**PRODIUS PROJEKT**

****

**Készítette:**

**Portik-Gyurka Botond**

**Vass Csaba**

**Vass lehel**

**TARTALOM JEGYZÉK**

Tartalom

[BEVEZETŐ 3](#_Toc501485065)

[ISMERTETŐ 4](#_Toc501485066)

[ELVÁRÁSOK 4](#_Toc501485067)

[FELHASZNÁLÓI KÖVETELMÉNYEK 5](#_Toc501485068)

[TERVEZÉSI FOLYAMAT 5](#_Toc501485069)

[A TERV IMPLEMENTÁLÁSA 8](#_Toc501485070)

[\_GameObject osztály 8](#_Toc501485071)

[\_Sprite osztály 9](#_Toc501485072)

[Colider osztály 9](#_Toc501485073)

[SpawnFactory osztály 10](#_Toc501485074)

[Score és HighScore osztályok 10](#_Toc501485075)

[Könyvészet 11](#_Toc501485076)

# BEVEZETŐ

**Már a saját számítógépek megjelenésével együtt járt a számítógépes játékok fejlesztése. Eleinte egyszerű volt a grafikájuk és a logikájuk is. Ahogyan fejlődött a technológia úgy egyre komplikáltabb ötleteket valósíthattak meg a játékokban is.**

**A két dimenziós játékok a 90-es évek közepén érték el az arany korukat a Super Nintendó TV játékokon keresztül. E közben a számítógépes játékok stílusa nagyon eltért a TV játékok stílusától. A számítógépes játékok nagy hangsúlyt fektettek a logikai puzzle-ekre és a történet alapu játékokra. Míg a TV játékok inkább akciódúsabbak voltak és a játék érzés tökéletesítésére törekedtek. Itt jelent meg az egyik leglegendásabb játék, amely a későbbiekben nagyban befolyásolta a két dimenziós játékok struktúráját és designe-ját. Ez volt a Super Metroid. A Számítógépes játékokat ebben az időben a kaland játékok uralták, mint például a Monkey Island, Day of the Tentacle a Lucas Arts-tol.**

**A 90-es évek végén a játékok nagy átalakuláson ment keresztül. Ekkor jelentek meg a három dimenziós játékok. Evel párhuzamosan az akció dús játékok áttértek a számítógépekre is. Ez a változás nagyrészt egy játéknak köszönhető, a Doom-nak. Ez a játék vezette be a 3d-s lövöldözős játékok fogalmát melynek designe stílusa még a mai napig is elterjedt. A 2010 es évek elején újból megjelentek a két dimeziós játékok mivel, hogy a nagy cégek csak 3d-s játékokkal foglalkoztak, kisebb játék tervező cégek 2d játékokat fejlesztettek, mert csak arra volt pénzügyileg lehetőség. Mi is úgy döntöttünk, azaz Csaba úgy döntött, mi csak, mint a szolgák követtük őt, hogy két dimenziós játékot alkotunk, mert könyebb megvalósítani egy 2d engine-t mint egy 3d-st.**

**A játékunk stílusa egy régi Nintendó játékból inspirálódott, a Gradius-ból. Az ehez hasonló játékok elterjedtek voltak a 80-as években az arcade-okban.**

# 

# ISMERTETŐ

**A mi játékunk a “régi” akció játékok stílusát képviseli ezért nincs hangsúly a történeten, hanem inkább a játék mechanikai részeire összpontosít. A játékos egy űrhajót irányíthat, amely ahogy az űrben halad különféle ellenségeket kell, elpusztítson a pont szerzés érdekében. Minden játékos 3 élettel kezd és minden 100 elért pont után kap egy újabb bónusz életet. Amennyiben a játékos egy ellenséggel ütközik, akkor veszít egyet az életéből. Az ellenségek nehézségi szintje különbözik. Egyes ellenségeket nehezebb elpusztítani, mint másokat.**

** ábra 1. menü ábra 2. játék**

# ELVÁRÁSOK

**- egyszerű, retro 2d grafika**

**- szimpla irányítási módszer: wasd, nyilak és space**

**- repülő ellenségek véletlenszerű megjelenési pozícióval**

**- több fajta ellenség, különböző nehézségi szinttel**

**- a játékosnak legyen élete**

**- a játékos tudjon korlátozott számú lövedéket kibocsátani**

**- eredmény nyilvántartás és mentés**

**- egyszerű menü**

**- űrszerű stílus**

# FELHASZNÁLÓI KÖVETELMÉNYEK

**- java runtime envieronment**

# TERVEZÉSI FOLYAMAT

**A játék megtervezése során több féle problémával ütköztünk. Első sorban meg kellett oldjuk az:**

* **objektumok kirajzolását a képernyőre**
* **objektumok mozgatását a térben**
* **ütközések érzékelését.**

**Ez a három ötlet volt a magja a projektnek. A Processing könyvtár vektoriális és grafikus alapja jó kiindulási pontnak bizonyosult ezeknek az ötleteknek a megvalósítására. A Processing vektoriális struktúrája intuitívnak bizonyult és a mechanikai háttér ismereteink sokat segítettek a mozgások megtervezésében. Az ütközések kezelését a vektorok egyszerű összeadásával és kivonásával valósítottuk meg.**

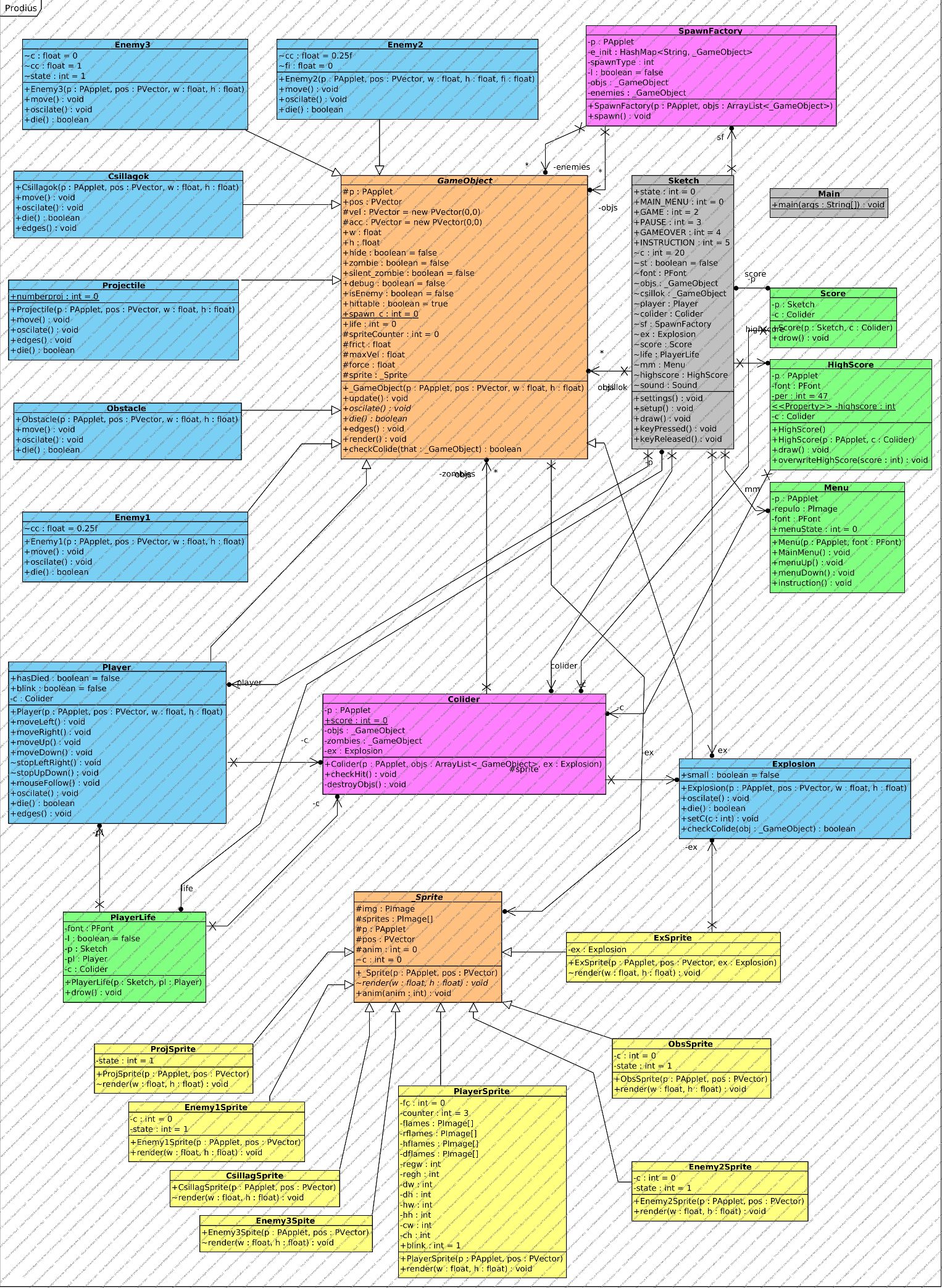
**A grafika megvalósításához egy jól bevált módszerhez folyamodtunk, a sprite-okhoz. Az ötlet a következő: Az objektumok animációja lebontható képek gyors sorozatban való frissítésére. ez hasonló a filmek működéséhez, mivel hogy az emberi szem nem tud értelmezni gyors változásokat csak maximum 24-et másodpercenként, ezért ahogy a képek gyorsan frissülnek, az agyunk gyorsan összemossa őket és folyékony mozgásként értelmezi. A sprite-oknál a különféle mozdulatokat egy képfájlban tároljuk egymás mellett, hogy később könnyen feldolgozható legyen és ez által kevesebb tároló egységet is foglal, mintha mindegyiket egy külön fájlban tárolnánk.**

**Az objektumok térbeni elmozdításához a létező vektor teret koordináta rendszerként kezeltük és newton jól bevált mechanikai törvényeihez folyamodtunk. Newton rájött, hogy a sebesség értelmezhető a helyzet koordináta időbeli változása ként, deriváltja ként. Ugyan így a gyorsulás értelmezhető a sebesség időbeni deriváltja ként. Ezekkel az eszközökkel könnyű volt implementálni a mozgást a játékban. Minden Objektum csak akkor mozdul el, ha egy gyorsulási vektor hat rá és ez a viselkedés mindegyik objektum része.**

**Az ütközést elég egyszerű volt megvalósítani, egyszerűen azt kell ellenőrizni, hogy ha egy objektum be hatol egy másik objektum intim szférájába. Ezért minden objektum rendelkezik saját kerülettel.**

**Ahogy kidolgoztuk ezeket a megoldásokat, osztály diagramot állítottunk össze, a főbb osztályok átlátása érdekében(3. ábra).**

**A \_GameObject absztrakt osztály leírja minden egyes fizikai objektum viselkedését a világban. Az összes több objektum ebből származik. Miden egyes objektumhoz csatlakozik egy sprite osztály, amely a vizuális reprezentációjáért felel. Ezt a viselkedés az \_Sprite osztály írja le. A Colider osztály ellenőrzi az objektumok helyzetét minden ciklusban és kezeli az esetleges ütközéseket. a SpawnFactoru osztály hozza létre az ellenségeket véletlenszerűen, korlátozva a létszámukat. A Sketch osztály az egész program szíve mivel hogy a Prcessing könyvtár PApplet osztályának implementálása. Itt inicializálódik az összes példány és itt van jelen a játék ciklus. A HighScore és a Score auxiliáris osztályok nyilvántartják a játékos által elért eredményt és elmentik a legjobb eredményt.**

**3.ábra: Osztály diagram**

# A TERV IMPLEMENTÁLÁSA

**A processing könyvtár PApplet absztrakt osztályát kellett implementáljuk , ami két fő függvényt tartalmaz: - setup()**

**- draw()**

**A setup metódus csak egyszer hívódik meg a program létrejövésekor.**

**A draw pedig egy ciklus alapján hívódik meg aminek a sebességét a frameRate mező határozza meg. A drawban mindig csak objektumok változásait érdemes írni, mert ha a drawban hozunk létre objektumokat, az sok erőforrást kíván.**

**A mi objektumainkat mind a setup-ban inicializáltuk és a draw-ban módosítottuk őket.**

**A PApplet absztrakt osztálynak még használtuk két metódusát a keyPressed és keyReleased. Ezek a metódusok a billentyűzet gombjainak kódjait kezelik. Itt kezeljük a felhasználótól érkező inputokat.**

**A draw-ban elsősorban egy nagy switch állítja a játék állapotait, amiből van 5:**

**-** **MAIN\_MENU – fő menü**

**-** **GAME – maga a játék**

**-** **PAUSE – a játék szüneteltetése**

**-** **GAMEOVER – a játék vége**

**-** **INSTRUCTION – útmutató**

**A főmenü kinézetét és navigálhatóságát a Menu osztály kezeli. Itt is egy switch váltja a menüpontok állapotát.**

## \_GameObject osztály

**Ez az absztrakt osztály a legnagyobb osztály a projektünkben leírja minden játék objektum általános tulajdonságait. A főbb mezői a következők:**

**- PVector pos – vektor változó az objektum helyzetére.**

**- PVector vel – vektor változó az objektum sebességére.**

**- PVector acc – vektor változó az objektum gyorsulására.**

**- float h, float w – az objektum szélessége és magassága.**

**-** **\_Sprite sprite – referencia az objerktum Sprite osztályára.**

**- boolean zombie – egy ellenőrzés hogy halott-e az objektum.**

**- float frict – súrlódási tényező ami hat az objektumra.**

**-** **float maxVel - maximális sebesség**

**-** **float force – a gyorsulási erő**

**A főbb metódusai:**

**- update() – ebben vannak a transzlációs algoritmusok amik módosítják az objektum pozícióját minden ciklusban.**

**-render() – itt van meghívva az objektum Sprite osztályának a render metódusa. Ez felel az objektumok kirajzolásáért a képernyőre.**

**-** **edges() – ez korlátozza az objektumok mozgását, hogy ha elhagyják a képernyőt, mi történjen. Ez egy absztrakt metodus amit minden objektum saját maga kell implementáljon.**

**-checkColide – ezt a Colider osztály hívja, hogy ellenőrizze az ütközéseket.**

## \_Sprite osztály

**Ez az abstakt osztály határozza meg az objektumok kinézetét.**

**A konstruktorában töltjük be a neki megfelelő, úgy nevezett SpriteSheet-et. Ez egy kép fájl ami tartalmazza az összes animáció rámát. A konstruktorban feldolgozzuk ezt a képet és egy tömbben tároljuk a Sprite-okat**

**A \_Sprite osztály render metódusában végig megyünk a Sprite tömbön a framerate-nek függvényében és kirajzoljuk a Sprite-okat a képernyőre. Ez egy animáció illúzióját kelti, de lényegében csak a Sprite-ok frissülnek majdnem minden ciklusban.**

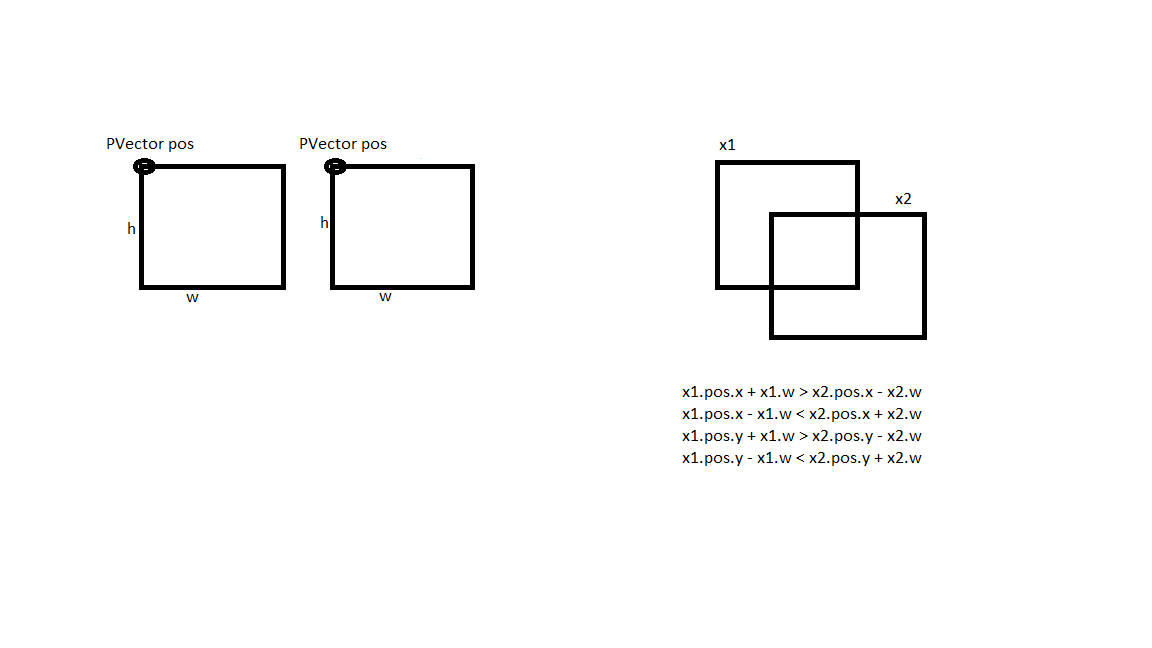
**Minden játék objektum saját osztályokkal rendelkezik, amelyek a \_GameObject és \_Sprite osztályokból származnak.**

## Colider osztály

**Ennek az osztálynak az egyik mezője egy lista ami tartalmazza a referenciákat az összes létező objektumra.**

**A checkHit metódus dupla foreach ciklusban végigmegy ezen a listán és meghívja az objektumok checkColide metódusát ami ellenőrzi, hogy az objektumok ütköznek-e.**

**Az ütközés ellenőrzést úgy végezzük, hogy megvizsgáljuk azt az esetet ha két objektum egymás sugarába kerül (4. ábra).**

****

**ábra 4. ütközés modell**

**Ha érzékeli az ütközést akkor az objektumok Zombie mezőjét igazra állítja.**

**A destroyObjs metodus végig megy megint az objektumok listáján és ha egy objektumnak az zombie mezője igaz akkor elrejti az objektumot és kirajzol egy robbanást azon a pozíción.**

## SpawnFactory osztály

**Ez az osztály tartalmaz egy referencia listát az összes ellenség objektumáról, a konstruktorában hozza létre az ellenség példányokat. Ellenség formátumokba rendezi őket.**

**A spawn metodus egy nagy switchet tartalmaz amely véletlenszerűen válogat a az ellenség formátumok közül és ha kevesebb mint 20 ellenség létezik a képernyőn akkor megjelenít egy új formátumot a kiválasztottak közül.**

## Score és HighScore osztályok

**A Score és a Highscore osztályok figyelik a játékos által kibocsátott lövedék és az ellenséges objektumok ütközését. Minden ilyen sikeres ütközés után a játékos +1 pontot kap. Minden sikeresen összegyűjtött 100 pont után a játékos egy újabb élettel gazdagodik.**

**A Highscore osztály fájlba menti és onnan is olvassa az adatokat. A játék során ha a játékos jobb eredményt ér el mint a fájlban található adat. akkor A képernyőre kirajzolódik az mint új High Score és ezt elmenti egy eredmény.txt -nevü állományba.**

# Könyvészet

- Sprite -okat kölcsönöztük a Gradius Gaiden nevü PlayStation 1-es játékból.

<http://gradius.wikia.com/wiki/Gradius_Gaiden>

- A Processing java könyvtárat használtuk alapként.

<https://processing.org/>