**OpenGL grafikus könyvtárra épülő számítógépes játékmotor fejlesztése**

Önálló laboratórium

Beszámoló

*Rittgasszer Ákos*

Z8WK8D

Konzulens: Dr. Tóth Balázs György

# Feladat leírása

A feladat egy játékmotor fejlesztése OpebGL grafikus könyvtárra alapozva. A játékmotor megvalósítás követően pedig egy demo játék elkészítése. Az első kérdés a játékmotor fejlesztése során, hogy 2 vagy 3 dimenziós motort szeretnék implementálni. Itt úgy döntöttem, hogy először implementálom a 2D-t és utána a 3D-t. Ezek után kiválasztottam egy játékot, amivel majd fogom demózni a motort és aminek a tulajdonságaihoz fogom illeszteni a motor funkcióit.

A játék választásom, egy labirintus játékra esett. Ezt úgy kell elképzelni, hogy a játékost helyeződik valahova a labirintuson belül, és ki kell találnia belőle. Azért választottam ezt a játékot, mert kellően egyszerű ahhoz, hogy már minimális funkciók mellett lehessen implementálni, ugyanakkor könnyen lehet bonyolítani a játékmenetet, amivel folyamatosan tudom szemléltetni a játékmotorban lefejlesztett újabb funkciókat. Az is előnye a játéknak, hogy meg lehet valósítani mind 3 mind 2 dimenzióban.

Ezek alapján összegyűjtöttem mik azok a motorbeli funkciók amiket célszerű lesz megvalósítani, a jó fejlesztői- és játékélményhez.

| **2D** | **3D** |
| --- | --- |
| 2D megjelenítés | 3D megjelenítés |
| 2D fizika | 3D fizika |
| 2D kamera | 3D kamera |
| 2D karakter modellezés | 3D karakter modellezés (nem feltétlen kell) |
| 2D karakter animáció | 3D karakter animáció (nem feltétlen kell) |
| 2D fények | 3D fények |

A felsorolt képességeken kívül még szeretném, ha a motor képes lenne modelleket betölteni és azokat megjeleníteni, valamint azokkal működni. Illetve szeretném ha tudna labirintust is készíteni. Ehhez egy egyedi fileformátumot használ, ami leírja a labirintust.

A kész motorban meg lehetne adni a labirintust leíró filet, a karaktert (2D-ban mindenképp) és a játékdinamikát, ha változtatni szeretnénk az alapértelmezetten.

Ezen kívül lehet, hogy készítek, egy GUI-t is ami egyszerűbbé teszi játékmotor használatát.

A projektet verziókövetéssel szeretném fejleszteni.

# Fejlesztési folyamat

Az első lépésként összeszedtem milyen lépéseken keresztül szeretném fejleszteni a projektet:

1. Feladat pontosítása
2. Feladat specifikálása
3. Követelmények felvázolása
4. Információszerzés a témában
5. Tervezés, a követelményeket figyelembe véve
6. Implementálás
   1. Hibák rögzítése
   2. Hibák javítása
   3. Refaktorálás és újratervezés
7. Tesztelés

# Követelmények

A követelmények összeszedése a feladat specifikáció alapján történt:

* Grafikus megjelenítés OpenGL segítségével
* Inputkezelés
* Fizika
* Modellek betöltése és kezelése
* Játéklogika megadása
* Multiplatformságra nyitott program
* OO elvek követése

# 

# Információszerzés a témában

Információ szerzéshez elsősorban a játékfejlesztés során használatos design patternökkel foglalkoztam. Ekkor megismerkedtem azokkal a tervezési mintákkal amiket játékfejlesztés során szoktak használni. Az alábbiakban összefoglaltam azokat a mintákat[[1]](#footnote-0), amiket leghasznosabbnak találtam.

| **Minta** | **Leírás** |
| --- | --- |
| Command | Használható az inputokhoz kötött műveletek végrehajtására. Kiemelten hasznos, ha lehetőséget adunk, arra, hogy a felhasználó válassza ki mit melyik parancshoz köt. |
| Observer | Elterjedt minta, használják a Model-View-Controller architectúránál. Használható például achievement rendszer kialakításához. Felhasználása nagyon gyakori a játékfejlesztésben. |
| Singleton | Egy osztályból csupán egy példányt enged létezni. Túlhasználása komoly gondokat okoz, ezért alaposan át kell gondolni ha használni szeretnénk, |
| State | Az állapotok osztályként vanak implementálva. Az egyes állapotok osztályai végzik az állapothoz tartozó események végrehajtását. Használat egyszerűbb esetekben fölösleges |
| Double buffering | A renderelés szebbé tételéhez szükséges. Ettől lesz simább és összehangoltabb a kép. Alapja, hogy amikor kirajzolunk egy képet akkor már minden pixelének színe ismert. |
| Game loop | A játékfejlesztés egyik legfontosabb tervezési mintája. Ez oldja meg, hogy a játék folyamatosan fusson és közben megfelelően kezelje az eseményeket és frissítse az állapotok valamint a képernyőt. Használatához hasznos az idő bevezetése. |
| Update method | A játék objektumok megfelelő frissítéséért felelős minta. Framenként frissít a játék összes frissíthető objektumának az állapotát. A Game loop minta használatával működik jól. |
| Dirty flag | Optimalizációs minta. Akkor van jelentősége, hogy ha adat alá-fölé rendeltség van. Mint minden optimalizációs mintánál itt is nagyon oda kell figyelni mikor és hol szabad használni. |
| Object pool | Optimalizációs minta. Akkor van jelentősége, ha valamely objektumokat úgy használjuk, hogy sokszor létrehozzuk és megsemmisítjük őket. Fontos, hogy milyen objektumokra használjun és nem szabad túlhasználni mert akkor többet árt mint segít a memóriahasználatban. |
| Spatial partition | Optimalizációs minta. Akkor hasznos, ha meg akarjuk keresni a hozzánk legközelebbi objektumot. Alapja, hogy az objektumokat pozicí alapján tároljuk. |

Ezen kívül információforrás volt számomra a külső könyvtárak dokumnetációi.

# Tervezés

Tervezés során az első lépésként eldöntöttem milyen nyelvet fogok használni, milyen fejlesztői környezetet és milyen platforma fogom elsősorban készíteni a programom.

Mivel OpenGL alapú játékmotor fejlesztése a feladat eléggé kézenfekvő volt a c++ nyelv választása. Ez azért is előnyös mert ezen a nyelven íródott a legtöbb OpenGL-hez kapcsolódó könyvtár. Emellet úgy terveztem igyekezni fogok a c++ minél újabb szabványát használni, ezért is döntöttem úgy, hogy a Microsoft Visual Studio 2019 c++17-es fordítójával fogom készíteni a projektet. Ezen kívül a verziókövetéshez githubot fogok használni. Illetve szeretnék precompiled header-t használni, a sok külső könyvtár miatt.

Az előbbiekből adódóan úgy döntöttem, hogy fejlesztői környezetnek a Microsoft Visual Studio 2019-et fogom használni. Emiatt az elsődleges fejlesztési platformom a Windows. Emellet úgy terveztem, hogy a tervezési folyamat folyamán nagy figyelmet fogok szentelni annak, hogy könnyen át lehessen vinni más platformra és más grafikus könyvtárra a játékmotort, még ha én nem is implementálom másra. Emiatt is igyekeztem úgy felépíteni a projektet, hogy a natív részeket mindig interface-en keresztül használjam, ami mögött úgy le lehet cserélni a kódot, hogy az egyéb részek nem igényelnek változtatást.

Ezek után szétszedtem több részre a feladatomat. Az egyik, az maga a játékmotor, amit fel lehet használni tetszőleges játék fejlesztéséhez. Egy másik rész maga a játék, amit felhasználja és személyre szabja a motort, a specifikus játéknak megfelelően. Itt olyanokra kell gondolni mint a játéklogika, grafikus elemek kinézete, vagy az inputok személyre szabása. Az opcionális harmadik rész arra való, hogy egy grafikus interface-en keresztül tudjuk beállítani a motort. Ezzel a résszel azonban az elején nem fogok foglalkozni, csupán akkor akkor amikor a többi már elkészült.

Ezek ismeretében úgy terveztem, hogy a játékmotorért felelős rész fordítása után generál egy könyvtárat, amit majd maga a játék tud használni.

Mivel a játék motor elsősorban labirintusos játékokhoz készül ezért szeretném beépíteni, hogy fileból tudjon labirintust beolvasni. Ehhez kitaláltam egy egyszerű szöveges file formátumot, ami elsősorban 2D labirintusok leírásához alkalmazható. A labirintus leíró file-okat a .lab kiterjesztéssel jelölöm meg. A leírásban meg van adva, hogy mekkor a labirintus, ez egy téglalap alakú terület, ami kis négyzetekből áll és mindegyik lehet út vagy fal. A leírásban szóköz jelöli az utat és # a falakat. Valamint a labirintus bejáratát egy I jelöli, a kijáratát egy O. A szélén körben fal található, a kijáratot és bejáratot kivéve.

Itt látható egy példa:

| .lab leíró file | Ahogy kinéz |
| --- | --- |
|  |  |

Ezek után eldöntöttem, hogy milyen külső könyvtárakat fogok használni. Itt elsősorban a konzulensem által javasoltakból indultam ki.

**Bullet Physics SDK[[2]](#footnote-1):** A fizikai motorhoz felhasznált c++ könyvtár. A funkciói közül számomra fontos, hogy képes valós idejű ütközésdetektálásra és fizikai szimulálásra. Használatával nem nekem kell a fizikai motort megvalósítani.

**FreeImage[[3]](#footnote-2):** A legelterjedtebb képformátumok betöltéséhez használható cross platform c++ könyvtár.

**Open Asset Import Library[[4]](#footnote-3):** A könyvtár képes modelleket és textúrákat fileból betölteni, amiket utána tud használni a motor.

**OpenAL[[5]](#footnote-4):** A könyvtár képes a hangok kezelésére és élethű használatára 3 dimenziós környezetben.

Az OpenGL egy szabvány ami lehetőséget ad a grafikus kártyák programozására a hardvertől függetlenül. A szabvány által definiált függvényeket a támogató grafikus kártyákra a gyártónak implementálni kell. Ahhoz, hogy hatékonyan és egyszerűen tudjuk használni, nem ártanak egyes kiegészítő könyvtárak.

**GLFW:** Elvégzi az ablakok létrehozását és kezelését helyettünk. Továbbá képes kezelni az egér, billentyűzet vagy más forrás által generált inputokat.

**GLEW:** Egy cross platform könyvtár, ami segít betölteni az OpenGL extension-öket, ezzel megkönnyíti a munkánkat.

**Glad:** Egy összetettebb könyvtár, ami ellátja ugyanazokat a feladatokat mint a GLEW. Csak egyikükre lesz szükség.

**ImGui[[6]](#footnote-5):** Egy nyílt forráskódú c++ könyvtár ami képes hatékonyan, optimalizáltan renderelni. Vannak beépített widget-jei, amik kész elemek és megkönnyítik a fejlesztést. Használatához szükséges az utóbbi három könyvtár, amik megmondják mit kell renderelni és kezelik az inputokat.

Ezeken szerettem volna, ha a programnak van egy jól használható és testreszabható log könyvtára, ami megkönnyíti a hibakeresést.

**spdlog[[7]](#footnote-6):** Egy gyors multiplatform c++ könyvtár, ami log rendszer kialakításához nyújt támogatást. Segítségével tetszőleges formátumú és színű log opciókat lehet létrehozni, amik nagyban hozzájárulnak a projekt átláthatóságához.

A felhasználandó külső könyvtárak kiválasztása után felvázolom a fejlesztés kezdetét jelentő lépéseket. Elsőként létrehozom a projekteket és kialakítom köztük a kapcsolatot. Ezt követően importálom a használandó külső könyvtárakat, a grafikus megjelenítéshez szükségesekkel kezdve. A beimportált könyvtárakat összekötöm egymással és bekötöm a projektbe. A projekt ezek után képes renderelni a megadott képet és kezelni az inputokat. Ezek után következik az egyéb funkciók implementálása.

Minden lépés után igyekszek teszteket készíteni, amivel le tudom ellenőrizni, hogy működnek-e az egyes részek. Ezek a tesztek fognak a későbbiekben támpontot nyújtani abban, hogy tudom ellenőrizni az új funkciók implementálása befolyásolta-e a már meglévők működését. A fejlesztés során git-et fogok használni ami használható, hogy bármi hiba esetén visszatérjek a legutóbbi stabil állapothoz.

Ezek után megkezdődhet az implementálás a tervek alapján.

# 

# Implementálás

Az implementálást egy Visual Studio solution létrehozásávak kezdtem. Ebben készítettem két projektet. Az egyik a játékmotor, a másik maga a játék. A játék projekt használj a motor projektet.

A játék projekt leszármazás segítségével és a szükséges függvények implementálásával létrehozza magát a játékot. Viszont a belépési pont és a fő ciklus is a motorban található. Lehetőség van a játékban személyre szabni a bemenetek és a hozzájuk kapcsolódó eseményeket. A játék úgy lett beállítva, hogy fordítás után egy futtatható állományt generáljon

A motor projekt definiálja azt az osztály, amiből való leszármaztatása lehet játékként példányosítani. Ez az osztály tartalmazza a játék fő ciklusát is, ami a Game loop és Update method tervezési minták felhasználásával működik. A motor úgy lett beállítva, hogy fordítás után egy könyvtárat generáljon (statikus vagy dinamikus), amit azután a játék fel tud használni.

Következő lépésként importáltam és testre szabtam az spdlog külső könyvtárat. Amire azért volt nagy szükség, hogy egyszerűen tudjam logolni a történéseket. Ezzel könnyebben tudom lokalizálni az esetleges hibákat és pozitív visszajelzést is ad ha minden a terv szerint működik.

Ezek után importáltam a grafikus megjelenítéshez szükséges könyvtárakat. Ezek a GLFW, GLEW, Glad és ImGui. Ezek közül van amelyik source kódként valamelyik pedig fordított könyvtárként kell importálni. Ez elég sok és macerás include és linker beállítást vont maga után.

A sok külső könyvtár használata miatt úgy döntöttem használok precompiled headert. Ennek lényege, hogy a header-ben include-olt könyvtárakat csupán egyszer fordítja le. Ez nagyban gyorsítani tudja a program fordítását. Ide olyan könyvtárak kerültek be amiket viszonylag gyakran használok, de csak ritkán vagy soha nem kell módosítani. Ilyenek például a harmadik fél által készített könyvtárak amiket használok, vagy a gyakran használt c++ header-ek (string, vector, stb...).

Ez után elkészítettem az ablakot. Az ablakot egy interfaceként hoztam létre, amit megvalósítanak a különböző nativ osztályok. Ezzel el rejtettem a platform specifikus részeket. Az ablak használatához az ImGui külső könyvtárat használtam. Ezzek könnyen tudok ablakokat létrehozni, amihez egyszerűen lehet különböző grafikus elemeket (gomb, input box, stb...) létrehozni. Ezen kívül a neki megadott képet hatékonyan képes renderelni. Viszont az inputok kezeléséhez meg neki azokat külső forrásból. Ehhez mindenképp szükség van egy imputkezelőre.

Az inputok kezelése, az ablakokhoz hasonlóan egy platformfüggő feladat. Ezt az ablokoknák használatos módon egy interface-szel rejtettem el, hogy könnyen lehessen lecserélni a natív input kezelést. Az inputokat ezek után hozzákötöttem az ablakhoz.

Ezután felmerült az a probléma, hogy a program csupán egy input kezelését képes elvégezni egyszerre. Ennek megoldásához elkezdtem áttervezni, módosítani és refaktorálni a projektet. Ekkor azonban nem várt nehézségekbe ütköztem és nem sikerült úgy megoldani ahogy szerettem volna. Emiatt visszatértem a legutóbbi stabil verzióhoz a github repository segítségével.

A visszatérést követően észrevettem, hogy a Visual Studio solution nem jól őrizte meg a megadott beállításokat. Ez azt eredményezte, hogy a projekt szerkezete szétesett és nem működtek már az include és linker beállítások megfelelően. Ezek után két megoldást vázoltam fel a továbbhaladáshoz.

Az egyik, hogy újra beállítom az include és linker beállításokat valamint visszaállítom a projekt felépítését. A másik, hogy használok valami projekt generátor programot, ami megteszi ezt helyettem. A döntéshez összeszedtem azokat az érveket amiket figyelembe vettem.

| **Újra beállítás** | **Generálás** |
| --- | --- |
| Gyorsabb | Lassabb |
| Később is előjöhet ez a probléma | Megoldja a problémát véglegesen |
| Nem oldja meg igazából a problémát | Segíti a multiplatform fejlesztést |
| Egyszerűbb | Bonyolultabb |
|  | Új dolgokat kell használni hozzá |

Az érveket figyelembe véve úgy döntöttem megpróbálom megcsinálni, úgy, hogy generálom a projektet. Ehhez első lépésben el kellett dönteni, milyen programot akarok használni. Itt az utánanézést követően két opcióra szűkítettem le a választás. Az egyik a CMake a másik pedig a Premake.

**CMake[[8]](#footnote-7):** Egyik legelterjedtebb ilyen program. Egy cross platform program ami rengeteg környezetet támogat. Nyílt forráskódú.

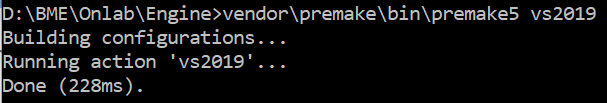
**Premake[[9]](#footnote-8):** Kisméretű, könnyen használható projekt generáló program ami számos platformot támogat.

A két opció közül végül a Premake-re esett a választásom. Ez főleg annak köszönhető, hogy kisméretű és egyszerűbb a használata. Valamint nem lesznek kihasználva a CMake plusz funkciói.

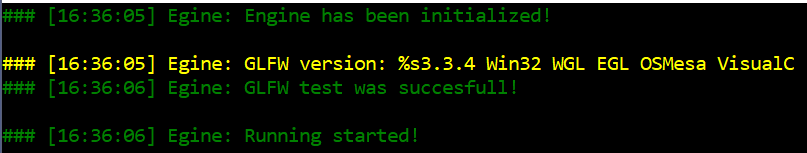
A Premake a generálásához lua scripteket használ. A scriptekben meg kell adni a projekt felépítését, az importálandó könyvtárakat és a precompiled headert. Ezen kívül a fordításokhoz meg kell adni platformot, fordítót és kimeneti könyvtárakat is.

Ez a megoldás abból a szempontból is előnyös, hogy a külső könyvtárakat elég source kódként eltárolni, a premake lefordítja azokat is.

A scriptek megírása után már egy egyszerű parancs kiadását követően legenerálódik a projekt a megfelelő beállításokkal és függőségekkel a kívánt platformra. A platform kiválasztása a parancs kiadásában lehetséges. Visual Studio 2019 esetén például így néz ki a parancs:



A fejlesztés jelen pillanatban abban az állapotban van, hogy a projekt legenerálódik, viszont a külső könyvtárak közül nem importálja mindegyiket hibátlanul. Ennek oka a könyvtár nem megfelelő fordítása, de kijavítására még nem került sor. Most a tesztek közül egy fut le és sikeresen és elindul az engine.



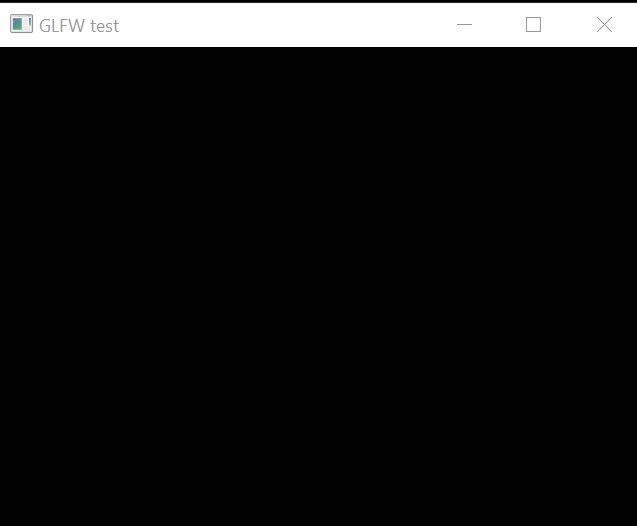
Jelenleg nem csinál semmit a fő ciklus.

# 

# Tesztelés

A tesztelés elengedhetetlen része annak, hogy nyomon tudjuk követni, hogy a programunk a legújabb változtatások után is jól működik-e. Ezeknek az egyik legjobb eszköze a Unit tesztek, viszont grafikus felületeknél ezek nem igazán használhatóak. Ez esetben manuális tesztek voltak használva, elsősorban a grafikus könyvtárak importálásának sikerességének ellenőrzésére.

A GLFW könyvtár tesztelésére egy olyan programrészt használok, ami létrehoz egy egyszerű ablakot. Emellett ellenőrzi, hogy sehol sem keletkezik exception. A jó lefutást követően ez látható:



A GLEW könyvtár tesztje egy színes hátterű ablakot hoz létre amiben kirajzol egy háromszöget. Ez a teszt jelenleg sajnos nem működik, a hiba oka a GLEW könyvtár sikertelen importálása.

Az ImGui tesztje egy ablakot hoz létre, amin megjelenik néhány widgetet check boxokkal meg gombokkal, amiknek hatására kirajzlódik egy háromszög. A működéséhez szüksége van a GLEW könyvtárra (Glad is lehetséges de nem ahhoz készült a teszt), de annak hibás működése miatt sem működik a teszt.

Egyenlőre a tesztek csupán a külső könyvtárak megfelelő integritását ellenőrzik, de a továbbhaladás során ez ki fog egészülni magára a játékmotorra irányított tesztekre is. Ezek közül azok amik a grafikus felület működésével függnek össze manuális tesztek lesznek, de a belső logika ellenőrzésére Unit tesztek lesznek használva. A tesztek ki fognak egészülni példa kódokkal is.

# 

# Felmerült problémák, tapasztalatok

A projekt kezdetén zökkenőmentesen ment a feladat specifikálása és megtervezése. Az implementálás eleje is jól ment, csupán egy probléma merült fel, hogy nem tudott a program több inputot kezelni egyszerre. Ennek kijavítása nem sikerült, de a komoly gondok ott kezdődtek, amikor a github repository segítségével vissza akartam térni egy régebbi verzióhoz. Ekkor kiderült, hogy a letöltött solution file nem megfelelő.

Ez a probléma nagyban hátráltatta a projekt fejlesztését, viszont szerencsére még egészen az elején kiderült, így nem okozott olyan nagy károkat. Egy új nézőpont bevezetését igényelte, ami alapján generálva lett a projekt script file-ok alapján.

Igaz, hogy a félév végére kitűzött célt nem sikerült emiatt elérni, de pozitív változások lettek eszközölve a projekten, aminek köszönhetően egyszerűbb lesz a fejlesztés a jövőben.

Megtapasztaltam, hogy mennyire megéri inkább több időt tölteni a tervezéssel és csak később nekikezdeni az implementálásnak. Az alapos tervezés nem csak könnyebbé teszi a fejlesztést, de lehetőbbé teszi a hibák könnyebb megoldását is.

A másik fontos tapasztalat amit szereztem az az, hogy mennyire fontos a külső forrásból származó könyvtárak és programok ismerete, kiváltképp ha azok adják a projekt alapjait. Alapos ismeretekkel sokkal könnyebb használni őket és kijavítani a bennük jelentkező hibákat.

Összességében elégedett vagyok a tanultakkal és olyan tapasztalatokat szereztem amik hosszú távon segítik a munkámat. Ezenk kívül a felmerülő problémáknak is örülök, mert így kisebb eséllyel fogom elkövetni őket újra, olyankor amikor már nehezebb lenne kijavítani őket.

# 

# Összefoglalás

Sikerült, szerintem alaposan és minden eshetőségre felkészülten megtervezni a projektet. A tervezés során figyelembe vettem minden olyan tényezőt amit fontosnak gondoltam. Utólag kiderült, hogy van amit nem sikerült figyelembe vennem és ez eléggé nagy mértékben befolyásolta a munka menetét.

Sikerült sokat tanulnom a félév során. Olyan dologgal foglalkozhattam ami érdekel. Ez is az oka annak, hogy szerettem volna minél jobban átlátni és tisztába tenni a dolgokat mielőtt valójában is belekezdek. Emiatt több időt töltöttem a olvasással és tervezéssel mint amennyit magával a kódolással, de úgy érzem ez növelni fogja a végső, elkészült eredményt.

Összességében nézve elégedett vagyok a munkámmal, még ha nem is sikerült eljutnom addig ameddig szerettem volna, fontos dolgot tanultam, ami pozitív hatással lesz a jövőbeli projektjeimre. Úgy érzem, hogy jól megalapoztam a szakdolgozatomat a féléves munkámmal, mégha nem is tudok látványos eredményeket bemutatni.

# Források

* Tervezési minták: <https://gameprogrammingpatterns.com/contents.html>
* Fizikai motor: <https://github.com/bulletphysics/bullet3>
* FreeImage: <https://freeimage.sourceforge.io/>
* Open Asset Import Library: <https://www.assimp.org/>
* OpenAL: <https://openal.org/>
* ImGui: <https://github.com/ocornut/imgui>
* spdlog: <https://github.com/gabime/spdlog>
* Premake: <https://premake.github.io/>
* CMake: <https://cmake.org/>
* <https://learnopengl.com/>
* <https://shahriyarshahrabi.medium.com/write-your-own-game-engine-df1132908cdf>
* <https://www.scratchapixel.com/>
* https://graphicscodex.com/

# 

# Tartalom

[Feladat leírása](#_e4guwmg765m9)

[Fejlesztési folyamat](#_6o42jakg7o0b)

[Követelmények](#_fl2yhgezpjk8)

[Információszerzés a témában](#_x5k9bwwebvlm)

[Tervezés](#_pp79i0qum71i)

[Implementálás](#_u567anlwefw3)

[Tesztelés](#_igdhx6a7doa2)

[Felmerült problémák, tapasztalatok](#_v2ykobggnuew)

[Összefoglalás](#_3dt67xhn3mel)

[Források](#_1m1jlujd8tkg)

[Tartalom](#_swuc4zfr6ai9)

1. <https://gameprogrammingpatterns.com/contents.html> [↑](#footnote-ref-0)
2. <https://github.com/bulletphysics/bullet3> [↑](#footnote-ref-1)
3. <https://freeimage.sourceforge.io/> [↑](#footnote-ref-2)
4. <https://www.assimp.org/> [↑](#footnote-ref-3)
5. <https://openal.org/> [↑](#footnote-ref-4)
6. <https://github.com/ocornut/imgui> [↑](#footnote-ref-5)
7. <https://github.com/gabime/spdlog> [↑](#footnote-ref-6)
8. <https://cmake.org/> [↑](#footnote-ref-7)
9. <https://premake.github.io/> [↑](#footnote-ref-8)