGUI::setCheckBoxesFont("fonts/creamy.ttf", 3);
    Beryll::MainImGUI::setTextsFont("fonts/proggy.ttf", 2);

    // camera properties
    Beryll::Camera::setPerspectiveNearClipPlane(2.0f);
    Beryll::Camera::setPerspectiveFarClipPlane(500.0f);
    Beryll::Camera::setObjectsViewDistance(120.0f);
```

```
    // Simulation precision (1-20 range), also increace CPU usage
    // Increace it if your ball penetrates wall but you want collision
    Beryll::Physics::setResolution(2);

    // Create own class MyLayer and inherit from Beryll::Layer provided by engine
    std::shared_ptr<GUILayer> guiLayer = std::make_shared<GUILayer>();
    std::shared_ptr<SceneLayer> sceneLayer = std::make_shared<SceneLayer>(guiLayer);

    // create object of GameState provided by engine + set ID
    std::shared_ptr<Beryll::GameState> mainState = std::make_shared<Beryll::GameState>();
    mainState->ID = Beryll::GameStateID::MAIN_MENU;
    // Push MyLayer to GameState->layerStack
    mainState->layerStack.pushLayer(sceneLayer);
    mainState->layerStack.pushOverlay(guiLayer); // pushOverlay for GUI

    // Push gameState to GameStateMachine
    Beryll::GameStateMachine::pushState(mainState);

    Beryll::GameLoop::run();

    return 0;
}
```

Call chains from main() to specific game object on 3D scene or GUI layer



Shaders

```
precision highp float; // highp mediump lowp
```

In shaders is important. Can reduce Z-fighting in 3D scene