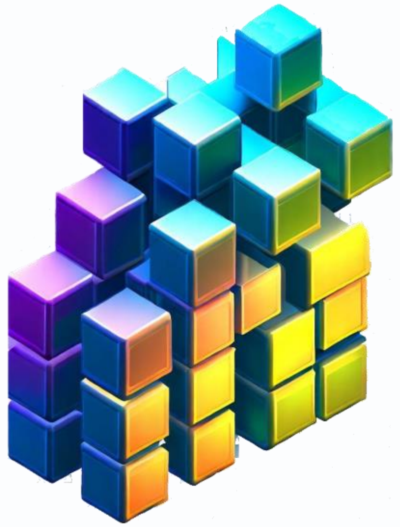
Italian C++ Community
++it
10 Years
www.italiancpp.org

Unreal Engine in C++, from scratch

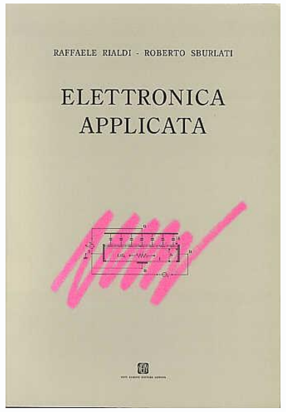


M.Eng. Raffaele Rialdi, @raffaeler
Microsoft MVP Most Valuable Professional

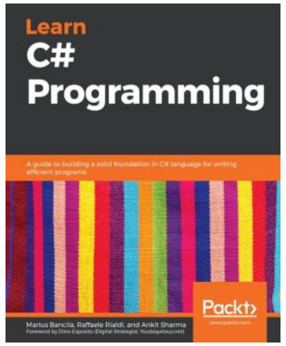
Italian C++ Conference 2023
June 10, Rome

Who am I?

- Raffaele Rialdi: @raffaeler also known as "Raf"
  - Master degree in Electronic Engineering, University of Genoa (Italy)
  - Occasional teacher at the Informatics Engineering University of Genoa
- Consultant in many industries
  - Manufacturing, racing, healthcare, financial, …
- Speaker and Trainer around the globe (development and security)
  - Italy, Romania, Bulgaria, Russia, USA, …
- Proud member of the great Microsoft MVP family since 2003


RAFFAELE RIALDI - ROBERTO SBURLATI

ELETTRONICA APPLICATA


Learn C# Programming

Packt>

Agenda for today

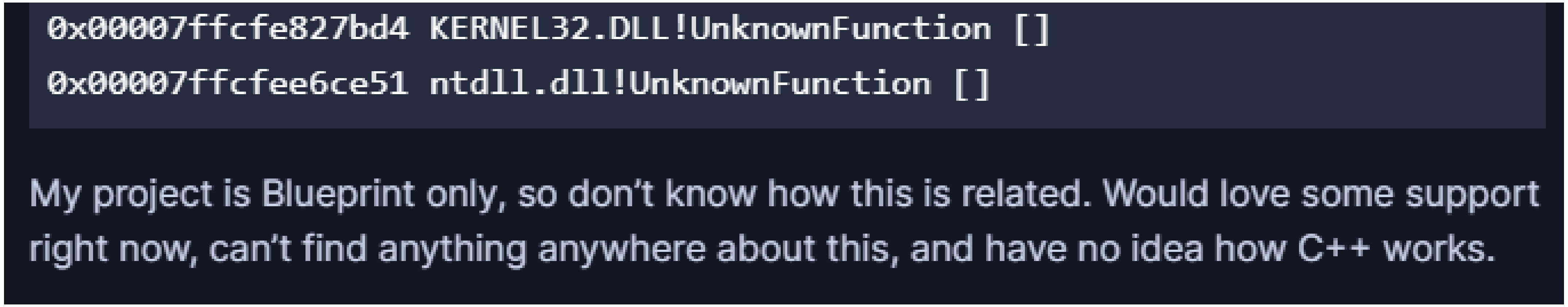
- Getting started with UE
  - A brief overview on the UE architecture
  - UE Runtime fundamentals
  - Installation tips, project types, project structure and git source control
- A retro-game Demo project
- Project structure
  - The fundamental building blocks needed in a project
  - Creating a C++ class and getting the right base class
  - Creating Blueprints based on your classes

What is Unreal Engine

- A framework to create 3D games, presentations, movies, …
- Includes a visual editor to build virtual worlds, the graphic engine, many libraries, toolchain and code generation tools.
  - Entirely written in C++ apart from the build system which uses C#
  - Visually generated classes are called Blueprints
- Many other developers mix Blueprints with C++
  - We will see why Blueprints are desirable even when using C++
- The STL library cannot be used (with few exceptions)
  - This limits the portability of standard algorithms.

Blueprints vs C++

- Many UE developers do not write a single C++ line of code
  - The editor visually creates C++ classes called Blueprints
  - The developer builds the logic graphically



```
0x00007ffcfe827bd4 KERNEL32.DLL!UnknownFunction []
0x00007ffcfee6ce51 ntdll.dll!UnknownFunction []
```

My project is Blueprint only, so don't know how this is related. Would love some support right now, can't find anything anywhere about this, and have no idea how C++ works.

https://forums.unrealengine.com/t/unreal-4-16-1-debug-game-build-error-icu-data-directory-was-not-discovered/395338/37

- Using C++, the stack trace can easily show the issue culprit.

Setting up the development environment (Windows)

- Minimum HW requirements
  - Windows 10 64 bit, 4+ Core Intel/AMD, 8GB Ram, DirectX 11 or 12
  - Far better: a powerful GPU with 8+GB Video Ram
- Install the Epics Game Launcher + VS2022 (C++ workspace)
  - Ensure to have the most updated video drivers
- On the first run, be <u>very</u> patient (thousands of shaders)
- VS2022 is <u>strongly</u> suggested
  - Recently the VC++ team provided specific support to UE developers
- Install the plugin "Visual Studio Integration Tools"

Source control

- Perforce is the most popular UE source version control system

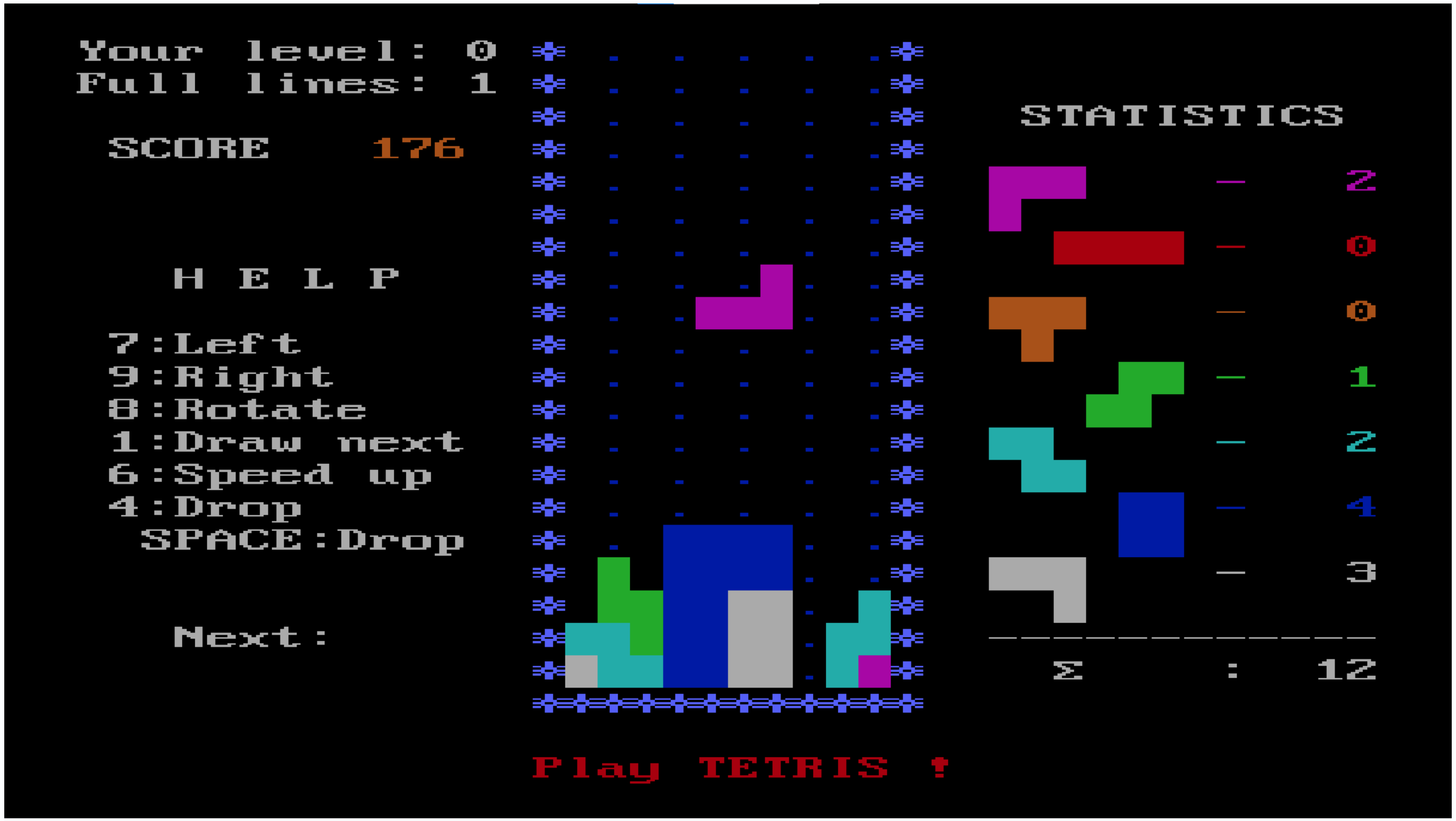
- BUT we can use GitHub as well if:

1. Do NOT use the standard UE or VisualStudio .gitignore files
2. You will find my .gitignore in the demo repository
3. If you then need storing larger files (typically assets)
    - Configure the Large File Support in GitHub (it will cost you money)
        - https://medium.com/projectwt/setting-up-git-large-file-storage-for-unreal-engine-projects-1854d6337177
        - Add *.vcxproj and *.pdb to the .gitattributes file **before** committing!

Demo time!

https://github.com/raffaeler/UnrealEngineCppIntro

Dissecting the remake of the Tetris game



```
Your level: 0
Full lines: 1

SCORE    176

   H E L P

7:Left
9:Right
8:Rotate
1:Draw next
6:Speed up
4:Drop
 SPACE:Drop

   Next:
```

```
STATISTICS
     — 2
     — 0
     — 0
     — 1
     — 2
     — 4
     — 3
-------------
Σ    :   12
```

Play TETRIS !

https://www.retrogames.cz/play_136-DOS.php

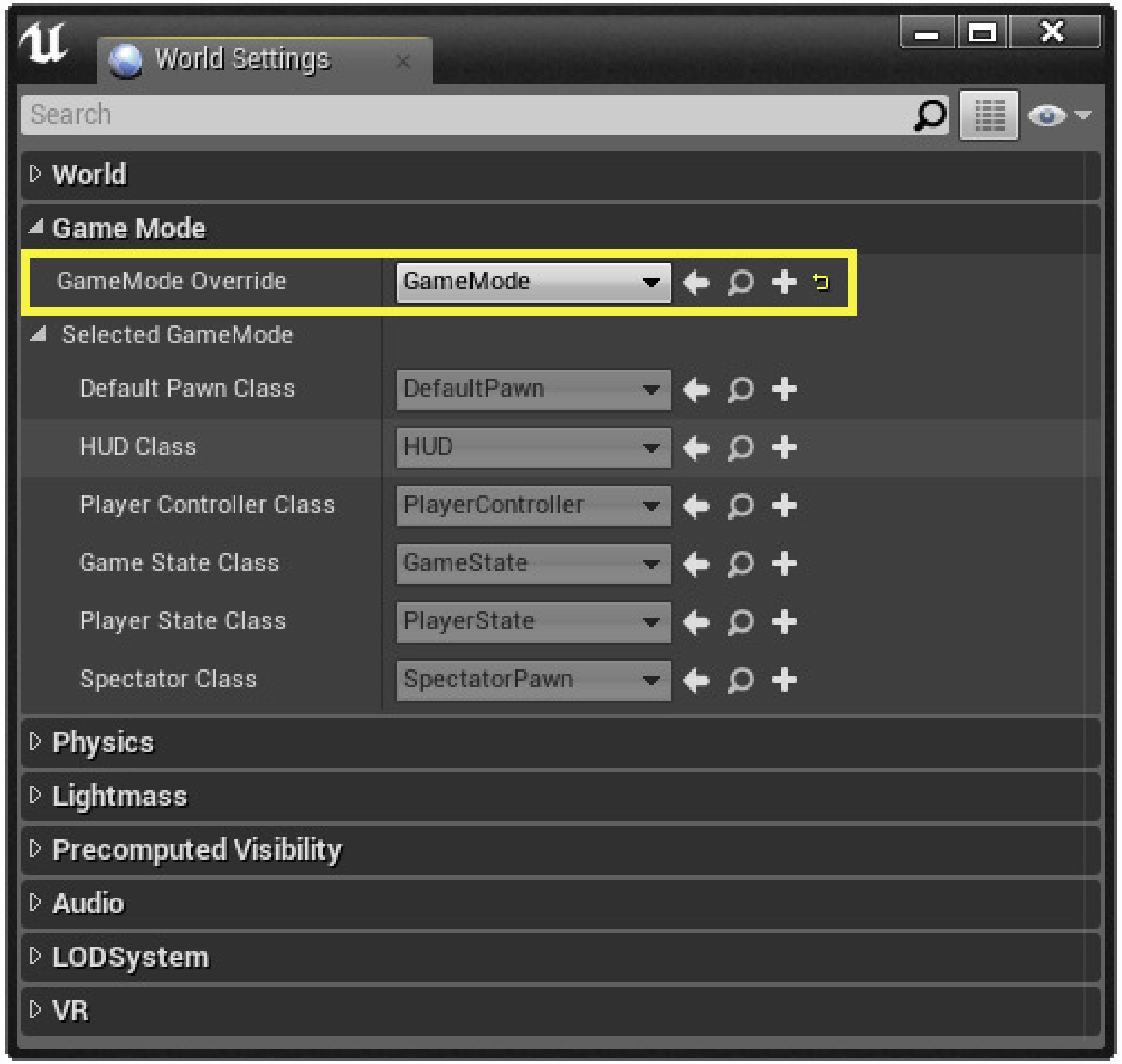
The main UE classes / objects

- UGameInstance has the longest lifetime
- AGameMode stores players/spectators along with:
  - Rules to spawn actors / players (location, when, etc.)
  - Ability to pause/resume the game
  - Transitions between levels
- AGameState stores the game state (independent from any player)
  - Time of join for each player
  - The base class for the Game Mode
- APlayerState stores the player specific state
- APlayerController dispatches the Input to its player
- APawn (derived from AActor) is anything participating to the scene
- AHUD displays the score
- Game Field (AActor).

The basics

Where is my main() ?

- Window – World Settings
- Your GameMode should be here
  - Global settings for Physics are here
- The rest of the settings are also available in the GameMode editor
- Most of the classes should override BeginPlay() to init
- Actors may override Tick() to draw/move/action.


World Settings

- World
- Game Mode
  - GameMode Override: GameMode
  - Selected GameMode
    - Default Pawn Class: DefaultPawn
    - HUD Class: HUD
    - Player Controller Class: PlayerController
    - Game State Class: GameState
    - Player State Class: PlayerState
    - Spectator Class: SpectatorPawn
- Physics
- Lightmass
- Precomputed Visibility
- Audio
- LODSystem
- VR

Garbage Collection and Reflection

- UE provides two key features: GC and Reflection
- Every object deriving from UObject is managed by the UE GC
  - This is true also for UE objects declared as fields
- The GC registers all the UObjects with AddToRoot and analyze the dependent UObjects tree via UE Reflection
  - Circular references do not prevent their removal from memory
- Standard C++ classes can still be used with new and delete
  - They can't contain UE libraries if you dynamically allocate them
  - The new and delete operators are overloaded by UE

Participating to GC and reflection

- Members are exposed to Blueprints
  - Blueprints derive can derive this class
- Enable serialization and introspection
- *UnrealHeaderTool* generates code:
  - StaticClass() is defined in this file
  - The base class is callable through "Super::"
- Use NewObject<T>() instead of new
- GetClass() returns introspected data
- A Class Default Object is an instance getting just the C++ initialization values without any Blueprint values
- UE5 recommends wrapping pointers in TObjectPtr<T>

```
#pragma once
#include "CoreMinimal.h"

#include "Foo.generated.h"

UCLASS()
class TETRIS_API UFoo : public UObject
{
   GENERATED_BODY()
public:
   UPROPERTY() int32 Count;
   UFUNCTION() void DoSomething();
}
auto cls = UFoo::StaticClass();

// Create an instance of UFoo
UFoo* foo = NewObject<UFoo>();
UClass* fooClass = foo->GetClass();
auto cdo = fooClass->GetDefaultObject<UFoo>();
```

Macros used to decorate the UE classes and members

- UCLASS(specifier) typical specifiers:
  - Blueprintable: can be used as a base class from Blueprints
  - BlueprintType: can be used as a type from Blueprints variables
  - https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/GameplayArchitecture/Classes/Specifiers/
- UPROPERTY(specifiers, meta) typical specifiers:
  - BlueprintAssignable, BlueprintCallable, BlueprintReadOnly, BlueprintReadWrite
  - Category="*abcd*"
  - Config (persist to a .ini file)
  - https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/GameplayArchitecture/Properties/Specifiers/
- UFUNCTION(specifiers, meta) typical specifiers:
  - BlueprintCallable
  - https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/GameplayArchitecture/Functions/Specifiers/
- Take a look at UENUM() and USTRUCT() as well

Actors and Components

- Actors are object in the scene like rocks or humans
  - You can enable physics and assign a mass, velocity, acceleration and also enable collision detection
- Components are special object attached to Actors
  - Examples: Scene (typically root), Meshes, Camera, ActorComponents, Sound
  - They are added as sub-objects
  - Actors can use GetComponents to access their own Components
- Meshes are 3D drawing shapes, *painted* with Materials
- Actors may want to enable the "Tick" method to enable periodical rendering

Actors and characters

- Create and destroy:
  - auto actor = GetWorld()->SpawnActor<T>();
  - auto actor = GetWorkd()->SpawnActor(UClass*);
  - actor->Destroy();

- Translate, rotate and scale
  - Use the FMatrix, FVector, FRotator types
  - actor->SetActorLocation, actor->SetActorRotation, SetActorScale3D, …+

Timers

- Managed through the *Timer Manager*
- The handler can be:
  - A member function
  - A lambda function
  - A delegate referencing a UFunction by name (reflection)
- They can be set, clear, paused and unpaused.

```
FTimerHandle ItemFallTimer;

GetWorld()->GetTimerManager().SetTimer(
    ItemFallTimer,
    this,
    &AGameField::OnItemFall,
    ItemFallDuration,
    false);

GetWorld()->GetTimerManager().SetTimer(
    MyTimer,
    [this]() { ... },
    ItemFallDuration,
    false);

GetWorld()->GetTimerManager()
    .IsTimerActive(ItemFallTimer);

GetWorld()->GetTimerManager()
    .ClearTimer(ItemFallTimer);
```

Delegates

- Delegates are function pointers available through UE reflection

```
FTimerDelegate MyTimerDelegate;
UFUNCTION() void MyTimerExpired();
```

- You bind member functions to a delegate object

```
MyTimerDelegate.BindUFunction(
    this, "MyTimerExpired");
```

- The member function has the same signature as before

```
void AGameField::MyTimerExpired()
{ ... }
```

- Similarly, you set the timer passing just the delegate

```
GetWorld()->GetTimerManager()
    .SetTimer(MyTimer,
    MyTimerDelegate,
    duration,
    false);
```

Primitives in the library

- FString — A new glorious type for the string ☺
  - Accessing the underlying buffer with the * prefix
- Containers (ranges-enabled):

TArray<T>	Similar to std::vector, used also for stack
TMap<T, K>	Similar to std::map
TQueue<T>	similar to std::queue
TSet<T>	similar to std::set / std::unordered_set

- Tuple container

TTuple<T1, T2, …>	std::tuple

- Math operations
  - See FMath static methods

Tips

Setting VS2022 as the default IDE/compiler

- Modify the BuildConfiguration.xml
  - <user>/AppData/Roaming/Unreal Engine/UnrealBuildTool

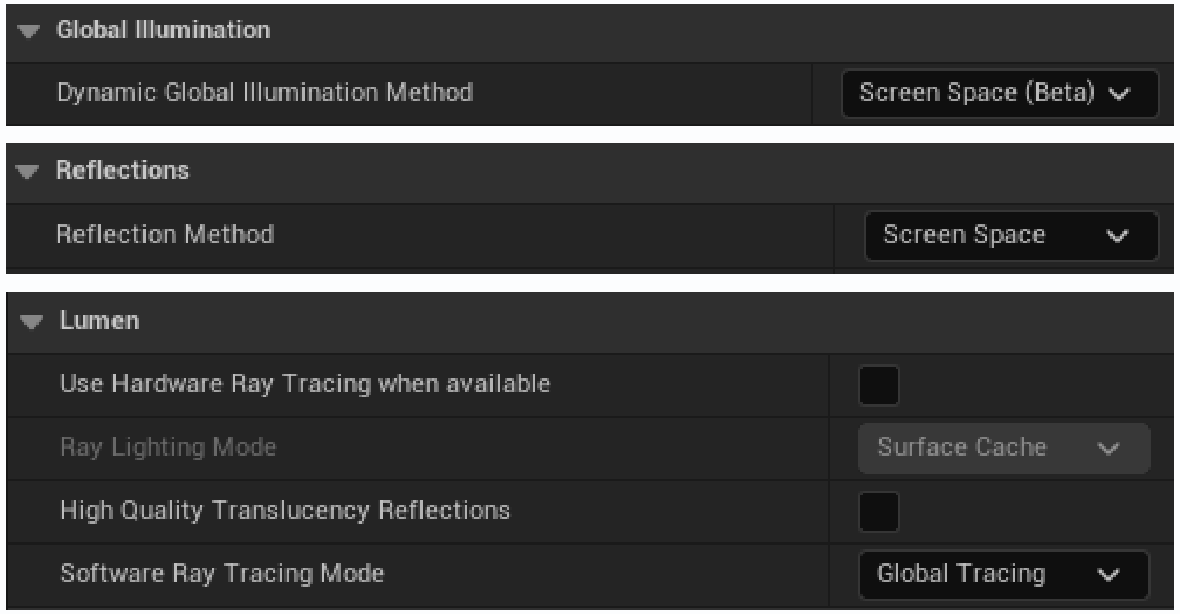
```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration xmlns="https://www.unrealengine.com/BuildConfiguration">
  <ProjectFileGenerator>
    <Format>VisualStudio2022</Format>
  </ProjectFileGenerator>
</Configuration>
```

How to create a new (really empty) project

- Create a new project from the wizard and build it
  - Shaders compilation will take a huge amount of time
  - Optimize the project settings if needed (see next slide)
- Do not save the current level
  - Create a new Level inside the Content folder, save (all black)
  - Select "Landscape Mode". Left+Right click to move over the landscape, Click on Create
  - Add a "Directional Light" from "Lights"
    - Ctrl-L + move mouse to change direction
    - Build the lights
- Project settings – filter by "Default Maps"
  - Change editor and game maps to your level name
- Enter in Modeling mode and draw the Static Meshes

Optimize (video) memory usage

- The default settings were: Lumen, Lumen, Default Tracing



Global Illumination

Dynamic Global Illumination Method	Screen Space (Beta)

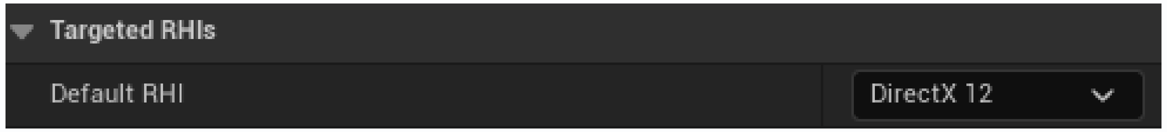
Reflections

Reflection Method	Screen Space

Lumen

Use Hardware Ray Tracing when available	☐
Ray Lighting Mode	Surface Cache
High Quality Translucency Reflections	☐
Software Ray Tracing Mode	Global Tracing

- If DirectX 12 is not available, change it to DirectX 11



Targeted RHIs

Default RHI	DirectX 12

Questions?

https://github.com/raffaeler/UnrealEngineCppIntro