Objectives:

The purpose of this assignment is to give you experience working with a component-based architecture that isolates all the core game functionalities learned so far in this course into interchangeable components. In this Quest you will:

- ➔ implement a Component class hierarchy, topped by abstract class Component;
- ➔ implement a SpriteComponent class which provides graphical support;
- ➔ implement a BodyComponent class which provides basic physics support; and,
- ➔ implement multiple BehaviorComponent classes.
- ➔ implement a UserInputComponent class

Summary

- ➔ You are required to re-implement all the functionality of Assignment 2 with a component architecture.
- ➔ You will modify your input device to allow for multiple key inputs.

Requirements:

- ➔ **OBJECT CLASS**
  - acts as a container with interfaces for objects derived from class `Component`
  - contains a `vector<Components>` components;
  - NO CHILD CLASSES OF OBJECT!!!!
  - `void addComponent (Component* component)`
    - adds a component to the vector of components.
  - `bool initialize (ObjectFactory::Initializers initializers);`
    - `ObjectFactory::Initializers` is a struct that I put in the ObjectFactory.h file that contains everything needed to initialize any component. This allows for quick updating of parameters needed for initializing the components.
    - Be careful what order you initialize your components in, as some depend on others!
  - `Object* update()`
    - method calls each component's update method.
    - returns an `Object`* in case an object (like a bullet) is created during the update method. For now, there is no need to do anything with a returned object!
  - `void draw();`
    - method finds the sprite component, and if it exists,
      - casts it to a sprite component
      - calls that sprite's draw method.

➔ Component Hierarchy

- `bool initialize (GAME_OBJECTFACTORY_INTIALIZERS initializers);`
- `virtual Object* update() = 0;`
- the `SpriteComponent` class will require a draw() method.

➔ Minimum Components

- `SpriteComponent` (for graphic functionality)
- `BodyComponent` (for physics functionality, but currently just stores position and angle)
- `InputComponent` (for player input handling)
- `Behavior1Component` (change to name to be appropriate for your object's behavior)
- `Behavior2Component` (change to name to be appropriate for your object's behavior)

➔ ObjectFactory Update

- You should have a single object factory to which you pass an object's tinyXML element. The factory would:
  - Create a new object
  - Read each component's information from the XML
  - Create an instance of the component
  - add it to the object
  - Create an `ObjectFactory::Initializers` variable and store the required information.
  - Call the object's initialize method which, in turn, calls each of the components' initialize methods.
  - Return the newly created object.
- It is also suggested that you overload the `ObjectFactory's` create method with one which takes a vector of component names and a `ObjectFactory::Initializers` variable. This can be used to create objects at runtime.

➔ Input Device Update

- Implement a keyboard-state processing class which maintains a `map<GameEvent, bool>`. This class has three methods:
  - `bool Initialize()`: which assigns false to all possible `GameEvent` enumerations.
  - `void Update()`: which maps an `SDL_Event` type to its corresponding `GameEvent` type and sets the value of the `GameEvent` type (within the vector) to true or false depending on a `KEYUP` or `KEYDOWN` has been detected for the key event; and,
  - `bool GetEvent(GameEvent event)`: which returns true/false by accessing the value associated with the `GameEvent` in the vector.
- Be sure your player makes use of this new ability, such as the ability to move diagonally or attack and move at the same time!

Suggested Division of Labor

TOGETHER:

- ➔ Update Object Class

MEMBER 1:

- ➔ Sprite Component
- ➔ Body Component
- ➔ Object Factory update

MEMBER 2:

- ➔ Input Component
- ➔ Behavior Components

Submission

All code should be submitted to your repository using GitHub desktop to "Commit" and "Push" your code. The last push date will be considered your date of submission.

Before the last "late day" you must submit a video via the Assignment 2 submission link in Blackboard. You **MUST** use Kaltura to submit the video and should be less than 10 minutes long. Focus on the new content. No need to rehash old code!

!!!!!Videos submitted directly into blackboard, not using Kaltura, will not be accepted!!!!!! *We have limited space on Blackboard for our class!*

Both members of the class will need to submit a video walkthrough of the code. DO NOT read the same script!! I need to be able to tell you understand ALL the code, not just the portions you wrote!

The map should be a small rectangle (proportional to the game window) that appears in the corner of the main game window. This map should update in real-time.