

Pannon Egyetem

Műszaki Informatikai Kar

Rendszer- és Számítástudományi Tanszék

Mérnökinformatikus MSc

## DIPLOMAMUNKA

"Free to play, run to win" játék Androidra

Nyitrai Tamás

Témavezető: Dr. Hegyháti Máté

2016

## KÖSZÖNETNYILVÁNÍTÁS

Meg szeretném köszönni édesanyámnak támogatását és folyamatos bátorítását. Továbbiakban köszönnettel tartozom témavezetőmnek, Dr. Hegyháti Máténak, aki ötleteivel és tanácsaival végig a helyes úton tartott.

Továbbá szeretném megköszönni Hollósi Tamásnak a dream-iso-android elkészítőjének a segítségét, akihez bátran fordulhattam, ha bármi kérdésem volt a játékmotorral kapcsolatban. Valamint köszönöm Böröndi Evelinnek a grafikák elkészítésében vállalt segítségét.

## TARTALMI ÖSSZEFOGLALÓ

Az utóbbi évtizedben az okoseszközök népszerűsége évről-évre folyamatosan növekedett és ennek következtében már hazánkban is a lakosság több, mint fele rendelkezik legalább egy ilyen eszközzel. Egy ilyen okoseszközt rengeteg módon lehet felhasználni: például újságot olvashatunk rajta, bevásárolhatunk, böngészhetünk menetrendek közt, vagy akár játékokkal is elüthetjük az időnket. Az okoseszközök térhódításával egyre elterjedtebbé váltak az olyan játékok, amelyek az úgynevezett micro-paymenteken alapszanak, vagyis a játékos kis összegű vásárlással könnyítéshez, előnyhöz juthat. Ezekre a játékokra szokás használni a "free to play, pay to win" kifejezést.

Mindezekkel párhuzamos jelenség, hogy a rendszeres testmozgás egyre több ember minden napjaiból hiányzik. Sétálás vagy kerékpározás helyett sokan kényelmi okokból az autós vagy tömegközlekedést választják; a szabadidő eltöltésében pedig egyre kevésbé az aktív tevékenységek játszanak szerepet, mely részben az okoseszközök térhódításának is köszönhető. Sok ember számára nincs elegendő motiváló hatása annak, hogy a rendszeres testmozgás egészségügyi szempontból fontos része kellene legyen a minden napoknak. A hiányzó motiváció pótlására különböző lehetőségek tárulkoznak az emberek felé, mint amilyen a virtuális díjazás, kedvezményekre becserélhető jutalom pontok kilométerenként vagy akár a készpénzzel való díjazás.

Munkám során elkészítettem egy Androidos szerepjátékot a két trend összekapcsolása érdekében. A játékban a felhasználó csak úgy tud előre jutni, illetve akkor kaphat az előrehaladást segítő ajándékokat, ha kimegy a szabadba futni, kerékpározni, túrázni vagy bármilyen más sport formájában kilométereket gyűjteni. Lényegében a micro-paymentek szerepét vette át a sportolás, így a játék "free to play, run to win" filozófiát követ.

**Kulcsszavak:** dream-iso-droid, Android, testmozgás, szerepjáték

## ABSTRACT

In the last decade, the popularity of smart devices has increased steadily year by year and therefore more than half of the population in our country already own at least one of those devices. These smart devices can be used in lots of different ways: for example, we can read newspapers on it, can do shopping online, browse through schedules, or even can play games to spend our time. With the expansion of smart devices, games have become more common which are based on so-called micro-payments, meaning that the player can gain advantages in the game by spending just a small amount of money on purchasing. We usually use the "free to play, pay to win" phrase for these games.

With all these, a parallel phenomenon is that more and more people are missing their regular daily exercises. Because of their convenience, many people use car or public transportation instead of walking or cycling; in their free time, the active exercising doesn't take an important role anymore, which is part of the expansion of smart devices. For many people, there is not enough motivating effect that regular exercise should be an important part of their health in everyday life. To make up the lack of motivation, various opportunities revealed themselves to the people, such as virtual remuneration, allowances exchanged for bonus points per kilometer or even to pay with cash.

During my work, I have prepared a roleplaying game to Android in purpose to link the two trends. In the game, the users are only able to improve, or get gifts to help them progress if they go out into the open to run, bike, hike or any other forms of sport to collect kilometers. Essentially, sport took over the role of micro-payments, so the game follows the "free to play, run to win" philosophy.

**Keywords:** dream-iso-droid, Android, sports, RPG game

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>7</b>
<b>2. Hasonló fejlesztések, szerepjátékokról általánosan</b>	<b>9</b>
2.1. A videójátékok története . . . . .	9
2.2. Sportjátékok . . . . .	11
2.2.1. Zombies, Run! . . . . .	12
2.2.2. Tep . . . . .	13
2.2.3. Pokémon Go . . . . .	14
2.3. Az RPG játékokról általánosságban . . . . .	14
<b>3. Követelmények, technológiák</b>	<b>17</b>
3.1. Követelmények . . . . .	17
3.2. Felhasznált technológiák . . . . .	18
3.2.1. Játékmotorok . . . . .	18
3.2.2. Fejlesztőkörnyezet . . . . .	20
3.2.3. OAuth protokoll . . . . .	20
3.2.4. OrmLite . . . . .	21
<b>4. Felhasználói kézikönyv</b>	<b>22</b>
4.1. Kezdőképernyő . . . . .	23
4.2. Karakter . . . . .	23
4.3. Város . . . . .	24
4.4. Harcmező . . . . .	28
4.5. Harcjelenet . . . . .	28

<b>5. Fejlesztői dokumentáció</b>	<b>30</b>
5.1. A játék indítása . . . . .	31
5.2. Adatbázis felépítése . . . . .	35
5.3. Játék megjelenítése . . . . .	37
5.4. A játéklogika . . . . .	40
<b>6. Továbbfejlesztési lehetőségek</b>	<b>45</b>
<b>7. Összefoglalás</b>	<b>48</b>

# Ábrák jegyzéke

2.1.	Tennis for two - első videójáték . . . . .	10
2.2.	Donkey Kong játék egy jelenete . . . . .	10
2.3.	A Zombies, Run! felülete . . . . .	12
2.4.	Tep játék felülete . . . . .	13
2.5.	Pokémon GO játékelemei . . . . .	14
2.6.	Dungeons & Dragons karakterlap . . . . .	15
3.1.	Játékmotorok összehasonlítása . . . . .	19
3.2.	Az OAuth protokoll absztrakt működési ábrája . . . . .	20
4.1.	A főmenü és a Runkeeper csatlakozási felület . . . . .	22
4.2.	A város látképe . . . . .	24
4.3.	Részletes tárgyinformáció . . . . .	25
4.4.	Pillanatképek a boltokról . . . . .	26
4.5.	Karakterlap és jutalomlista . . . . .	27
4.6.	Harcmező és harcjelenet . . . . .	29
5.1.	A program architektúrája . . . . .	30
5.2.	Jutalomcsoportok és esélyek táblázata . . . . .	35
5.3.	Az adatbázis szerkezete . . . . .	36
5.4.	A játékoshoz tartozó forráskép . . . . .	39

## 1. fejezet

# Bevezetés

Az okos-telefonok térhódítása miatt már hazánkban is a lakosság fele rendelkezik valamilyen okos eszközzel, ez a szám pedig a jövőben egyre csak növekedni fog. A minden nap élet megkönnyítésére rengeteg féle alkalmazás születik napról napra. Egyetlen eszközön olvashatunk újságot, tudakozódhatunk a közlekedésről, lehetünk ebédet magunknak, vagy unaloműzésként játszhatunk. Manapság minden korosztály talál kedvenc való játékot, legyen szó akár ingyenes akár fizetős verzióról. Napjainkban igencsak elterjedtek az olyan játékok, ahol a felhasználók interakciókba léphetnek egymással. Ezek túlnyomó többsége az úgynevezett micro-paymentekre alapszik, ahol a játékosok csekély összegekért cserébe előnyökhöz, könnyítésekhez juthatnak. A „free to play, pay to win” kifejezést azokra a játékokra szokták használni, ahol az előbb említett vásárlások nélkül képtelenség megnyerni a játékot, mert túlzottan befolyásolják a játékosok fejlődését.

Egyre több ember életéből hiányzik napjainkban a rendszeres testmozgás. Sokan választják a séta és a bicikli helyett az autós vagy a tömegközlekedést, főleg kényelmi szempontokból. A technológia fejlődésével pedig egyre több olyan eszköz jön létre, amik az emberek életének kényelmesebbé tételeit szolgálja. Ha rendelkezünk okostelefonnal, ma már a nagybevásárlást is el tudjuk intézni pár kattintással otthonról. Az ilyen alkalmazások célja az volt, hogy kényelmesebbé tegyék minden napjainkat, nem az, hogy elkényelmesítsenek minket. Az, hogy egyre több időt töltünk ezen eszközök előtt, csak súlyosbítja a rendszeres testmozgás hiányát. Sajnálatos módon a felhasználók nagy részét nem motiválja önmagában elégé az, hogy a mozgás jót tesz az egészségnak, szükség lehet valamelyen fajta ösztönző módszerre. Ezek többsége módon is megnyilvánulhatnak, léteznek különböző virtuális díjazások, sportolásért járó pontok, amik később tárgynyereményekre válthatóak, sőt vannak tényleges pénzzel való díjazások is.

Célom a két trend összekapcsolása oly módon, hogy a játékban a gyorsabb fejlődést nem pénzkifizetéssel, hanem sportolással lehessen kiváltani.

A 2. fejezetben ismertetem az elkészült játék hátterét, bemutatok különböző játéktípusokat.

A 3. fejezetben ismertetem a program tervezett funkcionalitását és követelményeit. Illetve kitérek a játék egy fontos alapelemére az általam választott dream-iso-droid játékmotorra.

A 4. fejezetben bemutatom az elkészült alkalmazást felhasználói szemmel.

Az 5. fejezetben részletesen kifejtem a megvalósított játék fő alkotóelemeit, és azok implementációját.

Végül a 6. fejezetben bemutatok pár lehetséges továbbfejlesztési ötletet, melyeket érdemes lenne megvalósítani.

## **2. fejezet**

# **Hasonló fejlesztések, szerepjátékokról általánosan**

A videójátékok által kínált szórakozás napjainkban nagy népszerűségnek örvend, akár a konzolokra készült játékokról, akár a számítógépes platformra készültekéről van szó. Azonban nem volt mindig ilyen populáris, fejlődése a technológia előrehaladásával valósulhatott meg. A következő fejezetben bemutatom hogyan is változtak a videójátékok az idő múlásával. Ezt követően ismertetek néhány játékot, amik jelenleg a piacon vannak és hasonló célkitűzéssel születtek meg, mint az általam fejlesztett alkalmazás: az emberek mozgásra ösztönzése miatt. Végül pedig áttekintést adok az RPG-ről, mint játékstílusról. Célom, hogy betekintést adjak az említett műfajról, illetve hogy miért erre a műfajra esett a választásom a játék elkészítésekor.

### **2.1. A videójátékok története**

Videójátékoknak nevezük azokat a típusú játékokat, ahol a játékosok egy felhasználói felületen keresztül lépnek interakcióba a játékkal. Története az 1950-es évekre vezethető vissza, ebben az évtizedben kezdte el foglalkoztatni az embereket az az ötlet, hogy a szórakozás élményét elektromos eszközök segítségével érjék el. Az első videójáték egy asztalitenisz szimulátor volt, 1958-ban készítette el William Higinbotham és a „Tennis for Two” nevet viselte, mely a 2.1. ábrán látható.

A játék egy asztali analóg számítógépre készült el, és egy oszcilloszkópot használt a megjelenítésre. Az első olyan játék, amelyet már egy kontroller űréseket nevezhető eszközzel irányítottak 1962-ben készült el, és a Spacewar nevet kapta. Itt két űrhajót irányítottak a játékosok, a cél pedig a másik fél letorpedázása volt.



2.1. ábra. Tennis for two - első videójáték

A Spacewar változataként jelent meg a világ első pénzbedobós játéktermi gépe a Computer Space. Ennek ellenére ez a játék feledésbe merült, a népszerűséget az 1972-ben megjelent PONG játék szerezte meg. A játék annyira népszerű volt, hogy 3 év múlva az otthoni verziója is elkészült a játéknak. Ennek a sikernek köszönhető, hogy megindult az otthoni konzolok fejlesztése. Az emberek korábban játéktermekbe jártak ha szórakozni vágytak, viszont ettől kezdve rá tudták kötni a TV-re is otthon és onnan élvezni a játékot.



2.2. ábra. Donkey Kong játék egy jelenete

Ezt követően megindult a játéktermi gépek térhódítása a világon. Az Atari mellett a Sega és a Midway cégek is belekezdték a saját játékgépeik fejlesztésébe. Ekkoriban mindenki be akart szállni a játékgyártásba, emiatt gyorsan sok videójátékot termeltek ki a vállalatok. Az

1980-as évek elején a sok silány minőségű játék miatt halottnak nyilvánították a videójáték piacot, mert a emberek nem kívántak rossz minőségű játékokat venni, ennek következtében csökkent a keresletük. Az első olyan videójátékot, ami történetet mesél el a Nintendo cégnak köszönhetjük. A játék az 1981-ben kiadott Donkey Kong volt, ahol a ma is híres Mario (akkori nevén Jumpman) karakterével kellett megmenteni a bajbajutott lányt, a játékból egy jelenet a 2.2 ábrán tekinthető meg.

1985-től a videójáték piac újból felemelkedett a Nintendo Entertainment System nevű otthoni játékkonzolnak köszönhetően. Egymás után jelentek meg különböző vállalatok játékai, ezek között találhatóak voltak stratégiai, verekedős továbbá kalandozós játékok is. Az 1990-es évektől kezdve folyamatosan jelentek meg a felfejlesztett játékkonzolok. A hozzájuk készült játékok nem csak a fajtájukban térnek el, hanem különböző megjelenésükkel és irányítási rendszerükkel egyedivé varázsolják a játékélményt. A közeljövőben elérhetővé válik majd az a játékforma, amikor a játékos érzékeit becsapják a képek és a hangok, és teljesen úgy fogja érezni, mintha belecsöppent volna a játék világába.

A játékok egy bizonyos fajtáját, a mobiljátékokat úgy definiálhatjuk, hogy hordozható telekommunikációs platformra készült játékok. Az első ilyen játék nem is igazán mobilra készült, hanem egy számológépen futott. A játék maga pedig egy Snake nevű program volt, ahol egy kígyót kellett irányítani és minél hosszabbra növeszteni azzal, hogy a képernyőn megjelent falatokat megetetjük vele. A Gameloft játékfejlesztő cég volt az első, aki nem volt mobiltelefongyártóhoz kötve, hanem az önálló játékfejlesztést tűzte ki céljául. A 2000-es évek elején megjelentek a színes kijelzővel rendelkező telefonok, amik új lehetőségeket biztosítottak a játékfejlesztőknek. Ekkoriban a játékok minőségét főleg a kijelző nagysága korlátozta. A játékok grafikai javulása miatt nőtt a méretük, amíg az első telefonos játékok 60 kilobájt körül mozogtak, addig 2007-re méretük elérte a 600 kilobájtot is.

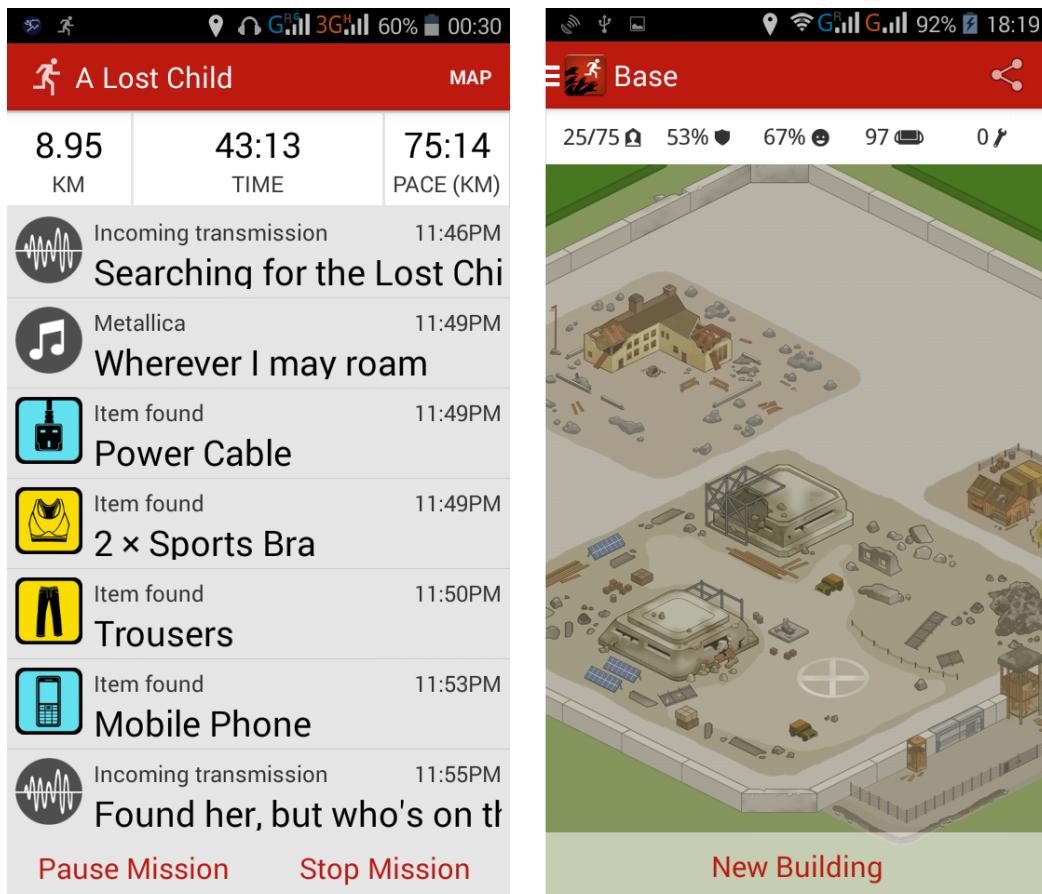
A következő fordulópontot az első okostelefon megjelenése jelentette, nemcsak mérete, de érintőképernyős kezelőfelülete miatt is. A mobiltelefonok fejlődésével a játékok is fejlődésnek indultak. A mai okostelefonra készült játékok egy része vetekszik a számítógépre készült játékok színvonalával. Léteznek olyan mobilok amelyek kifejezetten játékok gyakori használatára terveztek, illetve elérhető már olyan kontroller, amit mobileszközökre lehet csatlakoztatni és igazi konzolos játékélményt nyújt a felhasználóknak.

## 2.2. Sportjátékok

Az ötlet, hogy a játékokat és a sportokat össze lehet kapcsolni már egy ideje létezik, ezeket a játékokat "exergames" névvel illetik, a gyakorlat és a játék szavak összeolvásztásából. Léteznek konzolos játékok is amelyek a mozgáskövető rendszerük segítségével állapítják meg, hogy a játékos helyesen végzi e a gyakorlatokat. Továbbá léteznek mobilos alkalmazások is, ezek

főleg ösztönzésre vagy sportolás közbeni szórakoztatásra fókusznak. Ezek közül mutatok be pár sikeresebb példát, amelyek jelenleg a piacon találhatóak.

### 2.2.1. Zombies, Run!

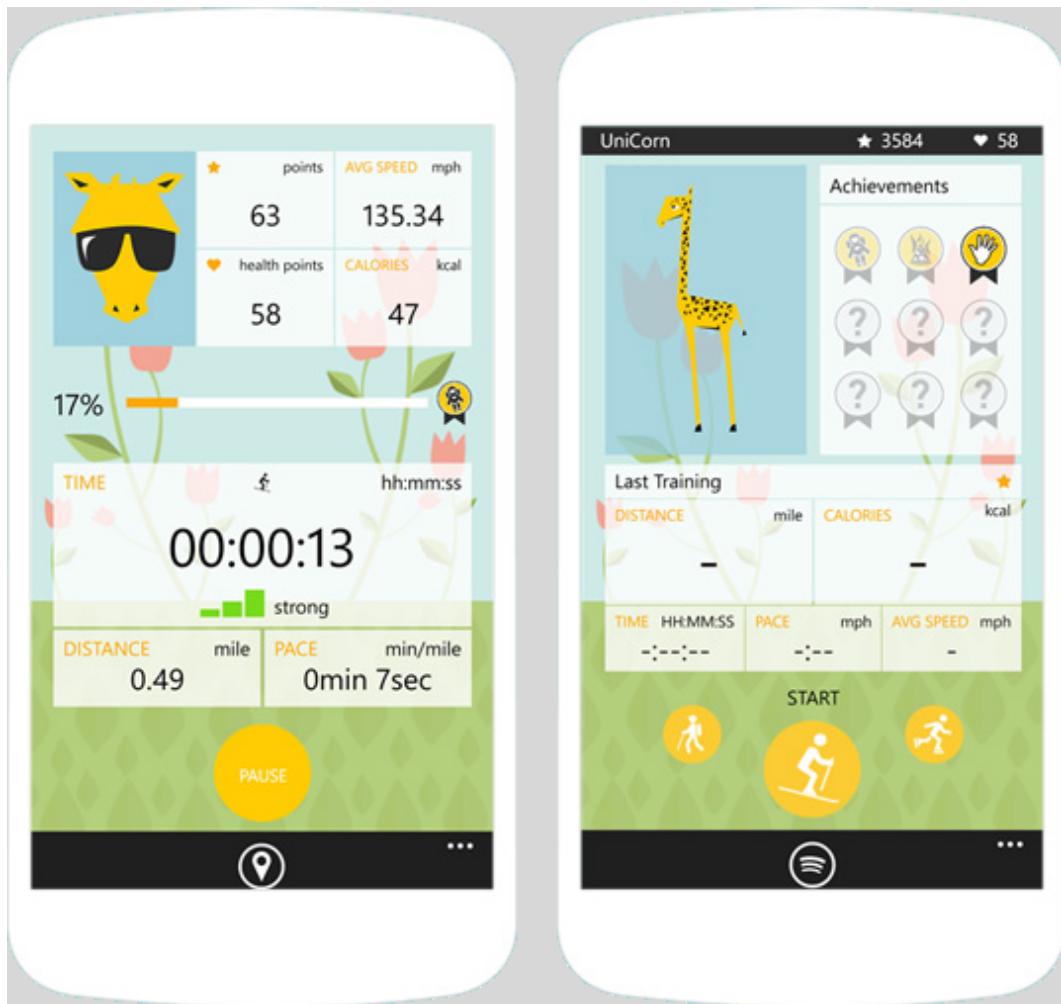


2.3. ábra. A Zombies, Run! felülete

A Zombies, Run! egy népszerű futó alkalmazás, ahol a felhasználó egy zombi-apokalipszisbe csöppen bele túlélőként. A játék sikerét mutatja, hogy Android felületen fél millió felhasználó használja jelenleg [1]. A játékos feladata, hogy minél több zsákmányt szerezzen futás közben, amivel egy bázist kell folyamatosan fejlesztenie, hogy az ott állomásoszt túlélőket biztonságban tudhassa. A megszerzett zsákmányokról és az építendő bázisról demonstráló képeket a 2.3 ábrán látható. A felhasználó kezdetben elindítja az alkalmazást, majd a futás közben elkezdődik a játék történetének mesélése. Ez nem folyamatos, a felhasználó tud közben zenét hallgatni. Futás közben találkozhatunk zombikkal, melyeket vagy a futás közben talált ellátmányért cserébe rázhatunk le, vagy egy rövid ideig 20%-kal gyorsabban kell futnunk. Ez a folyamatos váltakozás a futás közben egyfajta intervallum edzésnek felel meg, melynek igen

sok előnyös tulajdonsága van. Az alkalmazás önmagában egy sporttracker, ami szenzorok segítségével követi nyomon a felhasználó sportolási tevékenységét.

### 2.2.2. Tep

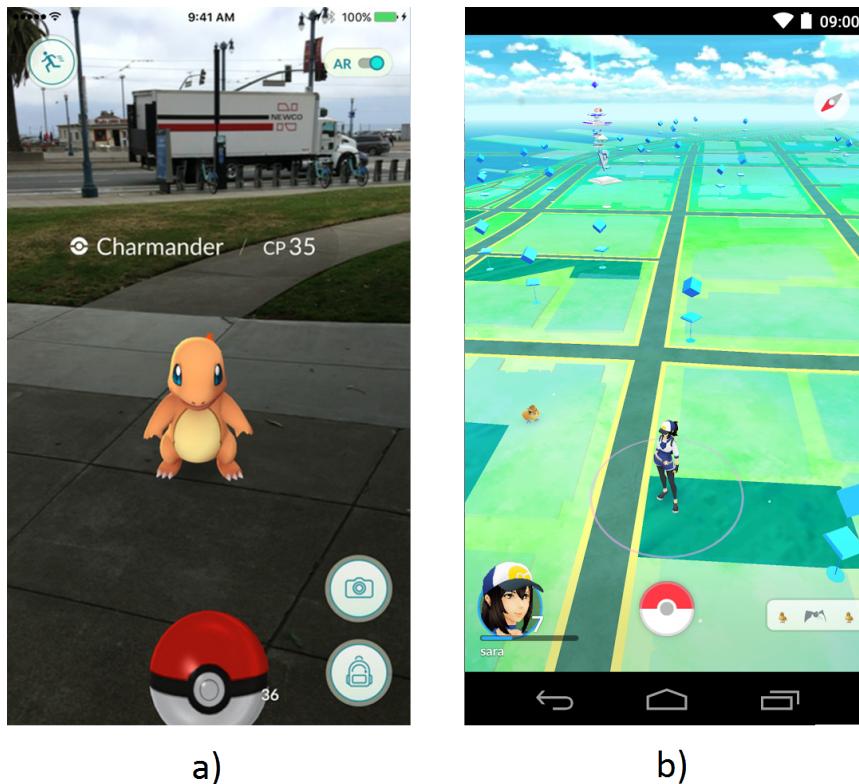


2.4. ábra. Tep játék felülete

Meg kell továbbá említeni a Tep-et, amely magyar fejlesztésű, a játékról pillanatfelvételeket a 2.4 ábrán láthatunk. A Tep szintén motivációs sport-nyomkövető alkalmazás, mely a valós teljesítmények után ad jutalmat a játékban. A játék stílusa a népszerű Tamagotchi játékhoz hasonló, azaz egy virtuális állatkát kell gondoznunk minden naposan. A kapott jutalmakat beválthatjuk a virtuális állatunk részére különböző étel, ital és dekoratív elemre is. Ez az alkalmazás is különálló sport-nyomkövetőként működik, azaz nem külső forrásból szerzi be az adatokat, ugyanakkor össze lehet kötni hordozható eszközökkel, a Fitbit-tel és a Jawbone eszközökkel. Annak ellenére, hogy a játék motivációs célt szolgál, a felhasználó kevésbé van ösztönözve a sportolásra, ugyanis ha nem sportol folyamatosan, az egyetlen változás, ami

bekövetkezik, hogy ha az állatkát "simogatjuk", akkor nem csóválja a farkát és éhezik.

### 2.2.3. Pokémon Go



2.5. ábra. Pokémon GO játékelemei

A Pokémon Go az RPG játékok egy speciális fajtába az MMORPG-be tartozik. A játék kizárolagosan csak mobil eszközre készült abból az okból kifolyólag, hogy közvetett vagy közvetlen módon sportolásra vagy legalább mozgásra ösztönözze az embereket. Emiatt szükség volt arra, hogy az eszköz, amin játsszanak, mobilis legyen. Mindezek mellett a játék félén a valóságban félén pedig a virtuális világban játszódik. A 2.5. ábra b) képén látható módon a pontos pozíciókat megjeleníti a térképen, ami a valós világ útjaira, épületeire alapszik. Annyiban viszont eltér, hogy a játék egyes elemeit például a pokémonokat (kitalált állatszerű lények) a virtuális világban a térképre helyezi, majd a felhasználónak a való világban fizikailag oda kell jutnia hozzá, hogy elkapphassa, amely a 2.5. ábra a) képén látható.

## 2.3. Az RPG játékokról általánosságban

A következő alfejezetben az RPG-t (role playing game), vagyis szerepjátékot fogom bemutatni. Ezek a fajta játékok arra épülnek, hogy a felhasználó egy karakter "szerepébe" bújik bele,

őt irányítva végzi el a feladatokat, kalandozik a világban. Eredete az ókorba vezethető vissza, elődjének tekinthetők a különböző harci játékok amelyek az ütközetek szimulálására szolgáltak. Az első szerepjáték az 1974-ben megjelent Dungeons & Dragons volt, ami hasonló szabályrendszerrel rendelkezett, mint a napjainkban megjelenő RPG-k. Előnyük, hogy sok ember számára elérhetők, ugyanis a játékhoz dobókockákra, papírra és képzőerőre van szükség. A mesélőnek kinevezett személy vezeti végig a kalandozáson a többi szereplőt. minden játékoshoz tartozik egy karakter, aki fölött rendelkezhet, illetve a karakterlapján vezetheti a statisztikákat és jellemzőket, amint a 2.6. ábrán is látható. Egy elvégzett feladat vagy küldetés után különböző jutalmakat kaphatnak a karakterek, amelyek fejlődésük során egyre erősebbek lesznek, emiatt pedig sikerül elérniük a játék elején kitűzött céljukat.



2.6. ábra. Dungeons & Dragons karakterlap

Nem kellett sok idő ahhoz, hogy az RPG meghódítsa a számítógépes közeget is. Az 1970-es évek közepe után sorra jelentek meg a többfelhasználós kalandjátékok, amik a szerepjátékok szabályait követve nyújtottak szórakozási lehetőséget azoknak, akik rendelkeztek internetkapcsolattal. Ennek a mintájára napjainkban már nem csak webes felületen érhetők el a hasonló típusú játékok, hanem az okostelefonok terjedésével, már mobil felületen is. A grafikai kártyák fejlődése következében az utóbbi két játékforma grafikai elemek felhasználásával szimulálja a

különböző akciókat, amelyek a régi típusú szerepjátékban a képzeletre voltak bízva. Továbbá egy jelentős különbség még, hogy míg az eredeti szerepjátékokban a harcok kimenetele többnyire a szerencsén múlik, addig az online játékoknál különböző képletek és algoritmusok segítségével számolják ki, egy egy támadás mértékét. Feltehetőleg a komplexebb harcrendszer miatt alakult ki több változata a küzdelem lebonyolításának. Egy népszerű formája a "turn based" vagyis körökre osztott összecsapás. A játékos karaktere és az ellenfél felváltva támad, minden fél a saját körében dönt arról, hogy milyen cselekvést fog végrehajtani. Ez az akció lehet támadás, öngógyítás vagy esetleg menekülési kísérlet. Ezen kívül vannak szimulált harcot implementáló játékok, ahol a harc kimenetele időközben nem befolyásolható. Itt az összecsapás az ellenfél és a saját karakter tulajdonságainak a felhasználásával kerül kiszámításra. A játékos a végeredményt látja csak, hogy sikerült-e legyőzni az ellenséget vagy sem.

Továbbá különbözetet tehetünk abban, hogy az online felületen játszható játékok hosszú távon tudnak szórakozási lehetőséget biztosítani, nem szükséges a játékosok folyamatos jelenléte. Az online szerepjátékokban jellemzően nincs kitűzött végcél, a felhasználók kalandoznak, fejlődnek és egyre erősödő ellenfeleket győznek le. Ha a játékban nincsenek maximálisan elérhető értékek definiálva, akkor csak a játékos kitartása és eltökéltsége szab határt a játék végének. A cél egy olyan játék megvalósítása volt, ami hosszú távon képes ösztönözni a felhasználót a sportolásra.

### 3. fejezet

## Követelmények, technológiák

A fejezetben szó esik a szoftverrel szemben támasztott követelményekről, valamint a felhasznált programok, technológiák kerülnek bemutatásra.

### 3.1. Követelmények

Az alkalmazással szemben különféle követelményeket támasztottam, melyeknek mindenképp meg kellett felelnie, ezek az alábbiak:

**Kiterjeszthetőség** A játék alapköve, hogy a különböző testmozgásokat különböző pozitív jutalmakkal díjazza. A sportolási tevékenységeket különböző sport-trackerekkel tudjuk mérni. Az alkalmazásban ezt kétféle módszerrel lehet megvalósítani, egy hasonló működésű modult hozok létre, amely különböző szenzorok segítségével méri a sporttevékenységeket (GPS, gyorsulásmérő) vagy már meglévő szolgáltatásuktól szerezzük be ezeket az adatokat. Az utóbbi megoldás előnye, hogy a népszerűbb sport-tracker alkalmazások felhasználóbázisa milliós nagyságrendű, így rengeteg potenciális felhasználó számára nyílik lehetőség a játékba való kapcsolódáshoz. Számos ilyen szolgáltatással találkozhatunk, elvárás volt a játékkal szemben, hogy minél több tracker integrálható legyen, és a legismertebbek közül legalább kettő meg is legyen valósítva.

**Jutalmak** A felhasználó számára elvégzett sportteljesítményei alapján jutalmakat kell kapnia, melyeket a játék közben felhasználhat.

**Érdeklődés fenntartása** Olyan játékmenetet kell kialakítani, amely hosszabb távon is leköti a játékos figyelmét. Ezt elérendő fokozatosan nehezedő területeket kell a felhasználó számára kínálni, mely ösztönzi a továbbhaladásra.

**Kis erőforrás igény** A játéknak alacsony erőforrás mellett is megfelelően kell működnie, hogy az esetleges régebbi készülékeken is kielégítő játékélményt nyújtson.

## Funkcionális követelmények

Az alábbiakban a főbb funkcionális követelmények kerülnek bemutatásra.

- A minél nagyobb számú támogatottság elérése érdekében úgy kell kialakítani az alkalmazást, hogy a későbbiekkben könnyedén lehessen integrálni különböző sport-tracker alkalmazásokat.
- Csatlakozás után az alkalmazásnak le kell töltenie a felhasználó legújabb sport tevékenységeit. Erőforrás takarékkosság szempontjából először meg kell bizonyosodni, hogy van-e új tevékenység. Törekedni kell, hogy a felhasználóhoz tartozó összes adatot csak az első csatlakozás alkalmával, vagy más eszközön való bejelentkezés esetén töltük le.
- A különböző integrált alkalmazások adatainak tárolására létre kell hozni egy egységes adatszerkezetet, így elkerülve az inkonzisztens adatokat.
- Bejelentkezés után az újonnan letöltött adatok alapján a felhasználó sztaminát (kittartást) kap. Egy játékosnak maximum 100 sztaminája lehet. A kapott sztamina mennyisége összhangban kell lennie ezzel a maximális értékkel, a játékos szintjével, és a tevékenységen szereplő adatok nagyságával. Vagyis az alacsony és magas szintű felhasználóknak is egyaránt élvezetessnek kell maradnia a játéknak, nem szabad se túl sokat, se túl keveset kapniuk. Túl sok sztamina esetén nagyon könnyen haladhatna a felhasználó a játékban, így egy idő után beleunna, túl kevés esetén viszont a folyamatosan túl nagy kihívást, és nagy megerőltetést jelentő tevékenységek szintén ugyanezt a hatást érnék el.
- A jutalomként megkapott sztaminát a felhasználó a játékosa fejlődésére használhatja fel különböző módokon. Az egyik ilyen mód a világban való "barangolás", ami közben szörnyek támadhatnak a játékosra, amelyeket legyőzve játékbeli pénzt és tapasztalati pontot kap a játékos.
- A játékosnak ezen kívül rendelkeznie kell tulajdonságokkal is, melyek a szörnyek elleni csatában segíthetnek számára. Tulajdonságot növelni szintlépéssel képes a játékos.
- További tárgyat is érdemes lenne megvalósítani a játékos számára, melyek védelemmel vagy támadóerővel növelhetnék a játékos erejét.

### 3.2. Felhasznált technológiák

#### 3.2.1. Játékmotorok

Játékmotornak nevezzük a játékok - legyen az akár számítógépre vagy konzolra készült – azon részét, amely a program alapjául szolgáló technológiát adja. Szerepe, hogy megkönnyítse a

fejlesztést illetve segítségével több platformon is futtatható lesz a játék. Egy játék elkészítése az alapktól nagyon nehéz, erőforrás-igényes feladat. Hamar világossá vált hogy szükség van olyan eszközökre, amelyek támogatják egy játék alapvető funkcióinak gyors implementálását, mint a megjelenítés és felhasználó input kezelése, hiszen ezek a legtöbb játék esetében nagy hasonlóságot mutatnak. Ezen funkciók megvalósítása után történhet az elkészülendő játék sajátosságainak kialakítása.

A fejlesztés megkezdése előtt több fajta játékmotort is megvizsgáltam abból a célból, hogy kiválasszam a legmegfelelőbbet a diplomamunkám elkészítéséhez.

A fő szempontom az volt, hogy ingyenesen elérhető legyen, illetve illeszkedjen a választott játéktípus játékmenetéhez.

A két legnépszerűbb motorral kezdtem az ismerkedést, a Unity és az Unreal engine-ekkel. Mivel a programomat Android platformon terveztem elkészíteni, amit Java nyelven kell implementálni, ezek a motorok pedig a C++ nyelvet támogatják, így nem lehet közvetlenül Java nyelven használni őket ezért hamar kiestek. Méretük alapján túl nagynak is bizonyultak volna egy ilyen kisebb méretű projekthez. A következő játékmotor, amit megvizsgáltam a HexEngine volt, amit Szabó László készített el MSc diplomamunkájaként. A motor előnye, hogy rengeteg hasznos funkciót támogat szerepjátékok elkészítéséhez, viszont a játéktér hatszögű blokkokra van osztva, amivel megbonyolította volna a közlekedést a játékon belül. A választásom így Hollósi Tamás által készített dream-iso-droid játékmotorra esett, amit témavezetőm ismertetett meg velem. [2] Mivel készítője elérhető közelégen volt, ezért könnyebben sikerült megismerkednem a motor nyújtotta funkciókkal.

	<b>Unity</b>	<b>Unreal</b>	<b>HexEngine</b>	<b>dream-iso-droid</b>
<i>nézet</i>	3D	3D	2D izometrikus	2D izometrikus
<i>platform</i>	Több	Több	Több	Android
<i>nyelv</i>	C++, C#	C++, UnrealScript	C++	Java

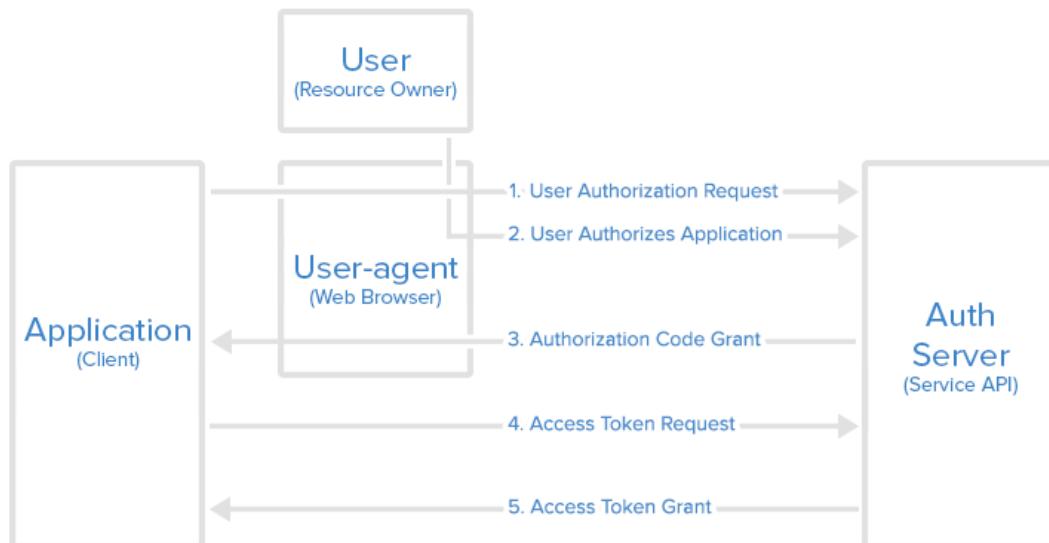
3.1. ábra. Játékmotorok összehasonlítása

A dream-iso-droid egy olyan speciális játékmotor, ami kifejezetten Android platformra készült és a két dimenziós izometrikus nézetet támogatja. Ez a két funkciója pontosan megfelelt az elvárásaimnak, amit a játékmotor felé támasztottam, aminek segítségével fejleszteni szerettem volna az RPG játékomat. A 3.1. ábrán látható egy összefoglaló táblázat, melyben megtekinthetők a megvizsgált játékmotorok azon tulajdonságai, melyeket figyelembe vettet a választás során.

### 3.2.2. Fejlesztőkörnyezet

A megvalósítás során az Android Studio fejlesztőkörnyezetet használtam, melyet az Android operációs rendszer fejlesztője, a Google készített el. Integrálva van benne minden olyan szolgáltatás, mely nagyban megkönnyíti a fejlesztők munkáját, mint például a Gradle keretrendszer, amely többek közt a projekt építéséért és a különböző külső függőségekért felelős. Továbbiakban található benne grafikus felületszerkesztő is, ahol drop&down módszer segítségével a grafikus felület szerkezetét könnyen össze tudjuk rakni. Integrálva van továbbá több verziókövető is, így a fejlesztőkörnyezet elhagyása nélkül tudjuk az újabb verziójú fájlokat a megfelelő távoli tárolóoldalra eljuttatni.

### 3.2.3. OAuth protokoll



3.2. ábra. Az OAuth protokoll absztrakt működési ábrája

Az OAuth protokoll [3] egy nyílt autorizációs szabvány mely segítségével a felhasználók megoszthatják bizonyos privát információit anélkül, hogy azonosítási adataikat kiadnák. A 3.2. ábrán látható a protokoll működési folyamata. Ha egy alkalmazás el akar érni olyan adatot ami bizalmasan van kezelve, ahoz előbb engedélyt kell kérnie hozzá. Az alkalmazás továbbirányítja a felhasználót az adott szolgáltatás felületére - többnyire valamilyen webböngészőbe -, ahol engedélyt tud adni számára. Ekkor a felhasználónak be kell jelentkeznie a fiókjába, és megadni a kért engedélyeket. Az engedély megadása utána a hitelesítő szerver egy megerősítő kódot juttat el az alkalmazás számára. Az alkalmazás ezt a megerősítő kódot tudja "elcserélni" a szerverrel egy tokenre. A későbbi adatelérés alkalmával minden kéréshez

csatolnia kell az alkalmazásnak ezt a tokent. A szerver ezt a tokent vizsgálva tudja eldönteni, hogy a kért információhoz a felhasználó engedélyt adott-e.

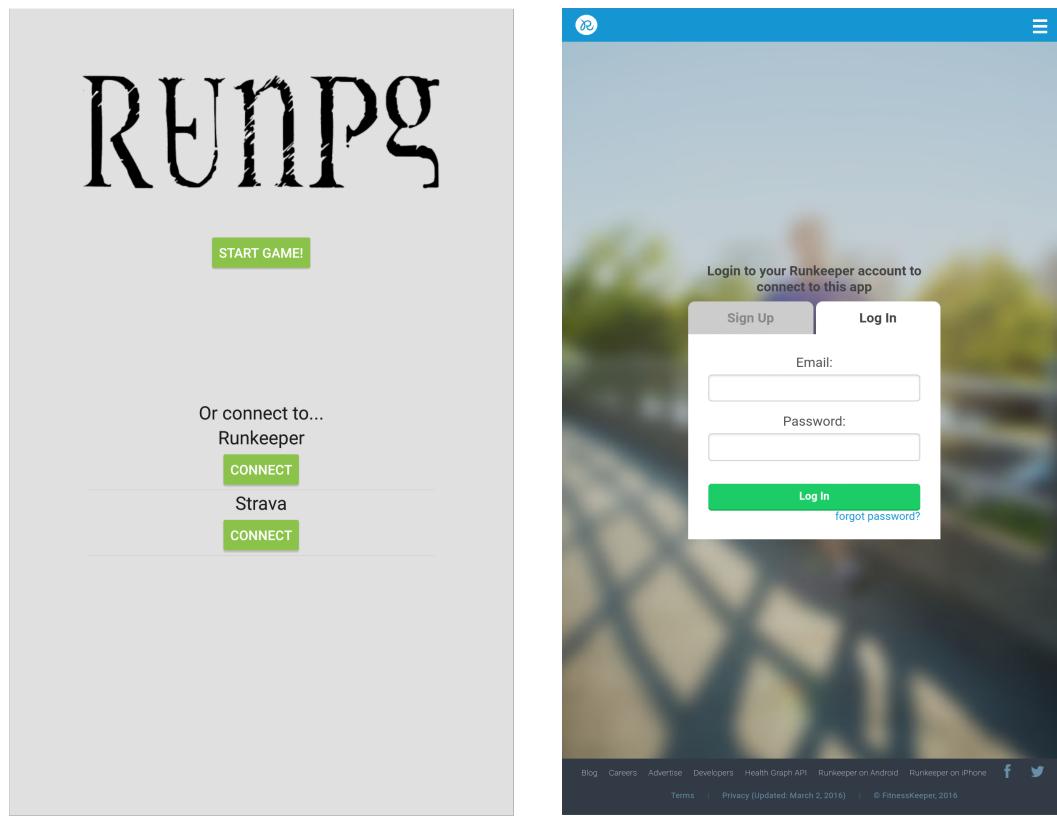
### **3.2.4. OrmLite**

Az alkalmazáson belül az adatok hosszútávú tárolására az OrmLite könyvtárat [4] használtam, amely képes Java objektumokat az alkalmazás helyi SQLite adatbázisába írni. Több platformon is elérhető, az Androidos készülékeken natív API hívásokkal kezeli az adatbázist. Használata könnyű, egyszerű Java osztályokat kell a megfelelő annotációkkal ellátni, majd az annotációk alapján a könyvtár képes a kívánt módon kiolvasni és eltárolni az adatokat. A könyvtár a DAO, azaz Data Access Object tervezési mintát használja, amely elkülöníti az alacsony szintű API hívásokat a magasabb szintű szolgáltatásoktól.

## 4. fejezet

# Felhasználói kézikönyv

A fejezet során bemutatásra kerül az alkalmazás felhasználói szemszögből: milyen felületekkel találkozhat a felhasználó, ezeken milyen akciókat végezhet, valamint bemutatásra kerülnek a játékmechanikai elemek is.



4.1. ábra. A főmenü és a Runkeeper csatlakozási felület

## **4.1. Kezdőképernyő**

A játék telepítése után a többi Androidos alkalmazás közül választható ki futtatásra. Az indulás után a felhasználó a kezdőképernyőn találja magát, ahol kétféle funkciót érhet el: csatlakozhat valamelyen meglévő sport-tracker fiókjához vagy elkezdheti a játékot. A menü tetején látható a játék logója, ez alatt található a játék indítására szolgáló gomb, ahogy a 4.1 ábra a) képe is mutatja. A képernyő alsó részén egy listában tárolódnak a csatlakoztatható sport-trackerek, soronként az adott sport-tracker neve, és egy gomb, amire kattintva elkezdődhet a csatlakozási folyamat. A csatlakozási folyamat során a kiválasztott sport-tracker szolgáltatás online felületére kerül át a felhasználó, a Runkeeper szolgáltatáshoz tartozó csatlakozási felület a 4.1 ábra b) képén tekinthető meg.

Miután a felhasználó megadta a belépési adatait és belépett a fiókjába, az alkalmazás számára meg kell adnia a kért engedélyeket. Miután megadta ezeket az engedélyeket, visszanavigálódik a főmenübe, ahol az eddig kattintható gomb - amivel a játékot lehet indítani - nem kattintható tovább, továbbá az alsó listából nem található meg a csatlakoztatott szolgáltatás. A játék felugró értesítés formájában jelzi a felhasználó felé, hogy elkezdte letölteni a szolgáltatástól az eddigi adatait. Miután a letöltés befejeződött - amit szintén egy felugró értesítés formájában hozunk a felhasználó tudtára - a játék kezdetét jelentő indító gomb aktívvá válik.

A felhasználó a játékot anélkül is elkezdheti, hogy bármelyik sport-tracker alkalmazást csatlakoztatta volna a játékhoz, de a későbbiekben ez nagyban hátráltatná a játékban való előrehaladásban. Természetesen a későbbiek során bármikor, amikor elindítja az alkalmazást lehetősége nyílik sport-tracker alkalmazások csatlakoztatására, és amennyiben a játékban már haladást ért el, a csatlakoztatás hatására ez nem veszik el.

Amennyiben a felhasználó korábban csatlakoztatott sport-tracker alkalmazásokat, akkor a játékban való belépés során az alkalmazás a csatlakoztatott sport-trackerektől lekéri a legfrissebb mentett sporttevékenységeket. Miután beszerezte a friss adatokat, megnézi, hogy ezek teljesítették-e a minimális elvárt szintet, és amennyiben igen, jutalmakat sorsol ki a játékos számára, melyeket a játékba való belépés után a hátizsákjában tud megnézni a felhasználó. Továbbá minden új sportteljesítmény után, kritériumok nélkül a játékos számára jóváírásra kerül a sportteljesítménnyel arányos mennyiséggű sztamina is.

## **4.2. Karakter**

A felhasználóhoz egy karakter tartozik, akit a játékban tud fejlődni. A játékos többfajta eszközt is birtokolhat, és különböző varázsitalokkal módosíthatja a tulajdonságait. Kezdetben a karakter szintje 1, életpontjainak száma 110, továbbá a játék folyamán bármikor maximálisan 100 sztaminaponttal rendelkezhet. Számon van tartva, hogy az adott szinten

mennyi tapasztalati pontot gyűjtött össze, és hogy összességében mennyit kell elérnie a szint teljesítéséhez. A karakter gyűjthet virtuális aranyat magának, amelyet a játék során használhat fel. Három alaptulajdonsággal rendelkezik: erővel, kitartással és szerencsével. A karakterhez lehet rendelni egy fegyvert is, amely a játékos sebzését hivatott növelni. A játékos háromféle varázsital közül választhat, ezek az alaptulajdonságokat növelik meg és háromféle méretben kaphatóak.

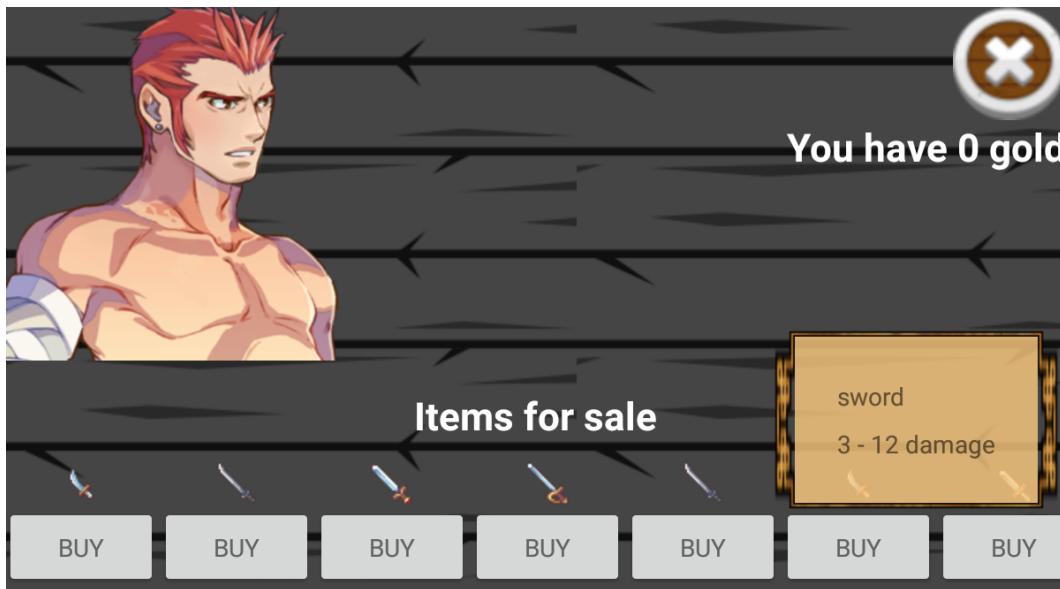
### 4.3. Város



4.2. ábra. A város látképe

Ebben a jelenetben a játékos egy kitalált kis városban közlekedik, és különböző dolgokat végezhet el. Amennyiben az első alkalommal lép be a játékba, vagy amikor bezárta a játékot ezen a jeleneten volt, akkor ide fog kerülni. Játékon belül többféleképp is eljuthatunk ide, a harcmezőn található portálkapun keresztül, ha meghal a karakter, vagy ha az összes ellenfelet legyőzte. A városban több épület is található, továbbá itt helyezkedik el egy portálkapu is, aminek segítségével átjuthat a felhasználó a harcmezőre, ahogyan a 4.2 ábrán is látható. A játékos ezen a jeleneten tetszőleges ideig barangolhat, nincs semmilyen korláthoz kötve. Az itt található épületek közül a kovácsműhelyben tud a felhasználó új fegyvereket szerezni a játékosa számára, illetve a régi, használaton kívüli fegyvereit eladni. A templomba belépve a játékoshoz hozzárendelhető varázsitalokkal lehet kereskedni.

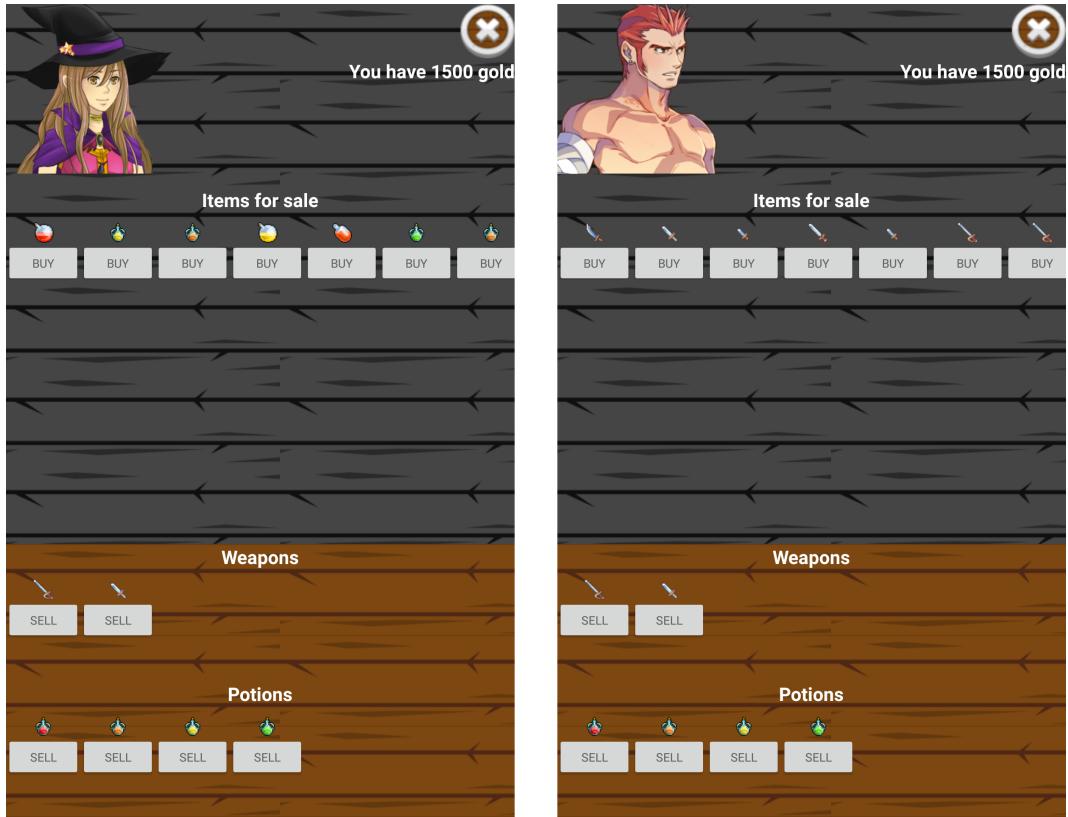
A két bolt belső nézete hasonlít egymáshoz, a boltba belépve a képernyő két részre van bontva. A képernyő alján látható, hogy a játékos jelenleg milyen tárgyat birtokol, a felső részen pedig a bolt által kínált termékek és a boltos arcképe található. Az ablak jobb oldalán látható egy szöveges információs rész, ahol a játékos tulajdonában lévő virtuális arany mennyisége jelenik meg. Mindkét tárgytároló egy vízszintesen mozgó lista, melyben oszlopokként találhatóak a tárgyak, és a hozzájuk tartozó gombok. A gombok szövege annak függvényében változik, hogy a hátizsák elemeihez tartozik, vagy a boltban található tárgyakhoz.



4.3. ábra. Részletes tárgyinformáció

A tárgyat ábrázoló képek hosszú lenyomására egy felugró ablak jelenik meg, amiben a kiválasztott tárgyról kaphatunk részletesebb információkat, melyet a 4.3 ábrán tekinthetünk meg. Amennyiben a hátizsákban szereplő tárgyakhoz tartozó gombot nyomja meg a felhasználó, azt a tárgyat eladja amiért cserébe aranyat kap és egy hely felszabadul a hátizsákjában. A boltban található tárgyak esetén a gombnyomás hatására a kiválasztott tárgyat megveszi a

felhasználó, amennyiben van elegendő aranya rá. A két bolt belső nézete a 4.4 ábrán látható.

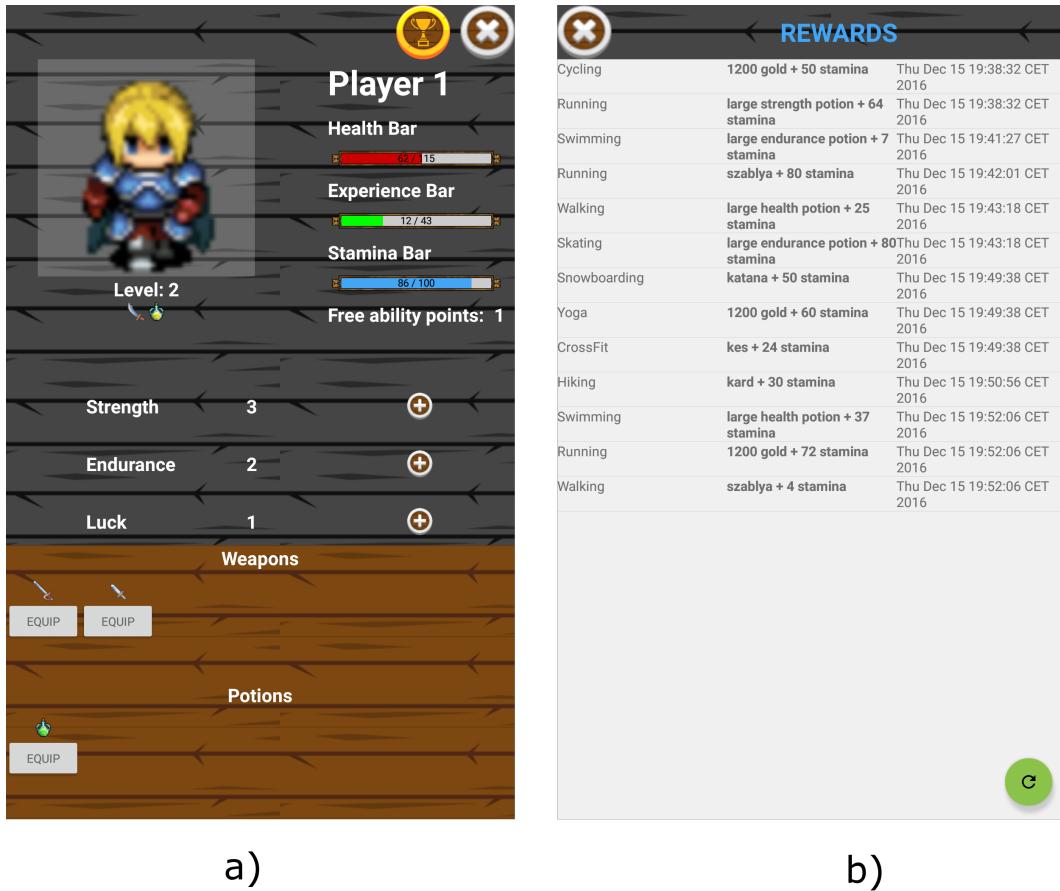


4.4. ábra. Pillanatképek a boltokról

## Karakterlap

Minden jelenet jobb alsó sarkában található a karakterlapot megnyitó gomb. A gomb megnyomása után a karakterlap felülete teljesen eltakarja a játékteret, ahogy a 4.5 ábra a) képén láthatjuk. Itt a játékos tulajdonságait tekinthetjük meg, továbbá itt rendelhetjük hozzá a fegyvereket és varázsitalokat is. A nézet itt is két részre tagolódik, alul található a hátizsák tartalma - akárcsak a boltok esetén - felül pedig a karakter fontosabb tulajdonságait jelző panel jelenik meg. A panel bal felső részében található a karakter képe, alatta pedig az aktuális szintje jelenik meg. A szintjelző szöveg alatt találhatjuk meg azt a tárolót, ahol a karakterhez rendelt fegyver és varázsitalok találhatóak. Amennyiben kicséréljük az aktuális hozzárendelt fegyvert - egy, a hátizsákba rakott másik fegyverre - a felület automatikusan frissül. Ezektől az elemektől jobbra találhatók az életerőt, tapasztalati-pontot és sztamina mennyiséget jelző sávok, melyek mindegyike grafikusan és számokkal is megjeleníti a mennyiségeket. A sávok alatt látható, hogy a karakter mennyi pontot tud elosztani a három alaptulajdonságára. A karakter számára négy elosztható pont íródik jóvá minden szintlépése után. Az előzőekben tárgyalta két panel között található az a felület, ahol ezeket a pontokat lehet elosztani. A

tulajdonságokat külön sorban jelezve, mindegyik sorban jelezve az aktuális nagyságát és egy gomb, amivel - ha van elosztható pont - növelhető az. A képernyő jobb felső sarkában található a karakterlap bezárására alkalmas gomb, és a jutalomlista megnyitására szolgáló gomb. Amennyiben a karakterlapot zárja be a felhasználó, az azt megnyító gomb újra megjelenik és a játéktér újra láthatóvá válik.



4.5. ábra. Karakterlap és jutalomlista

## Jutalomlista

A karakterlapról tudunk idekerülni, és a korábbi karakterlap felülete teljes egészében lecserélődik a jutalomlista nézetére, melyet a 4.5 ábra b) képén tekinthetünk meg. A felületen egy listába vannak rendezve a sporttevékenységek, illetve ezekhez hozzárendelve a sportolásért kapott jutalmak. A lista három oszlobból áll: az első oszlopban a sporttevékenység típusát láthatjuk, a középső oszlopban a kapott jutalom megnevezése és a megszerzett sztamina mennyisége található, míg az utolsó oszlopban a jutalmazás ideje van. A felület jobb alsó sarkában található egy frissítés gomb, mellyel manuálisan ellenőrizhetjük, hogy van-e olyan új sportteljesítmény feljegyezve a csatlakoztatott sport-tracker alkalmazásokban, amelyért

jutalom jár a karakter számára. Amennyiben van ilyen, az a listába automatikusan bekerül. A képernyő bal felső sarkában található a jutalomlistát bezáró gomb, melyre kattintva visszatérünk a karakterlapra. Amennyiben manuálisan frissítettük a listát, és kaptunk új jutalmat, a karakterlapra visszatérve a kapott jutalmat a játékos hátizsákjában találjuk.

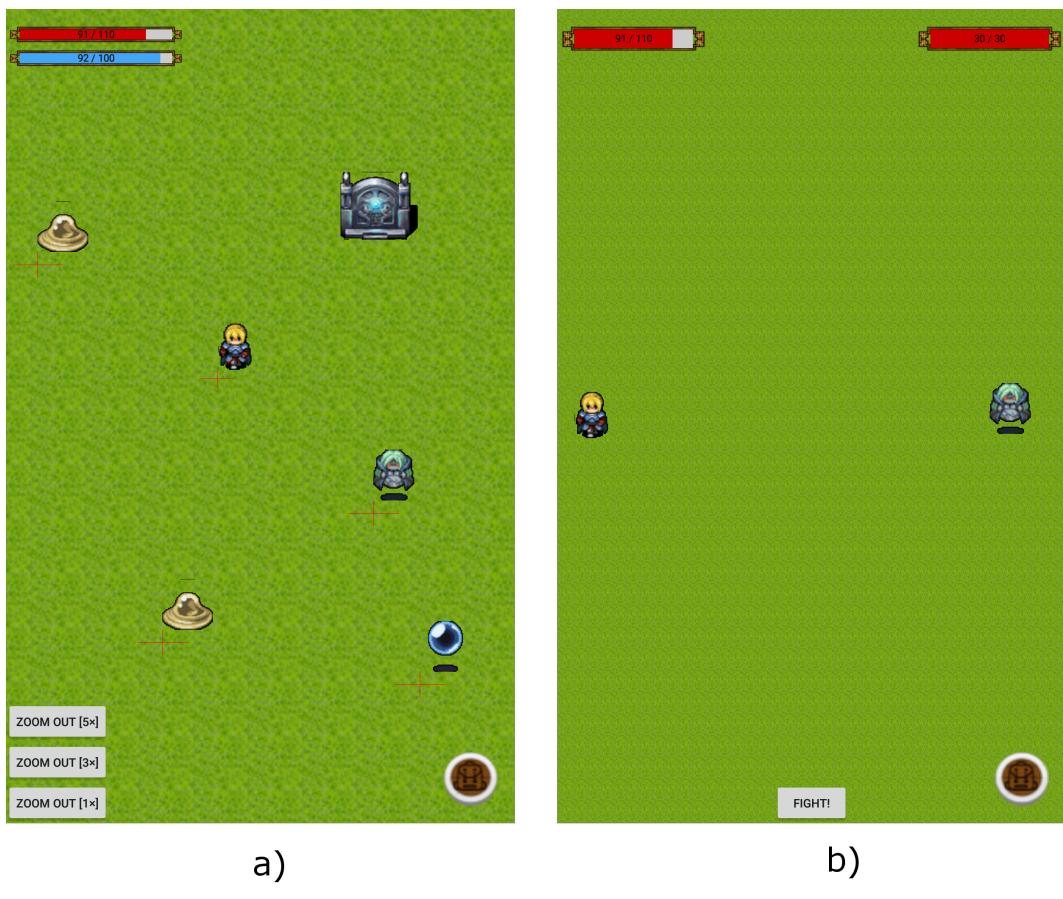
## 4.4. Harcmező

A karakter a harcmezőn találja magát a játék elindítása után azonnal, ha a legutóbbi futás során a harcmezőn tartózkodott az alkalmazás bezárásakor. A játékmenet közben a városban is megtalálható portálkapun keresztül képes idekerülni, és a területet ezen keresztül képes elhagyni. A várossal ellentétben ezen a képernyön a játéktér bal felső részén található két sáv, amelyek a felhasználó életerő- és sztaminaszintjét jelzik. A területen való távolságmegtétel a karakter sztamináját csökkenti, így ha túl sokat barangolt a területen és elfogyott, nem lesz képes továbbhaladni. Mozgás közben a sztaminajelző sáv folyamatosan követi a karakter aktuális sztaminaszintjét, így a felhasználó minden tudatában lesz, hogy hozzávetőlegesen mekkora távolságot képes megtenni a karakterrel. A felület bal alsó sarkában található három gomb, amelyek segítségével csökkenthető a játékterület teljes mérete, így az egy képernyőn nem látható elemek is láthatóvá válnak a felhasználó számára. Különbséget képez még, hogy ezen a területen az épületek helyett szörnyek találhatóak, melyeket le kell győzni. A karaktert és a legyőzendő ellenségeket a 4.6 ábra a) képén tekinthetjük meg. Az ellenféllel való csatához az ellenség közelébe kell érni, és ha megfelelően megközelítette a játékos az adott szörnyet, automatikusan átkerül a harcjelenethez. Ha a harctéren található minden ellenfelet legyőzött, a játékos automatikusan visszakerül a városba, ahol a tartalékeit visszatölve újra elindulhat a harcmezőre, ahol új, erősebb ellenfelek várnak rá.

## 4.5. Harcjelenet

Miután a harctéren közel került a karakter egy szörnyhöz, erre a jelenetre kerül át, ahol csak a harcban résztvevő két fél látható. A harcmezőn található két sáv közül a sztaminaszintet jelző eltűnik - hiszen most erre az információra nincs szüksége a felhasználónak - ehelyett a felület jobb felső részében megjelent az ellenfél életerejét jelző sáv. A felület alsó részén látható továbbá egy új gomb, amellyel a harc szimulálását lehet elindítani. Amíg nincs megnyomva ez a gomb, a felhasználónak ideje van felkészülni a csatára, például a karakterapon új fegyvert rendelhet a karakterhez vagy valamilyen varázsital segítségével növelheti az alaptulajdonságát, vagy visszatöltheti részlegesen a karakter életét. A szimulálás során a felhasználó nem tud közbeavatkozni a csatában azon kívül, hogy a szimuláció során is képes megnyitni a karakterlapot, ahol az előbb leírt lehetőségei vannak a harc során is. Amennyiben a harc

során győztesen kerül ki a karakter, újra a harcmezőre kerül, és folytathatja az ellenségekkel való megmérettetéseket. Ellenkező esetben visszakerül a városba, ahol az életpontjai újra visszatöltődtek, viszont a hozzárendelt összes varázsital eltűnik. A harcjelenetről egy áttekintő kép a 4.6 ábra b) képéén látható.

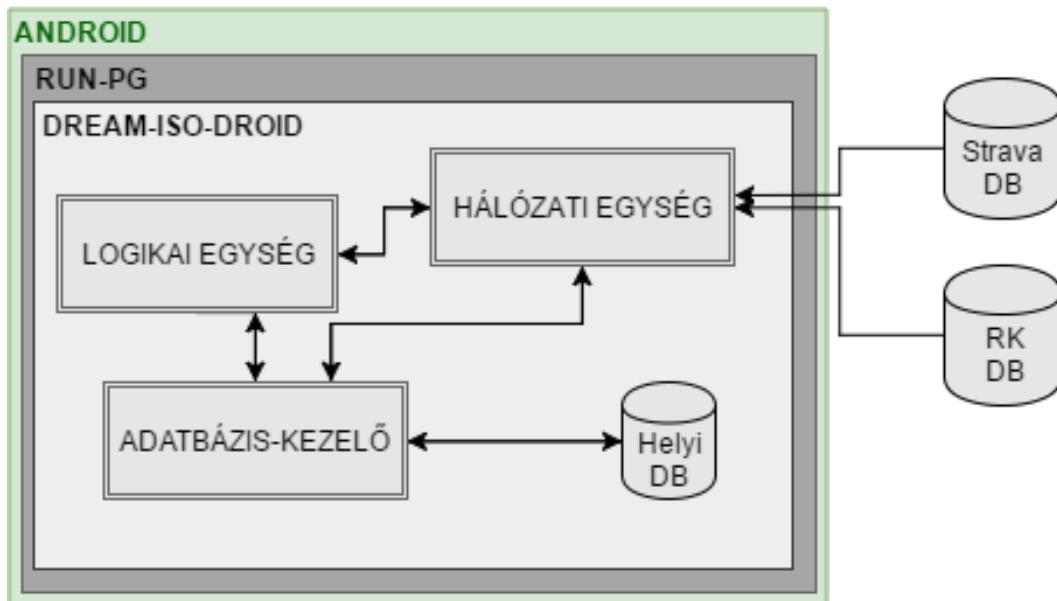


4.6. ábra. Harcmező és harcjelenet

## 5. fejezet

# Fejlesztői dokumentáció

Az 5.1. ábrán látható a játék architektúrája. Két fő részre bontható fel, egy megjelenítő egységre és egy logikai egységre. A megjelenítő egység elkészítéséhez Hollósi Tamás által készített dream-iso-droid nevű keretrendszer használtam fel. A játékban megjelenő grafikai elemek túlnyomó részét ez kezeli. A logikai egység feladata a megtervezett szabályok és játékmechanikai elemek felügyelete. Ez az egység több kisebb modulra bontható fel, melyeknek minden megvan a saját jól elkülöníthető feladata. Az egység ezeket a modulokat kezeli, és a belső működésükbe nem szól bele, elfedve azokat. Elkerülhetetlen, hogy két modul kommunikáljon egymással, ezt is a logikai egység szabályozza. A két egység szoros együttműködéseként valósul meg a játék. A továbbiakban bemutatásra kerülnek a megvalósítás részletei.



5.1. ábra. A program architektúrája

## **Engedélyek, rendszerkövetelmények**

Ahhoz, hogy a játék minden funkcionálitása elérhetővé váljon, néhány rendszerkövetelménynek meg kell felelnie a felhasználó eszközének. Az eszközön legalább 4.4-es Android operációs rendszernek kell futnia, ami a 19-es API szintnek felel meg. Ezt a feltételt a *build.gradle* fájlban adhatjuk meg, ahol más, fontos paramétereket is megadunk. Ebben a fájlban kell megadni többek közt, hogy mely harmadik féltől származó könyvtárakat használ az alkalmazás, és ezen könyvtárak használt verziószámát is.

Az alkalmazásnak csupán egy engedélyre van szüksége, amit az *AndroidManifest.xml* fájlban határozhatunk meg a következő módon:

```
1 <uses-permission android:name="android.permission.INTERNET" />
```

Ezen kívül a fájlban az alkalmazást leíró metaadatok, általános konfigurációs beállítások találhatóak, például hogy melyik legyen az induló Activity.

## **5.1. A játék indítása**

Az alkalmazás indítása után a felhasználót egy egyszerű menü fogadja a *StartActivity* nevű osztályban. Itt két lehetőség tárul a felhasználó elé, képes csatlakozni új sport-nyomkövető szolgáltatásokhoz, vagy elkezdhet játszani. A csatlakoztatható szolgáltatások egy egyszerű listában tárolódnak soronként, ami tartalmazza a nevét, és egy gombot, amely elindítja a csatlakozási folyamatot.

### **Csatlakozás nyomkövető alkalmazáshoz**

Jelenleg a két legnagyobb felhasználói bázissal rendelkező sport-trackerek a Runkeeper és a Strava. Mindkettő rendelkezik publikusan elérhető API-val [5] [6], mely segítségével könnyedén készíthetünk hozzájuk saját alkalmazásokat. Mindkét szolgáltatás esetén ahhoz, hogy a játékkal össze tudja kötni a felhasználó a sport-tracker fiókját, az OAuth protokollra van szükségünk, melyet a 3.2.3 fejezetben mutattam be.

A protokoll teljes működésének implementálása sok biztonsági és hibakezelési kérdést vet fel, így úgy döntöttem, hogy egy már kész könyvtárat használok hozzá.

A választott könyvtáram a ScribeJava [7], mely a protokoll több verzióját is támogatja és számos szolgáltatáshoz már kész API-val rendelkezik. Az általam integrált két sport-nyomkövető nem voltak elkészítve, így ezeket nekem kellett megvalósítanom. Szerencsére a könyvtár úgy lett kialakítva, hogy minden, a protokollt használó szolgáltatáshoz egységesen lehessen API-t készíteni.

A Runkeeper nyomkövetőhöz elkészített API legfontosabb része a következőként néz ki:

```
1 private static final String AUTHORIZATION_URL = "https://runkeeper.com/apps/
  authorize?client_id=%s&response_type=code&redirect_uri=%s";
2 private static final String ACCESS_TOKEN_URL = "https://runkeeper.com/apps/
  token";
3
4 @Override
5 public String getAccessTokenEndpoint() { return ACCESS_TOKEN_URL; }
6
7 @Override
8 public String getAuthorizationUrl(OAuthConfig config) {
9     Preconditions.checkNotNull(config.getCallback(), "Must provide a
  valid url as callback.");
10    final StringBuilder sb = new StringBuilder(String.format(
11        AUTHORIZATION_URL, config.getApiKey(), OAuthEncoder.encode(config.
  getCallback())));
12    ...
13    return sb.toString();
14 }
```

Mivel ez a kommunikáció hálózati tevékenységgel jár, nem történhet az Android fő programszálán. Egyrészt ha a fő szalon folyna ez a kommunikáció, az alkalmazás nem tudna tovább futni, amíg a hitelesítési folyamat be nem fejeződik. Ez akár több másodpercbe is telhet a hálózati körülményeket figyelembe véve, így addig az alkalmazás blokkolódna. A felhasználói élmény miatt ez nem megengedhető, így ezt a folyamatot a háttérben kell elvégezni, hogy az alkalmazás zavartalanul futhasson tovább. Az Android SDK-ban több beépített lehetőség segítségével is meg tudjuk valósítani ezt:

- Egyszerű Java szálak
- AsyncTask
- IntentService

Ezeknek a lehetőségeknek megvan a maguk előnye és hátránya. A normál Java szálak használata széles körben elterjedt, de nagyobb, bonyolultabb programszerkezet mellett használatuk nehézkes. A következő választási lehetőség az Android SDk-ban bemutatott AsyncTask osztály használata. Ennek segítségével könnyedén indíthatunk háttérben futó kódrészleteket. Az osztályon belül felülírhatóak metódusok, amelyek az adott tevékenység elején, közben, vagy a feladata végeztével hívódnak meg. Ez a fajta megoldás sokkal kötetlenebb, viszont ahogy az előző pontban taglalt sima szálhoz hasonlóan, ha az adott Activity, amelyikből el lett indítva háttérbe kerül, az operációs rendszer meg tudja szakítani a folyamat futását. Az *IntentService* esetén ez nem történik meg, tovább egyrészt nincs *Activity*-hez kötve, másrészt a kéréseket egy sorba teszi, amelynek a későbbieknek még fontos szerepe lesz.

A háttérben folyó munkákért a *FetchService* nevű osztály felelős, mely a beépített *IntentService* osztályból származik, és felülírja az *onHandleIntent()* metódust. Ez a metódus már a háttérben

fut. Paraméterként egy Intent-et kap, melyet a *FetchService* osztály statikus metódusain keresztül hozunk létre, így példányosítás nélkül is képesek vagyunk ilyen folyamatot indítani. Egy adott sport-trackerhez tartozó összes eddigi elmentett sportteljesítmény letöltéséhez szükséges Intent elkészítése az alábbi kódrészletben látszódik.

```
1 public static void startFetchActivities(Context context, TrackerService
2     tracker) {
3     Intent intent = new Intent(context, FetchService.class);
4     intent.setAction(ACTION_FETCH_ALL_ACTIVITY);
5     intent.putExtra(EXTRA_TRACKER_SERVICE, tracker);
6     context.startService(intent);
}
```

Ez a metódus egy sport-tracker alkalmazáshoz való sikeres csatlakozás esetén automatikus meghívódik, második paramétere az a TrackerService objektum, amelyhez csatlakoztunk. A beállított akció alapján tudjuk majd meghatározni az *onHandleIntent()* metódusban, hogy milyen típusú műveletet kell végrehajtani.

Mivel az itt futó kód futási ideje nemdeterminisztikus, előre nem tudhatjuk hogy mikor ér véget, hisz nagyban függ az elérhető hálózati csatlakozások jelerőssége-től, így nem tudjuk mikor engedhetjük tovább a játékost a játékkelületre. Ahhoz, hogy valahogy tudjuk mikor ért véget az adatok letöltése, az *IntentService* esetén rendelkezésre áll a *BroadcastReceiver* osztály. Amint az összes letöltés befejeződött ezt el kell juttatnunk az *Activity* számára, ahol a gombot megnyomtuk. Az üzenet létrehozása során egy új Intent-et hozunk létre speciális akció típussal, amelyet broadcast üzenet formájában kiküldünk. Az ilyen broadcast üzenetekre feliratkozhatnak a különböző activity-k, továbbá az is lehetséges, hogy csak bizonyos akciójú broadcast üzenetekre reagáljon, mely implementálásának főbb részlete az alábbi kódrészletben figyelhető meg:

```
1 IntentFilter mStatusIntentFilter = new IntentFilter();
2 mStatusIntentFilter.addAction(FETCH_NEW_ACTIVITY_DONE);
3 mStatusIntentFilter.addAction(FETCH_ALL_ACTIVITY_DONE);
4 LocalBroadcastManager.getInstance(this).registerReceiver(receiver,
    mStatusIntentFilter);
```

Ebben az esetben regisztrálunk egy fogadó osztályt, melyet a *BroadcastReceiver* osztályól örökoltettünk és *onReceive* metódusát felülírtuk. A második paramétere az az Intent lesz, amelyet a háttérben futó kód befejeztével sugároztunk szét, és csak is akkor jutunk el ide, ha az adott intent objektum akciója megfelel a szűrőbe beállított akciók valamelyikének. Amennyiben nem csatlakozni akarunk egy lehetséges sport-trackerhez, hanem el szeretnénk kezdeni játszani, abban az esetben hasonló folyamat játszódik le, annyi eltéréssel, hogy a már csatlakoztatott trackerektől csak a legújabb sportteljesítményeket kérjük le.

## Sportadatok letöltése

A különböző sportadatok letöltése olyan feladat, amely hasonlóságot és különbséget is mutat az integrált sport-trackerek között, ami az adott szolgáltatás API-jának kialakításában, és az eltárolt adatok típusában keresendő. Kialakítottam egy olyan egységes felületet, mely elfedi a szolgáltatások egyedi megoldásait, ezzel megkönnyíti további sport-trackerek integrálását a jövőben. Ehhez létrehoztam egy *IProvider* nevű interfész, amelyben deklaráltam azokat a metódusokat, amelyek az egységes működést hivatottak megvalósítani. Kialakítottam továbbá egy absztrakt *BaseProvider* nevű osztályt, mely implementálja ezt az interfészt, valamint tartalmazza a mindenképp szükséges adatmezőket, amik minden szolgáltatás esetén azonosak. Mivel ez az osztály absztrakt, nem jöhét létre belőle példány, ezért nem is muszáj konkrétan megvalósítania az *IProvider* metódusait, hanem csak a nem absztrakt gyerekeinek kell majd. Egy integrált sport-trackerhez el kell készíteni egy osztályt, amely ebből az osztályból öröklődik. A Strava-hoz elkészült osztály fontosabb elemei:

```
1  @Override
2  public List<SportActivity> getAllActivityFromService(Context context,
3      TrackerService tracker) { ... }
4
5  @Override
6  public List<SportActivity> getNewActivityFromService(Context context,
7      TrackerService tracker) { ... }
8
9  @Override
10 protected SportActivity convertActivity(Object item) { ... }
11
12 public SportActivityType getUnifiedType(String typeFromTracker) { ... }
```

Mindegyik metódus az interfészben lett deklarálva, így elérve, hogy csak az adott szolgáltatóhoz tartozó osztály legyen felelős az onnan történő adatletöltés részleteiért. A *convertActivity()* metódusra azért van szükség, mert már a két integrált sport-tracker esetén is előfordult, hogy ugyanazon célt szolgáló információk nem ugyanolyan típusban tárolódnak. A metódus segítségével egy sporttevékenységhez tartozó adatot az alkalmazásban elvárt formátumba hozható, így segítve továbbá az egységes felületet, mivel ezt a metódust az összes letöltött adatra meg kell hívni.

## Jutalmazás

A 3.1 fejezetben követelményként lett lefektetve, hogy a játékos a valós sportteljesítménye alapján jutalmakat kaphasson, amit a játékban felhasználhat. Miután az adatok letöltése meg-történt és a kívánt formátumba kerültek, megkezdődhet a vizsgálat, hogy az újonnan letöltött adatokért jár-e a felhasználó számára ajándék. A *Requirements* osztály tudja megvizsgálni,

hogy az adott sportolásért járhat-e ajándék. Ennek egyetlen statikus metódusa van, amely igazzal tér vissza, ha megfelelt az elvárásoknak, hamissal, ha pedig nem érte el a minimum megszabott követelményeket. A jutalom kisorsolására létrehoztam a *RewardDrawer* osztályt, amely minden új adatot megvizsgál. A jutalmazás függ a sportolás típusától, hosszától, esetleges egyéb paramétereitől is. Maga a jutalom is többféle lehet, ugyanakkor minden esetben mindenféle feltétel nélkül kiszámolásra kerül az adott sportteljesítmény alapján egy sztaminamennyiség, amit a játékos megkap.

A kiosztható jutalmak a következők:

- Arany
- Fegyver
- Varázsital

Továbbá elkészült három csoport, amelyekbe besorolásra kerültek a leggyakrabban előforduló sporttípusok. Az előbb felsorolt jutalmak a három csoport esetén más-más esélyel adományozódnak. A kialakított csoportok és a hozzájuk tartozó esélyeket az 5.2. ábrán látható táblázatban nézhetjük meg.

	Sporttipusok	Jatalomesélyek
Csoport 1	Futás Túrázás CrossFit	Arany: 10% Fegyver: 60% Varázsital: 30%
Csoport 2	Biciklizés jógázás snowboard-ozás	Arany: 60% Fegyver: 30% Varázsital: 10%
Csoport 3	Sétálás úszás jégkorcsolyázás	Arany: 30% Fegyver: 10% Varázsital: 60%

5.2. ábra. Jutalomcsoportok és esélyek táblázata

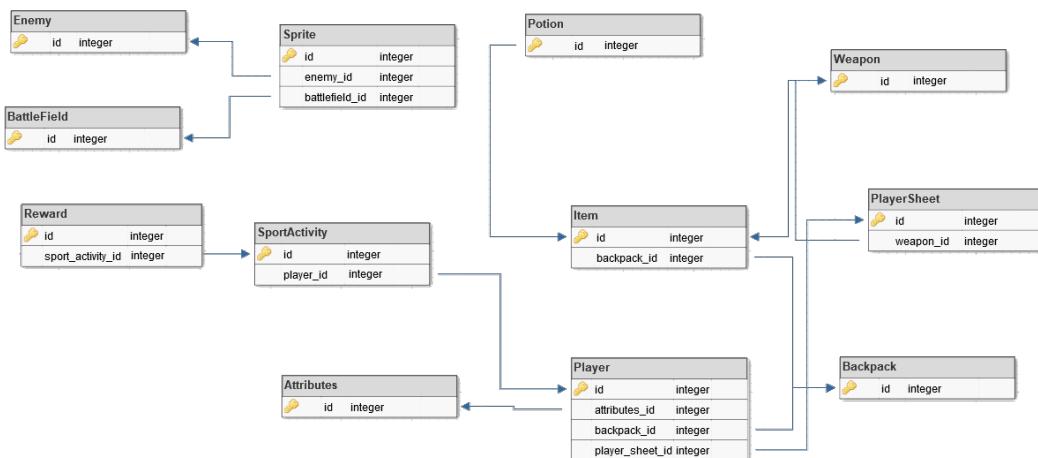
## 5.2. Adatbázis felépítése

Minden Androidos alkalmazás rendelkezhet egy helyi SQLite adatbázissal, amelybe tárolhatja azokat az adatokat amelyeket két indítás között is meg akarunk őrizni az alkalmazásban. Az SQLite egy több platformra fejlesztett, kisméretű relációs adatbázis-kezelő rendszer, mely tökéletesen illik a felhasználó okos-eszközének véges erőforrásaihoz. Az adatbázissal való

kommunikáció SQL utasításokkal történhet, amely nagyobb alkalmazások esetén nagyban megnövelheti a kódbázis méretét és potenciális hibákhoz vezethet. Ezt elkerülendő felhasználtam az ORMLite könyvtárat, amelyet részletesebben bemutattam a 3.2.4 alfejezetben.

A könyvtár használatához el kell készítenünk egy olyan osztályt, amelyen keresztül végre tudunk hajtani különböző akciókat az adatbázissal. Ehhez örököltetni kell a könyvtárban található *OrmLiteSqliteOpenHelper* osztályból és felül kell írni az *onCreate()* és *onUpgrade()* metódusokat. Az *onCreate()* metódus az első adatbázis-elérés alkalmával fut le, az alkalmazás futása során egyetlen egyszer, így itt kell létrehozni az adattáblákat és esetleges kezdeti értékekkel feltölteni. Az *onUpgrade()* metódus akkor hívódik meg, ha az alkalmazás egy újabb verziója telepítődik az eszközre, így ha valami strukturális változás történt az adatbázis szerkezetében a legutóbbi verzió óta, itt lehet a megfelelő lépéseket megtenni a kompatibilitás megőrzése érdekében.

Az elkészült adatbázis fontosabb adattáblái és azok kapcsolata az 5.3. ábrán látható.



5.3. ábra. Az adatbázis szerkezete

Egy adattábla elkészítéséhez elegendő egy egyszerű Java osztályt létrehozni, és ellátni a megfelelő annotációkkal, erről egy egyszerű példa az alábbiakban látható:

```

1  @DatabaseTable(tableName = "player")
2  public class Player implements CreatureData {
3
4      @DatabaseField(columnName = "id", generatedId = true)
5      private int id;
6
7      @DatabaseField(columnName = "player_gold", defaultValue = "0")
8      private int gold;
9

```

```

10     @DatabaseField(foreign = true, foreignAutoRefresh = true)
11     private Attributes attributes;
12
13     @ForeignCollectionField(eager = true)
14     private ForeignCollection<SportActivity> sportActivities;
15
16     ...
17 }

```

Az osztály tetején elhelyezett `@DatabaseTable` annotációval lehet megadni, hogy a hozzá tartozó adatbázistáblát milyen néven hozza létre az OrmLite. Miden olyan osztálynak rendelkezni kell egyedi azonosító attribútummal, amit szintén annotáció segítségével lehet megadni. Képes a mező neve alapján automatikusan oszlopnevet generálni, de lehetőség van egyedileg is megadni azt. Amennyiben nem használnánk semmilyen keretrendszeret az adatbázis felett, akkor nekünk kéne gondoskodni, hogy az idegen kulcsokat rendben vissza legyenek fejtve objektumokká, ezt a terhet leveszi rólunk a könyvtár.

Miután elkészült az összes entitás, mindegyikhez létre kell hozni egy DAO objektumot, amely segítségével képesek leszünk az adott entitást kezelni az adatbázisban. Az `IManager` interfész tartalmazza azokat a metódusokat, amelyek segítenek egy egységes felület kialakításában. A technikai részletek elfedése érdekében létrehoztam egy absztrakt `DataManager` osztályt amely implementálja ezt az interfészt. Ez az osztály továbbá tartalmaz egy objektumot a korábban említett `@OrmLiteSqliteOpenHelper`-ből származtatott osztályból, és az adott entitáshoz tartozó DAO-ból. A gyerekosztályok implementálása után ugyanazon felületet használva tudunk a megfelelő adatbázis-táblával kommunikálni.

### 5.3. Játék megjelenítése

A játékban megjelenő grafikai elemekért a 3.2.1 alfejezetben megvitatott indokok alapján választott dream-iso-droid játékmotor a felelős. Az alkalmazás indulása és az esetleges új sportadatok letöltése után a felhasználói interakcióra betöltődik a játék fő grafikus felülete. A korábbiakkal ellentétben, ahol is natív Androidos grafikai felületekkel találkozhattunk, az itt található legtöbb elemet a játékmotor rendereli le. Ez a felület egy új `Activity`-ből származtatott gyerekosztályban található, aminek a neve `GameActivity`. Az osztályhoz tartozik egy xml leíró fájl, amiben megadhatjuk, hogy az Activity által megjelenített vizuális elemeknek milyen szerkezete van. A legfőbb adatok a fájlban található elemekről az alább tekinthető meg:

```

1 <RelativeLayout>
2   <com.diploma.lilian.engine.GLCanvas android:id="@+id/glcanvas" ... />
3
4   <FrameLayout android:id="@+id/hud_container" ... />
5

```

```
6      <ImageButton android:id="@+id/inventory_open" ... />
7  </RelativeLayout>
```

A *RelativeLayout* tulajdonsága miatt az egymás után következő elemek, ha nincs meghatározva pozíciójuk, a előttük lévő elem fölé kerülnek. Ezt kihasználva a legelső elem a játékmotorban megtalálható úgynevezett vászon, a *GLCanvas* osztály mely minden megjelenítendő vizuális elemet itt rajzol ki. Efölött található egy *hud\_container* nevű elem, amely teljes egészében a vászon fölé terül, feladata hogy megjelenítse a játékoshoz kapcsolódó legfontosabb információkat.

A vászonra a játékmotor minden a beállított *GameScene* osztály által szolgáltatott képkockákat rajzolja ki. Ez szintén a játékmotorban található, az osztály maga absztrakt, így önmagában nem, csak örököltetett gyerekei képesek tartalmukat megjeleníteni a vászonon. Az elkészült játék három fő jelenetre bontható, amelyek a következők:

- Városi jelenet
- Harcmező jelenet
- Harci jelenet

A három jelenetnek jól elhatárolt feladatköre van és más-más grafikai elemek tárolására szolgálnak, ugyanakkor rendelkeznek olyan tulajdonságokkal is, amik mindenkor esetén megegyeznek. Ennek érdekében létrehoztam egy absztrakt osztályt *BaseScene* néven a *GameScene* osztályból örököltetve, melyben ezen közös funkcionálitásokat valósítottam meg, így az egyes jeleneteknél ezeket nem kellett redundáns módon megvalósítani. minden gyerekosztályban rendelkezik az örököltetés révén egy *init()* metódussal, amelyben az úgynevezett sprite-ok inicializálása történik. Ezen sprite-ok tulajdonképpen kétdimenziós képek, így a vászonon csupán ezen képek sokasága látszódik, és folyamatos váltakozása kelti az animáció képzétét.

Ebben az esetben is ugyanazon funkciót kell megvalósítania jelenetfüggően, így itt is elkészítettem egy *ISpriteProvider* nevű interfész, amelyben definiálásra kerültek azon metódusok amelyek segítségével ez a folyamat egységesen végrehajtható minden jelenetben. Elkészült egy *BaseSpriteProvider* nevű absztrakt osztály, ami megvalósítja ezen interfész metódusait, továbbá tartalmazza a beolvásásához szükséges adattagokat. Egy jelenet példányosítása során beállítjuk a hozzá tartozó *ISpriteProvider*-t implementáló objektumot. Amikor egy adott jelenet *init()* metódusa lefut, az interfészen keresztül el tudjuk készíteni a jelenethez szükséges sprite-okat, és ezeket a játékmotor már a vászonra tud rajzolni. A sprite-ok különböző adatait, amiket az *assets* könyvtárban tárolt xml fájlok ból olvasunk be, az engine-ben található *IsoSprite* osztályban tároljuk. A fájlban található animation tag-ekkel lehet megadni, hogy az adott sprite milyen animációk végrehajtására képes. A frame tag-ek között megadjuk, hogy a forráskép milyen pixeltartománya tartalmazza azt a területet, amelyet a játékmotornak meg

kell jeleníteni az animáció futása során. Egy ilyen xml fájl szerkezete az alábbi kódrészletben látható, a hozzá tartozó kép az 5.4. ábrán látható:

```
1 <isosprite name="male_light_sprite"      imgname="player_set">
2   <animations imgwidth="384" imgheight="384">
3     <animation name="up_move" steptime="150" startframe="0">
4       <frame left="0" right="32" top="144" bottom="192"/>
5       <frame left="32" right="64" top="144" bottom="192"/>
6       <frame left="64" right="96" top="144" bottom="192"/>
7     </animation>
8   </animations>
9 ...
10 </isosprite>
```



5.4. ábra. A játékoshoz tartozó forráskép

Ahogy korábban említésre került, teljes egészében a vászon fölé van kiterítve egy alapesetben átlátszó elem, ami a HUD, azaz a head-up display megjelenítésére szolgál, továbbá minden olyan grafikai elemre, amit nem a játékmotor kezel. A három jelenethez más-más HUD szolgál, amin az adott jelenethez tartozó legfontosabb információk jelennek meg. A HUD megjelenítésén kívül ezen a felületen jelenítődnek meg az egyéb elemek is, mint a hátizsák vagy a boltok, így biztosítva hogy natív Android grafikai elemeket is fel lehessen használni a felhasználói interakcióhoz a játék során. Ahhoz, hogy minden megfelelő nézet legyen megjelenítve, minden jelenethez és minden egyéb interakciót követelő nézethez létrehoztam egy *Fragment*-ből örökölhetett osztályt. Mindegyik osztályhoz tartozik egy hasonló xml fájl akárcsak az *Activity*-khez, ahol az adott osztályhoz tartozó grafikai elemek struktúráját lehet megadni, így elérve, hogy a kívánt tartalom jelenjen meg vászon felett.

## 5.4. A játéklogika

A megjelenítés mellett a legfontosabb eleme az alkalmazásnak a játék logikai része, azaz hogy bizonyos esemény hatására milyen művelet hajtódjon végre. Emiatt létrehoztam egy *GameLogic* osztályt, amely segítségével egy helyen kezelhetőek a különböző feladatok. Ilyen feladatok közé tartoznak a megfelelő jelenethez történő *ISpriteProvider* objektumok rendelése, a jelenetek cseréje a vászonon, a felhasználó interakcióra való reagálás és az ezekhez szükséges adatok tárolása. Ahhoz, hogy az osztály egységesen tudja kezelni a jeleneteket, létrehoztam egy *BaseSceneHandler* absztrakt osztályt, és ebből örököltettem osztályokat amelyek egy-egy jelenet kezeléséért felelősek. A származtatott gyerekosztályokon belül készülnek el a jelenetek, a jelenetek itt történik az *ISpriteProvider* objektum jelenethez rendelése, és ezen keresztül tudja elindítani a játéklogika a jelenetet. Az osztály három interfész implementál, amelyek úgynevezett eseményfigyelőként működnek, így az osztályobjektum kezelhető az implementált interfések objektumaiként is. Az interfésekben található metódusokat más osztályokból lehet meghívni amikor a megfigyelni kívánt esemény bekövetkezik. Az implementált felületek a következők:

- OnFightListener
- OnLevelUpListener
- OnGateListener

Ezek olyan eseményeket figyelnek, amelyeknek közvetlen hatása van a vászonon megjelenő elemekre. Példaként megemlítte az *OnFightListener* eseményfigyelő metódusait a harcjelenetben hívjuk meg, ahova a *GameLogic* osztályt áadtuk *OnFightListener* objektumként.

Néhány sprite-hoz - mint amilyen a játékos sprite-ja és az ellenfelek sprite-jai - tartoznak másféle adatok is, nem csak a megjelenítendő adatok. Ezért elkészítettem egy *SpriteInfo* osztályt, amely tárol egy konkrét *IsoSprite* objektumot és opcionálisan további adatokat, mint például a játékos vagy ellenfél paramétereit.

### Játékos

A játékos adatainak tárolására elkészült szerkezet az alkalmazásban a *Player* osztály, mely minden, a játékshoz tartozó objektumot és információt tárol. Többek közt ilyen tárolt objektum a hátizsák és a karakterlap, valamint az attribútumait tároló osztály, melyekhez minden tartozik egy-egy adatbázistábla. A hátizsákban a játékos fegyvereit, illetve varázsitalait lehet tárolni, a karakterlap felelős az éppen aktív varázsitalok és a karakterhez rendelt fegyverekért. Az attribútumosztályban a játékos tulajdonságait tároljuk, ezek befolyásolják a játékos minden más tulajdonságait. A tárolt tulajdonságok a kitartás, az erő és a szerencse, melyek

rendre a maximális életerőt, a játékos sebzését és a kritikus találat esélyét befolyásolják. Ugyanebben az osztályban tárolódik a játékos életpontja, sztaminája, a szinten eddig megszerzett tapasztalati pont és az elérődő tapasztalati pont. minden fegyvernek van egy minimális és egy maximális sebzési értéke, melyeknél nagyobbat nem képes sebezni. Varázsitalok között három típust különböztettem meg, melyek mindegyike egy-egy tulajdonságot növel meg, és ezeknek három fajta méretét határoztam meg. Az ital mérete befolyásolja a hatás hosszát és mértékét, így a kis méretű 2 harcon keresztül 10%-kal, a közepes méretű 4 harcon át 15%-kal, a nagy méretű ital pedig 6 harcon át 25%-kal növeli az adott tulajdonságot. Ezeket felhasználva elkészültek azok a formulák amelyek megadják a játékos végleges tulajdonságait. Az alábbiakban látható a fontosabb adatok kiszámolására szolgáló formulák:

**Maximális életerő** Játékos kitartása \* 5 \* (játékos szintje + 1) + 100

**Sebzés** Játékosnál lévő fegyver átlagsebzése \* ( 1 + játékos ereje / 10)

**Kritikus találat esélye** Játékos szerencséje \* 5 / (ellenség szintje / 2)

## Leltár

A leltár jelenetén a felhasználó számára több interakció lehetséges, így ezt a felületet natív Android elemekből építettem fel, és ezt a nézetet a HUD-on jelenítettem meg, olyan módon, hogy az eddigi áttetsző HUD ezután teljesen elfedje a vásnat. A hátizsákban található tárgyak megjelenítési módjához hasonló megoldást máshol is használok a játék során, illetve a hátizsákok más felületen is megjelenítem, így létrehoztam egy egyedi osztályt *EquipmentItem-Row* néven, amelyet az Android SDK-ban található *RelativeLayout* osztályból örököttem. Ennek segítségével elkerülhettem ugyanazon kódrészlet redundáns használatát. Az osztály konstruktorában megkapja azon elemek listáját, melyet meg kell jelenítenie, a hátizsák esetén az adatbázisból visszakérdezett adatok ezek. A konstruktor paramétereit közé tartozik egy eseményfigyelő, amely abban az esetben aktiválódik, ha az egyik tárgyat kiválasztotta a felhasználó.

Erről a felületről érhető el az a lista, amelyben a sportteljesítményekért járó konkrét jutalmakat jeleníti meg. Ez a lista is Androidos elemekből épül fel, a játékmotorban dinamikusan változó adatok megjelenítésére jelenleg nincs lehetőség. Az adatbázisból lekért adatok egy *ListView* tárolóban jelenítjük meg, a felület tovább tartalmaz egy lebegő gombot is a lista felett, mely megnyomására az 5.1 alfejezetben bemutatott *FetchService* osztályt felhasználva elindítható az adatok manuális frissítés. Amint a frissítés befejeződött, erről is egy broadcast-olt üzenet érkezik, mely hatására frissíthetjük a lista tartalmát.

## Város jelenet

A városjelenetben a játékos kétféle művelet végrehajtására képes, átjuthat a harcmezőre vagy beléphet a boltba. Ezen műveletekhez a megfelelő sprite-ba kell ütköznie a mozgása során. Az ütközésdetektálást a játékmotor kezeli, és ha ilyen esemény történt, akkor azt közli a jelenet osztállyal. Csak azokat a sprite-okat veszi figyelembe ütközésdetektálás közben a játékmotor, amelyeket megadtunk számára. Amikor egy sprite-ot hozzáadunk figyelendő sprite-ok közé, meg kell adni, hogy milyen típusú ütközésben vesznek részt. Ha két sprite összeütközött, a játékmotor meghívja az adott jelenet osztály által felülírt *handleCollision()* metódust. A metódus paramétereként megkapja a két ütköző sprite objektumot, és az ütközés típusát. A metóduson belül az ütközés típusa és a két ütköző sprite-tól függően más-más eseményt hívhatunk meg. A városjelenetben megtalálható *handleCollision()* fontosabb részei a következők:

```
1  @Override
2  public void handleCollision(IsoSprite s1, IsoSprite s2, int collisionGroupMask
   ) {
3      player.getSprite().stopMove();
4      if(s1.getName().equals("gates") || s2.getName().equals("gates")) {
5          onGateListener.onGateCollision(GameLogic.SCENE_TOWN);
6      }
7      if(s1.getName().equals("fegyverbolt") || s2.getName().equals("fegyverbolt")
8 )){ ... }
9      if(s1.getName().equals("templom") || s2.getName().equals("templom")){ ...
}
9 }
```

A kódrészletben látszik, hogy amint ütközés történik, megáll a játékos mozgása, a való életet imitálva, ahol egy szilárd test szintén nem tud áthaladni egy másikon. Ezután ellenőrizzük, hogy milyen épülettípusba ütközött, és ez alapján tudjuk a megfelelő döntést meghozni. Ha a játékos a teleportkapunak ütközik, akkor ezt a *GameLogic* osztály számára az adott eseményfigyelő *onGateCollision()* metódusával tudjuk jelezni, ahol a játéklogika a kapott paraméter alapján eldönti melyik jelenetre kell továbbvinni a játékost. Ha az egyik boltba ütközik a játékos, a *GameLogic* osztály hasonló módon értesítjük, csupán annyi a különbség, hogy más eseményfigyelő hívódik meg. Ilyen esetben a bolt típusától függően a játéklogika az eddigi HUD helyére betölti a bolt felületét, amelyet natív Android grafikus elemekkel valósítottam meg, hisz találhatóak rajta olyan elemek, amik a játékmotorban nem találhatóak meg, és a motorhoz adásuk túl sok egyéb feladat megvalósítását is jelentette volna. A boltok felülete két részre van osztva, a boltos részre és a felhasználó hátzsákját megjelenítő részre. A bolti árukat és a hátzsákban található tárgyakat is a korábban bemutatott *EquipmentItemRow* osztály felhasználásával jelenítem meg. A két rész esetén más-más eseményfigyelőt adok meg a létrehozott objektumnak, így más esemény végrehajtását elérve egy adott tárgy kiválasztása során.

## Harcmező jelenet

Ez a jelenet némely aspektusában hasonlít az előbb bemutatott városi jelenetre. A területen való mozgás az előző jelenettel eltérően sztaminapontokba kerül, és amennyiben ez elfogy, a játékos nem tud tovább mozogni. A játékmotorban nem volt lehetőség megállapítani egy sprite-ról, hogy tett-e meg valamilyen távolságot a játéktérben, és ha igen, akkor mikor, így ezzel a funkcionálitással ki kellett egészítenem. Az *IsoSprite* osztályban elhelyeztem egy eseményfigyelő objektumot, amelyen keresztül értesülhetünk arról, ha mozgás történt. Miután inicializálódott a játékos sprite-ja, beállítottam számára az eseményfigyelő objektumot egy anonym osztály formájában az alábbi módon:

```
1 player.getSprite().setOnMoveListener(new OnMoveListener() {  
2     @Override  
3     public void onStep() { ... }  
4 });
```

Az *onStep()* metódusban így csökkenthető a játékos sztaminája.

Itt is található egy kapu, amellyel vissza tudunk menni a városba, és ennek működése is az előzőekben leírtak alapján történik. A fő különbség a két jelenet között, hogy az itt található ellenfeleket a játéklogika automatikusan generálja. Ha a játékos a területen található összes ellenfelet legyőzte, a játéklogika érzékeli ezt és egy újabb területet generál új, erősebb ellenfelekkel. Ahhoz, hogy a játék bezárása és újból megnyitása között megőrizze, hogy a játékos mely szörnyeket győzte le, és a hártya szörnyek hol helyezkednek el, ezeket az adatokat a generálás során elmentjük az adatbázisban található *BattleField* táblába. A jelenet első megjelenítése során ezeket az adatok kiolvassuk az adatbázisból, és a visszanyert elemek alapján hozza létre a jelenethez tartozó, korábban taglalt *ISpriteProvider* a sprite-okat.

Szörnyekkel való ütközést szintén az *handleCollision()* metódusban tudjuk kezelni, a játéklogika a két ütköző entitást átirányítja a harcjelenethez.

## Harcjelenet

A harcjelenet során a játékos küzd meg azzal az ellenféllel amelyikbe a harcmezőn beleütközött. A harc folyamatát a játéklogika leszimulálja, azaz a két ellenfél felváltva támadja egymást, melyet a jelenethez tartozó handler osztályban valósít meg. A szimulálás azzal kezdődik, hogy a játéklogika elindítja a játékos támadási animációját, majd minután ez véget ér, a korábban bemutatott formula alapján kiszámolja a sebzését. Ez az érték levonódik az ellenfél életpontjaiból, majd a játéklogika most az ellenfél animációját indítja el, és ennek a végén az ő támadási értékét számolja ki. Ahogy a játékos, úgy az ellenfélhez tartozó formulákat egy *Formulas* nevű osztályban tárolódnak, és a megfelelő statikus metódusok

meghívásával lehet felhasználni őket. A handler osztály tartalmazza a *GameLogic* osztály által implementált eseményfigyelőt, így ha bármelyik fél életpontja nulla vagy az alá esik, ezen keresztül értesíti az eseményről a *GameLogic* osztályt. Ha a játékos életpontja fogott el, akkor az eseményfigyelő az *onFightLost()* metódust hívja meg, amiben a *GameLogic* osztály visszarakja a játékost a város jelenetre, az életpontja visszatöltődik, viszont minden varázsitalát - aktív és a hátizsákban lévő is - eltünteti. Ellenkező esetben a *Formulas* osztály metódusát felhasználva kiszámoljuk mennyi tapasztalatpontot kap a felhasználó, melyet az eddig megszerzett pontjai közé írunk. Amennyiben ezzel a többlettel elérte a szint teljesítéséért szükséges tapasztalatszámot, az osztályban található másik eseményfigyelőn keresztül értesítjük a *GameLogic* osztályt, ami az esemény hatására aktualizálja a játékos tulajdonságait, mint például a maximális életpontja, új elérendő maximális tapasztalati pontszámot állít be és elosztható pontokat ad a játékosnak, amelyet a karakterlapján oszthat el.

## 6. fejezet

# Továbbfejlesztési lehetőségek

Az elkészült játék teljesíti a felé támasztott követelményeket ugyan, de ahogyan minden egyéb más program, ez sem tökéletes. Előfordulhatnak olyan nem várt hibák, amelyek a fejlesztés és tesztelés során nem jelentkeztek. Ezek a hibák lehetnek a játék nem megfelelő használata miatt, a külső függősségekben található hibák miatt, melyeket az adott könyvtárak nem kezeltek le, vagy akár az operációs rendszer hibájából. Az ilyen hibák számát a továbbfejlesztés során ha nem is lehet nullára csökkenteni, de a lehetséges legkisebb értékre kell minimalizálni. A feladatspecifikus programokkal ellentétben egy játékhöz megannyi továbbfejlesztési ötlet juthat a fejlesztő de akár a játékos eszébe is. Nem történt ez másként most sem, már a fejlesztési szakaszban kínálkoztak olyan továbbfejlesztési lehetőségek, amiket érdemes lenne megvalósítani a szórakoztatóbb játékélmény érdekében.

### **További sport-trackerek integrációja**

A játék szempontjából nagyon fontos tényező, hogy a játékos elérje az általa használt sport-tracker alkalmazással rögzített adatokat. Jelenleg integrálásra került a két legnagyobb ilyen tracker, de természetesen vannak olyan emberek akik nem ezeket használják. Ahhoz, hogy ezeket az embereket is elérhesse a játék, a jövőben minél több sport-trackert kellene integrálni a játékba.

### **Online játékszolgáltatások**

Első ilyen ötletként az online játékszolgáltatások integrációja merült fel, a platformból adódóan főként a Google Play Games szolgáltatásé, melyhez külön SDK is tartozik. Amennyiben integrálásra kerül, lehetőség nyílik valós idejű többjátékos módra, az adatok felhőbe való mentésére, közösségi és nyilvános ranglisták készítésére, kitüntetések osztására és illegális szoftverhasználat elleni védekezésre is. A többjátékos mód esetén a játékban nem csak az

adott játékos karaktere jelenne meg, hanem minden online, az adott helyen tartózkodó játékosé is, így egymással kommunikálni is tudnának a játékosok, mely a legtöbb széles körben elterjedt online szerepjáték alapvető eleme. Az adatok felhőbe mentésével elérhetővé válik hogy a felhasználó egy eszközön elérte eredményeit egy másik eszközön folytathassa, így készülékcseré esetén nem kellene előlről kezdeni a játékot. A ranglisták és kitüntetések bevezetésével a játékosok közt kialakulna egy természetes versenyszemmel, hiszen mindenki szeretne minél jobb helyezést elérni, főleg, ha azt ismerőseinek is meg tudja mutatni. Az illegális szoftverhasználat segítségével meggátolhatóvá válnak a játékkal való visszaélések, így a tisztegesen játszó emberek munkája nem veszik kárba.

## **Játékmechanika javítása, új elemek hozzáadása**

A játékélmény fokozására a meglévőeken kívül új játékmechanikai elemeket lenne érdemes bevezetni, és a meglévőket folyamatosan frissíteni, finomhangolni. Ilyen új elemek lehetnek például zárt ajtók, amelyeken a játékos csak az ajtóhoz tartozó kulccsal tudna átjutni, a játékos számára az ugrás képességének implementálása, hogy a magasabban található elemekhez is hozzáérhessen. Továbbá a meglévők mellé új fegyverek hozzáadása, illetve más kiegészítők elkészítését, melyek mindegyike a harc kimenetelét befolyásolnák. A játékos figyelmének napi szinten való fenntartásához létre lehetne hozni egy új épületet, ahol minden napos gyakorisággal küldetéseket vállalhatna. Ezen küldetések teljesítési feltételét és jutalmát megjelenítenénk a felhasználó számára, így a játékos eldöntheti, hogy képes-e teljesíteni a feladatot, vagy megéri a jutalomért elvégeznie azt. A játékosok számára saját házat lehetne készíteni, mely többféle feladatot is betölthetne, mint például a különböző tárgyak tárolása vagy éppen olyan különleges elemek elhelyezése benne, amiknek permanens pozitív hatása lenne a játékosra nézve.

## **Játékosok közötti küzdelem**

További ötletként felmerült a játékosok közötti küzdelem lebonyolítása, mely során kiderülne mely felhasználó karaktere az erősebb. Ez a plusz funkció nagyban növelheti a játékosok sportolásra való motivációját, hisz a vesztes fél fel akarná erősíteni a karakterét, hogy a következő összecsapásban ő kerekedjen felül, a nyertes játékos pedig azért sportolna még többet, hogy a karakterét még tovább növelte megvéhesse a nyertes pozícióját. Természetesen különböző korlátokat kellene behozni, elkerülve hogy egy nagy szintű karakter ne küzdhessen meg csak bizonyos szinthatáron belüli karakterrel. Továbbá korlátozni lehetne az ilyen típusú harrok gyakoriságát is, vagy akár feltételhez kötni, esetleg valamelyen minimális szintű testmozgás esetén válna elérhető.

## **Pályaszerkesztő**

Érdemes lenne megvalósítani egy online pályaszerkesztő szolgáltatást is. Ennek segítségével bárki készíthetne a játékon belül játszható pályákat, ezeknek szerkesztés közben meg lehetne adni minden fontosabb paramétert, mint amilyen a méretei, nehézsége és a teljesítésének feltételét. A szerkesztés közben el lehetne helyezni adott pozícióra ellenfeleket és díszítő elemeket. Az így elkészült pályák igen szórakoztatónak tűnnek, hiszen a szerkesztők kreativitásának köszönhetően olyan megoldások is születhetnének, melyek addig nem jelentek meg a játékban.

## **Szerver**

Az utóbbi két továbbfejlesztési lehetőséghöz el kellene készíteni egy online kiszolgálót, így nem játékos eszközét terhelve ezen feladatok végrehajtásával, valamint ezen keresztül biztosítva folyamatos működésüket. Ezen kívül a szerveren ki lehetne alakítani egy fórumot is, ahol a játékosok megoszthatnák tapasztalataikat a többi játékossal, segítve a kezdőket.

## 7. fejezet

# Összefoglalás

Diplomamunkám célja egy lehetséges megoldás adása volt arra a problémára, hogy míg egyre több időt töltünk okoseszközök képernyője előtt, addig az egészséges életmódhoz szükséges tesmozgás gyakran hiányzik a minden napjaink ból. Az alapötlet egy olyan játék elkészítése volt, ami a micropayment-ek módszereit használja fel oly módon, hogy az előnyöket nem pénzáért, hanem sportteljesítményekért adjá.

Ezen ötletet követve készítettem el egy Android platformon működő szerepjátékot, melynek a RunPG nevet adtam. A játék magában hordozza a szerepjátékok jellemző stílusjegyeit, úgy mint a fejleszthető karakter, tárgyak kezelése, csaták, térképen barangolás, stb. A hosszú, és kihívást jelentő játékmenet biztosítása érdekében a játékbeli hősnek véletlenszerűen generált, de egyre nehezebb pályákat kell teljesítenie. A világban utazva azonban a folyamatosan csökkenő sztamináját csak valós sportteljesítmények után tudja visszatölteni, így ösztönözve a felhasználót a testmozgásra. A sportteljesítmények mérését külső alkalmazások végzik, a játékhoz példaképp két meghatározó sport-adatbázis lett illesztve.

A szoftver moduláris felépítésének köszönhetően a jövőben könnyen illeszthetők lesznek további funkcionálisok, mint például további adatbázisok illesztése, Google Play integráció, vagy PvP mód.

# Irodalomjegyzék

- [1] Google play áruház: Zombies, run! URL <https://play.google.com/store/apps/details?id=com.sixtostart.zombiesrunclient>.
- [2] Hollósi Tamás. *Izometrikus játékmotor Androidra*. Pannon Egyetem, 2012.
- [3] Oauth protocol. URL <https://tools.ietf.org/html/rfc6749>.
- [4] Ormlite weboldala. URL <http://ormlite.com/>.
- [5] Runkeeper api. URL <https://runkeeper.com/developer/healthgraph/overview>.
- [6] Strava api. URL <http://strava.github.io/api/>.
- [7] Scribejava könyvtár. URL <https://github.com/scribejava/scribejava>.

## MELLÉKLET

A mellékelt CD könyvtárszerkezete

– **Dokumentum**

- **Forrás** - A szakdolgozat szerkeszthető formátumban
- **Hivatkozások** - A szakdolgozatban lévő internetes hivatkozások letöltve
  - szakdolgozat.pdf
- **Forrás** - A program forrásállománya
  - **Az alkalmazás forrásfájljai importálható Android Studio projektként**
- **Program** - A futtatható program
  - **app-debug.apk** - Az alkalmazás telepítéséhez szükséges .apk fájl