Modules in engine:

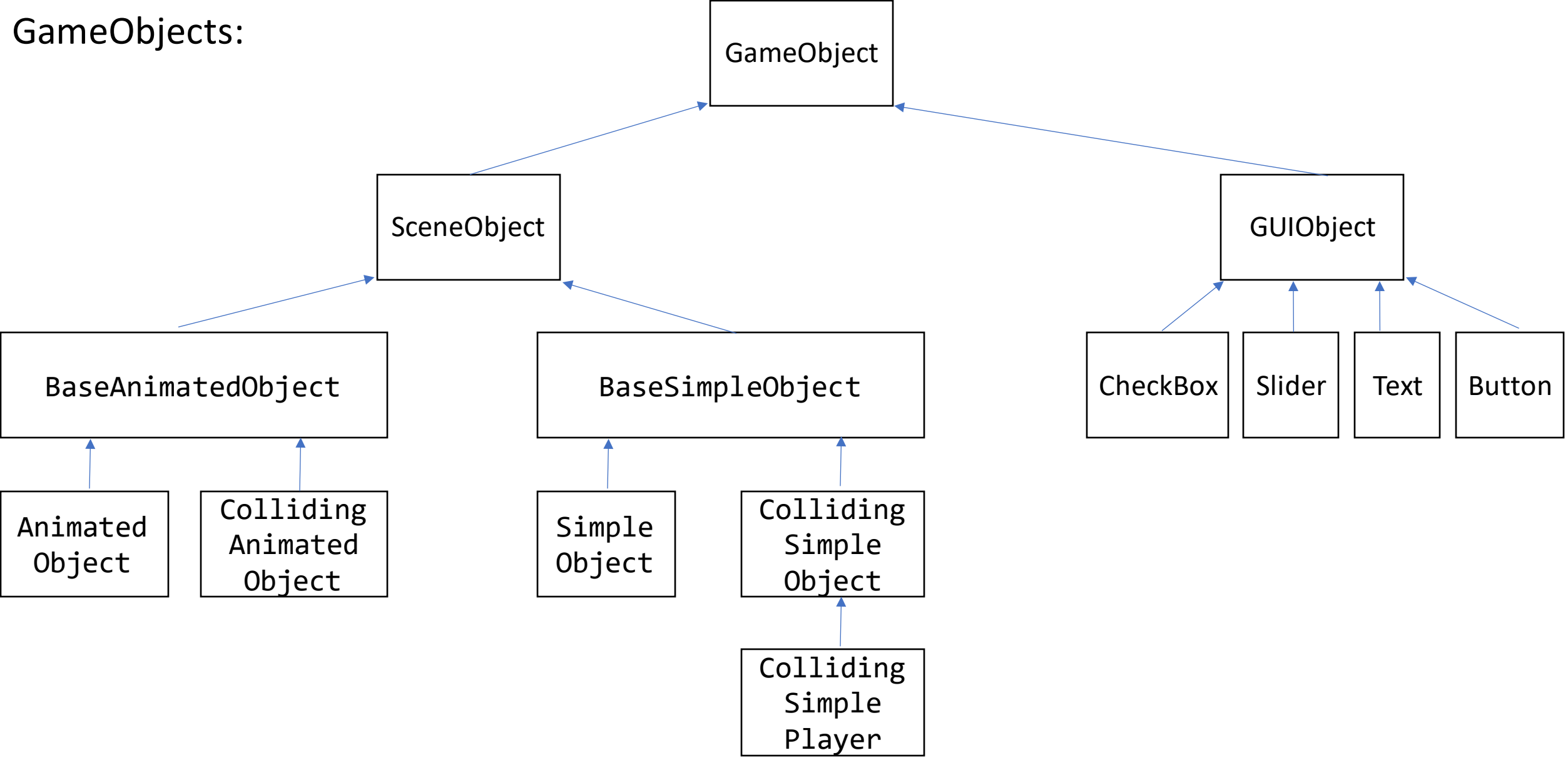
Core:

Contains main classes which are basics for game engine.
Like Constants, GameLoop, EventHandler, GameStateMachine + States and Layers, Log, Timer, TimeStep, ….

Async:

Can perform same function for all vector elements in multithreading way.

GameObjects:



DataBase:

Not implemented.

GUI:

GUI elements:

Button, CheckBox, Text, Slider.

Network:

Not implemented.

LoadingScreen:

Represent image which appearing partially and shows progress.

ParticleSystem:

Can emit two types of particles: squares and cubes with given parameters.

Physics:

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

Platform:

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

Renderer:

Contains classes and interfaces required for create graphics content:
Buffers, Shader, Texture, VertexArray, Camera, Renderer.

Utils:

Classes for help with files, matrixes, quaternions.

Libraries included in engine

SDL2: window, input handler, time,…

SDL2_image: load and handle images to textures

SDL2_mixer: sounds and music

SDL2_net: networking

Assimp: load 3D models + animation

Bullet: physics simulation

Glm: math operations

ImGUI: base library for all GUI elements

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(); // 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 increase CPU usage
    // Increase it if your ball penetrates wall but you want collision
    Beryll::Physics::setResolution(3);
```

```
    Beryll::LoadingScreen::setTextures({"loadingScreenImages/lowPolyTerrain1.jpg",
                                        "loadingScreenImages/lowPolyTerrain2.jpg"});

    // 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> playState = std::make_shared<Beryll::GameState>();
    playState->ID = Beryll::GameStateID::PLAY;
    // Push MyLayer to GameState->layerStack
    playState->layerStack.pushLayer(playSceneLayer);
    playState->layerStack.pushOverlay(playGUILayer); // 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