Tutorial 6 – Screen Manager

Before starting this tutorial, you must complete the last one and have it working. Failure to do so will result in not being able to complete these steps. It is your choice but is recommended to crate a new branch each tutorial.

For us to progress our program more toward a game framework, we need to be able to construct separate levels. We will be working with screens. A screen can be a level, an intro screen or even a credits page. We will require a manager class to deal with the current level, and to change for another. This is what we will be implementing today.

We will start by constructing a game screen, which will form the basis for all our screens.

1. Create your new branch and open VS.
2. Create a GameScreen class via the solution file or Shift+Alt+C. Upon doing so VS should open the header file for you ready to begin work. Add the MACRO and include the SDL header.
    a. Add the following protected variable:

    ```
    SDL_Renderer* m_renderer;
    ```

    b. And the following public functions:

    ```
    GameScreen(SDL_Renderer* renderer);
    ~GameScreen();

    virtual void Render();
    virtual void Update(float deltaTime, SDL_Event e);
    ```

3. Next, move over to the GameScreen.cpp file and add the relevant headers and copy your function prototypes over for editing and creating function bodies with.
    a. The constructor should setup the renderer, just as you have done in previous classes. Have the protected variable equal that of the passed renderer.
    b. The destructor should then set the m_renderer to nullptr.
    c. Both the Render and Update functions contain no code. These were setup as virtual and will be overridden. Non the less, they still require function bodies. Here is the Render function as an example:

    ```
    void GameScreen::Render(){}
    ```

    Note that the virtual keyword is not needed and remember that the class name comes after function return types.

Now that we have the basis for our game screen, we need to implement a specialised one. This will cater for everything specific that level 1 requires, and will override the Render() and Update()

functions that we set up in the base. The reason for this is that the GameScreenManager class that we will create later will store a reference to a GameScreen. Therefore, we MUST implement the virtual functions.

1. Create a GameScreenLevel1 class.
    a. This requires two includes to header files, GameScreen and Commons.h. It will require SDL.h but that is included through the GameScreen.h we include.
    b. Next, we need to do a forward declaration of the class Texture2D, like so:

```
class Texture2D;
```

Forward declarations are like promises made to the compiler. What we are saying with this line of code is that we will be using Texture2D, but don't worry about what that is until we use it in the .cpp file. To keep this promise, we must include Texture2D.h at the top of the .cpp file.

    c. As we are inheriting functionality from the GameScreen class, we need to state this when we set the class type for GameScreenLevel1.

```
class GameScreenLevel1 : GameScreen
```

    d. Create the following private variable:

```
Texture2D* m_background_texture;
```

We can use Texture2D here because of our forward declaration we made above.

    e. Declare the following public functions:

```
GameScreenLevel1(SDL_Renderer* renderer);
~GameScreenLevel1();

void Render() override;
void Update(float deltaTime, SDL_Event e) override;
```

    The keyword override used here is optional, the complier will be able to tell without it but it's good practice when learning.

    f. Finally, declare the following private function:

```
bool SetUpLevel();
```

2. Move over to the GameScreenLevel1.cpp file.
    a. Include GameScreenLevel1.h (should be done for you), iostream and "Texture2D.h"(promise kept!).
    b. Add the function bodies as normal, however, the constructor will require something additional. To ensure the base class constructor is called we need to pass through

the required parameters from the GameScreenLevel1 constructor. Done like so:

```
GameScreenLevel1::GameScreenLevel1(SDL_Renderer* renderer) : GameScreen(renderer)
```

c. Withing the body of the constructor, call the SetUpLevel function. We'll complete that function later.

d. In the destructor, we will need to delete the Texture2D pointer (m_background_texture) and set to nullptr.

e. The Update function can be left empty for the moment so focusing on the Render function, this requires a call to the Texture2D pointer's Render function:

```
//draw the background
m_background_texture->Render(Vector2D(), SDL_FLIP_NONE);
```

f. Finally, the SetUpLevel function. This will set up the background texture.

```
//load texture
m_background_texture = new Texture2D(m_renderer);
if(!m_background_texture->LoadFromFile("Images/test.bmp"))
{
	std::cout << "Failed to load background texture!" << std::endl;
	return false;
}
```

The function should then return true if the texture was set up correctly.

Now that we have a game screen class for level 1 setup, we need to extend the program's functionality by enabling it to switch between screens. Currently, we only have the one screen, but you could easily extend this to create a frontend or game over screen.

1. In the Commons.h file, add an enumeration of the for the screen types, like so:

```
enum SCREENS
{
	SCREEN_INTRO,
	SCREEN_MENU,
	SCREEN_LEVEL1,
	SCREEN_LEVEL2,
	SCREEN_GAMEOVER,
	SCREEN_HIGHSCORES
};
```

This should be placed under the struct Vector2D. If you hover over the screen types, you can see the value they hold. As we covered in semester 1, enums start at value 0.

2. Next, create a GameScreenManager class. Include the SDL and Commons headers.
   a. Forward declare GameScreen, just like you did with Texture2D.
   b. Add the following private variables:

```
SDL_Renderer* m_renderer;
GameScreen* m_current_screen;
```

c. And the following public functions:

```
GameScreenManager(SDL_Renderer* renderer, SCREENS startScreen);
~GameScreenManager();

void Render();
void Update(float deltaTime, SDL_Event e);

void ChangeScreen(SCREENS new_screen);
```

3. Once done, move over to the cpp file and add the includes, (the GameScreen, GameScreenLevel1 and GameScreenManager).
   a. Within the body of the constructor, set up the renderer pointer and set the current screen to nullptr. Also, call the ChangeScreen() function, passing through the startScreen parameter. (We will create the ChangeScreen() function later).

```
ChangeScreen(startScreen);
```

   b. Next, the destructor.
      i. Set the render to nullptr
      ii. Delete the m_current_screen and then set to nullptr.
   c. The Render function just need call the m_current_screen pointers Render function using the arrow notation.
   d. Similarly, the Update function needs to do the same with m_current_screen but call the Update. Ensure to pass deltaTime and e.

```
m_current_screen->Update(deltaTime, e);
```

   e. Lastly, the ChangeScreen function.
      i. First, we need to perform a check on if the change screen pointer is nullptr or not. If it is not, we need to delete it.

```
//clear up the old screen
if(m_current_screen != nullptr)
{
	delete m_current_screen;
}
```

      ii. Next, we will implement a switch statement to determine which new screen we need to set up, but you cannot setup a pointer inside a switch statement, so we must declare them first:

```
GameScreenLevel1* tempScreen;
```

      iii. Now create the switch statement that switches based on the SCREEN enum that was passed through. Below is an example of how to cast the

GameScreenLevel1 type to a GameScreen type.

```
switch (new_screen)
{
case SCREEN_LEVEL1:
	tempScreen = new GameScreenLevel1(m_renderer);
	m_current_screen = (GameScreen*)tempScreen;
	tempScreen = nullptr;
default: ;
}
```

To test that everything we have done today works, we need to adapt our main source file Source.cpp in the following manner.

1. First, remove all references to g_texture we created in earlier tutorials.
2. Next, include the GameScreenManager header at the top of your source file.
3. Create a global pointer of GameScreenManager type:

   ```
   GameScreenManager* game_screen_manager;
   ```

4. You should have noticed that we reference delta time in our program, but what is it? We are going to be keeping track of the delta time and will be calculating this and passing it through to the update functions. Add the following time declaration below the GameScreenManager pointer you just created.

   ```
   Uint32 g_old_time;
   ```

5. In the main function.
   a. Immediately in the if(InitSDL()) statement (if returned true), set up the GameScreenManager pointer. Pass through the level 1 enum, and then start time by using the in-built SDL function GetTicks:

      ```
      game_screen_manager = new GameScreenManager(g_renderer, SCREEN_LEVEL1);
      //set the time
      g_old_time = SDL_GetTicks();
      ```

   b. In the CloseSDL function, add the following code to clean up the GameScreenManager pointer:

      ```
      //destroy the game screen manager
      delete game_screen_manager;
      game_screen_manager = nullptr;
      ```

   c. In the Render function, where we removed the Texture2D render call we want to add the GameScreenManager Render call:

      ```
      game_screen_manager->Render();
      ```

   d. The Update function requires a couple of changes. We need to add a call to the GameScreenManager's Update() function, as well as altering the current time.

i. At the top of the function create a new variable for time:

```
Uint32 new_time = SDL_GetTicks();
```

ii. Then, just before we return false at the close of the function add:

```
game_screen_manager->Update((float)(new_time - g_old_time) / 1000.0f, e);
```

iii. Right after this set g_old_time to equal new_time:

```
g_old_time = new_time;
```

6. Try building and running the program. You should have exactly the same output as before.

Possible issues

If you are getting redefinition error, then it is possible that you have not included the #ifndef macros as explained in Tutorial 5. Reread Tutorial 5 and add the macro definitions to all headers.

It is possible that your program runs but does not function how you expect. This will most probably be due to copying the code snippets from this tutorial in the wrong place. Go through each step and check that all the code is where it should be. This will be the first thing that the tutor will do, so it is in your best interest to do this first before asking for assistance. You can also refer to the videos for help.

Additional Work

Try creating a title screen that draws a different background image. Have the GameScreenManager switch from the title to the level on a key press.