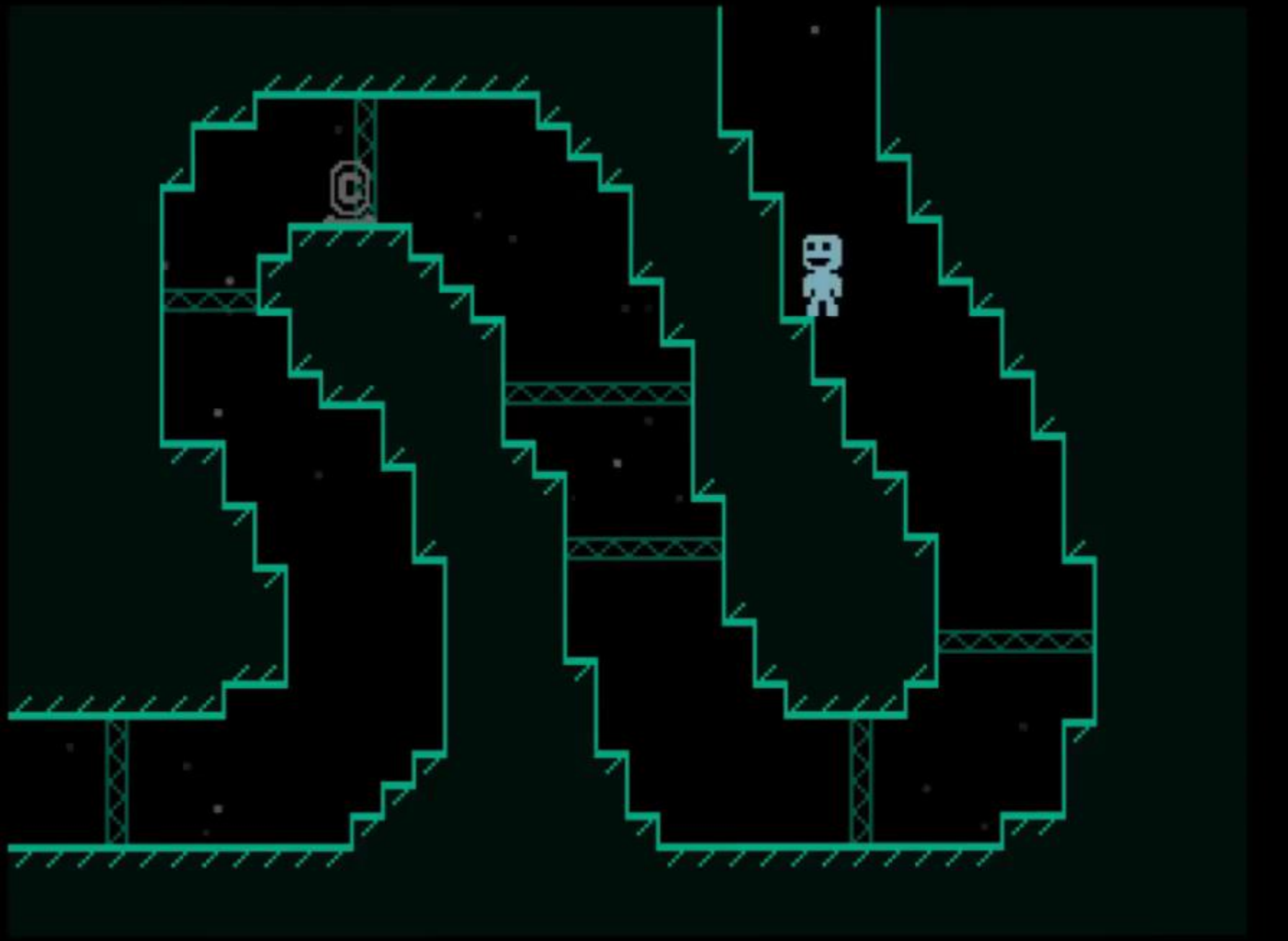
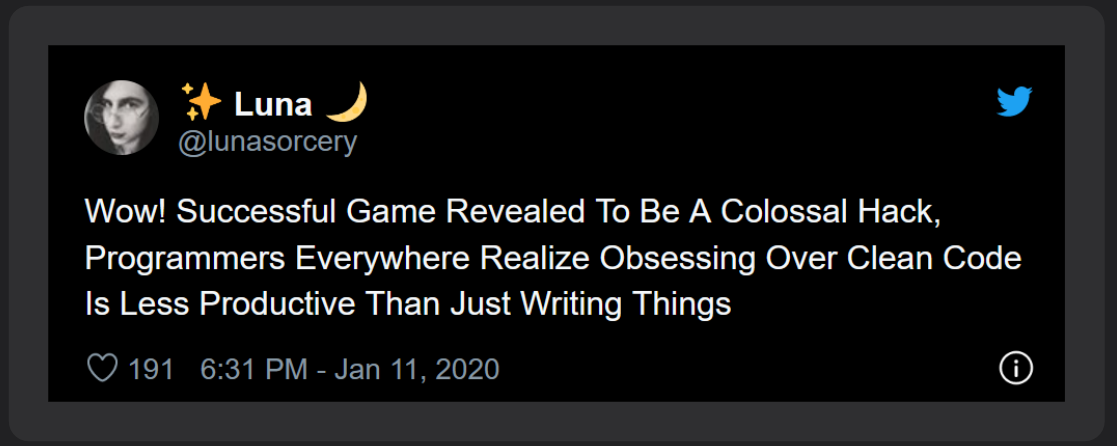

GAME PROGRAMMING IN C++
2020

Digresija





https://github.com/TerryCavanagh/VVVVVV/tree/master/desktop_version

3rd party libraries

Zašto?

- Dupliranje posla
- Potencijalno su osobe čija je biblioteka eksperti iz te oblasti
- Ubrzava naš razvoj
- Fokusiramo se na ono što stvarno želimo da uradimo
- Domaći: How Logging Made me a Better Developer

Kako?

Kopiranje celog koda	Biblioteke	Package manager
• **Header only** • Compiled	• Dinamičke biblioteke – distribuiranje uz program ili oslanjanje na postojeće • Statičke biblioteke – distribuiranje u izvršnoj datoteci • Tutorijal	• Conan • Cmake ExternalProject • git submodule • **NuGet** • vcpkg

Logging

Logging

- Logovanje je proces snimanja stanja i akcija na sekundarni interfejs
- Čemu služi?
  - Debugging
  - Analytics
- Nema uticaj na funkcionalnost programa *
- Bitno je da se može uključiti i isključiti na jednostavan način zbog uticaja na performanse

spdlog

- https://github.com/gabime/spdlog
- Header-only ili compiled
- Koristi {fmt} za formatiranje teksta
- Vrlo konfigurabilna biblioteka
- Dobra dokumentacija
- MIT licenca

Formatiranje teksta

- Pre C++20
  - printf
  - iostream
  - {fmt}
- Počev od C++20
  - std::format (link) – nastao iz {fmt}

```
fmt::print("Hello, {}!", "world");  // Python-like format string syntax
fmt::printf("Hello, %s!", "world"); // printf format string syntax
```

```
std::string s = fmt::format("I'd rather be {1} than {0}.", "right", "happy");
// s == "I'd rather be happy than right."
```

```
fmt::memory_buffer buf;
format_to(buf, "{}", 42);    // replaces itoa(42, buffer, 10)
format_to(buf, "{:x}", 42);  // replaces itoa(42, buffer, 16)
```

Logging API

- Uglavnom u vidu makroa (ili variadic template u C++11)
- Nivoi logovanja:
  - Info / Trace / Debug `LOG_INFO`
  - Warning `LOG_WARNING`
  - Error `LOG_ERROR`
  - Critical / Fatal `LOG_CRITICAL`
- Uključeno u Debug buildovima, isključeno u Release buildovima

Logger.h

Engine

```
21  #ifndef NDEBUG
22
23  #define LOG_CRITICAL(...) \
24      do \
25      { \
26          // TODO: Implementation
27      } \
28      while (0)
29
30  #define LOG_ERROR(...) \
31      do \
32      { \
33          // TODO: Implementation
34      } \
35      while (0)
36
37  #define LOG_WARNING(...) \
38      do \
39      { \
40          // TODO: Implementation
41      } \
42      while (0)
43
44  #define LOG_INFO(...) \
45      do \
46      { \
47          // TODO: Implementation
48      } \
49      while (0)
50
51  #define ASSERT(x, ...) \
52      do \
53      { \
54          if(!(x)) \
55          { \
56              // TODO: Implementation
57              __debugbreak(); \
58          } \
59      } \
60      while (0)
```

```
37  #define LOG_WARNING(...) \
38      do \
39      { \
40          // TODO: Implementation
41      } \
42      while (0)
```

Nivo apstrakcije

- Nikad ne želimo da koristimo strani kod direktno
  - Zbog želje da imamo uniforman API prema korisnicima
  - Nekad ne želimo da imamo ceo API strane biblioteke otvoren u našem kodu
  - Cross-platform
  - Lakša zamena implementacije

Nivo apstrakcije

Logger.h

Logger.h — E:\sandbox\matf\MATFGame\Engine\src\Logger\Logger.h

Engine — (Global Scope)

```
1   #pragma once
2
3   #include <spdlog/spdlog.h>
4   #include <memory>
5
6   using externLogger_t = ???;
7
8   namespace Engine {
9
10      class Logger
11      {
12      public:
13          static void Init();
14
15          inline static externLogger_t& GetLogger() { return *s_Logger; }
16      private:
17          inline static std::unique_ptr<externLogger_t> s_Logger;
18      };
19  }
```

spdlog

- Pruža nam svoje mogućnosti kroz klasu `spdlog::logger`
- Iz dokumentacije

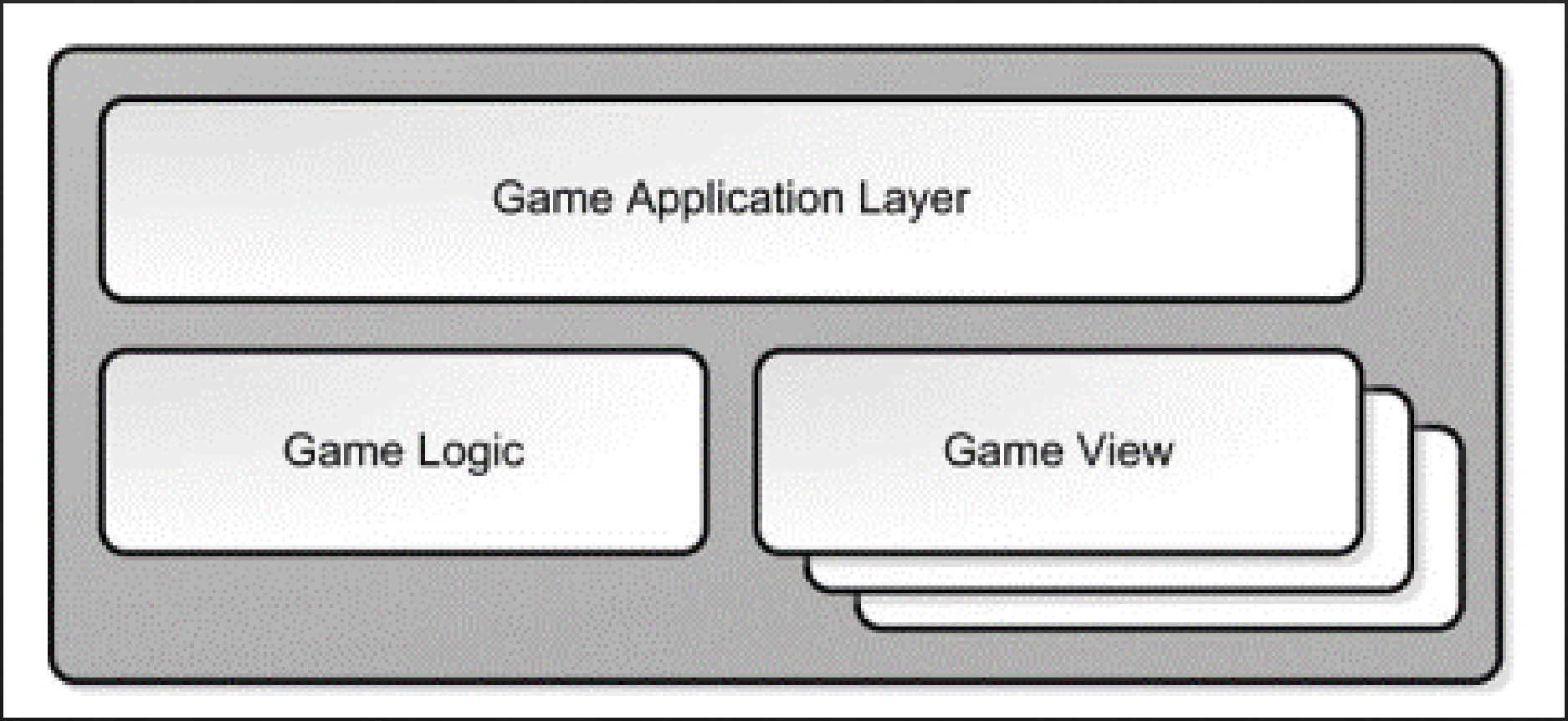
Logger with multi sinks - each with different format and log level

```
// create logger with 2 targets with different log levels and formats.
// the console will show only warnings or errors, while the file will log all.
void multi_sink_example()
{
    auto console_sink = std::make_shared<spdlog::sinks::stdout_color_sink_mt>();
    console_sink->set_level(spdlog::level::warn);
    console_sink->set_pattern("[multi_sink_example] [%^%l%$] %v");

    auto file_sink = std::make_shared<spdlog::sinks::basic_file_sink_mt>("logs/multisink.txt", true);
    file_sink->set_level(spdlog::level::trace);

    spdlog::logger logger("multi_sink", {console_sink, file_sink});
    logger.set_level(spdlog::level::debug);
    logger.warn("this should appear in both console and file");
    logger.info("this message should not appear in the console, only in the file");
}
```

Application layer



Application layer

- Stvari vezane za operativni sistem
- Životni ciklus aplikacije i sistema
- Inicijalizacija logike
- Bazna klasa u engine-u, igra treba da je nasledi i popuni

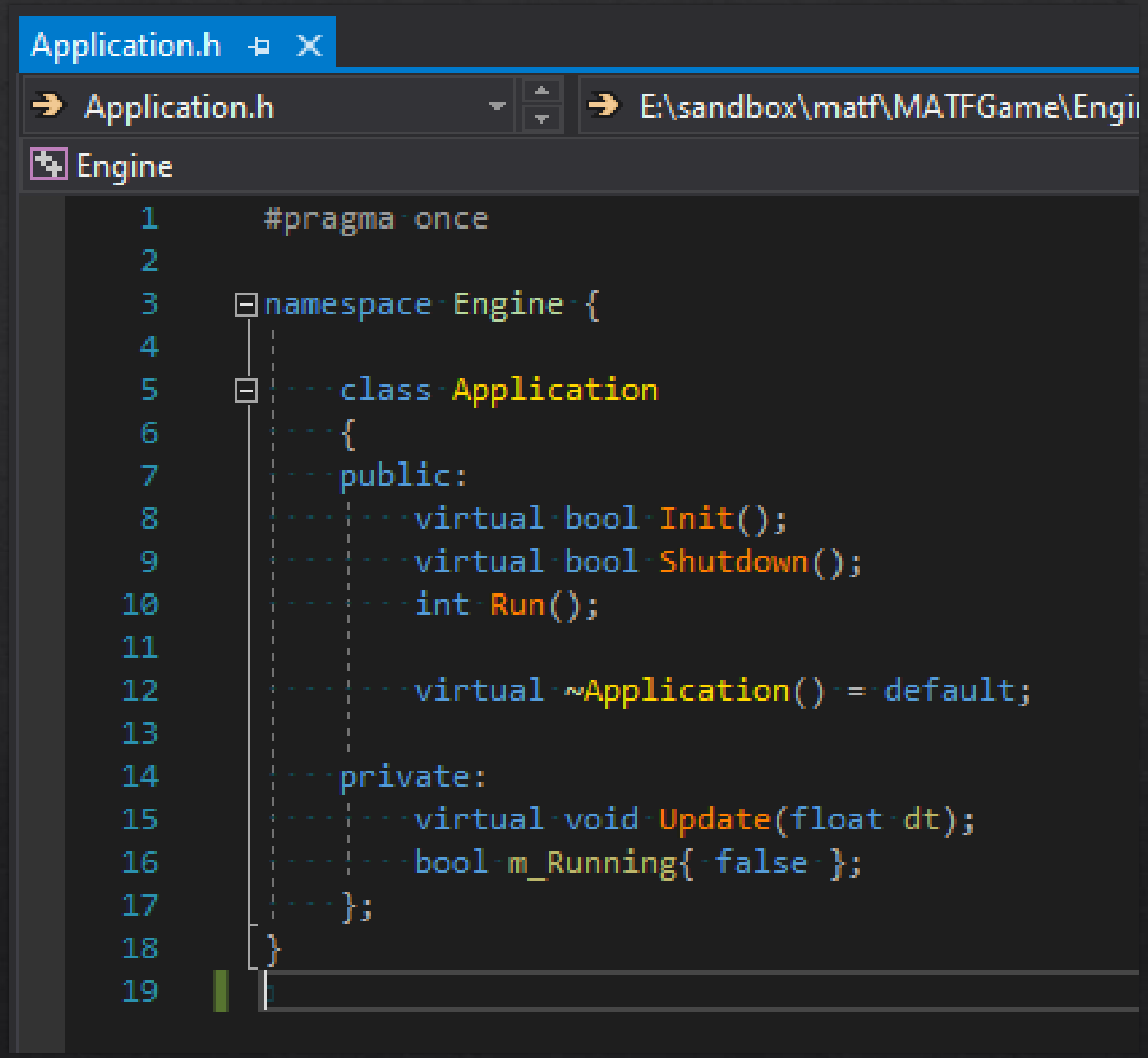
Digresija

Virtuelne funkcije

- Dynamic dispatch / Polymorphism
- Funkcija se bira u runtime-u
- vtable
- Zašto nisu sve funkcije virtuelne?

Two-phase construction

- Inicijalizacija statičkih varijabli
- Lakša reinicijalizacija / mogućnost pool-ovanja
- Povratna vrednost / Status inicijalizacije
- Poziv virtuelnih funkcija prilikom inicijalizacije


Application.h
Application.h
E:\sandbox\matf\MATFGame\Engi
Engine
#pragma once
namespace Engine {
class Application
{
public:
virtual bool Init();
virtual bool Shutdown();
int Run();
virtual ~Application() = default;
private:
virtual void Update(float dt);
bool m_Running{ false };
};
}

Application.cpp

Engine.Application.Init → bool Application::Init()

Engine

```cpp
#include "Application.h"
#include "Logger/Logger.h"

namespace Engine {

    bool Application::Init()
    {
        LOG_INFO("Initializing application");

        // Main Renderer initialize

        // Window initialize

        // InputManager initialize

        // Entity Manager initialize

        // Render system initialize

        // Camera Controller initialize

        // Physics system initialize

        return true;
    }

    bool Application::Shutdown()
    {
        LOG_INFO("Shutting down application");

        return true;
    }
```

```cpp
    int Application::Run()
    {
        m_Running = true;

        // Main loop
        while (true)
        {
            float argumentForUpdate = 1.0f; // TODO: Remove
            Update(argumentForUpdate);
        }

        m_Running = false;

        return 0;
    }

    void Application::Update(float dt)
    {
        // Update all systems
    }

}
```

Vežba

- Napraviti instancu klase Application u main.cpp
- Inicijalizovati pravilno
- Prekinuti izvršavanje koda
- Završiti program pravilno

```cpp
int main(int argc, char* args[])
{
    Engine::Logger::Init();
    LOG_INFO("Logger initialized!");


    // TODO Create Application and init
    Engine::Application* app = new Engine::Application{};
    bool success = app->Init();
    if (!success)
    {
        LOG_CRITICAL("Application failed to initialize!");
        return 1;
    }

    app->Run();

    app->Shutdown();

    delete app;

    return 0;
}
```

Na sledećem predavanju

- Počinjemo da radimo prave stvari
- Rendering
- Pravljenje prozora
- Još 3rd party biblioteka