Carcassonne társasjáték

# Követelmények feltárása

## Célkitűzés, projektindító dokumentum

### Rövid megfogalmazása a legfontosabb követelményeknek.

A projekt célja a Carcassonne nevű stratégiai társasjáték számítógépes reprezentációjának elkészítése, amely kellemes időtöltésként szolgálhat mindenki számára. A játékot egér segítségével lehessen irányítani, használata egyszerű és könnyű legyen. A játék folyamatosan küldjön visszajelzést a felhasználónak a lehetséges lépésekről, illetve a játék jelenlegi állásáról. Fontos, hogy megakadályozzuk, hogy a játékunk, a szabályoknak nem megfelelő, inkonzisztens állapotba kerüljön. Lehetőséget kell nyújtani az aktuális játékállapot mentésére és annak betöltésére is, valamint a több vagy egy gépen történő többszemélyes játszmák lebonyolítására.

## FOGALMAK

A fogalomjegyzék formátuma NÉV (ANGOL NÉV), ahol a név a magyar szabályok által használt kifejezés, az angol név pedig a program modelljében használt angol kifejezés.

### Játszma (Game)

Egy Carcassonne játszma addig tart, amíg minden a játékhoz tartozó Lapka át nem került a Pakliból a Pályára. A játszma legelején a Kezdőelem található csak a Pályán, és a játékosok folyamatosan pakolják át a többit. A játszma körökre osztva zajlik.

A játszma során a játékosok pontokat gyűjtenek, és a pakli elfogyását követően ezeknek a sorrendje alapján alakul ki a végeredmény – akinek a legtöbb több pontja lett, az a nyertes.

### Lapka (TILE)

A lapka a játék alapeleme. A pálya lapkákból épül fel, azzal a feltétellel, hogy annak folytonosnak kell lennie. A játék az eredeti társasjáték vizuális reprezentációját használja megjelenítésre, belső reprezentációban pedig egy lapka felfogható egy ötször ötös betűkből álló táblázatnak, amiben a betűk határozzák meg az adott részlet jelentését.



Két lapka akkor illeszthető adott forgatással egymás mellé, ha azonos betűket tartalmaz mind az öt szomszédos mezőjük.

A lapkákra bábukat lehet elhelyezni, aminek jelentése attól függ, hogy a lapka melyik pontjára helyezte le őket a játékos.

### Kör

Körnek nevezzük a játékban résztvevő Játékosok egy permutációját, amely során minden játékos egyszer lép. A kör sorrendjét a játék elején határozzák meg a játékosok, és a játék végéig fix marad. A körön belül egy játékos cselekvését Lépésnek hívjuk.

Az eredeti társasjátékban a legfiatalabb játékos kezd, majd tőle jobbra haladunk, a számítógépes verzióban viszont az egyszerűsítés miatt a játékba történő csatlakozás sorrendjében szerepelnek a játékosok. Egyszemélyes játék esetén az első játékos az ember, a továbbiak pedig a mesterséges intelligenciák.

### Pálya (Board)

Egy lazán kitöltött táblázatként vagy mátrixként fogható fel, amiben lapkák szerepelnek adott forgatással. A táblázatban lehetnek lyukak, de különálló szigetek nem, azaz a lapkáknak egy nagy komponenst kell alkotniuk.

### Pakli (Deck)

Meghatározza, hogy az adott játszmában milyen lapkák szerepelhetnek. Az eredeti társasjátékban adott volt, majd a kiegészítők tovább bővítették. A számítógépes megvalósításban a pakli két művelettel rendelkezik: egy, ami visszaadja a kezdőhelyzetet – ami állhat több lapkából is –, valamint egyből, ami megadja a következő lapkát – ez lehet akár tényleges pakliból választott, vagy teljesen véletlenszerű, csak valószínűségre épülő is, így több lehetőséget biztosítva, mint az eredeti játék.

### Kezdőelem / kezdőhelyzet (Starting)

Kezdőhelyzetnek nevezzük azt a lapkonfigurációt, ami a játékosoktól függetlenül eredetileg is szerepel a pályán. Az eredeti társasjátéktól különbözően a számítógépes verzió az üres kezdőkonfigurációt nem támogatja, amennyiben a játszma lapkák egy részcsoportjából történő húzással kezdődne, egy lapka alapból felkerül a pályára, hiszen ez a játszma tényleges kimenetelét nem befolyásolja.

### Játékos (Player)

A játékos a játék egyik szereplője, aki cselekvést végez a játszma során. A számítógépes változatban ez lehet ember, vagy mesterséges intelligencia is.

### Forgatás (Rotation)

Megmondja azt, hogy egy adott lapkát milyen irányban tettünk le. Eredeti (R0) forgatásnak nevezzük azt az esetet, ami megfelel az eredeti társasjáték hátoldala alapján a normális elrendezésnek. Ehhez képest a további három elforgatási szöget ettől az óramutató járásával megegyező irányba definiáljuk 90 fokonként, és ennek megfelelően R90, R180, és R270 kifejezéssel jelöljük.

### Bábu (Follower)

### Lépés (Step)

### Pontok (Score)

### Elhelyezés (Placement)

## Használatieset-modell, funkcionális követelmények



## Nem-funkcionális követelmények

**Fejlesztési módszertan:**

Egységesített Eljárás

**A fejlesztéshez szükséges hardver:**

* CPU: Pentium 4
* RAM: 1 GB
* videó: 1024x768

**A fejlesztéshez használt szoftverek:**

* Operációs rendszer: Windows 8.1
* Követelményelemzés: Microsoft Word
* CASE eszköz: Enterprise Architect 8
* Java fejlesztőeszköz: [IntelliJ](http://www.eclipse.org/downloads/packages/eclipse-standard-432/keplersr2) IDEA 14

**A futtatáshoz szükséges operációs rendszer:**

* Tetszőleges operációs rendszer, melyhez létezik JRE 8 implementáció

**Egyéb követelmények:** Intuitív felhasználói felület, könnyű kezelhetőség, könnyű telepítés, hordozhatóság, egyszerű üzemeltetés és megbízhatóság.

# Tervezés

## A program architektúrája

A program célja, hogy ugyanazt a funkcionalitást biztosítsa mind egyjátékos, mind többjátékos – legyen az egy vagy több számítógépen zajló – játékmód esetén is. Ennek segítésére a következő, más modern Java programoknál is jellemző technológiákat használja:

### Dependency Injection

A DI megvalósításáért a Spring Framework felelős, a program a játékmódtól függően különböző Beaneket használ.

Amennyiben szeretnénk bővíteni a játékot, szintén lehetőségünk van új komponenseket létrehozni, majd a konfiguráció átírásával az eredeti forráskód módosítása nélkül plusz funkcionalitást hozzáadni az eredeti programhoz.

### http Json kommunikáció

A több számítógépen történő többjátékos működés megvalósítására a program a http protokollt használja, és azon belül is a JSON adatformátummal történő parancs illetve adatátvitelt, egy egyszerű REST protokoll segítségével.

Ez egyrészt lehetővé teszi, hogy saját kommunikációs protokoll készítése helyett már jól bevállt megoldásokat használhassunk – A Jetty HTTP szerver, illetve a Jersey JAX-WS implementáció keretében –, illetve lehetővé teszi, hogy a program akár olyan környezetben is működjön, ahol a tűzfalak az egyedi protokollon történő kommunikációt megfognák.

A Dependency Injection segítségével mindössze a konfiguráció módosításával egyszerűen indíthatunk a kliensprogramon belülről is saját szervert, vagy létrehozhatunk akár dedikált szervert is.

Ezeknek megfelelően a program architechtúrája szolgáltatás-orientált: a játék modellje megváltoztathatatlan (immutable) objektumokból áll, a szolgáltatások műveletei pedig egy játékteret, és az azon történő módosítás leírását várjak, majd ebből visszaadnak egy új játékteret.

A szerver-kliens kommunikáció okán a játéktérnek két reprezentációja is létezik: egy, ami ideális műveletvégzéshez (Domain), illetve egy a kommunikációhoz használt (DTO). A jelenlegi program által megvalósított DAO mindössze memóriában tárolja az objektumokat, amennyiben szükség lenne egy olyan szerverszoftverre, amely leállás esetén is megtartja a játékokat, szükség lenne egy JPA vagy egyéb adatbázisréteg alapú DAO implementálására.

A fájlba történő mentés/betöltés műveletek megvalósítottak, ezek azonban kiesnek a tipikus DAO műveletek alól, így jelenleg kizárólag egyjátékos esetben megvalósítottak.

A felhasználói felület ezt a szolgáltatásréteget használja, így működése tulajdonképpen kimerül abban, hogy megjeleníti a szolgáltatás által visszaadott játékteret, illetve lehetőséget ad a játékosnak a lépésének megtervezésében, valamint végrehajtásában.

# Szolgáltatások

A program két szolgáltatást implementál, és használ:

A LobbyService a játék indulásáig történő műveletekért felelős, azaz tud játékot létrehozni, játékosokat (emberi vagy mesterséges intelligencia) csatlakoztatni, illetve játéklistát lekérni.

A GameService ehhez képest azokkal a műveletekkel foglalkozik, amik a játék indulása után történnek, azaz annak segítségével a játékosok lépéseit hajthatjuk végre, illetve a játék aktuális állását tudjuk lekérdezni.

A mentés / betöltés egyik szolgáltatásban sem jelenik meg, mivel ez a két művelet kiesik a szolgáltatás alapú megvalósításból, és egyedien lett megvalósítva:

A mentés elvégezhető, ha a játékteret átalakítjuk a kommunikációhoz is használt DTO formátumra, majd ennek segítéségével a játéktér JSON reprezentációját kiírjuk a fájlrendszerre.

A betöltéshez pedig ugyanezt a műveletet megfordíthatjuk, és inicializálhatjuk a GameService-t az így kapott Game objektummal.

## LobbyService

## GameService

## Save / Load

# Kommunikációs modell

A program belső működése erősen függ attól, hogy azt milyen üzemmódban használjuk: a beanek konfigurációja máshogy néz ki attól függően, hogy azt dedikált szerver vagy kliens módban indítottuk, majd utóbbi esetében még külön megkülönböztetjük azt az esetet, amikor egy számítógépen, illetve azt, amikor több számítógépen játsszunk.

A dedikált szerver, illetve a többszámítógépes kliens játék között lényeges különbség nincs, így ez a kettő együtt tárgyalható, annak megjegyzésével, hogy utóbbi esetben a szerver az egyik kliensnek felel meg.

## Egyszámítógépes eset

## TÖbb számítógépes eset

# Adatátviteli modell

Az adatátviteli modellben a játékteret olyan formára alakítjuk, hogy az egyszerűen és egyértelműen szöveggé alakítható legyen, majd abból vissza. A szöveges reprezentáció formátumának a JSON-t választottuk, de mivel a JAX-WS jópár formátumot támogat, ehelyett igen könnyen lehetne használni tetszőleges, általános leíró nyelvet (pl. XML vagy YAML)

## Lobby DTO

## Game DTO

# Domain Modell

# Műveletek a játéktéren