Kezdőlap

Témabejelentő: a szakdolgozat bekötve kell, hogy tartalmazza a kitöltött es jóváhagyott (az Informatikai Kar dékánja által aláírt) Szakdolgozat-téma bejelentőt. Ennek a lapnak a helyére kell bekötni.

Tartalomjegyzék

[1. Bevezetés 3](#_Toc469245476)

[2. Felhasználói dokumentáció 5](#_Toc469245477)

[2.1. A program témája 5](#_Toc469245478)

[2.2. Rendszerkövetelméy 5](#_Toc469245479)

[2.2.1. PC-n 5](#_Toc469245480)

[2.2.2. Androidon 5](#_Toc469245481)

[2.3. A program futtatása 5](#_Toc469245482)

[2.4. A program használata 6](#_Toc469245483)

[2.4.1. A játéktér. 6](#_Toc469245484)

[2.4.2. A HUD 6](#_Toc469245485)

[2.4.3. Irányítás 6](#_Toc469245486)

[2.4.4. Irányítás PC-n 7](#_Toc469245487)

[2.4.5. Irányítás Androidon 7](#_Toc469245488)

[2.4.6. A játék vége 7](#_Toc469245489)

[3. Fejlesztői dokumentáció 8](#_Toc469245490)

[3.1. A megvalósítandó feladat 8](#_Toc469245491)

[3.2. A megvalósítás eszközei 8](#_Toc469245492)

[3.2.1. Lehetőségek 8](#_Toc469245493)

[3.2.2. libGDX 9](#_Toc469245494)

[3.2.3. Animáció – Spriter 9](#_Toc469245495)

[3.2.4. Pályaszerkesztés – Tiled 10](#_Toc469245496)

[3.2.5. Audio 10](#_Toc469245497)

[3.2.6. Betűtípus 10](#_Toc469245498)

[3.2.7. Fizika – Box2d 10](#_Toc469245499)

[3.2.8. Projektgenerálás 11](#_Toc469245500)

[3.3. Az erőforrások előkészítése 11](#_Toc469245501)

[3.3.1. Textúrák 11](#_Toc469245502)

[3.3.2. Animáció 12](#_Toc469245503)

[3.3.3. Pályák 12](#_Toc469245504)

[3.4. Megvalósítási terv 12](#_Toc469245505)

[3.4.1. Controller 12](#_Toc469245506)

[4. Irodalomjegyzék 14](#_Toc469245507)

# Bevezetés

A dolgozat a témája a platformfüggetlen (PC és mobil) játékfejlesztés lehetőségeinek megismerése, előnyeinek és hátrányainak bemutatása egy 2D platformjáték elkészítésén keresztül. Bemutatom a játékok készítése során gyakran alkalmazott tervezési mintákat, játék architektúra opciókat, feltérképezem a (2D platform) játékok szokványos elemeit, mint az animáció, pályaszerkesztés, fizika, felhasználói interakció stb.

Mivel a platformfüggetlen programozás a dolgozat fő témaköre, külön kitérek az ebből következő kihívásokra, mint például az irányításra, ami az eszközök különbözősége miatt az egyik legnehezebb témakör a jó felhasználói élmény elérése szempontjából: a játék ugyanolyan könnyen irányítható kell, hogy legyen billentyűzettel, mint érintőképernyős, virtuális gombokkal. Hasonlóan kihívásokkal teli a játékmenet és a grafika is: az okostelefonok kisméretű (bár egyre növekvő) képernyőjén, és nagyobb méretű modern monitorokon (vagy akár televíziókon) egyaránt könnyű navigációt és minőségi grafikát kell tudni biztosítani.

Természetesen a technikai határokkal is foglalkozok: bár az okostelefonok rohamos ütemben fejlődnek, az olcsóbb, de még akár a középkategóriás készülékek is jóval erőforrás szegényebbek a modern PC-knél, így ügyelni kell az erőforrások hatékony csomagolására és betöltésére, vagy például a túlzott Garbage Collector használatra.

A platform játékok történelme az 1980-as években kezdődik, ebbe a műfajba tartoznak például a korábbi Mario játékok, a Contra, és a Castlevania sorozat, modern képviselői pedig például a Super Meat Boy vagy a Trine sorozat. A hagyományos 2D megoldások óta a játékok nagy fejlődésen mentek keresztül: ma már 3D vagy 2.5D megoldásokkal is találkozhatunk, és a játékmenet is sokféle: míg például Super Meatboy játékmenete relatív egyszerű (bár nagyszerű), a Salt and Sanctuary egy hatalmas képességfával és sokféle felszereléssel rendelkező oldalnézetű szerepjáték.

A mobil eszközök (okostelefonok, táblagépek) népszerűségének emelkedésével megjelentek a játékok is a platformokon, mára pedig az alkalmazások egyik legnagyobb részét teszik ki. Hatalmas sikernek örvendtek példeául az Angry Birds játékok, újabban pedig a Clash of Clans, vagy a Clash Royale. A készülékek fejlődése lehetővé tette a régebbi számítógépes és konzol játékok portolását is, és a korábban csak PC-ken elérhető címek iOS-en és Androidon is megjelentek, a népszerű kártyajáték, a HearthStone mobil verziói ugyanabban az évben jelentek meg, mint a PC kiadás, de a Minecraftank is van mobil verziója: a Pocket Edition.

Egy modern, bár 2D grafikával rendelkező játékot szerettem volna készíteni, átugorható akadályokkal; ellenségekkel, akikkel a játékos megküzdhet, vagy akiket megpróbálhat elkerülni. Csontalapú animációt és kézzel készült, csempe alapú pályákat használtam.

# Felhasználói dokumentáció

## A program témája

A játék egy - a középkori Európa által inspirált - fantáziavilágban a játszódik, a főhős pedig Ragnar Lothbrok, az izlandi sagák egyik szereplője. Ragnar a társaival éppen a frankok földjén fosztogatott, amikor katonák jelentek meg a semmiből, és Ragnar a keletkezett zűrzavarban elválasztódott bajtársaitól. Így egyedül kell megmenekülnie, úgy, hogy közben a szerzett kincsre is ügyelnie kell.

Műfaját tekintve a program 2D, oldalnézetes platform játék, a játékos Ragnart irányítja, és a pálya egyik feléből el kell jutnia a másikba, közben akadályokat átugorva, ellenségeket legyőzve, és ügyelve, hogy minél több kincs megmaradjon.

## Rendszerkövetelméy

A játék által támogatott platformok: Windows, Linux, macOS személyi számítógépen, Android mobil eszközökön. A minimális ajánlott felbontás 1280 x 720, az ajánlott képarány pedig 16:9.

### PC-n

* Java Runtime Environment 7
* OpenGL 4.1-képes videókártya

### Androidon

* Android 4.0.3
* OpenGL ES 2.0

## A program futtatása

### PC-n

A futtatáshoz szükség van a Java Runtime Environment szoftverre. Ha nincs a számítógépen telepítve, le kell tölteni a <https://java.com/en/download/> címről. A letöltött fájlt futtatva kövessük a megjelenő utasítások a Java telepítéséhez.

Ha telepítve van a Java a számítógépen, a játék futtatható a Lothbrok.jar fájlra való dupla kattintással, vagy parancssorból a következő parancs kiadásával: java -jar Lothbrok.jar

### Androidon

A Lothbrok.apk fájlt először át kell másolni az Android készülék Downloads vagy Letöltés mappájába a számítógépről. Ehhez csatlakoztatni kell az eszközt a PC-hez, majd Androidon kiválasztani a File Transfer opciót a megjelenő USB beállítások közül. Ezután megnyitható az Android készülék a számítógépen és át lehet másolni az .apk fájlt.

A telepítés előtt engedélyezni kell a külső forrásból érkező alkalmazások telepítését. Ezt a Settings menü Security alpontjában tehetjük meg, az Unknown Sources opció bepipálásával.

Az átmásolt fájl megjelenítéséhez és futtatásához szükség van egy fájl kezelő alkalmazáshoz az Android készüléken. Amennyiben nincs ilyen gyárilag telepítve, le kell tölteni egy ilyen alkalmazást, például a következő linken található appot: <https://play.google.com/store/apps/details?id=com.asus.filemanager&hl=en>.

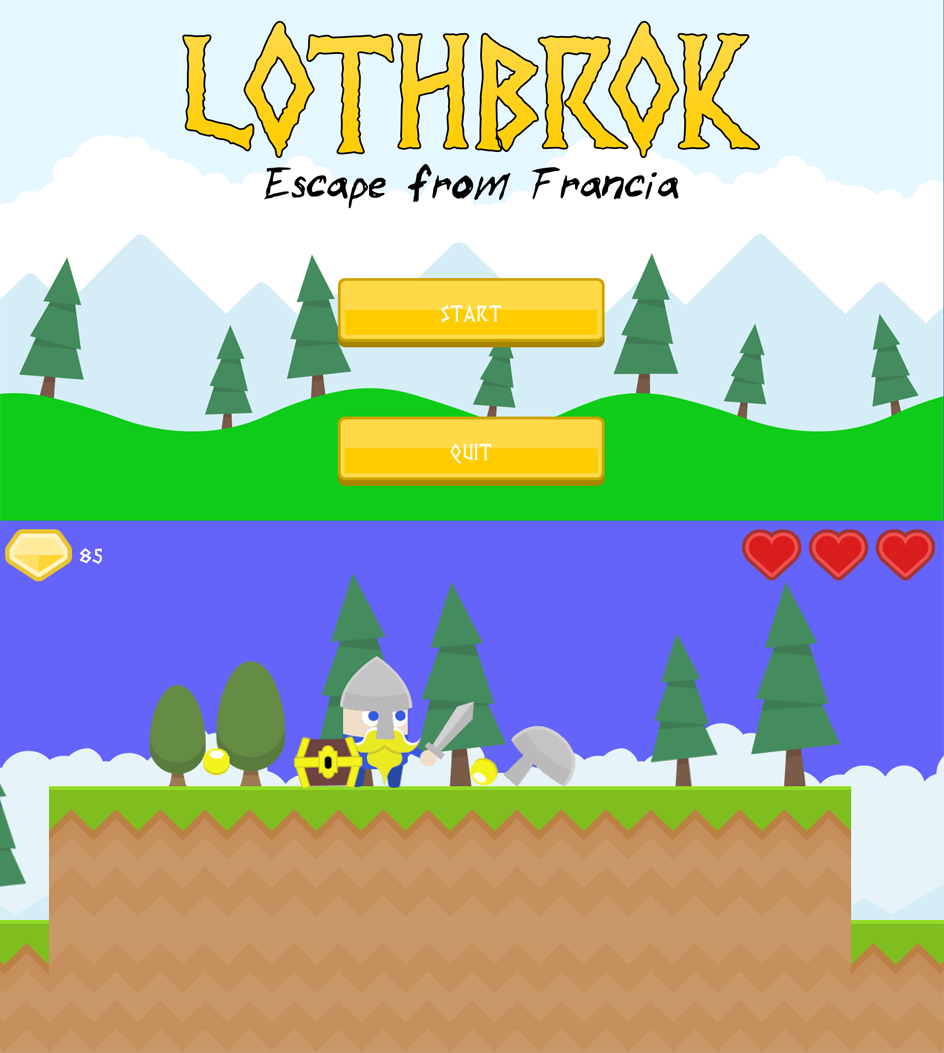
A fájl kezelő alkalmazásban keressük meg a Downloads/Letöltés mappát, és érintsük meg az átmásolt Lothbrok.apk ikont. Ezzel telepíthetjük a játékot a készülékre.

Telepítés után a program elérhető a telefon App Drawerjében, vagy parancsikonként valamelyik kezdőképernyőn.

## A program használata

A program indulásakor rövid töltés és töltési képernyő után a főmenü fogad. Itt két lehetőségünk van, a „Play” gombra kattintva indíthatjuk el a játékot, a „Quit” gombbal pedig kiléphetünk a programból.

A játék indulásakor Ragnar leesik az égből az egyik platformra, már ekkor irányítható. A cél: elérni a pálya túlsó oldalán lévő csillagot. A képernynő a pályán és a karaktereken kívül egyéb információkat láthatunk, mint például a játékos fennmaradó élete. Ez az ún. „Heads-up Display”, röviden HUD. A képernyő tehát két fő részből áll: a játéktérbő és a HUDból. Androidon láthatók az irányításhoz szükséges gombok és a virtuális joystick is. A játék szüneteltethető, ekkor a „Pause” menü jelenik meg. Innen kiléphetünk a játékból, vagy folytathatjuk azt.



### A játéktér.

A platformokon lehet jobbra-balra közlekedni, ugrálni. A kamera követi a játékost, így mindig Ragnar közvetlen környezete látható. A pálya alján helyenként víz található, ha ebbe beleesünk, az azonnali halállal jár és véget ér a játék.

A platformokon ellenségek járőröznek. A sárga szakállas, kék ruhás karakter a játékos, a többiek ellenfelek. Ha egy ellenfél közelébe érünk, az elkezd Ragnar felé mozogni, és ha elég közel ér, megtámadja a játékost. Ragnar három kardcsapást kibír, az ellenségeknek viszont egy is elég.

Minden mozdulat, a mozgás, ugrás, támadás esetén Ragnar kincse fogy: arany pénzérmék potyognak a poggyászából, ezért minden lépés megfontolandó!

### A HUD

A bal felső sarokban egy gyémánt ikon mellett láthatjuk a maradék kincsünket: ebből mindig egyet veszítünk, amikor a játéktéren is látható, hogy elgurul egy pénzérme.

A jobb felső sarokban látható a maradék élet: annyi szív ikon, ahány élete még maradt a játékosnak, ami kezdetben három.

### Irányítás

Az irányítás PC-n és Androidon különböző, a platform sajátosságai miatt.

Androidon a karaktert a képernyőn megjelenő gombokkal és joystickkal lehet irányítani, a játékot szüneteltetni és a „Pause” menüt is egy ilyen gombbal lehet elérni. PC-n az írányításhoz a billentyűzet használható.

### Irányítás PC-n

A játékos karakter a következő billentyűkkel irányítató:

* **„A”** mozgás balra
* **„D”** mozgásjobbra
* **„W”** ugrásh
* **„H”** támadás

A „Pause” menü az **„ESC”** billentyűvel hozható elő

### Irányítás Androidon

A karakter jobbra és balra mozgatható a virtuális joystcikkal, ami a bal alsó sarokban található. A jobb alsó sarokban lévő kard ikonnal jelzett gombbal támadni, a nyíl ikonos gombbal pedig ugrani lehet.

A „Pause” menü az alul középen található, négyszögletes gombbal jeleníthető meg.



### A játék vége

A játéknak kétféleképpen lehet vége: a játékos meghal (beleesik a vízbe, vagy legyőzi az egyik ellenfél), vagy eléri a csillagot és nyer.

Előbbi esetben a „Game Over” képernyő, utóbbiban pedig a „You won” képernyő jelenik. Mindkét esetben visszatérhetünk a főmenübe „Main Menu” gombra kattintva.



# Fejlesztői dokumentáció

## A megvalósítandó feladat

A megvalósítandó program egy 2D oldalnézetes platform játék. Rendelkezik egy főmenüvel, ahonnan elindítható a játék. A játékmenet szüneteltethető és a „Pause” menüből visszajuthatunk a főmenübe. A játék végén a „Game Over” vagy „You Won” menűből szintén visszajuthatunk a főmenübe. A navigációt gombok segítik.

A játék egy 2d, csempékből és dekorációbál álló pályán játszódik. A csempék alkotta platformokon a karakterek mozoghatnak. A háttérelemek parallax mozgással a 3D illúzióját keltik.

A játékos egy platform játék esetén elvárható fizikával rendelkezik: tud jobbra-balra mozogni, ugrani és esni, viszont nem tud felborulni, pattogni, vagy csúszni, és nem „ragad” a falhoz. Mozgás vagy támadás esetén kincset veszít, ami viszont valós fizika szerint mozog: esik, pattog és gurul.

Az ellenfelek jobbra-balra járőröznek a kezdeti pozíciójuk egy sugarában. Ha a játékos a sugáron belülre kerül, az ellenfél elindul felé, majd ha elég közel érnek, megtámadja. A játékos szintén meg tudja támadni az ellenfeleket, és legyőzhetik egymást.

A játékos meghal, ha az élete elfogy: ha a vízbe esik, az összes életét elveszíti, ha pedig egy ellenfél eltalálja, egyet veszít.

A játékos élete és kincse legyen számon tartható egy Heads-up Display (HUD) segítségével.

Az irányítás PC-n billentyűzettel, Androidon pedig a HUD-on megjelenő gombokkal és joystickkel történik.

## A megvalósítás eszközei

### Lehetőségek

A dolgozat írásakor több lehetőség is rendelkezésre állt játékfejlesztéshez használható programozási nyelvek és technológiák tekintetében, ezek három csoportba sorolhatók:

* saját játékmotor készítése
* keretrendszer használata
* 3rd party játékmotor alkalmazása

Ezek közül a középső opciót választottam, mivel a saját játékmotor fejlesztése önálló téma, 3rd party alkalmazása pedig a dolgozat témájához túl absztrakt, és sok elem már implementálva van.

Szrencsére játékfejlesztő keretrendszerekből is nagy a választék, szinte minden népszerű programozási nyelvhez találunk megoldást. Ilyenek a teljesség igénye nélkül:

* MonoGame, C#
* Löve, Lua
* Cocos2d-x, C++
* libGDX, Java

A választásom a libGDX-re esett, a minőségi és teljes dokumentáció, és a nagy méretű, segítőkész közösség miatt.

### libGDX

A játék Java nyelven, a libGDX keretrendszer segítségével készül. A libGDX egy nyílt forráskódú, cross-platform játékfejlesztő keretrendszer Java nyelven. Segítségével 2D és 3D játékok készíthetők, absztrakcióinak köszönhetően mentesít az alacsony szintű kód írásától (pl. OpenGL), és számtalan platformra kiadható az elkészült termék.

A jó teljesítményt az OpenGL ES alapú megjelenítés, és Garbage Collectort minimálisan használó gazdag API biztosítja.

### Animáció – Spriter

A karakterek mozgásához, tevékenységeinek megjelenítéséhez csont alapú animációkat használtam. Ezek lényege, hogy a szerkesztőprogramban a behúzott képek alá egy csontvázat állítunk, és a karaktert a csontok mozgatásával, forgatásával animáljuk. A csontok szülő-gyerek kapcsolatban állhatnak – ha a szülő mozog, a gyerekeit is magával viszi. Minden csonthoz beállíthatunk egy, vagy több, a karaktert alkotó képet, amit mozgat.

A csont alapú animáció előnye a hagyományoshoz képest, hogy gyorsabban lehet látványos eredményt elérni, és minden képből csak egy példányt kell tárolni, mivel az animációt a csontok időponthoz kötött koordinátái és elforgatási szögei határozzak meg.

Az animációk létrehozásához, szerkesztéséhez a Spriter animációs szoftvert használtam. A létrejött animációk a felhasznált képekből és egy .scml kiterjesztésű fájlból állnak – utóbbi tárolja az animáció adatait.



A libGDX nem támogatja az .scml fájlok beolvasását, ezért a közösség egy tagja, trixt0r által készített implementációt alkalmaztam, amit kiegészítettem. (1)

### Pályaszerkesztés – Tiled

A játékban csempealapú, ortografikus pályán játszhatunk, ami a platform csempéken kívül háttér- és dekorációs elemeket is tartalmaz, továbba a Box2d fizikai motor által használt poligonokat, és a játékos illetve ellenfelek kezdő pozícióját is tartalmazza.

Ilyen pályát a Tiled általános célú, csempe alapú 2D pályaszerkesztő programmal készítettem. Egy Tiled pályában egy csempéhez egy előre megadott méretű kép, és tetszőleges mennyiségű tulajdonság tartozik. A grafikus szerkesztőprogramokhoz hasonlóan rétegek hozhatók létre a különböző pályaelemek elválasztására, amik három típusúak lehetnek: Tile layer, Object layer és Image layer. A Tile layereken helyezhetjük el a csempéket, az Object layeren pedig minden mást: poligonokat, képeket, amiket tulajdonságokkal ruházhatunk fel, hogy felhasználjuk a játék futásakor. Az Image layerben egy képet lehet tárolni.



Az elkészült pálya egy .tmx kiterjesztésű xml fájlba kerül mentésre, és tömörítést is használhatunk. A Tiled pályák betöltését és kirajzolását támogatja a libGDX, ezért ezt az API-t használtam.

### Audio

A játékban használt zenét és hangaokat a libGDX API-val töltem be és játszom le.

### Betűtípus

A betűtípus fájlból a libGDX FreeTypeFont kiegészítőjét alkalmazva hozom létre a játékban megjeleníthető BitmapFontokat.

### Fizika – Box2d

Bár a játékos és az ellenfelek saját, egyszerűsített és a célra specializált fizikával rendelkeznek, a hulló, guruló, pattogó kincsek mozgásához a Box2d fizikai motort használtam, ami kiegészítésként elérhető a libGDX keretein belül.

Segítségével valós fizikai paraméterekkel rendelkező testek definiálhatók poligonok használatával – megadható a világ gravitációja, a testeket alkotó alkatrészek sűrűsége, rugalmassága stb.

### Projektgenerálás

A projekt létrehozására a libGDX projektgeneráló eszközt használtam(2). Az eszközben megadható a projekt neve, a csomag, a főosztály neve, a célkönyvtár, valamint be kell állítani az Android SDK helyét. Kiválaszthatók a célzott platformok is, valamint kiegészítéseket is megadhatunk, például a Box2d fizika könyvtárat.

Az eszköz futtatásának eredménye egy Gradle projekt, amelyben a különböző platformok (indító) kódja, valamint a közös kód (core) külön Gradle modulokba kerülnek.

### Disztribúció

#### PC

A desktop Gradle modul dist nevű taskját meghívva készíthetjük el a futtatható .jar állományt.

#### Android

Android Studioban a „Build” menü alatt található „Generate Signed APK” opcióval készíthető el az aláírt Android .apk fájl, ami a mobil készülékre másolva telepíthető és futtatható.

## Az erőforrások előkészítése

A PC-k és mobil eszközök különböző karakterisztikái miatt a játék négy méretet támogat, a csomagolt erőforrások szempontjából. Ezek előkészítésének és csomagolásának automatizálására Gradle taskokat alkalmaztam.

Elsősorban a textúrák szempontjából fontos ez – 4K felbontású monitorokon és kisebb képernyős, 720p kijelzővel rendelkező telefonokon is szép kell legyen a grafika. Mivel a térkép fájlok, animációk függhetnek a textúrák méretétől, ezeket is módosítani kell szükség szerint. A négy támogatott méret a következő:

* XL – 3840 x 2160
* L – 2560 x 1440
* M - 1920 x 1080
* S – 1280 x 720

Az erőforrások tárolására két mappát hoztam létre, két helyen: a core modul „assets\_raw” mappájába kerülnek a nyers, csomagolandó erőforrások. Az android modul „assets” könyvtárába kerülnek a csomagolt, kész anyagok. A libGDX is ebben, az utóbbi mappában keresi alapértelmezetten az erőforrásokat, innen kerülnek betöltésre a játékban.

### Textúrák

A játékban a hatékony rajzoláshoz a képi elemeket, textúrákat egy nagyobb, közös texúra fájlba kell csomagoli (Texture Atlas), ráadásul az összes támogatott méretben. Ez kézzel nem csak fáradalmas, de nem is feltétlenül eredményezi a leghatékonyabb méretű Atlast. E feladat automatizálására a libGDX TexturePacker kiegészítését alkalmaztam, Gradle taskok keretén belül. A TexturePacker segítségével a megadott paraméterek szerint hatékonyan elrendezett Atlasokat kapunk a kép fájlokból. A kívánt Atlas újrapakolásához csak meg kell hívni az adott Gradle taskot.

### Animáció

A Spriter animációk függnek a hozzájuk tartozó képektől, ezért el kell őket menteni az összes kívánt méretben, erre van lehetőség a szerkesztő programban.

### Pályák

A Tiled által készült pályák szintén tartalmaznak adatot a textúrák méretéről, viszont nincs lehetőség a különböző méretű mentésre, mint a Spriter esetén. Ezért a pályát csomagoló Gradle task a createAtlasReadyMapCopies.py script segítségével módosítja a .tmx fájlokat, hogy az összes támogatott méretben működjenek.

## Megvalósítási terv

### A Screen tervezési minta

A program háromféle fő komponensre oszthatók a „screen” játékfejlesztési design pattern szerint, ezek mind a libGDX Srcreen interface-ének implementációi – töltőképernyők, főmenü és játékmenet.

A program indulásokor először az EssentialLoadingScreen jelenik meg, betölti a legszükségesebb erőforrásokat. Ezt követi a MainMenuLoadingScreen, ami a többi erőforrást tölti be, és közben lejátszik egy animációt. Ezzel a töltés végére értünk.

Következő lépésben a MainMenuScreen töltődik be, ahonnan a megfelelő gombbal eljutunk a végső állapothoz – a GameScreenhez.



A töltőképernyő osztályok egyszerűek, feladatuk az erőforrások betöltése és töltés képernyő megjelenítése. Hasonlóan egyszerűbb a főmenü felépítése, itt a libGDX Scene2d API-ját használtam, rábízva a megjelenítés és az input kezelését.

Mivel a célom az volt, hogy a játék közben megjelenő menük – a „Pause”, a „Game Over” és a „You Win” – hátterében látható legyen a játékmenet, ezek a menük nem új Screen implentációként, hanem mint rendererek (inputfelülettel) jelennek meg a GameScreenben.

### MVC architektúra

A program játékmenettel foglakozó része, a GameScreen, egy MVC-szerű (Model – Modell, View – Nézet, Controller – Irányító) architektúra szerint készült: a modellben csak állapotokat és viselkedést tárolunk, a nézet pedig csak a rajzolásért, illetve hangok vagy zene lejátszásáért felel. A kettőt a controller rétegben kapcsoltam össze: ezek a libGDX Screen interface implementációja (GameScreen), illetve ezen belül az input- és mesterséges intelligencia alapú irányítók – Controller osztályok – ezek irányátják a játékos és az ellenség mozgását, illetve a „Pause” menü elérését is.

A modell közvetlenül nem kommunikál a nézet réteggel, állapotai az irányítón keresztül kerülnek a megjelenítőhöz. A nézet is az irányítón keresztül küld adatokat (az ütközésellenőrzéshez szükséges négyszögetket) a modellnek.



### A Game loop tervezési minta

A videójátékok klasszikus tervezési mintája, a Game loop szerint egy ciklusban másodpercenként többször (pl. 30, 60 stb.) frissítjük a teljes játékmenet (GameScreen), és újra rajzoljuk a világot. A főmenü és a töltőképernyők is hasonlóan működnek: előbbi figyel az eseményekre és frissíti illetve kirajzolja a felhasználói felület widgetjeit, utóbbi pedig a háttérben tölt és közben esetleg animációt jelenít meg.

A loop már implementálva van a keretrendszerben, ez az ApplicationListener interface render callback metódusát hívja minden iterációban. A Game osztály implementálja ezt az interface-t és egy Screen kezelő rendszerrel egíészíti ki – így az aktív Screen render metódusa is meghívódik.

Hasonlóan kezeli a keretrendszer a többi callback metódust, ami meghívódhat az applikáció élettartama során, például: resize, pause stb.

A program belépési pontja minden platformon a Game osztályból leszármazó Lothbrok osztály.

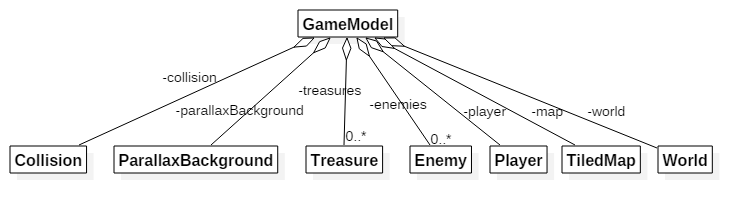


### MVC - Model

A játék világát a GameModel osztály modellezi, mezőként megtalálható benne a pálya, a játékos, az ellenségek, a harcrendszer ütközési rendszere, a kihullot kincs és a fizikáért felelős világ.

A pálya tárolja, hogy melyik csempék blokkolnak (a játékos és az ellenségek ütköznek velük, platformkén használhatják őket), és hogy hol van a játékos és az ellenségek kezdőpozíciója. Ezen kívül a Box2d fizikához szükséges poligonokat is a térképből nyerhetjük ki.

Ezek hozzáadva a Box2d világhoz, ütközőfelületként szolgálnak a játékos által elvesztett kincsek számára.

Kincs megadott időközönként jelenik meg, a játékos ládájánál. Ezt a pozíciót a GameScreenen keresztül a játékos animációjából (nézet réteg) kapjuk meg. Ez alapján rögtön a megjelenéskor erő hat az érme, így az elrepül, majd a platformokon elgurul a Box2d világ szabályai szerint.

#### Entity-component-system (ECS) tervezési minta

A játékos és az ellenség osztályok hasonló módon lettek implementálva. Hagyományosan, a játék entitásokat (játékos, ellenfelek, egyéb karakterek) több szintű leszármazással implementálták.

A közös Entity osztályból származott az összes szereplő, akár több köztes rétegen keresztül, ez azonban többször konfliktust okozott – egy osztály akár több őshöz is tartozhat. Feltételezve például Tree és Enemey osztályokat, az EvilTree osztálynak mind a kettő őse kéne legyen. (3)

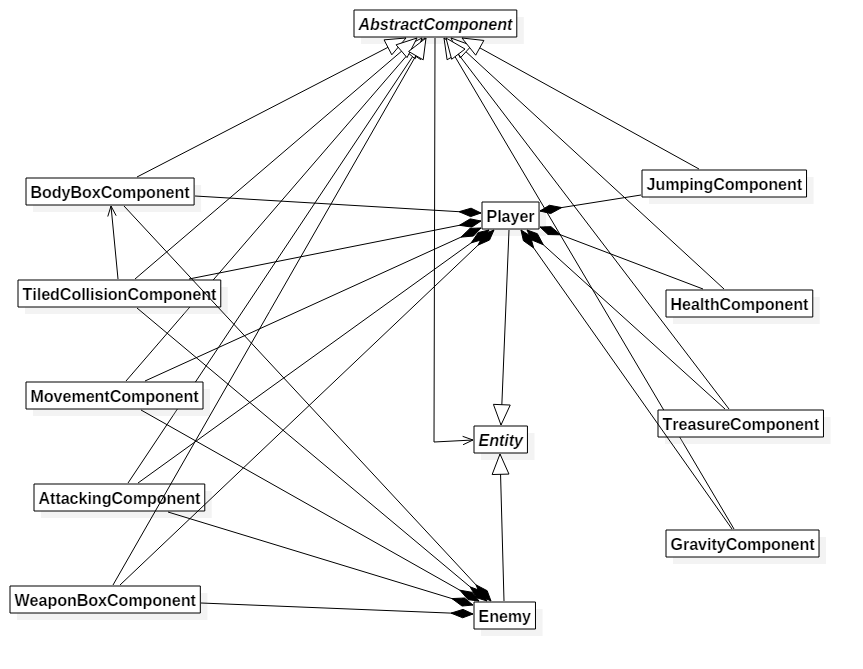
Esetünkben például a játékos egy mozgó, ugró, és támadó entitás. Ennek megfelelően lehetnek a szülei a MovingEntity, a JumpingEntity és AttackingEntity osztályok. Ez azonban többszörös öröklődés lenne, így egyszerűsíteni kellene – legyen a játékos ősosztálya a MovingJumpingAttackingEntity. Ez nem csak túlbonyolítja a modellt, de ha az ellenség nem képes például ugrani, létre kell hozni még egy újabb ősosztályt MovingAttackingEntity névvel.

Ezen rendszer helyett az Entity-component-system tervezési minta egy módosított változatát implementálva alkottam meg a játékos és ellenfél osztályokat.

Az ECS minta szerint az entitások komponenseket tartalmazó halmazok, a komponensek pedig adatok. Az adott komponensekből álló entitásokat pedig adott rendszerekkel lehet módosítani. Például a Movement komponenssel rendelkező entitásokon értelmezve van a MovementSystem rendszer: a rendszer a komponens adatait módosítja - az x, y koordinátákat.

A dolgozat implementációja szerint az Entity osztály egy háromféle állapottal (mozgás, élet, tevékenység), továbbá pozícióval és iránnyal rendelkező főosztály. Ebből származnak az Enemy és a Player osztályok. A többszörös, köztes származási rétegek helyett ezek az AbstractComponent osztályból leszármazó komponenseket tartalmaznak, mint mezők. Ezek a komponensek azonban rendelkeznek alapvető metódusokkal (szemben az ECS mintával), a mozgás komponensbe például már be vannak építve a moveTo, moveLeft és moveRight funkciók, így ezek a komponensek az ECS-szerinti rendszereket is magukba foglalják. A komponenseket együtt alkalmazó funckiók a Player és Enemy osztályokba kerültek.

Mivel a játékban csak két entitás van, elegendő a fenti rendszer és nem szükséges egy teljes ECS implementáció. Ennek előnye, hogy az entitásokat könnyebb elválasztani a program többi részétől, például az irányításól, ami logikailag nem tartozik a modellhez.



### MVC – Controller

A GameScreenen kívül a Controller osztályok alkotják az irányító réteget. Ezek feladata a szó szoros értelmében vett irányítás: a PlayerController a játékos karakter mozgását, ugrását, támadását, az EnemyController pedig az ellenfeleket irányítja. A PauseController azt figyeli illetve állítja, hogy a játék meg van-e állítva.

A PlayerController és a PauseController állapotát az input változtatja minden iterációban (tehát PC esetén a billentyűzet, Androidon pedig a képernyőn megjelenő gombok), a Controllerek pedig a beállított állapot alapján adnak parancsot az általuk irányított entitásnak. Ezzel szemben az EnemyController a magába foglalt egyszerű mesterséges intelligencia szerint irányítja az Enemy osztály példányait.



### MVC – View

A nézet réteget a különböző Renderer osztályok és az Audio osztály alkotják. Ezek a GameScreen mezőiként jelennek meg a programban. Közülük a legösszetettebb a GameRenderer, a játéktér megjelenítéséért felelős osztály.

Tartalmaz referenciákat a modell azon részeire, amelyek szükségesek a megjelenítéshez – a modell állaptának megfelelően választja ki a GameRenderer, hogy mit kell megjeleníteni. Például ha játékos nem mozog vagy támad, akkor az alap, „idle” animációt játsza le.

A többi Renderer a felhasználói interfész megjelenítéséért felelős.

### Koordináta-rendszer és képarány

Egyszerűbb játékok, vagy olyan programok esetén, amik csak előre adott hardveren, vagy előre ismert (kevés vagy csak egy fajta) megjelenítőn lesznek használva, elegendő lehet a pixeleket használni a játéktér vagy menük koordinátáiként.

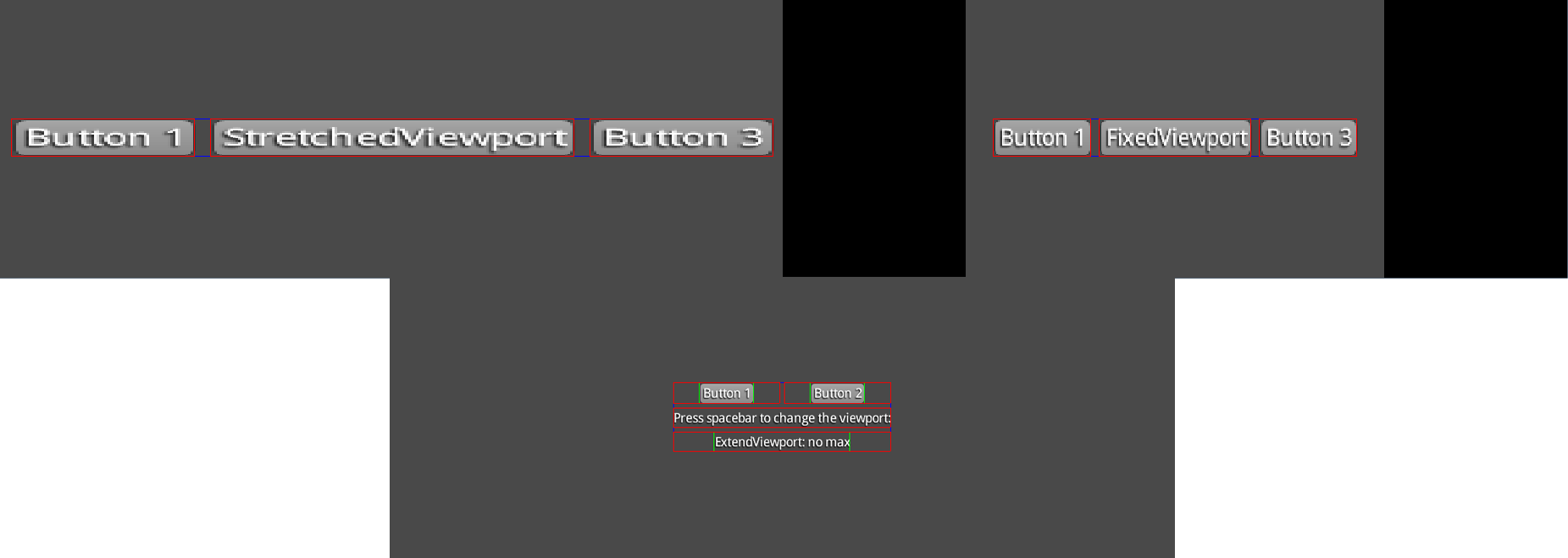
Sőt, ha a fenti feltételek nem teljesülnek, akkor is használható lehet a pixel alapú koordinátarendszer - szükséges esetén skálázással – ha a grafikai elemekhez megfelel a stratégia, például pixel art (olyan grafikai stílus, amelyben a karakterek, tárgyak stb. négyzetekből, azaz „pixelekből” állnak, felelevenítve a korábbi játékok technikai határaiból következő megjelenését) használata esetén, Nearest Neighbour skálázás mellett.

Mivel a dolgozat témájának egyik fenti feltétel sem felel meg, célszérű a játékvilágnak egy új koordinátarendszert definiálni. Így ugyanazok a koordináták használhatók minden platformon, a megjelenítők méretétől és képarányától függetlenül. Ez jó ötlet továbbá azért is, mert a Box2d fizika motor, ami az elveszett kincs mozgásáért felel, szintén saját koordinátarendszert használ: 1 egység = 1 méter.

További probléma a képarány kezelése, amire a modern megjelenítők különbözősége miatt kell odafigyelni. A legegyszerűbb stratégia, hogy figyelmen kívül hagyjuk. Ennek eredményeként a tervezettnél különböző képarányú képernyőkön a kép horizontális vagy vertikális irányban nyúlik, ami (hacsak nem ez volt a cél) nem kellemes látvány a felhasználó számára. Egy másik stratégia szerint, ha különbözik a tervezettől a készülék képaránya, akkor nem nyújtjuk el a képet, hanem kirajzoljuk a középre, így a széleken vagy fekete csíkok jelennek meg, vagy a játéktér egy részét le kell vágni. A legelegánsabb (bár nem mindig megfelelő) megoldás szerint a játéktér kitölti a képernyőt, aránytól függetlenül és nyújtás nélkül, tehát a különböző megjelenítőkön több vagy kevesebb fog látszani a játéktérből.

A játéktér és a képernyő pixeleinek koordinátarendszere közötti egyszerű átváltás a libGDX Viewport API-ja segíti. A Viewport egy alosztályának példányosításakor megadható a világ kívánt mérete, ezután ebben a koordinátarendszerben dolgozhatunk.

A Viewport leszármazott osztályaiból válogatva a fenti három képarány stratégia mindegyikét támogathatjuk a libGDX keretrendszert alkalmazva, ezek rendre: StretchViewport, FitViewport és FillViewPort, ExtendedViewport.



3.1. ábra A három képarány stratégia illusztrációja (4)

Annak megfelelően, hogy a játékteret, vagy a menüt kell megjeleníteni, kétféle stratégiát alkalmaztam.

#### Játéktér

A játéktérben 1 egység 1 méternek feleljen meg, és 1 csempe legyen 1 egység (azaz méter) széles illetve magas. A koordináták tehát törtek lesznek, az első csempe közepének x koordinátája például 0,5.

A játéktér 4 egység magas, a szélessége pedig a képarány szerint változik.

A képarány változása tehát úgy befolyásolja a játéktér megjelenését, hogy szélesebb vagy szűkebb részletet mutat, míg a magasság mindig 4. Így elkerülhetők a fekete csíkok a képernyő szélein, bár különböző képaránnyal rendelkező eszközökön több, vagy kevesebb fog látszani a pályából. Ez többjátékos vagy kompetitív mód esetén nem eredményezne fair játékmenetet, azonban mivel a játék ilyen funkciókat nem támogat, megfelelő a módszer.

#### Főmenü

# Irodalomjegyzék

1. **Trixt0r.** Trixt0r/spriter: A Generic Java importer for Spriter animation files. *GitHub.* [Online] [Hivatkozva: 2016. 06. 29.] https://github.com/Trixt0r/spriter.

2. **libGDX.** libGDX. *libGDX.* [Online] [Hivatkozva: 2016.. 06. 28.] https://libgdx.badlogicgames.com/download.html.

3. **Games, Boreal.** gamedev.net. [Online] 2013.. 04. 02. [Hivatkozva: 2016.. 08. 24.] http://www.gamedev.net/page/resources/\_/technical/game-programming/understanding-component-entity-systems-r3013.

4. **Maxim.** GitHub libGDX Wiki. *GitHub.* [Online] [Hivatkozva: 2016.. 10. 01.] https://github.com/libgdx/libgdx/wiki/Viewports.