Learning Objectives:

- ➔ understand how to apply SDL2's low-level rendering and event handling capabilities;
- ➔ demonstrate abstraction of SDL2 specific capabilities within Graphics and Input devices;
- ➔ understand how to apply Art and Game Asset Libraries;
- ➔ demonstrate the separation of the game state with the game view (i.e., game version of model-view control).

Task Summary

In this assignment you will be implementing the graphics and controls for your game. You don't need to worry about getting all the functionality of your game working. The sprites should appear on the screen and keyboard, or mouse input should do something!

Requirements:

Each group's assignment will be slightly different since we are all working towards different final projects. You will need at least 3 sprites that are children of the object class, which show polymorphic behavior. One of the sprites should be controllable by the user. There should also be a view that changes.

Finally, the following classes should be included.

- ➔ Objects loaded from XML
- ➔ Object hierarchy
  - ◆ virtual update()
  - ◆ virtual draw(View*)//drawing of the sprite adjusted by view!
- ➔ Update Library class to include a map<string, std::shared_ptr<Texture>> artLibrary;
- ➔ **GraphicsDevice - use mine!**
- ➔ **Texture Class - use mine!**
- ➔ InputDevice
  - ◆ std::unique_ptr<SDL_Event> event; //tracks current event.
  - ◆ gameEvent translate(); //gameEven is an enum of all user input events for the game.
  - ◆ bool getEvent(gameEvent);
  - ◆ void update(); //polls SDL & calls translate and update event map
- ➔ View class
  - ◆ adjusts object's position for display (does not change the object's members)
  - ◆ members:
    - InputDevice*

- Vector2D center; //Vector2D is a struct with a float x; and float y;
- Vector2D position;

◆ methods
- View(InputDevice*, Vector2D);
- bool update(); Initially the current gameEvent is requested from the InputDevice. The View then responds to keyboard events by shifting its position in accordance with the events. If the view of your game does not change, then not much will be done here.

➔ **Timer class - just use mine!**
➔ Engine class
◆ library and device members
◆ timer member
◆ view member
◆ vector of std::unique_ptr<Object>
◆ Methods:
- void reset(); destroys view, maybe the library & Object vector
- bool loadLevel(string levelConfig, string libraryConfig);
- bool run(); does everything for a single frame
- bool update(); calls update of all objects
- bool draw(); calls draw of all objects

Suggested Division of Labor

MEMBER 1:

➔ update Object update/draw methods for all classes
➔ View class
➔ Input Device
➔ update game loop

MEMBER 2:

➔ update XML with sprite path;
➔ Graphic Device/Texture Class
➔ update Library class
➔ Timer 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!