2D Engine: Structure

Ricard Pillosu | Marc Garrigó

Learning Goals

- Understanding how game engines and video games are structured
- Fast track into creating a 2D game with SDL
- Put your C++ to test
- **Be able to explain all ins and outs of this code**
- Fit in your portfolio of github code
- Be a testbed for other prototypes

How games are structured

- Video games are **real time**:
  - We will iterate forever until application finishes
  - We will control time accurately
- Video games are **large and complex**:
  - We will structure the code to allow it to grow
  - Divide and conquer: Individual modules with minor dependencies
- Video games use **lots of resources**:
  - (Later) we will look into optimization techniques

About SDL

- SDL is our library of choice to pretty much everything
- It is fast, stable and multiplatform
- It's well organized C API makes it easy to work with it
- Take a look at the documentation, you will need it
  - The web is being ported, here you can find the old one
- It has other library extensions to render fonts, audio, texture loading, etc
- It is considered good practice to hide SDL details inside each Module
  - In theory we should be able to replace SDL with other library maintaining the same API

Let’s start with Main

- Check **main.cpp**
- Usually the goal is to have a very thin main function
- In this case we use a simple state machine approach
- All complexity should be delegated to the main **application controller**

Application controller

- Check **application.cpp / .h**
- Is the main organizer of the modules for the game
- All modules that want to talk to each other need to pass through it
- Its responsibility:
  - Allocate and free the modules
  - Call all the modules in the right order
  - Organize the flow of execution:
  - Init -> Start -> (PreUpdate / Update / PostUpdate) -> CleanUp

Module base class

- Acts like an interface / blueprint to create modules
- Allows application to deal just with this type of pointer

  (it does not need to know implementations of each module)
- Important decision if we make methods virtual or pure virtual
- Your take ?

Implemented modules

- *ModuleWindow:* deals with window context creation
- *ModuleRender:* deals with rendering to buffers
- *ModuleInput:* handles keyboard/mouse/gamepads detection
- *ModuleTextures:* loading and freeing textures
- Which other modules you think we will add ?

TODO 1

“Make the application properly close when ESC is pressed (do not use exit())”

- Check the documentation for SDL_GetKeyboardState
- Remember to exit **properly**

TODO 2

“Create options for SDL_WINDOW_BORDERLESS, SDL_WINDOW_RESIZABLE, SDL_WINDOW_FULLSCREEN_DESKTOP (same way as with FULLSCREEN)”

- Simple enough if you follow the code style around FULLSCREEN
- Remember to use the awesome F12 in visual studio

TODO 3

*“Create a new method **Start()** that should be called for all modules just before the first frame”*

- Check how **Init()** and **Update()** are implemented
- **Start()** is very useful since you know that all modules have “Init()” before it
  - Loading resources (textures, audio, ...) after everything is initialized
  - Start could be called several times

TODO 4

*"We need to have three updates, add them: **PreUpdate Update PostUpdate**"*

- To structure correctly the order of events we need more granularity than just one Update
  - PreUpdate: prepare everything for the upcoming frame
  - Update: normal logic update like moving stuff
  - PostUpdate: frame cleanup and render to screen
- Again, follow the code around you
- Things can get pretty crazy, here you have a good example

TODO 5

“Now that we have PreUpdate/PostUpdate/Update move the code around so we can properly render”

- Check documentation for SDL_RenderClear and SDL_RenderPresent
  - Think where each should be properly called

TODO 6

“Free all module memory. We should remove all App memory on close.”

- Remember how to free containers:
  - Free memory from the container
  - ... then the container itself!

TODO 7

*“Create a new **scene** module that loads a texture and draws it on the screen”*

- We will put all this work to test
- Your own module! check how they are created in **Application.cpp / .h**
- Pick the right method to load the texture and render it
  - You can find a texture inside the *“Game/”* folder, set as the current *WorkingDirectory*

Homework

- Open the homework project and check the new modules:
  - Audio, FadeToBlack
  - Player, SceneKen
- Follow the exercises in “TODO” comments
- Get familiar with the code structure, it will be the base for your 3D Engine
- The code has **a lot** of room for improvement.
  - Suggest your own improvements in the next class (8 Nov.)