The Legend of Zelda: Quest 1 Specification

Deadlines:

See the World Map for availability and dude dates.

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 ability to combine XML parsing, document object modelling (DOM) abstraction, Object Factories, and Assets Libraries into a single cohesive Game-Level initialization; and,
- understand and demonstrate the separation of the game state with the game view (i.e., game version of model-view control).

Task Summary:

We will be programming an open-world, Zelda 1 simulation in the 2-dimensional plane, viewed top-down. The instructor has provided all the necessary images (although you are allowed to create and use your own sprites if they have the same roles and complexity). Your task is to implement an SDL2-driven, window-based game through which you can observe the independent simulation of multiple types of objects.

What is more important than the simple functionality in this programs is that it builds up an abstract and flexible back-end with which we can build substantially more complex programs later in the semester. The instructor has provided you with the Source.cpp file which contains the declaration and construction of a game object.

The Game object contains a GraphicsDevice and InputDevice which are wrappers that abstract all of the SDL2-specific calls necessary to interact with and display the game state to User-defined calls. The GraphicsDevice conducts all the screen management and hardware rendering and the InputDevice handles all of the keyboard inputs. Abstracting these calls as objects allow for the Game to be written independently of the Media Library (e.g., if done properly, this game could be written to run in SDL, OpenGL, or DirectX simply by rewriting the GraphicsDevice and InputDevice classes. The Game object also contains the game logic and game asset management.

Game Class:

To achieve the desired functionality, the Game class should have the following properties:

- `unique_ptr<GameAssetLibrary> gLibrary; (see below)`
- `unique_ptr<ArtAssetLibrary> aLibrary; (see below)`
- `unique_ptr<GraphicsDevice> gDevice; (see below)`
- `unique_ptr<InputDevice> iDevice; (see below)`
- `unique_ptr<Timer> timer; (similar to that described in the videos)`

- `GAME_FLT gameTime; (maintains the overall in-game time)`
- `unique_ptr<View> view; (see below)`
- `vector<unique_ptr<Object>> objects;`

and the following capabilities:

- default constructor and destructor;
- `bool Initialize(); //initializes or constructs all the members of the class`
- `void Reset(); //destroys the View object and all Objects stored in objects, and restes its size to zero;`
- `bool LoadLevel(string, string); //constructs the View object, loads objects into the ArtAssetLibray from an XML file, and loads from XML file (by both parsing the XML and appropriate calls to the Art and Game Assets Libraries) and constructs all Objects in the objects container;`
- `bool Run(); // conducts a single frame of the game (i.e., Updating, Drawing, and frame timing)`
- `bool Update(); //calls the Update method of all Objects in objects`
- `void Draw(); //calls the Draw method of all Objects in objects`

ArtAssetLibrary

This class contains the following (as well as initialization capabilities which are assumed for all classes):

properties:

- `map<string, shared_ptr<Texture>> library;`
- `GraphicsDevice* gDevice;`

capabilities:

- `bool AddAsset(string, string);`
- `shared_ptr<Texture> Search(string);`

GameAssetLibrary

This class contains the following (as well as initialization capabilities which are assumed for all classes) at minimum:

- property: `map<string,unique_ptr<ObjectFactory>> library;`
- capability: `unique_ptr<Object> Search(string);`.

InputDevice

The InputDevice class is a wrapper for the SDL_Event. (we poll the SDL_Event and then, if an event exists, we translate it to a game event and return it). The purpose here is to provide a layer between the Game and the SDL2 Libraries. The InputDevice has the following properties:

- `unique_ptr<SDL_Event> event;`

and the following capabilities:

- `bool Initialize(); //initializes the InputDevice by creating the "event" unique pointer`
- `GAME_EVENT GetEvent(); //polls SDL for the latest event and calls "Translate" to convert it to a game event`
- `GAME_EVENT Translate(); //this private method translates SDL_Event information into GAME_EVENT type information (see Definitions.h)`

Object Class

The Object class is an abstract class meant to be a generic on which the specific game objects are derived. In addition to default construction, destruction, and Initialization capability, the Object class has the following properties:

- `GraphicsDevice* gDevice;`
- `InputDevice* iDevice;`
- `shared_ptr<Texture> texture;`
- `GAME_VEC position;`
- `GAME_VEC startPosition;`
- `GAME_FLT angle;`

has the following capabilities (as well as necessary accessor methods):

- `virtual void Update(GAME_FLT) = 0;`
- `virtual void Draw(GAME_FLT, View*) = 0;`

From the Object class we will derive multiple other classes (which will correspond with artistic assets and XML configuration files provided by the instructor).

- Blue Octorok
- Red Octorok
- Blue Leever
- Red Leever

Each of these objects should contain properties and special capabilities necessary to provide unique behavior that suits them. Each object must overload the purely virtual method of its parent class in order to be instantiable. Each object must change its internal state via the Update method. Moreover, each object should display itself appropriately using the Draw method (see Viewing the Game Space as well as the demonstration video).

Construction of Game Objects via Factories

Each of the derived classes should be instantiated using an appropriate Factory class derived from a ObjectFactory class with respective overloaded unique_ptr<Object> create() methods (see class notes, example, and solution key to Homework 0). See the GameAssetLibrary class for how these factories will be used.

Viewing the Game Space

You must implement an omniscient view of the game state in the form of a class View. In addition to default construction, destruction, and Initialization capability, the View class has the following properties:

- `InputDevice* iDevice;`
- `GAME_VEC position;`
- `GAME_VEC center;`

and the following capabilities:

- `bool Initialize(InputDevice*, GAME_FLT, GAME_FLT); //the GAME_FLT values correspond to the initial x- and y-coordinates of the View object.`
- `bool Update(GAME_FLT); //The view responds to GAME_EVENTs. Initially the current GAME_EVENT is requested from the InputDevice. The View then responds to keyboard events by shifting its position in accordance with the events (in the example program, the instructor shifts the view by 2 pixels per detected event, e.g., GAME_LEFT event causes the view's x-position to be decremented by 2 pixels without any change to the view's y-position).`

Texture Class

A simple way to implement the desired Viewing functionality is to modify the Texture class. The Texture class is a conduit by which art resources represent objects in the game. However, the texture may apply conditions to this representation, e.g., by modifying the object's visual representation to include a perspective as encoded in the View class. In this task, the Texture class has identical properties and capabilities as the examples shown in class except that the Render and RenderEx capabilities have been replaced with the following capability:

- `void Draw(SDL_Renderer* renderer, View* view, GAME_VEC position, GAME_FLT angle, SDL_Rect* clip = nullptr );`

Using this data the object can determine the appropriate transformation of its position and angle in order to achieve the appropriate graphical effect.

Game Functionality

A voice annotated video demonstration of the homework's functional requirements is available on Blackboard.

Submission

You should submit your Visual Studio project as a compressed archive. First you must name your project to adhere to the format: `userid.quest.1` where userid is your UALR user ID (i.e., the part of your UALR E-mail to the left of the @). You then should create a zipped archive, and submit this archive: `userid.quest.1.zip`.

Grading Rubric

Your job is to design and implement a C++ procedural paradigm program that performs the tasks according to this specification (and any supplemental specifications provided to clarify program tasks). As would be true in a real game development scenario, you will be evaluated primarily on your code's ability to implement all of the submission and functional requirements correctly. A secondary consideration, but also important, will be the structure, readability, and documentation of your code.

Program Element Values

Category	***XP***
Incorrect submission format*	*-100*
Does Not Compile	*-300*
Lack of Comments (max)**	*-200*
Feature (function) completely missing	*-250*
Some part of feature not working	*-100*
Ineffecient Code (each instance)	*-50*
Undescriptive Identifier (each declaration)	*-50*
Late (each day, up to 4)	*-150*

The minimum number of XP you can receive for a valid attempt is 1040XP. Therefore, be sure to submit an attempt on the due date, and for each day after in which you at least an additional 150XP in completed work.

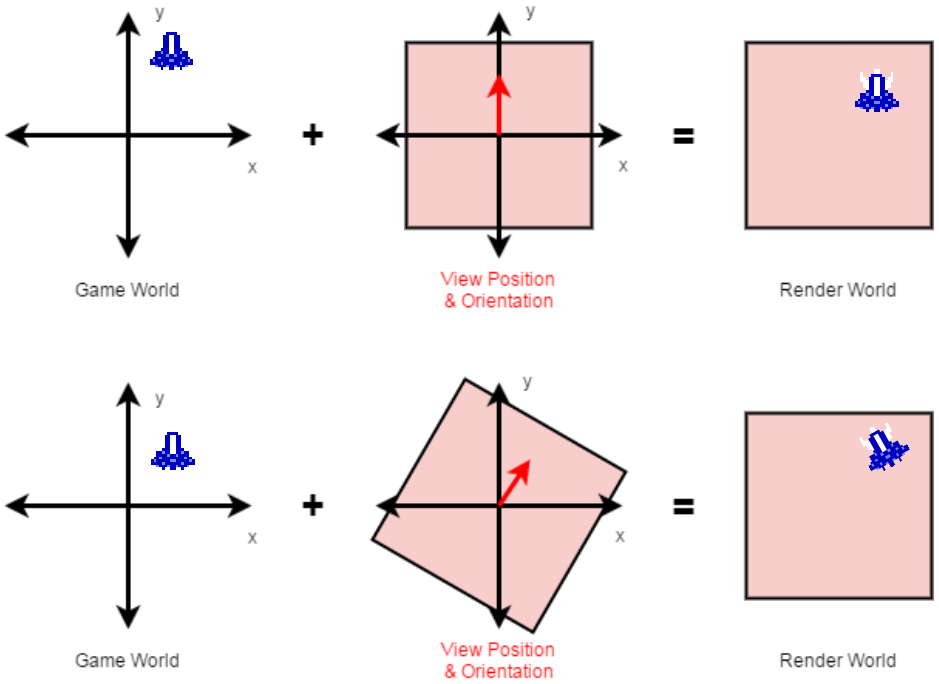
Please check to make sure you submit your project properly. There is no reason to lose points and make the instructor's life more difficult.

****Comment your code. Comments are worth major points in this assignment.***

Badge Points

You can receive up to 450 Badge points on this assignment.

(200 points) Rotational view: Implement code such that View contains both a GAME_VEC position and a GAME_FLT. The position will denote the center of the window in the game space whereas the angle member describes the relative direction in the game world's x-, y-axis that the View is facing. See the figure below (the center vector describes the position of the view with respect to the game window).



The Update method of the view will take as argument the InputDevice. It will interpret at least the following four commands

- LEFT ARROW: rotate the view counter-clockwise
- RIGHT ARROW: rotate the view clockwise
- UP ARROW: move the position of the view forward in the direction of its angle
- DOWN ARROW: move the position of the view backwards from the direction of its angle

(100 points) Object perspective: Implement code such that pressing the Tab key cycles the view to each of the game's objects such that the view center's the object with its sprite facing vertically (Requires Rotational View be implemented)

(150 points) Toggleable Map: Implement code such that pressing the 'm' toggles on/off a pixel-based map of all of the objects in the game space with the center of the current view centered in the map view. 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.

When submitting your assignment, indicate in the comments of the Blackboard submission link that you attempted the extra credit problem, and how you went about solving it.