2D Top-down Shooter

Készítette:
Hegyi Botond - Sámuel
Fekete János Szilárd
Kovács Katalin

Tartalomjegyzék

Előszó

Az 1980-as évek óta a videojátékok a szórakoztatóipar egyre fontosabb részévé váltak, és akár tekinthetőek a művészet egyik formájának is. A videójátékokra használt elektronikai eszközök a „platform” megnevezéseként ismertek, amelynek egyik csoportja a személyi számítógépek, a másik a videojáték-konzolok.
A videójátékok is különböző műfajokba sorolhatóak, számos különböző jellemvonás alapján, mint a játékmenet, a célok fajtái stb.
Műfajok: akció, ügyességi, kaland, szerepjátékok, stratégiai játékok
A videójátékoknak vannak hasznos hatásai, többek között amelyek a taktikai készséget, reflexet, térlátását, ujjaink koordinációját, memóriát és problémamegoldó és együttműködési készséget, vagy a társalgást fejlesztik. Hátránya, hogy függőség is kialakulhat vagy olyan erőszakos magatartás mintákat és reflexeket rögzíthetnek, amelyek a társadalmi életben nem kívánatosak, ezért a gyermekek számítógépes játékát ajánlatos egy felnőttnek felügyelnie.

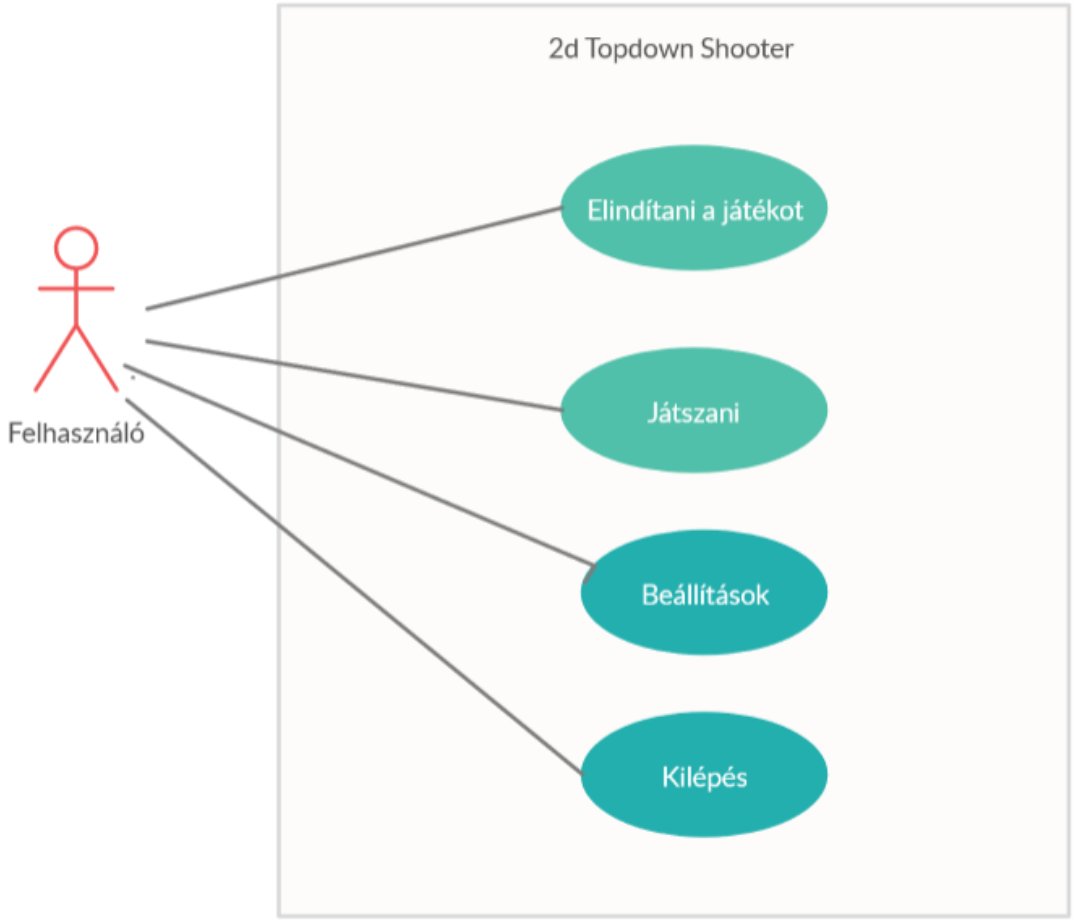
Bevezető

Projektünk célja az volt, hogy megismerkedjünk a játékfejlesztés alapjaival és elsajátítsuk azokat. Fejlesztői környezetnek az OpenGL és C++ párosát választottuk a régi múlt, és a rengeteg dokumentációnak köszönhetően. Játék témájának egy felülnézetes Shooter-t választottunk mert 2D-ben kell dolgozni ami sokkal leegyszerűsíti a dolgokat.

Funkcionalitás követelmények

1. Legyen egy billentyűvel illetve egérrel mozgatható karakter
2. A karakter tudjon lőni
3. Legyenek ellenségek, amelyek golyó lövések által meghalnak
4. Az ellenségek kövessék a karaktert és támadják meg
5. A játékkarakterünknek legyen egy életereje

Felhasználói követelmények definíciói

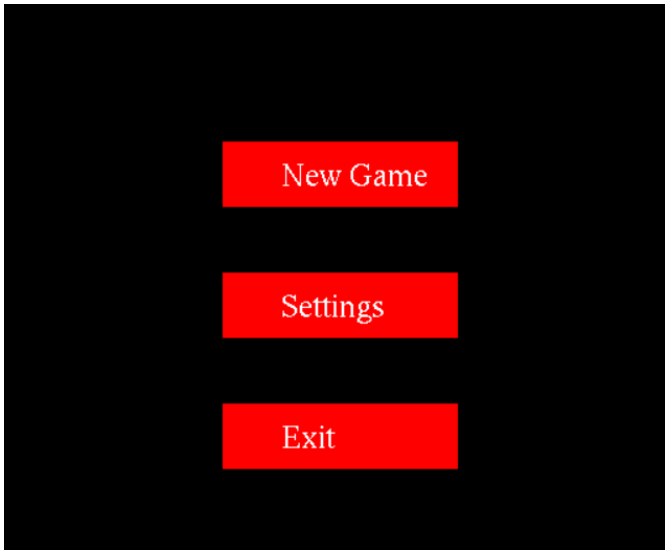


Ábra 0.1: Use Case diagram

A játéknak csak egy szerepköre van ez a *0.1 Ábrán* látható Felhasználó. A felhasználó következő funkcionalitásokat végezheti:

1. Elindítja a játékot: ezzel egy új játékmenetet kezd el. Kezdetben a játékkarakter életereje maximumon van és a pálya bal felső sarkán található.
2. Játszani. Miután elindult a játék mozgathatja a karakter helyzetét illetve irányát.
3. Beállítások esetén a játékra érvényes beállításokat tudja módosítani.
4. Kilépés esetén megáll a játékmenet, illetve a teljes alkalmazás bezárul.

Vizuális felület



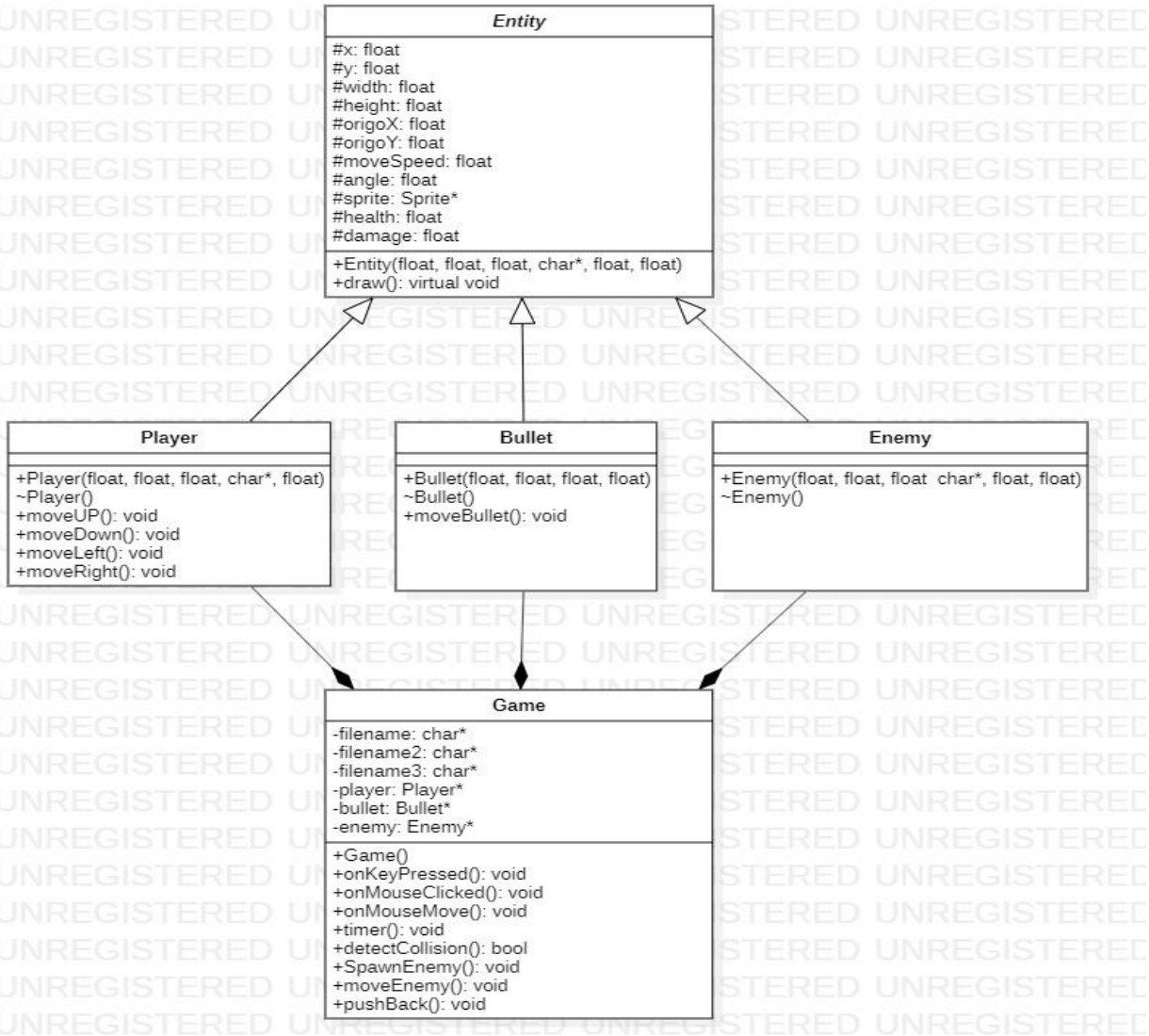
A menü három gombot tartalmaz:

New Game: ez megfelel a Játekot elindító funkcionalitásnak.
Settings: ez a Beállitások funkcionalitás.
Exit Game: Kilépés funkcionalitás.

A “New Game” menüpont kiválasztása után megjelenik a játékfelület. Itt láthatjuk a mozgatható karakterünket, illetve az ellenfelet.

A játék felépítése

A játékmenet



Ábra 1.0: Class Diagram

A fenti ábrán, azaz a class diagramon magának a játéknak a lelke látható. A fő cél az volt, hogy minél inkább lebontsuk a feladatokat, minél kevesebb felelősséget adjunk egy objectnek, ezért is hoztuk létre az Entity classt, amiből származtattuk a Playert, Bulletet és az Enemyt. A hatalmas előnye ennek az, hogy sokkal átláthatobb a kód, és egyszerűbb debuggolni. A legelső verzióban csak egyetlen egy object-ünk volt, de amint egyre több feladatot kellett elvégeznie szükségessé vált ezen osztályok létrehozása.
A Game osztályban történnek a fontosabb metódusok a játékra vonatkozóan, és lényegében ott hozzuk létre az irányítandó entityt, és a legyőzendő enemyket.

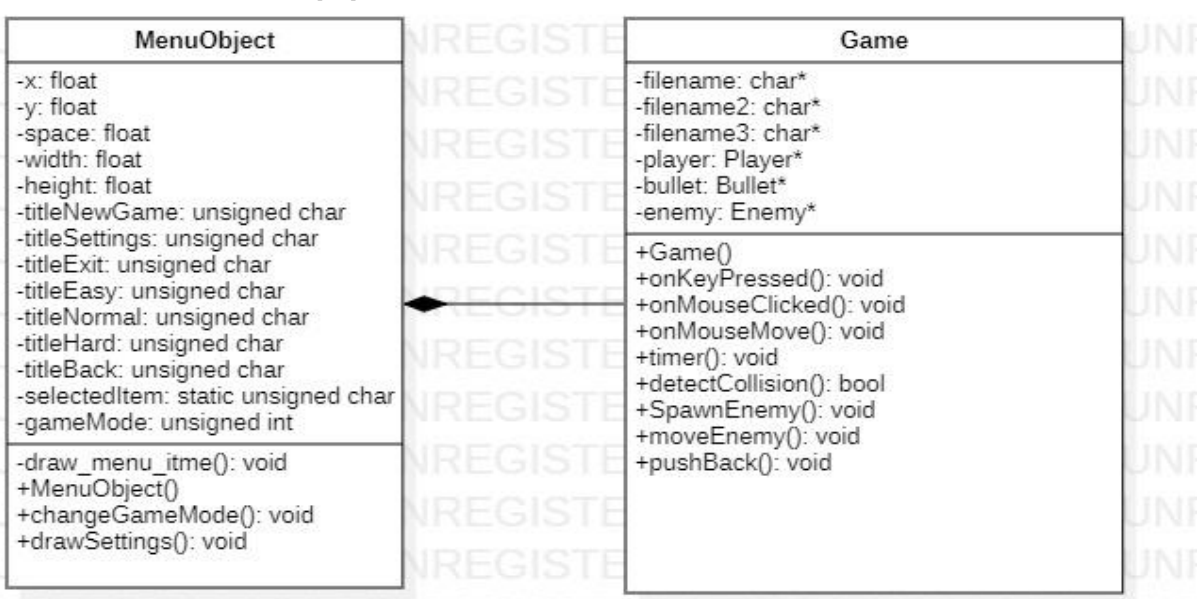
Példák a Game-ben lévő metódusokra

1) Az alábbi pszeudokód a Player entity-nek az irányítását mutatja be. Ezen megoldással csak az adott 4 irányba tudjuk mozgatni a játékost.

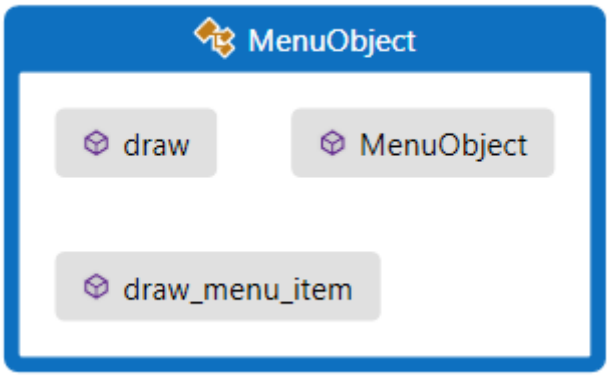
```
Void onKeyPressed
    switch( which key has been pressed)
        Case ‘w’:
            Player move up
            break
        Case ‘s’
            Player move down
            break
        Case ‘a’
            Player  move left
            break
        Case ‘d’
            Player move right
            break
```

2) Az alábbi pszeudokód bizonyos időközönként meghívja önmagát, ez köszönhető a freeglut-ba beépített függvénynek. Ezen felül a kilőtt golyó és az enemy mozgásáért felelős, egyúttal az ütközést nézzük az entitások között, ez konkrétan a damage system alapját jelenti.

```
Void timer
    If  bullet  exists
        Move bullet
        If the bullet is out of the windows screen
            Delete the bullet
    If the enemy is alive
        Move enemy
    If enemy had collision with bullet
        Reduce the enemy’s life
        Push back the enemy
        Delete the bullet
        If enemy’s life is 0
            Delete enemy
    If player had collision with enemy
        Reduce the player’s life
        Push back the player
        If Player’s life is 0
            Exit Game
    Redisplay the game window
```



Az egyes Menü Elemek ábrázolása a MenuObject osztályban van megvalósítva.



A draw publikus függvény segitsegevel kirajzoljuk az összes menüpontot. A draw_menu_item kirajzol egy darab menüpontot.

```
//kirajzolja az osszes menupontot
void MenuObject::draw() {
    ……
    glEnable(GL_STENCIL_TEST);
    glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);

    glStencilFunc(GL_ALWAYS, 1, -1);
    draw_menu_item(tmpBuffer, offset); //kirajzol egy darab menu pontot
     ……
}
```

A glEnable(GL_STENCIL_TEST); illetve glStencilOp függvényhívások segítségével azonosítható es ezáltal kiválaszthatóvá tudjuk tenni az egyes menüpontokat. A glStencilFunc függvény felhívása után kirajzolt elemek bele kerülnek egy úgynevezett Stencil Bufferbe és ezeket az elemeket a későbbiekben a koordináták alapján be lehet azonosítani. A függvény második paraméterben egy egyedi azonosítót lehet adni az elemeknek.
Hogyha rákattintunk egy Menü elemre annak beazonosítása végett megpróbáljuk elérni a stencil Buffert. Ezt a következő függvényel tesszük:

```
glReadPixels(x, window_height - y - 1, 1, 1, GL_STENCIL_INDEX, GL_UNSIGNED_INT,
&index);
```

A glReadPixels függvény utolsó paramétere az index, amelyet kiolvasunk a Stencil Bufferből, hogy ha az index egyenlő a korábban beállított értékkel azt jelenti, hogy az adott indexszel ellátott menüpont lett kijelölve.

Projekt menedzsment

Verziókövető rendszernek a GitHubot választottuk az egyszerű használat miatt. Nagyon hasznosnak bizonyult, mindenki külön branchen dolgozott a saját feladatán, megelőzve azt hogy a már működő alkalmazást egy commit elrontsa. Tartottunk kód reviewkat egymásnak, és beállítottuk, hogy review nélkül ne lehessen bemergelni a *master* branchre. Emellett még használtuk GitHubon a projekt tabot is, ahol három kisebb projektet hoztunk létre az átláthatóság érdekében, és a projekten belül hoztuk létre a kisebb feladatokat és osztottuk ki egymás között az alapján, hogy ki mihez ért vagy mivel szeretne foglalkozni. A projektek befejezése után mindig próbáltuk refaktorolni a kódot, hogy könnyebben olvasható legyen és könnyebb legyen a későbbi munka vele.

Összegzés

Jövőbeli tervek

A játék jelenlegi formájában már játszható, azonban még jó pár funkció hiányzik belőle amire már nem volt időnk.
Néhány ötlet amit meg lehetne valósítani vagy tovább lehetne fejleszteni:

- Settings menü -> több nehézségi szinttel
- Jobb képmegjelenítő algoritmus
- A játékos életének megjelenítése játék közben
- Game over screen
- Több enemy megjelenítése
- Pálya design

Következtetés

Összességében meg vagyunk elégedve a munkánkkal, hogy sikerült véghez vigyük azt, amit az elején elterveztünk. Persze még rengeteg munka van vele, és ha időnk engedi szeretnénk majd a későbbiekben is foglalkozni a projekttel, a meglévő állapotán javítani és további funkciókat megvalósítani.
A csapatmunkára nem lehet panasz. Szerintem bátran mondhatjuk, hogy rengeteget tanultunk a játék fejlesztése során amit akár a későbbiekben is alkalmazhatunk.

Bibliográfia

OpenGL Game Development By Example - Robert Madsen, Stephen Madsen

STACKOVERFLOW