GAME PROGRAMMING IN C++

2020

Sitne izmene

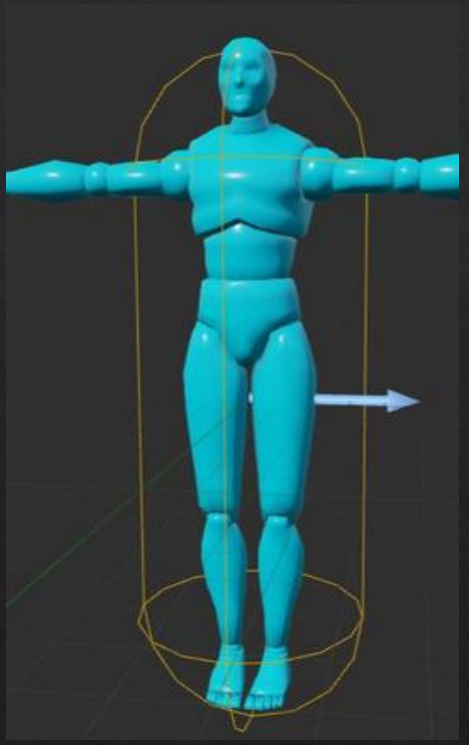
- Obrisao sve copy contructor-e i operatore dodele
- Uključio optimizacije u kompajleru (agresivniji inlining, korišćenje SIMD instrukcija)
- Probao Clang kao kompajler – testiranje performansi
- TextureManager klasa (hashmap tekstura i imena, Create, Get) – pogledati kod van časa

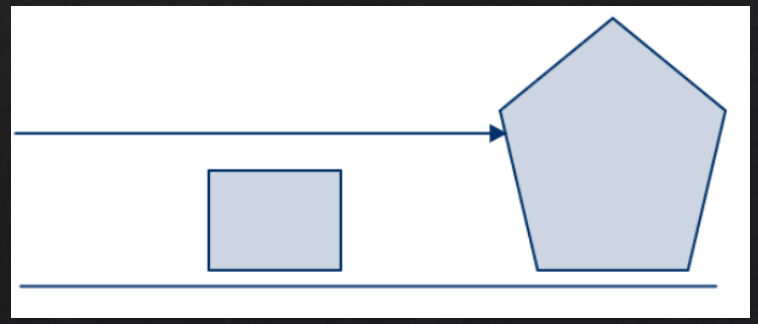
Fizika u igrama

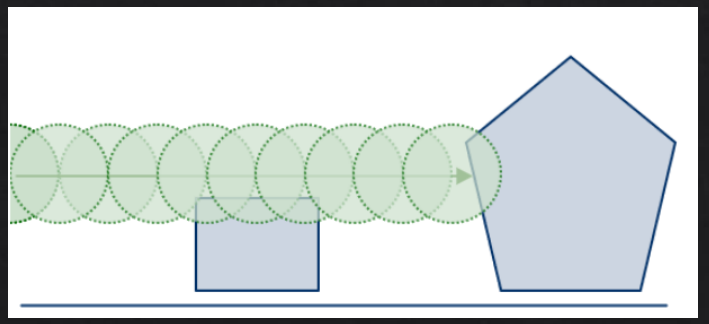
- Detekcija kolizija između objekata u igri
- Simulacija krutih tela
- Uništavanje terena, zgrada i drugih objekata
- Raycast i shape cast
- Trigger volumes
- Vozila
- Simulacija tekstila, kose, fluida

Fizika u igrama

- Slobodno kretanje
  - Pozicija, brzina, ubrzanje, sila, masa, inercija, ugaona brzina, obrtni moment
- Interakcija između objekata
  - Statičko trenje, dinamičko trenje, restitution
  - Collision mesh (!= visible mesh)
  - Raycast, Shapecast, collision reporting

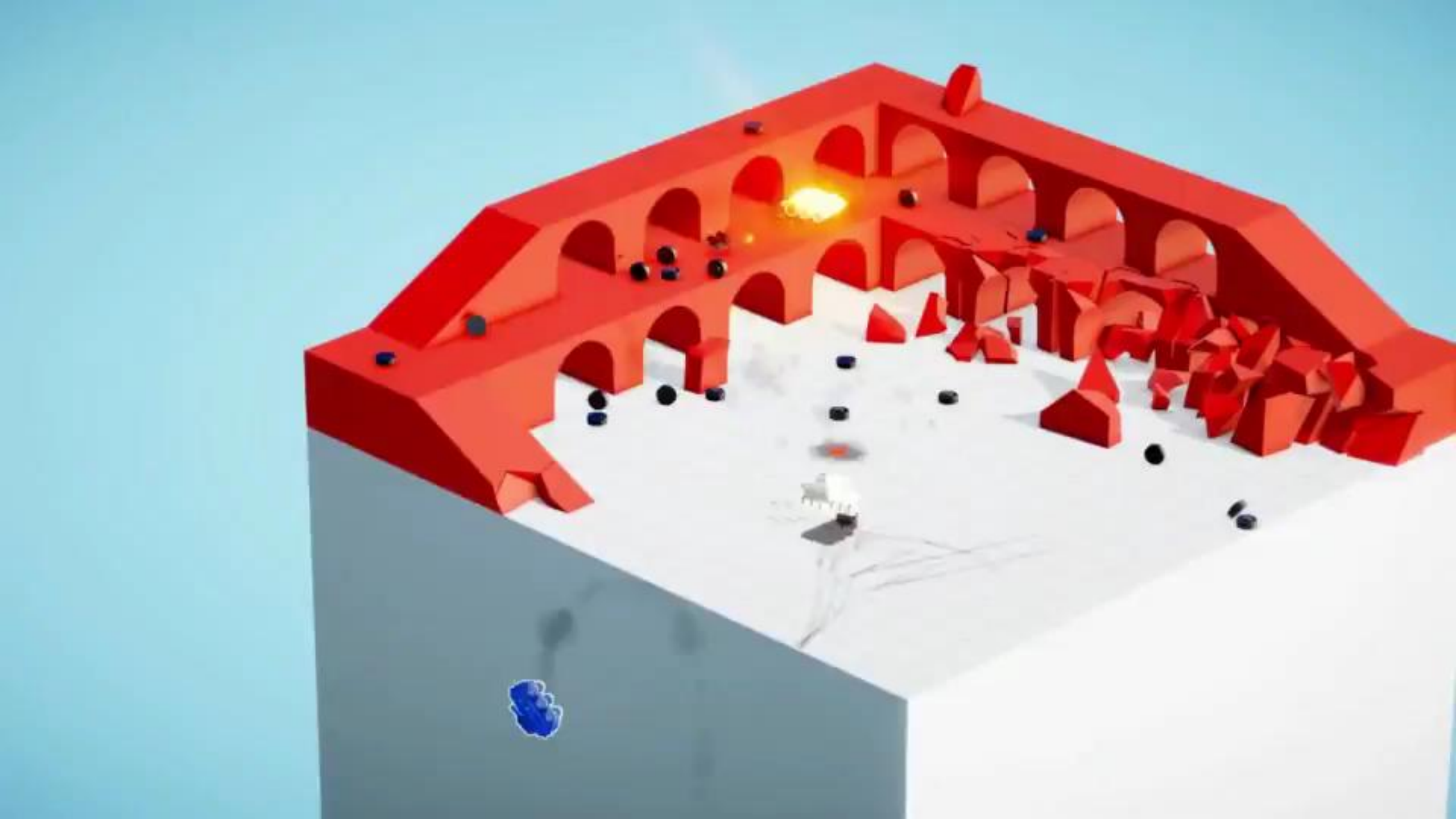






Uticaj fizike na druge sisteme

- Grafika – utiče na bounding volumes, osvetljenje i senke
- Networking – fizika koja utiče na gemeplay „mora“ da se simulira na serveru
- Snimanje i playback događaja u igri – Fizika mora biti deterministička
- AI – pathfinding
- Kompleksniji art pipeline – Objekti moraju da imaju masu, trenje…




目標： 06:00.00
目前： 00:43.28


39 Voodoo
1 [BACN]TURDLES
16 xDs|TylerDurden
5 [1501] JigsawXV
QUARTERMASTER
MAIL
33 [BS9] shmoop
43 [BS9] Stanky
Fade sits
Supply Drops E


ELIMINATE JANUS
ELIMINATE NOLAN CASSIDY
FIND CLUES
GET JANUS' DIARY
?
Q
SILENT ATTACK
PACIFY
CONCEALED
All right I'd better go.


ーーフ
幻術だ
wwww
ネル効果かミ
チキン
JPN
ATHENS 2004
000
JPN
4800
3回目
走高跳 男子


1 #2 --
2 #33 --
3 #67 --
4 #95 --
5 #21 --
6 #17 --
7 #8 --
8 #29 --
9 #67 --
10 #4 --
RACE
LAP 1 of 10
NASCAR ON FOX
BlacKNred81
Nascar 2003

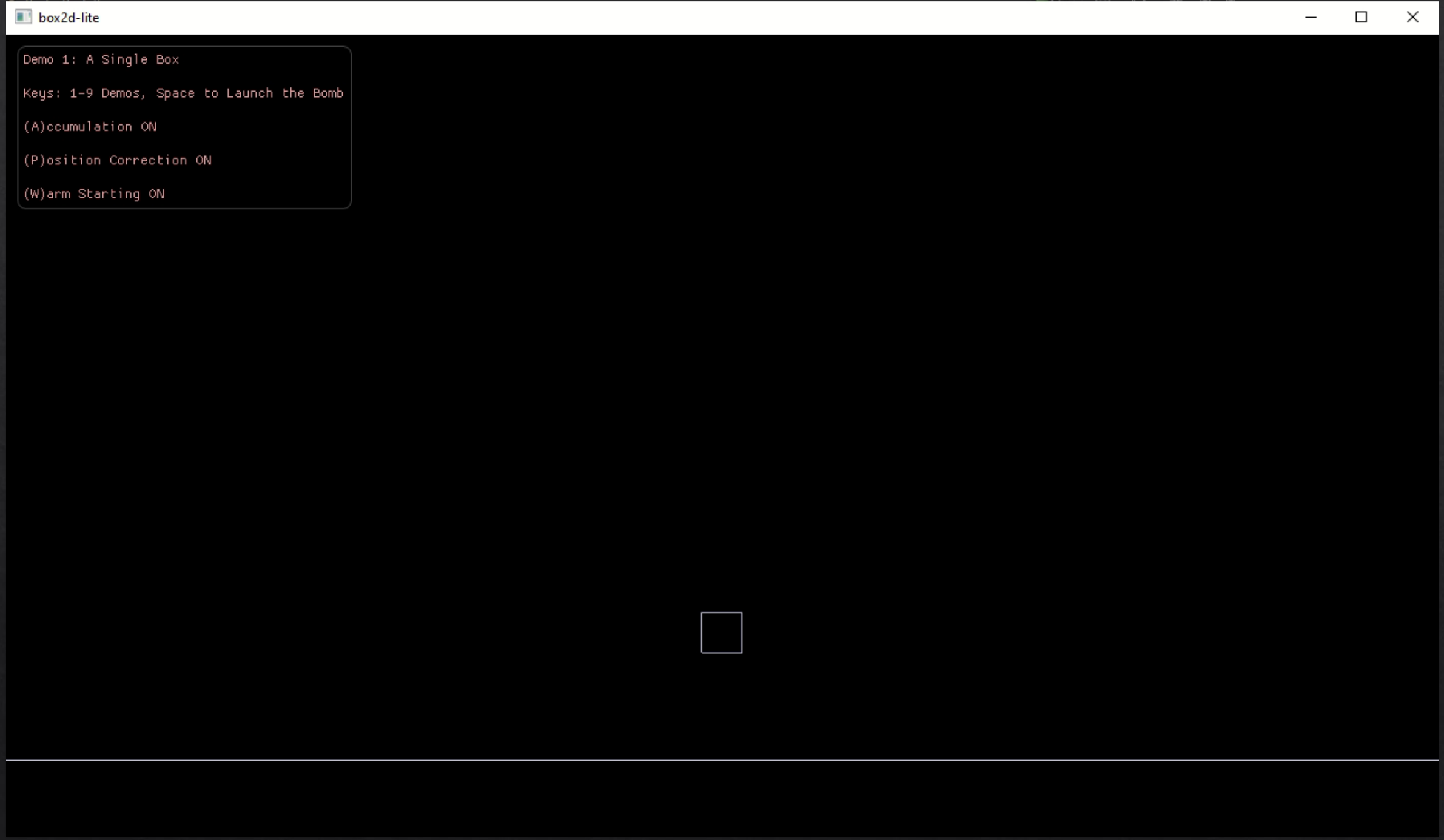
3rd party fizika

Open Source

- box2d, box2d-lite, Bullet, Chipmunk, Open Dynamics Engine, PhysX, Unreal Chaos

Proprietary

- Havok , Digital Molecular Matter, Unity RigidBody



https://github.com/erincatto/box2d-lite

Fizika u našoj igri

```cpp
struct MoverComponent : public Component
{
    vec2 m_TranslationSpeed{};
    float m_RotationSpeed{};
```

```cpp
struct TransformComponent : public Component
{
    vec2 m_Position{};
    vec2 m_Size{};
    float m_Rotation{};
```

```cpp
enum class ECollisionShape
{
    AABox,
    Circle,
};

struct CollisionComponent : public Component
{
    ECollisionShape m_Shape{ ECollisionShape::Circle };
    vec2 m_Size{}; // only using first element for circle
    std::vector<Entity*> m_CollidedWith{};
```

```cpp
void PhysicsSystem::Update(float dt, EntityManager* entityManager)
{
    // Move
    auto entitiesToMove = entityManager->GetAllEntitiesWithComponents<TransformComponent, MoverComponent>();

    for (auto& entity : entitiesToMove)
    {
        auto transform = entity->GetComponent<TransformComponent>();
        auto mover = entity->GetComponent<MoverComponent>();

        transform->m_Position += mover->m_TranslationSpeed * dt;
        transform->m_Rotation += mover->m_RotationSpeed * dt;
    }

    // Collide
    auto entitiesToCollide = entityManager->GetAllEntitiesWithComponents<TransformComponent, CollisionComponent>();

    for (auto& entity : entitiesToCollide) { entity->GetComponent<CollisionComponent>()->m_CollidedWith.clear(); }

    for (auto& entity1 : entitiesToCollide)
    {
        for (auto& entity2 : entitiesToCollide)
        {
            bool collided = CheckForCollision(entity1, entity2);

            if (collided)
            {
                entity1->GetComponent<CollisionComponent>()->m_CollidedWith.push_back(entity2);
                entity2->GetComponent<CollisionComponent>()->m_CollidedWith.push_back(entity1);
            }
        }
    }
}
```

FPS

- Update loop

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

    // Main loop
    SDL_Event event{ };
    while (m_Running)
    {
        while (SDL_PollEvent(&event) != 0)
        {
            if (event.type == SDL_QUIT)
            {
                m_Running = false;
            }
        }

        float argumentForUpdate = 1.0f; // TODO: Remove
        Update(argumentForUpdate);
    }

    m_Running = false;

    return 0;
}
```

FPS

- Update loop

Fixed delta time

The simplest way to step forward is with a fixed delta time, like 1/60th of a second:

```
double t = 0.0;
double dt = 1.0 / 60.0;

while ( !quit )
{
     integrate( state, t, dt );
     render( state );
     t += dt;
}
```


123
FINAL DEPARTURE
Investigate Morgantown Airport
PERSONAL MATTERS
Find the next journal in Sutton
HP
E
AP

FPS

- Update loop

```
double t = 0.0;

double currentTime = hires_time_in_seconds();

while ( !quit )
{
    double newTime = hires_time_in_seconds();
    double frameTime = newTime - currentTime;
    currentTime = newTime;

    integrate( state, t, frameTime );
    t += frameTime;

    render( state );
}
```

https://web.archive.org/web/20130328024036/http://gafferongames.com/game-physics/fix-your-timestep/

Syntax

Toggle line numbers

```
Uint32 SDL_GetTicks(void)
```

Return Value

Returns an unsigned 32-bit value representing the number of milliseconds since the SDL library

Code Examples

Toggle line numbers

```
unsigned int lastTime = 0, currentTime;
while (!quit) {
  // do stuff
  // ...

  // Print a report once per second
  currentTime = SDL_GetTicks();
  if (currentTime > lastTime + 1000) {
    printf("Report: %d\n", variable);
    lastTime = currentTime;
  }
}
```

Remarks

This value wraps if the program runs for more than ~49 days.

FPS

https://wiki.libsdl.org/SDL_GetTicks

Use this function to get the current value of the high resolution counter.

```
Uint64 SDL_GetPerformanceCounter(void)
```

Frame Rate

A common application of timing is to calculate the FPS, or frames per second, your program is running at. A frame is simply one iteration of your main game or program loop. Hence, timing it is quite straightforward: log the time at the start and end of each frame. Then, in some form output the elapsed time or its inverse (the FPS).

```
bool running = true;
while (running) {

	Uint64 start = SDL_GetPerformanceCounter();

	// Do event loop

	// Do physics loop

	// Do rendering loop

	Uint64 end = SDL_GetPerformanceCounter();

	float elapsed = (end - start) / (float)SDL_GetPerformanceFrequency();
	cout << "Current FPS: " << to_string(1.0f / elapsed) << endl;

}
```

FPS

https://wiki.libsdl.org/SDL_GetPerformanceCounter

Post C++11, možemo da koristimo i std::chrono

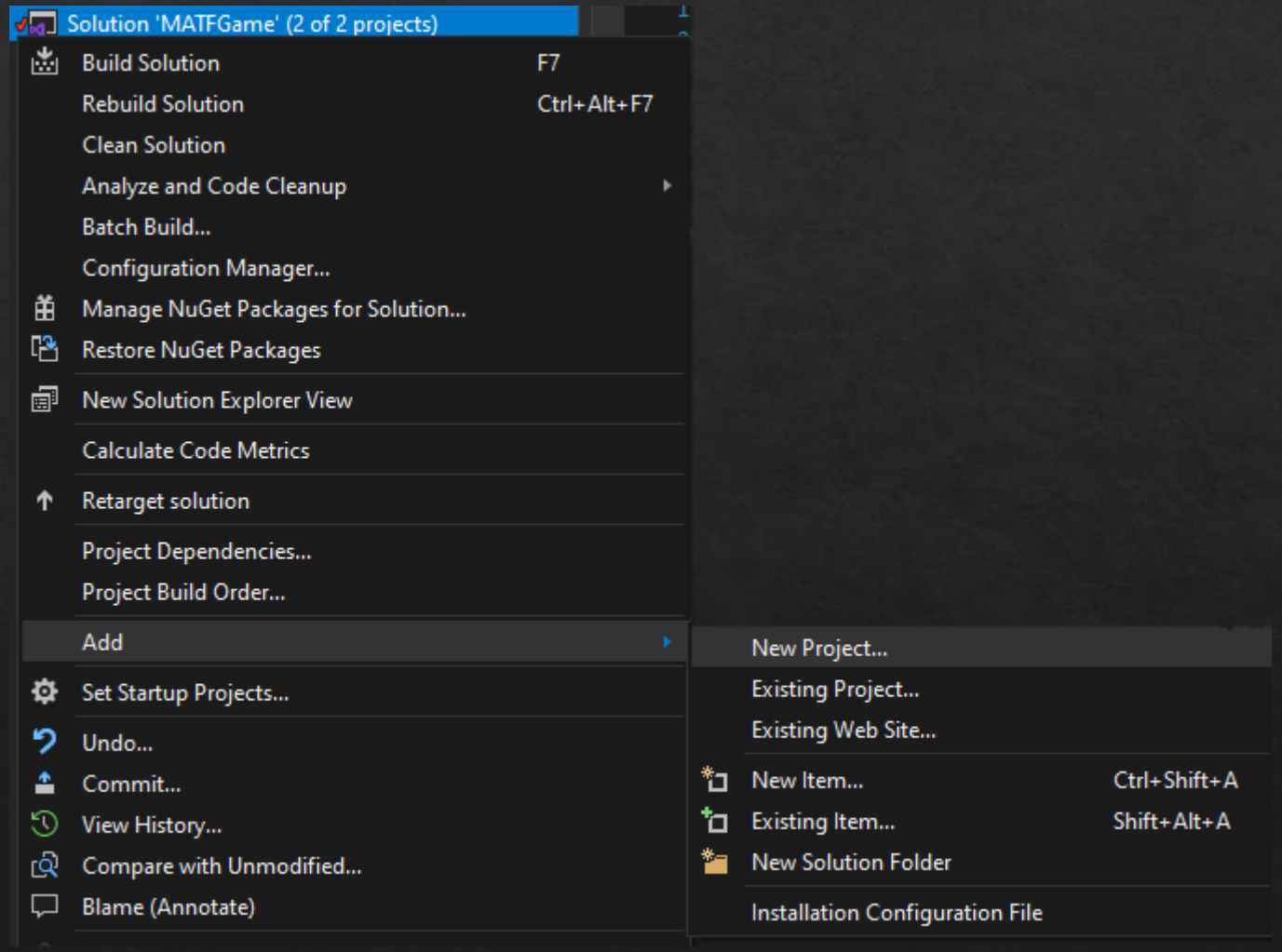
https://thenumbat.github.io/cpp-course/sdl2/08/08.html

```cpp
int Application::Run()
{
    m_Running = true;
    static auto previousFrameTime = SDL_GetPerformanceCounter();

    // Main loop
    SDL_Event event{ };
    while (m_Running)
    {
        while (SDL_PollEvent(&event) != 0)
        {
            if (event.type == SDL_QUIT)
            {
                m_Running = false;
            }
        }

        auto frameTime = SDL_GetPerformanceCounter();

        float deltaTime = (frameTime - previousFrameTime) / static_cast<float>(SDL_GetPerformanceFrequency());

        // LOG_INFO("Current FPS: {}", 1.f / deltaTime);
        Update(deltaTime);

        previousFrameTime = frameTime;
    }

    m_Running = false;

    return 0;
}
```

The Game

- Želimo da koristimo isti engine za sve igre na ovom kursu
- Želimo da odvojimo zajedničke funkcionalnosti od specifičnih

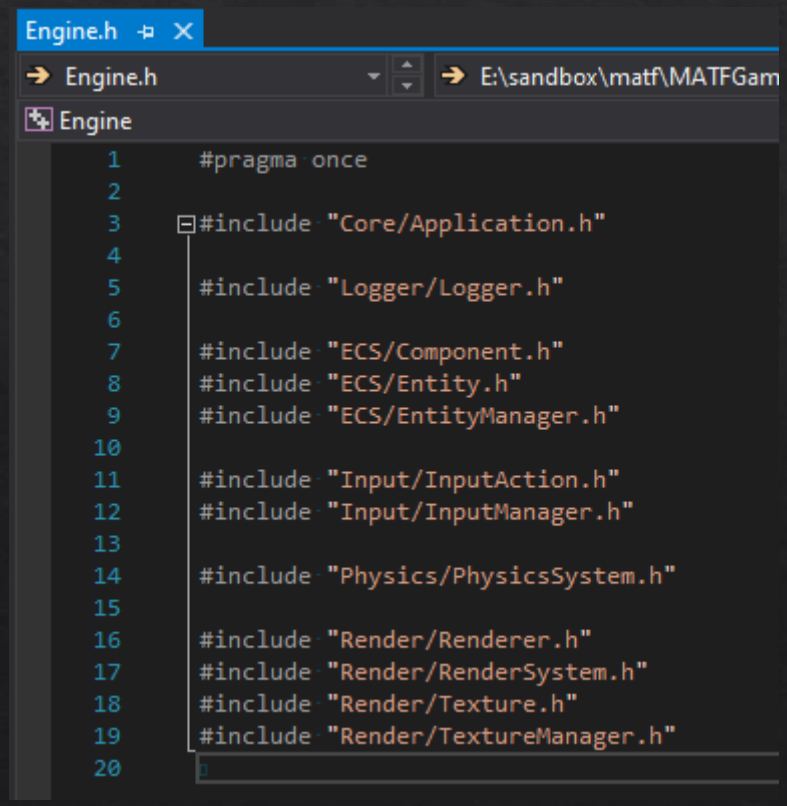
The Game



Empty Project -> Game Name

The Game

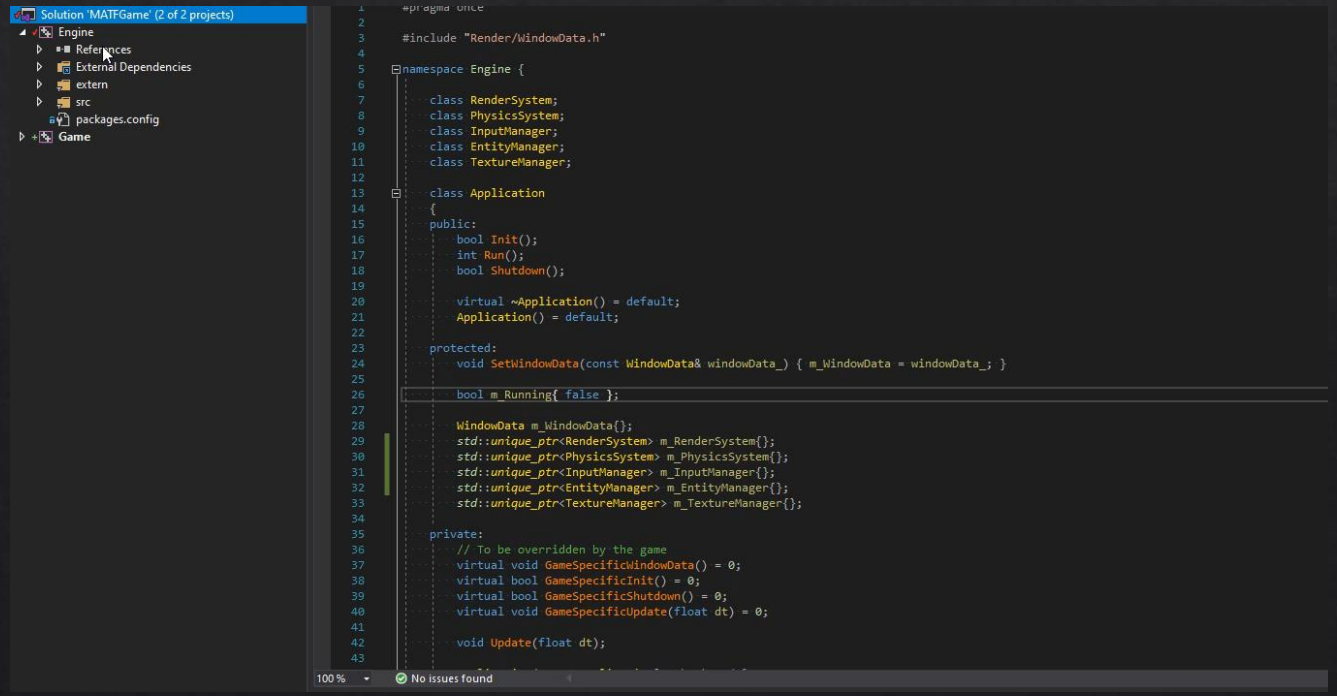
- Treba nam način da include-ujemo tipove iz Engine.h
- Pošto je Engine biblioteka, možemo da uradimo isto što i SDL, ubacimo sve javne Engine headere u novi fajl: Engine.h, koji dodamo u precompiled header Game projekta



The Game

- Pošto koristimo nuget i SDL kao dll, i ovaj novi projekat mora da ima SDL2 i SDL2_image kao zavisnosti ☹

- Druga zavisnost je sam Engine projekat



The Game

- Ukoliko želimo da dodajemo ručno neke biblioteke: https://docs.microsoft.com/en-us/cpp/build/adding-references-in-visual-cpp-projects?view=vs-2019

- Ne želimo da korisnici imaju mogućnost da menjaju `main()`

EntryPoint.h

E:\sandbox\matf\MATFGame\Engine\src\Core\EntryPoint.h

Engine

(Global Sco

```
1   #include "precomp.h"
2   #include "Core/Application.h"
3   #include "Render/RenderSystem.h"
4   #include "Render/Renderer.h"
5
6   int main(int argc, char* args[])
7   {
8       Engine::Logger::Init();
9       LOG_INFO("Logger initialized!");
10
11      auto app = Engine::CreateApplication();
```

Game-specific code

```
class Application
{
private:
    // To be overridden by the game
    virtual void GameSpecificWindowData() = 0;
    virtual bool GameSpecificInit() = 0;
    virtual bool GameSpecificShutdown() = 0;
    virtual void GameSpecificUpdate(float dt) = 0;
```

```
namespace Game
{
    class GameApp final : public Engine::Application
    {
    private:
        void GameSpecificWindowData() override;
        bool GameSpecificInit() override;
        void GameSpecificUpdate(float dt) override;
        bool GameSpecificShutdown() override;
    };

}

Engine::Application* Engine::CreateApplication()
{
    return new Game::GameApp();
}
```