Engine Layout and Architecture:

1. This engine uses a component based design. There are components and compositions. Components perform a specific task. Multiple components are held together in a composition to perform complex task(s). GameComponent is the base class of components. A new type of component can be created by extending from this class. All composition have a pointer to the composition that they are in.
2. All components are instantiated by the 'GameObjectFactory' class. All components need to register their component creators with this factory to be able to be instantiated. There is a templated ComponentCreatorType class which can be used to readily define a creator class for a new component or a component can write its own specific componnet creator by extending the base class ComponentCreator. GameObjectFactory also creates empty compositions. It also serializes a archetype of a composition and creates that composition. Components have unique IDs with which the GameObjectfactory can keep track of them. These IDs are defined in the file ComponentTypeIds.h . Currently, basic components like Transform, Sprite, Controller, Body are defined. Any other components can be created and added as defined above.
3. The game loop is contained in a class called 'CoreEngine'. This class does two things: 1. update systems every frame with the appropriate delta time and broadcast messages to the systems. The global pointer 'CORE' is available for all at all times to access this CoreEngine.
4. A broadcast style messaging style is used. 'Message' is the base class of all types of messages, located in file Message.h. Whenever a message needs to be broadcast, create an instance of the appropriate message type and dispatch it by calling 'BroadcastMesssage()' on CoreEngine(if the message has to be broadcast to the world) or a particular system(if the message has to broadcast only on a particular system) or a specific composition(if the message is meant for a specific composition or a specific component of that composition) or on the parent composition(if the message has to be broadcast to peer(s)). Those who are not concerned with any message that they receive can just ignore it.
5. A simple text based serialization is used. Components take care of their own specific implementation of serialization. Each object archetype is contained in its own text file. When composition with an archetype is to be serialized it uses this file to create the individual components required for that composition. Currently, values like only int, float and string are supported. The format of the data is specific for each component since the components themselves take care of their serialization.
6. User inputs are recorded and dispatched using Windows messaging system from the class WindowsSystem defined in WindowsSystem.h. Most of the usual keyboard and mouse inputs are defined by their own specific message classes; ex: messageCharacterKey, MouseButton, MouseMove, etc. Any undefined message classes can be added in this file to extend their availability. Classes need to listen to the specific message types that they are interested in to get the input information.
7. There is debug draw in place. It can draw basic stuffs like circles, rectangles, lines in different colors. Defined in DebugDraw.h