The core of this whole project is the underlying game engine named *Shingine* that was built using third party libraries and is a wrapper around the required functionalities provided by those libraries so the user of the engine does not have to know about the intricate underlying details and go through the learning curve to understand all the concepts and functionality provided by those libraries.

**Libraries Used**

This engine was made using following (32-bit) libraries and therefore this engine will only work with x86 configuration, so beware of that. If your machine does not support these libraries this engine would not work for you.

1) SDL2 <https://www.libsdl.org>

2) Box2D <https://box2d.org>

3) glm <https://glm.g-truc.net/0.9.9/index.html>

**How to use**

This game engine is meant to be compiled as a static library so you have to link your project made in your favorite IDEs with *Shingine* and add Shingine.lib in your additional library directory path and add *Shingine* folder having all header and cpp files in your additional include directory paths. The details of doing this vary across different IDEs so consult the documentation of IDE you are using.

After linking to engine be sure to add *Shingine::* as fully qualified name before all classes and functions you use from this the library (engine) because everything written in this engine is in namespace *Shingine*.

**Why use it?**

It is little early to ask this question because we have not discussed functionality provided by this game engine which of course is code based meaning it does not support graphical interface. If you are a coder that like to program game using plain code and code alone (no GUI game engines) maybe because you are learning how to code but you have hard time coming around making an engine for yourself that you can use again and again in your projects or you just don’t feel like you want to invest you time making a simplistic game engine like this in one’s eye which can get quite a lot complicated actually in another’s, for your simple game project. This engine then has most of what you want. You can make your own engine using this engine to add additional functionality and features. See licensing below.

**Features**

This game engine (from now on referred as *Shingine*) has several functionalities available for you in form of different header files. Every functionality a header file provides is briefly mentioned below (*see* interfaces and function declarations in header files to get better understanding of what parameter member functions / functions require):

***Shingine.h****:* This header files provides two functions, init and quit to initialize and quit SDL subsytems. You should never require it although. See IMainGame.h

***IMainGame.h***: This header files provides you with IMainGame interface that you should inherit your application/game class from to be free from initializing subsystem and other minute engine details.

***IGameScreen.h****:* This header file provides you with IGameScreen interface that you have to inherit your classes for different screens from. If you are using IMainGame then this is a must because IMainGame calls members function of IGameScreen.  
***ScreenList.h****:* This header files provides you with functionality to handle variables amount of IGameScreen s. Again, you do not require this when you are using interfaces like IMainGame provided by the engine.

***Timing.h***: This header file provides you with two classes FPSLimiter and Timer. First class provides functionality to cap your frames and second class helps you measure time passed in second or in terms of frames passed.

***STexture.h***: This header file provides you with a class STexture that you load an image into using resource manager or member function setImage and it provides functionality to draw it on window with specified coordinate, rotation etc.

***TextureCache.h*:** This header file provides a class TextureCache that stores the texture you want to load from file into a C++ map (binary tree) and returns that texture in form of STexture to you. This done to reduce IO operation and making of a new texture if game uses same texture is used multiple times in program.

***RosourceManager.h:*** This header file includes a static class ResourceManager that you guessed it by its name manages resource like textures etc. This class stores textureCache as static member and you actually use resource manager to load a texture for you from a file.

***TileSheet.h***: This header file provides class TileSheet that you can use to draw a part of an image mostly a single image from a sprite sheet used in animation after specifying rows and columns of total little images and index of the image inside that you want to render.

***SpriteAnimator.h***: This header file provides class SpriteAnimator with functionality to animate sprite sheets using TileSheet internally.

***Text.h***: This header file provides you with class Text to draw text on screen with specified font, size and color.

***Physics.h*:** This header file provides with class BoundingBox and two very important functions that initPhysicsWorld and updatePhysics in nested namespace *Physics*. All of your entities in the game must have bounding box attached with them if you want collisions, forces applied on them (movement, jump etc.). Bounding box can be used to create a ground for your game world on which your entities will stand. Call former function once in your gameplay screen and later every time in your update function for that specific screen, when you want to use functionalities of box2D. See the header file for more details.

***IOManager.h***: This header file provides a static class IOManager with static function readFileToBuffer to get input from file into a vector of strings. Can be really useful if you save your level files in asci format and want to load them in your game.

***InputManager.h***: The header file provides with a class InputManager that is of extreme importance. If you want to query which key is pressed, where the mouse is etc this class has functionality just for that which is very critical for event driven software like games.

***GraphicUtils.h***: This header file provides with a class Rectangle, which can be used to show health bar of entities and stuff and three function nested in namespace Graphics which can be used to draw rectangles (unlike the class one which holds onto state of rectangle), line between two points and just a point on screen.

***Errors.h***: This header file just includes a function fatalError for the purpose of debugging, you can use it if you are modifying engine and want to debug things. Basically, it prints out specified message, pauses console and just exits application. You can come with better solutions but it is there if you need it.

***Display.h***: This header file includes class Display that refers to window we are displaying things on. Again, if you are inheriting from interface IMainGame must never need it but if you are going with your own implementation you can use this class to create a window with additional arguments.

***Camera.h***: This header file includes class Camera1D for side scrolling only. Can be expanded for 2D easily. The translation got from camera after focusing it to player must be added to every entity in game you want to scroll, while drawing them to get side scrolling effect.

***Button.h***: This header file includes class Button for making a button for ButtonMenu. See ButtnoMenu.

***ButtonMenu.h***: This header file includes class ButtonMenu which has a list of buttons your assign it for making horizontal or vertical button menu for main menu screens etc.

***AudioEngine.h***: This header file provides you with three classes Music, SoundEffect and AudioEngine. AudioEngine is used to load Music and SoundEffect (in wav format) from a file and also caches them in map. Then you can play the returned Music or SoundEffect with their member function play.

***Note***: See the sample game I created

**Contributors**

* Shayan Ali Abbasi shayanali422@gmail.com

**License**

MIT License