**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Informatikatudományi Intézet**

**Programozáselmélet és Szoftvertechnológia Tanszék**

Szakdolgozat címe

Szerző: Témavezető:

Ábrahám Erik Név

Programtervező informatikus BSc. beosztás, titulus

**Budapest, 2025**

Ide kerül a hivatalos témabejelentő lap.

Tartalomjegyzék

[1. Bevezetés 2](#_Toc87521353)

[2. Felhasználói dokumentáció 2](#_Toc87521354)

[3. Fejlesztői dokumentáció 2](#_Toc87521355)

[4. Összefoglalás és további fejlesztési lehetőségek 2](#_Toc87521356)

[5. Irodalomjegyzék 2](#_Toc87521357)

[6. Melléklet 2](#_Toc87521358)

# Bevezetés

A számítógépes játékfejlesztés a szoftverfejlesztés egyik legsokoldalúbb és legizgalmasabb területe, amely egyesíti az algoritmikus gondolkodást, a vizuális megjelenítést, valamint a felhasználói élmény optimalizálásának kihívásait. A szakdolgozatom célja egy népszerű társasjáték, a *Bang!* digitális, asztali változatának megvalósítása Java nyelven. A projekt során egy önállóan futtatható, grafikus felhasználói felülettel rendelkező alkalmazást készítettem, amely hűen követi az eredeti játék szabályait, és lehetőséget biztosít egyjátékos (AI elleni) játékra.

A *Bang!* játék különlegessége, hogy a játékosok rejtett szerepek mentén, eltérő győzelmi feltételekkel versenyeznek, így a játékmenet változatos, és taktikai döntéseket igényel. A program célja ennek a játékmechanikának a pontos, de ugyanakkor élvezetes számítógépes adaptációja. A felhasználó egy grafikus felületen keresztül irányíthatja a karakterét, kártyákat húzhat, kijátszhat, támadhat vagy védekezhet, a játék során pedig a mesterséges intelligencia vezérelte ellenfelek a saját logikájuk szerint próbálják elérni győzelmüket.

A fejlesztés során a Java programozási nyelvet és a Swing grafikus felületkezelő könyvtárat választottam, mert ezek jól illeszkednek az asztali alkalmazásokhoz, illetve a Java objektumorientált szemlélete ideálissá teszi az összetettebb játéklogika kezelésére is. A játék fejlesztése során nagy hangsúlyt fektettem a rétegzett architektúra kialakítására, a kód újra felhasználhatóságára és a tesztelhetőségre.

A dolgozat célja, hogy bemutassa a játék működését a felhasználó szemszögéből, majd részletesen feltárja a szoftverfejlesztési folyamatokat, az architektúrát, a program szerkezetét és a megvalósítás során használt technológiákat. Az utolsó fejezetben a játék teszteléséről és a továbbfejlesztési lehetőségekről esik szó.

# Felhasználói dokumentáció

## A játék rövid leírása

A Bang! egy vadnyugati tematikájú, szerepalapú társasjáték, amelyet digitális formában valósítottam meg Java nyelven. A játék során a játékosok rejtett szerepek szerint versenyeznek egymással: Sheriff, Renegát, Banditák vagy Sheriff-helyettes szerepben. Minden játékos célja eltérő, ami izgalmas és taktikus játékmenetet eredményez.

Az alkalmazás egy asztali (desktop) környezetben futó, grafikus felhasználói felülettel ellátott Java program, amely lehetővé teszi a játék egyjátékos módját. Az egyjátékos módban a felhasználó számítógép által irányított (bot) játékosok ellen játszik, akik egyszerű mesterséges intelligenciával rendelkeznek, és saját, előre meghatározott stratégiájuk mentén cselekednek.

A program a valódi társasjáték szabályait követi, beleértve a kártyák húzását, kijátszását, a különféle támadó és védekező kártyák használatát, valamint az életerő (HP) kezelését. A játékosok karaktereket választanak, amelyek mindegyike egyedi képességgel rendelkezik, ez tovább fokozza a stratégiai lehetőségeket.

A digitális verzió célja, hogy a társasjáték hangulatát és mechanikáját a számítógépen keresztül is elérhetővé tegye, lehetőséget adva gyakorlásra, játékelemzésre, vagy egyszerű szórakozásra akkor is, ha nincs kéznél elég ember egy fizikai játékpartihoz.

## Játékszabályok bemutatása

A Bang! egyjátékos változatában a felhasználó egy választott, vagy véletlenszerű szerepű karaktert irányít, és mesterséges intelligencia által vezérelt ellenfelekkel (botokkal) mérkőzik meg. A játék célja az, hogy a játékos túlélje az összes ellenség támadását, és győzelemre vezesse karakterét azáltal, hogy minden ellenfelet legyőz.

### A lehetséges szerepkörök, céljaik és győzelmi feltételeik:

A játék elején a játékos kiválaszthat egy tetszőleges szerepet, vagy rábízhatja a programra, hogy osszon ki neki egy véletlenszerű szerepet (Sheriff, Bandita, Renegát, Sheriff-helyettes). Az összes többi szerep botokra van kiosztva, és a játékos a játék kezdetén nem tudja, ki milyen szerepben van. A szerepkörök céljai:

* **Sheriff**  
  Célja: Minden Outlaw (Bandita) és a Renegát likvidálása.  
  Ha a Sheriff túléli a játékot, és már csak ő és a Sheriff-helyettesek maradnak, akkor a Sheriff győz.
* **Outlaw (Bandita)**  
  Célja: Megölni a Sheriffet.  
  Ha a Sheriff meghal, és legalább egy Bandita életben van, akkor a Banditák nyernek.
* **Renegát**  
  Célja: Ő legyen az utolsó túlélő játékos.  
  Előbb el kell távolítania minden más játékost, legvégül a Sheriffet. Ha a Sheriff előtt bárki más túléli, a Renegát automatikusan veszít.
* **Deputy (Sheriff-helyettes)**  
  Célja: Megvédeni a Sheriffet, és segíteni a Banditák és a Renegát kiiktatásában.  
  Ha a Sheriff győz, a Sheriff-helyettesek is nyernek, de ha a Sheriff meghal, ők is elveszítik a játékot.

### A kör menete

A játék körökre osztott. Egy körben minden játékos (bot is) végrehajtja a következő lépéseket:

1. **Kártyahúzás** – A kör elején minden[[1]](#footnote-2) játékos két kártyát húz a pakliból.
2. **Kártyák kijátszása** – A játékos tetszőleges számú kártyát kijátszhat, a szabályoknak megfelelően (pl. egyszerre csak egy Bang! kártya játszható ki).
3. **Kártyadobás** – A kör végén a játékos annyi kártyát tarthat a kezében, ahány életereje (HP) van. A fölösleget el kell dobni.

### Kártyatípusok

A játékban különböző típusú kártyák találhatók, melyek a harc és védekezés alapját képezik.

* **Bang!** – Elsősorban más játékosokra lehet vele lőni.
* **Nem talált!** -Elsősorban a Bang! hatását semlegesíti.
* **Sör** – Gyógyít egy életerőpontot, de csak ha a játékos nem maximális életerőn van. És kettőnél több játékos van életben.
* **Kocsma – Ha kettőnél több játékos van, akkor vagy mindenkit gyógyít vagy csak a lapot kijátszó játékost.**
* **Gatling** – Minden ellenfél egy Nem talált! kártyával védekezhet, vagy veszít egy életpontot.
* **Indiánok!** – Minden ellenfél egy Bang! kártyával védekezhet, vagy veszít egy életpontot.
* **Párbaj – A célpontnak Bang! kártyát kell kijátszania, majd a Párbaj lapot kijátszó játékosnak. Aki először nem tud Bang! lapot kijátszani, az veszít 1 életpontot.**
* **Cat Balou – Eldobat egy kártyát a célponttal.**
* **Pánik** – Elvesz egy kártyát a hatótávolságon belüli célponttól.
* **Postakocsi – Húzhat 2 kártyát a kijátszó játékos.**
* **Wells Fargo – Húzhat 3 kártyát a kijátszó játékos.**
* **Szatócsbolt – Annyi lapból, ahány játékos életben van, mindenki választ egyet magának a kör haladása szerint.**
* **Musztáng** - A többi játékos eggyel messzebbről látja, míg aktív ez a lap.
* **Távcső** - A többi játékost eggyel közelebbről látja, míg aktív ez a lap.
* **Hordó** – Ha aktív ez a lap és meglövik egy Bang! kártyával, akkor húznia kell a pakli tetejéről és eldobni. Ha ez a lap kőr, akkor sikeresen blokkolta a Bang! lapot.
* **Dinamit** – Ha aktív a dinamit a játékos köre kezdetén, akkor húznia kell és eldobnia a lapot. Ha a lap pikk 2-9 közti magassággal, akkor a dinamit robban és 3 életpontot veszít a játékos.
* **Börtön** – Ha aktív a börtön a játékos köre kezdetén, akkor húznia kell és eldobnia a lapot. Ha a lap kőr, akkor kiszabadul és lejátssza a körét. Ellenkező esetben eldobja a börtön kártyát és kimarad ebből a körből. Ha a játékos előtt egyszerre van börtön és dinamit, a dinamitra húz először. A Sheriff nem célozható vele.
* **Fegyver – Megnöveli azt a távolságot, ahol Bang!-et játszhat ki a játékos.**
  + **Gyorstüzelő – Ha aktív, bármennyi Bang’ kártyát kijátszhat a játékos az adott körben.**

### Karakterek

A játékban minden karakter egyedi képességgel rendelkezik. A játék kezdetén a játékos kiválaszthat egy karaktert (vagy választhatja a véletlenszerű kiosztást), és a botokhoz is véletlenszerű karakterek kerülnek. Karakterek és képességeik:

|  |  |
| --- | --- |
| **Bart Cassidy** | **Minden elvesztett életpont után, húz egy lapot a pakli tetejéről.** |
| **Black Jack** | **A köre elején húzáskor megmutatja a második lapot, ha az kőr vagy káró, húz még egyet.** |
| **Calamity Janet** | **Bang! lapokat Nem talált! lapként is kijátszhat és fordítva. Támadó szándékkal csak egyet használhat körönként ugyanúgy.** |
| **El Gringo** | **Minden sebződés után húz egy lapot, ha van, az őt megsebző játékos kezéből.** |
| **Jesse Jones** | **A köre elején választhatja, hogy az első lapot egy másik játékos kezéből húzza a pakli teteje helyett.** |
| **Jourdonnais** | **Van egy beépített Hordó-ja. Kijátszhat még egy hordót, mint a többi játékos és ilyenkor kétszer húzhat, ha meglövik a Bang!-gel.** |
| **Kit Carlson** | **A köre elején megnézi pakli felső 3 lapját, választ kettőt és a maradékot visszateszi.** |
| **Lucky Duke** | **Ha Hordó, Dinamit vagy Börtön miatt húznia kell, akkor kettőt húz, kiválaszt egyet, majd mindkettőt eldobja.** |
| **Paul Regret** | **A többi játékos eggyel messzebbről látja, kijátszhat Musztángot is, ekkor kettővel messzebbről látják.** |
| **Pedro Ramirez** | **A köre elején az első lapot húzhatja a dobó pakli tetejéről.** |
| **Rose Doolan** | **A többi játékost eggyel közelebbről látja, kijátszhat Távcsövet is, ekkor kettővel közelebbről látja a többi játékost.** |
| **Sid Ketchum** | **Ha eldob két lapot, akkor visszanyer egy életpontot, egy körön belül többször is.** |
| **Slab the Killer** | **A Bang! kártyái elől csak kettő Nem talált! lappal térhet ki a célpont. A Hordó ugyan úgy le tudja védeni a lövést.** |
| **Suzy Lafayette** | **Ha üres a keze, húz egyet.** |
| **Vulture Sam** | **Ha kiesik valaki, megkapja a kiesett játékos lapjait.** |
| **Willy the Kid** | **Egy körben bármennyi Bang! kártyát kijátszhat.** |

Minden karakterképesség más és más, így érdemes ezek köré építeni a taktikát.

### Játékosok közötti távolság

A játékosok közötti távolságot úgy érdemes elképzelni, mintha egy körben ülnének egymás mellett. Adott játékos melletti játékosok egy távolságra vannak, azok szomszédjai, pedig kettőre és így tovább. Mivel két játékos között általában két különböző távolság van, egy az óramutató járásával megegyezően, másik az óramutató járásával ellentétesen. Ilyenkor a kisebb távolságot kell figyelembe venni. Ez a távolság vonatkozik a Pánik kártyára, amit olyan célpontot használhatunk, aki egy távolságon belül van. Ezt a távolságot módosítja a Musztáng és a Távcső kártya. Ha van aktív Musztáng lapja a játékosnak, akkor őt eggyel távolabbról látják, ha van aktív Távcső kártyája, akkor a játékos társait eggyel közelebbről látja. Tehát Ő és a szomszédjai között 0 a távolság és így tovább. A fegyverek csak a Bang! kártya hatótávolságát befolyásolják. Tehát például: Ha van egy Remingtonunk, ami három távolságot ad, akkor azokra az ellenfelekre lehet lőni, akik három távolságon belül vannak.

### Speciális szabályok

* A játékban nincs emberi játékostárs – mindenki más bot.
* Csak a Sheriff kiléte köztudott.
* A botok mesterséges intelligenciája egyszerű szabályok szerint működik, és elsősorban a kártyák típusára és lehetőségeikre reagálnak.

## A Bang! letöltése, telepítése

## Hardver- és szoftverigény

Az alkalmazás célja, hogy egy modern, könnyűsúlyú, egyjátékos asztali játékélményt nyújtson. A játék nem igényel jelentős számítási kapacitást vagy speciális eszközöket, így a legtöbb, Java-t futtatni képes számítógépen zökkenőmentesen használható.

### Ajánlott hardverkonfiguráció

* **Processzor**: 2 GHz-es, négymagos processzor vagy gyorsabb
* **Memória (RAM)**: 4 GB vagy több
* **Tárhely**: ~180 MB szabad lemezterület
* **Képernyőfelbontás**: 1920×1080 vagy magasabb, a kényelmes megjelenítéshez
* **Egér**: szükséges a játék futtatásához

### Szoftverkövetelmények

* **Operációs rendszer**: Windows 10
* **Java futtatókörnyezet (JRE)**: Java SE 21 vagy újabb verzió

### Egyéb megjegyzések

* Az alkalmazás nem igényel internetkapcsolatot a használathoz.
* Nem szükséges adminisztrátori jogosultság a futtatásához.
* Mivel a játék csak egyjátékos módban működik, nincs hálózati vagy multiplayer komponens, így különösen kis erőforrásigényű.
* Mivel a játék angolul van, ezért minimális angol tudás szükséges.

## Játékmenet

### A játék áttekintése

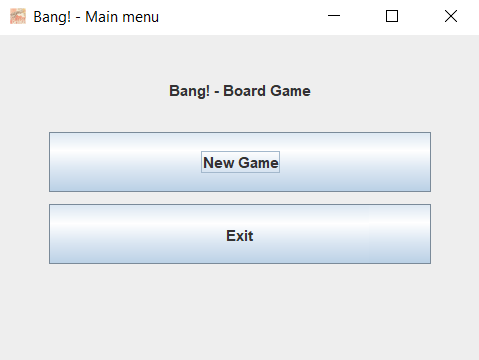
A játék egy digitális adaptációja a klasszikus Bang! társasjátéknak, kifejezetten egyjátékos módra fejlesztve. A játékos a sheriff szerepét veszi fel, és célja, hogy a mesterséges intelligencia által irányított ellenfeleket legyőzze a játék szabályainak megfelelően. A játék teljes mértékben egérrel vezérelhető, és a felhasználói felület intuitívan vezeti végig a játékost a körökön.

A játék körökre oszlik, és minden körben a játékos vagy a mesterséges intelligencia által vezérelt ellenfelek lépnek. Minden játékos körében kártyákat húz, kijátszik, támad vagy védekezik, és végül eldobja a felesleges kártyákat.

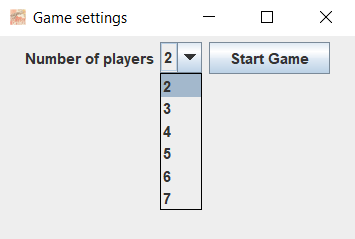
### Új játék indítása

A program elindításakor a főmenü jelenik meg, ahol kettő lehetőség közül lehet választani:

* **New game (Új játék indítása)**
* **Exit (Kilépés)**

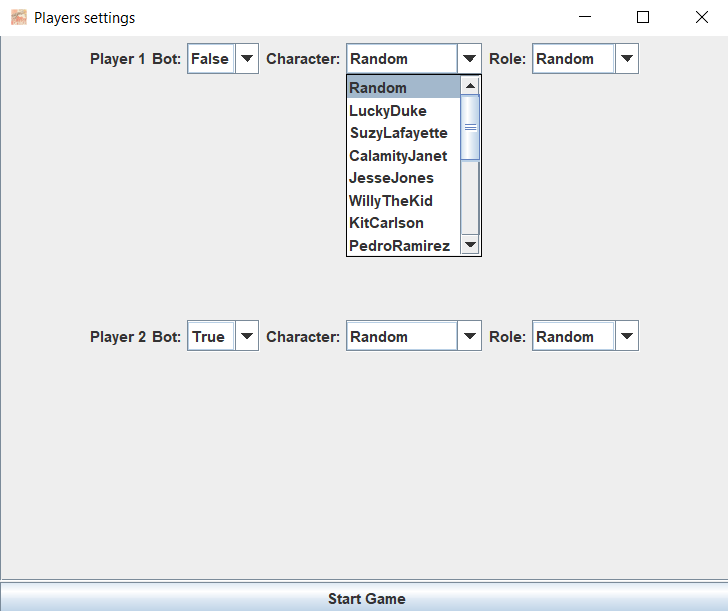


Az **New game** gomb megnyomásával megnyílik a *„Game settings”* (játék beállítások) ablak, ahol a játékos kiválasztja, hogy mennyi játékos legyen (2-7).



A **Start Game** (játék indítása) gomb megnyomásával megjelen a *„Players settings”* (játékos beállítások) ablak. Ebben a felhasználó kiválaszthatja:

* melyik player (játékos) szeretne lenni
* a saját, és a botok karakterét (vagy választhat véletlenszerű karaktert)
* a saját, és a botok szerepeit (szintén lehetőség van véletlenszerű beállításokra)



Az ellenfelek minden esetben mesterséges intelligencia által vezérelt botok. A konfiguráció beállítása után a **Start Game** gombbal indítható el a játék.

### A játéktér felépítése

A játék elindulása után megjelenik a fő játéktér, amely több vizuális komponensből épül fel. A felület kialakítása Swing alapon történt, és minden elem világosan elkülönül, hogy segítse a játékos tájékozódását. A főbb elemek a következők:

#### **Játékos lapjai (alsó sáv)**

Ez a szekció a játékos kézben tartott lapjait jeleníti meg: mindegyik kártya egyedi képe jelenik meg, ha a játékos föle viszi az egeret, akkor megnő, és kijelölhető.

#### **Ellenfelek (felső sáv)**

Az ellenfelek, azaz a botok, külön panelen helyezkednek el. Mindegyik ellenfél esetén látható:

* A nevük, aktuális, és maximális HP értékük.
* Egy kártya borítóval felfelé, benne egy fehér kör és benne a kézben tartott kártyáinak száma.
* Az aktív, asztalra lerakott kártyáik képpel felfele (felszerelések, fegyverek).

#### **Középső vezérlőpanel**

Ez a szekció a játékos asztalon lévő kijátszott, aktív lapjait jeleníti meg: mindegyik kártya egyedi képe jelenik meg, nem lehet velük interakcióba lépni közvetlenül, csupán azt az információt adják át, hogy a kártya aktív és befolyásolj a játékos képességeit.

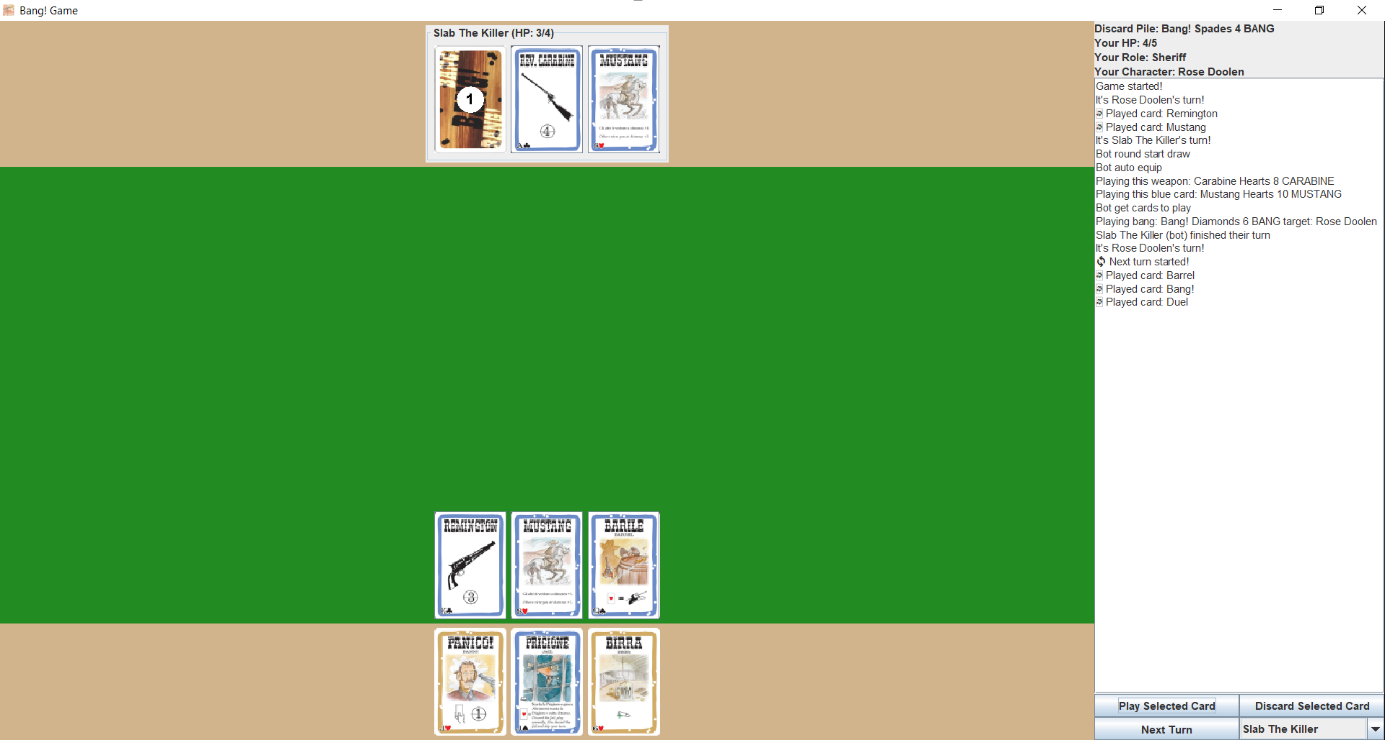
Itt található a játékmenetet vezérlő főbb elemek:

* **„Következő kör”** gomb: ezzel adható át a kör a következő játékosnak.
* **Célpontválasztó legördülő menü**: amelyben kijelölhető, hogy melyik ellenfelet kívánjuk megtámadni.
* **Kijátszott kártyák megjelenítése**: az asztalra kijátszott lapok itt is megjelenhetnek átmenetileg.

#### **Jobb oldali log panel**

* **Dobott lapok**: a „Discard Pile” jelzéssel jelenik meg az utolsó eldobott lap neve.
* **Aktuális/maximális életerő: a játékos aktuális és maximális életereje.**
* **Szerep: a játékos játékbeli szerepe.**
* **Karakter: a játékos karakterének a neve.**
* **Játéknapló**: itt látható az összes fontos esemény (pl. ki támadott kit, ki dobott lapot).
* **„Play this card” (kártya kijátszása) gomb: ezzel tudjuk kijátszani az aktuálisan kijelölt kártyát.**
* **„Discard this card” (kártya eldobása) gomb: ezzel tudjuk eldobni az aktuálisan kijelölt kártyát.**
* **„Next turn” (következő kör)** gomb: ezzel adható át a kör a következő játékosnak.
* **Célpontválasztó legördülő menü**: amelyben kijelölhető, hogy melyik ellenfelet kívánjuk megtámadni.

A felület célja, hogy világos, egyszerűen értelmezhető legyen, és megkönnyítse a játékos számára a döntéshozatalt.



### A körök működése és fázisai

A játék körökre osztva zajlik. Minden kör egy játékos (jelenleg mindig az emberi játékos) teljes körét jelenti, amely során végrehajthat különböző műveleteket. A botok automatikusan, a megírt mesterséges intelligencia logikája alapján hajtanak végre akciókat a saját köreikben. Ezek a fázisok automatikusan követik egymást a játék során. Az emberi játékos minden akcióját vizuálisan is támogatja az alkalmazás, míg a botok automatikusan és minimális késleltetéssel lépnek a saját logikájuk alapján.

Egy kör három fő fázisból áll:

#### Kártyahúzási fázis

A játékos a kör elején automatikusan húz két lapot a húzópakliból. Ha a húzópakli kifogy, a dobópakli újra megkeverésre kerül és húzópaklivá válik, a legutolsó eldobott lap nem keveredik bele a húzópakliba, ott marad a dobópakli tetején.

#### Akciófázis

Ez a fázis a leghosszabb, itt történik a játék fő része. A játékos a kezében tartott kártyákat különböző célokra használhatja:

* **Felszerelés kijátszása**:  
  A játékos kijátszhat olyan kártyákat, mint a Hordó, Musztáng, Távcső vagy fegyverek. Ezek az asztalra kerülnek és onnantól aktívak.
* **Támadás**:  
  A Bang! típusú kártyával a játékos egy ellenfelet támadhat meg, ha az lőtávolságban van. A célpont kiválasztása a célpontválasztó segítségével történik.  
  Egy körben alapesetben csak egy Bang! kártya játszható ki, kivéve, ha karakterképesség vagy más kártya ezt felülírja.
* **Különleges kártyák**:  
  Bizonyos kártyák minden ellenfelet érintenek („Gatling”, „Indiánok”), vagy más játékos lapját célozzák („Pánik”, „Cat Balou”). A játék automatikusan értelmezi ezek hatását.
* **Segéd kártyák**:Néhány kártya csak az aktuális játékosra van hatással, például gyógyítja („Sör”), vagy új lapokat huzat a játékossal („Postakocsi”, „Wells Fargo”).
* **Nem talált!** kártya („Missed!”):  
  Nem akcióként, hanem reakcióként használható. Amikor egy ellenfél megtámad minket, ezzel hárítható a támadás.

#### Dobás fázis

A kör végén a játékosnak el kell dobnia annyi lapot, hogy a kezében lévő kártyák száma ne haladja meg az aktuális életerőpontjainak számát. A felesleges kártyák automatikusan dobópakli tetejére kerülnek.

### Győzelmi feltételek és a játék vége

A játékos és a botok közötti küzdelem célja az egyéni **győzelmi feltétel** teljesítése, amelyet a játékos szerepköre határoz meg. A játék indulásakor minden résztvevő véletlenszerűen (vagy felhasználói választással) kap egy szerepet. A szerepek titkosak, kivéve a Sheriff szerepét, amely mindig ismert.

#### A játék vége

A játék vége automatikusan bekövetkezik, amikor valamelyik győzelmi feltétel teljesül. A játék végi képernyő kiírja a győztesek szerepét, valamint lezárja az aktív játékmenetet.

Az egyjátékos mód során a játékos egy előre beállított számú bot ellen küzd. A botok mesterséges intelligencia logikát követnek – támadnak, védekeznek, és megfelelő időben használják a kártyáikat. A játék vége után lehetőség van új játék indítására a főmenü segítségével.

# Fejlesztői dokumentáció

## A szoftver rövid leírása, specifikáció

A szoftver egy Java nyelven készült, asztali (desktop) környezetben futtatható egyjátékos kártyajáték, amely a Bang! nevű társasjáték digitális adaptációja. A játék célja a klasszikus szerepalapú játékmenet és a kártyahasználat élményének átültetése egy számítógépes környezetbe, a lehető leghitelesebb módon, miközben a felhasználó botok ellen játszik. A projekt célja egy oktatási célú demonstrációs alkalmazás megvalósítása, amely bemutatja a Java objektumorientált programozási eszköztárát, a Swing felhasználói felületépítést, valamint az egyszerű AI logika integrálását.

Főbb funkciók:

* Egyjátékos mód botok ellen.
* Véletlenszerű vagy felhasználó által választott karakterek és szerepek.
* Kártyák kezelése: kézben tartott, asztalra helyezett, és eldobott kártyák.
* Játékosinterakciók: támadás, védekezés, különleges akciók.
* Botlogika: minden bot szerepkörtől és kártyáktól függően dönt.
* Felhasználói felület: grafikus ablakos megjelenítés Java Swing segítségével.
* Mentés/betöltés előkészítve, de még nem teljesen implementált.

A szoftver működésének rövid összefoglalása:

A játék elindítása után a felhasználó a főmenüből új játékot indíthat, ahol kiválaszthatja a játékosok számát (önmagán kívül csak botokat), valamint beállíthatja a karaktereket és szerepeket (vagy véletlenszerűen oszthatja ki). Ezután a játékosok egymás után hajtanak végre köröket, kártyákat húznak, kijátszanak, támadnak, védekeznek. A játék automatikusan véget ér, ha valaki eléri a saját szerepkörének győzelmi feltételét.

A szoftver fejlesztése során elsődleges szempont volt a modularitás, a karbantarthatóság, valamint az objektumorientált szemlélet, különösen a karakterek, kártyák és botok kezelésében.

## Követelményelemzés

### Funkcionális követelmények

A funkcionális követelmények leírják azokat az alapvető műveleteket és képességeket, amelyeket a szoftvernek biztosítania kell a Bang! játék megfelelő implementálásához. A felhasználónak lehetősége kell legyen **új játékot indítani** a főmenüből, ahol kiválaszthatja a karaktereket és a szerepeket, vagy választhatja a véletlenszerű kiosztást is. A játék támogatja az **egyjátékos módot**, amelyben a felhasználó **gépi ellenfelek (botok) ellen** játszik. Ezek a botok önállóan, saját belső logikájuk alapján hajtják végre lépéseiket minden körben, beleértve a kártyahúzást, felszerelések kijátszását, támadást, védekezést és egyéb szükséges műveleteket.

Kulcsfontosságú a **kártyakezelés** megvalósítása: a játékosoknak képesnek kell lenniük kártyát húzni a pakliból a körük elején, kijátszani különböző típusú lapokat (mint a Bang!, Nem talált!, Sör stb.), felszereléskártyákat (fegyverek, védelmi tárgyak) az asztalukra helyezni, és szükség esetén lapokat eldobni a kezükből. A rendszernek automatikusan kell kezelnie a **játék végállapotát**, és meg kell állapítania a győztest, amint valamelyik szerepkör teljesíti a győzelmi feltételeit. Mindezekhez elengedhetetlen egy **grafikus felhasználói felület** biztosítása, amely Swing technológiával készül. Ezen a felületen a játékosnak tisztán kell látnia a saját kezében lévő kártyákat, az asztalon lévő lapokat (mind a sajátját, mind az ellenfelekét), valamint az ellenfelek életerejét. A felületnek lehetővé kell tennie a kártyák egyszerű, kattintással történő kiválasztását.

### Nem funkcionális követelmények

A nem funkcionális követelmények a rendszer működésének minőségi jellemzőire vonatkoznak. A **telepíthetőségnek egyszerűnek** kell lennie; a projekt futtatásához elegendő egy Java fejlesztői környezet (mint az IntelliJ IDEA) vagy a program parancssorból történő indítása a Main osztályon keresztül. Ehhez szükséges egy Java futtatókörnyezet (JRE 21 vagy újabb verzió)

A Swing alapú **grafikus felületnek felhasználóbarátnak** kell lennie, átlátható és logikus elrendezéssel, hogy a játékosok könnyen kezelhessék a játékot, még több ellenfél esetén is. Fontos szempont a **stabilitás és hibatűrés**, vagyis a szoftvernek képesnek kell lennie kezelni a váratlan helyzeteket és hibákat, mint például egy üres húzópakli vagy egy olyan szituáció, ahol nincs érvényes célpont. Végül, az objektumorientált tervezésnek köszönhetően a rendszernek **bővíthetőnek** kell lennie, hogy a jövőben könnyen hozzá lehessen adni új kártyákat, karaktereket vagy akár szabályvariánsokat a meglévő struktúra jelentős átalakítása nélkül.

## Tervezés

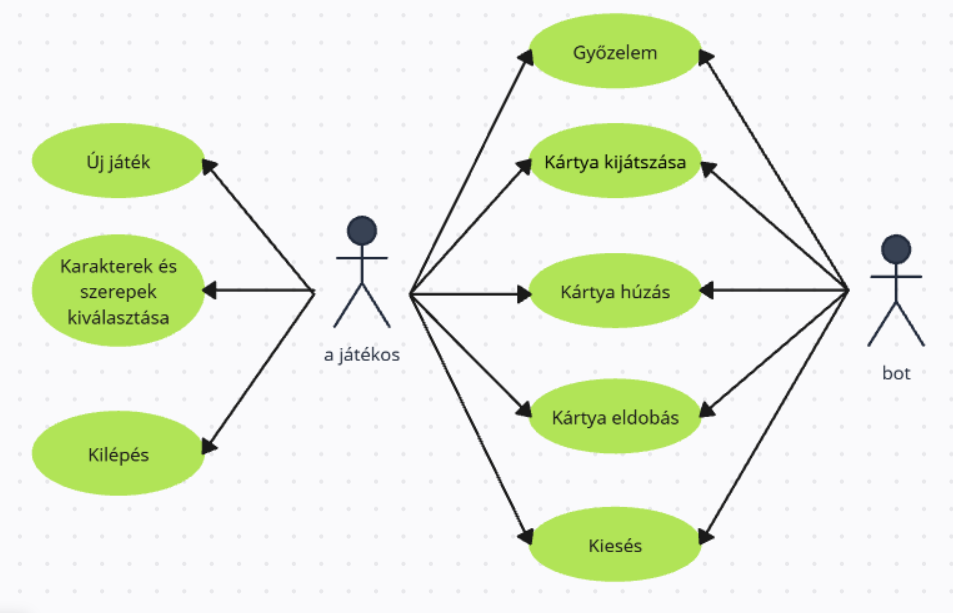
### A szoftver architektúrája

A Bang! játék megvalósítása egy háromrétegű architektúra szerint történt:

* **Prezentációs réteg (UI):**Swing alapú grafikus felhasználói felület (BangGameUI, MainMenuUI stb.), amely felelős a felhasználói interakciók kezeléséért.
* **Üzleti logika réteg (BL):**A játékmenet szabályait, a körök kezelését, győzelmi feltételek érvényesítését valósítja meg (GameLogic, Bot osztályok).
* **Adatkezelési réteg (Model):**A játék modelljeit tartalmazza, mint például a játékosokat (BaseModel leszármazottai), a kártyákat (Card és leszármazottai), a paklikat és az állapotokat.

Ez a rétegzett szerkezet biztosítja a fejlesztés átláthatóságát, az egyes rétegek független módosíthatóságát és a könnyebb karbantartást.

### Use-case diagram

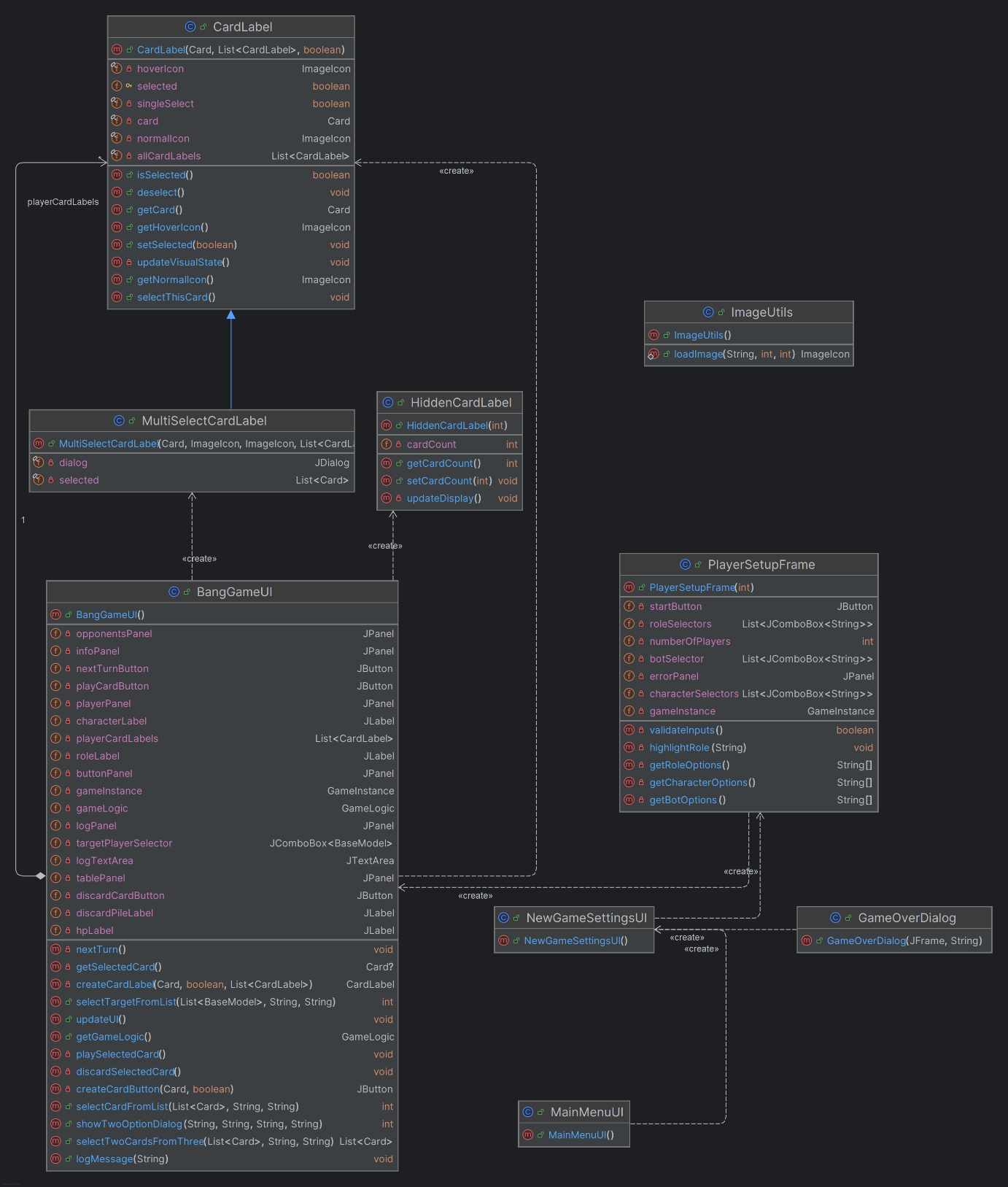


A Bang! játék egyjátékos változatában kétféle szereplő vesz részt: maga a játékos, akit a felhasználó irányít, és a bot, amely a gép által vezérelt ellenfél. Bár az irányításuk különbözik, a játék során nagyon hasonló tevékenységeket végeznek.

A játékos azzal kezdi, hogy új játékot indít, majd kiválaszt egy karaktert, amely meghatározza az életerejét és speciális képességeit. A szerepek kiosztása automatikusan történik, így nemcsak a karakter, hanem a győzelmi feltételek is egyértelműek lesznek. Miután a játék elindult, a játékos minden kör elején két kártyát húz a pakliból. Ezután dönthet arról, hogy milyen kártyákat játszik ki: támadhat, védekezhet, gyógyulhat, vagy speciális hatásokat idézhet elő. A kör végén figyelnie kell arra, hogy a kezében maradt kártyák száma ne legyen több, mint a megmaradt életereje — ha mégis több lenne, el kell dobnia a felesleges lapokat. A cél az, hogy teljesítse a szerepéhez tartozó győzelmi feltételeket, és ha sikerrel jár, megnyeri a játékot. Viszont ha az életereje nullára csökken, kiesik a játékból.

A bot, bár nem valódi játékos, szinte ugyanígy működik. A köre elején ő is húz két lapot, majd egy előre megírt logika szerint dönt arról, hogy melyik kártyát használja fel. A döntéseit egyszerű szabályok vezérlik: például ha van támadókártyája, és van támadható ellenfél, akkor nagy eséllyel azt játssza ki. A kör végén neki is el kell dobnia a felesleges kártyákat, ha szükséges. A bot is megnyerheti a játékot a saját szerepének teljesítésével, vagy kieshet, ha az életereje elfogy.

### Osztály- és objektumdiagram



Ez az osztálydiagram a Bang! játék implementációjának felhasználói felületét (UI) megvalósító Java Swing komponensek szerkezetét és kapcsolatait mutatja be. A diagram szemlélteti, hogyan épül fel a grafikus interfész az alkalmazás indításától kezdve a játékbeállításokon át egészen a tényleges játékmenet megjelenítéséig. Az itt látható osztályok mind a felhasználói interakciókért és a vizuális megjelenítésért felelősek.

A felhasználói interakció folyamata a MainMenuUI osztállyal kezdődik, amely a játék főmenüjét jeleníti meg, és innen indítható új játék. Új játék indításakor a MainMenuUI meghívja a NewGameSettingUI komponenst, amelynek feladata bekérni a játékosok számát. Ezt követően a NewGameSettingUI továbbítja a vezérlést a PlayerSetupFrame osztálynak. Ebben a keretben történik a játékosokhoz tartozó karakterek és szerepek részletes beállítása, lehetővé téve a játék testreszabását a kezdés előtt.

A beállítási fázis lezárultával aktiválódik a BangGameUI osztály, amely a felhasználói felület központi eleme és a játékmenet fő megjelenítője. Ez az osztály felelős a játék aktuális állapotának vizualizálásáért, beleértve a játékosok kezében lévő lapokat, az asztalon lévő felszereléseket, az ellenfelek adatait (pl. életerő, asztalon lévő lapok) és a játék közbeni eseményeket.

A BangGameUI működését több speciális segédosztály támogatja, amelyek szerves részét képezik a fő játékfelületnek. A CardLabel egy alapvető komponens, amelyet a BangGameUI kompozíció révén tartalmaz és kezel, felelőssége az egyes kártyalapok grafikus megjelenítése. Ebből az osztályból származik (öröklődés révén) a MultiSelectCardLabel, amely szintén a BangGameUI kompozíciójának része, és kibővíti az alap funkcionalitást azzal, hogy lehetővé teszi több kártya egyidejű kiválasztását, ami például lapok eldobásakor vagy bizonyos kártyaeffekteknél szükséges. Továbbá a BangGameUI kompozícióként tartalmazza a HiddenCardLabel komponenst is, amely olyan kártyák megjelenítésében nyújt segítséget, amelyek tartalmát a játékos nem láthatja (például a húzópakli hátlapja vagy az ellenfelek rejtett lapjai). Végül az ImageUtils egy hasznossági (utility) osztály a UI csomagon belül, amely a képfájlok betöltését és kezelését egyszerűsíti le, és vélhetően több UI komponens is függ tőle (dependency) a grafikus elemek (kártyaképek, ikonok) megjelenítéséhez.

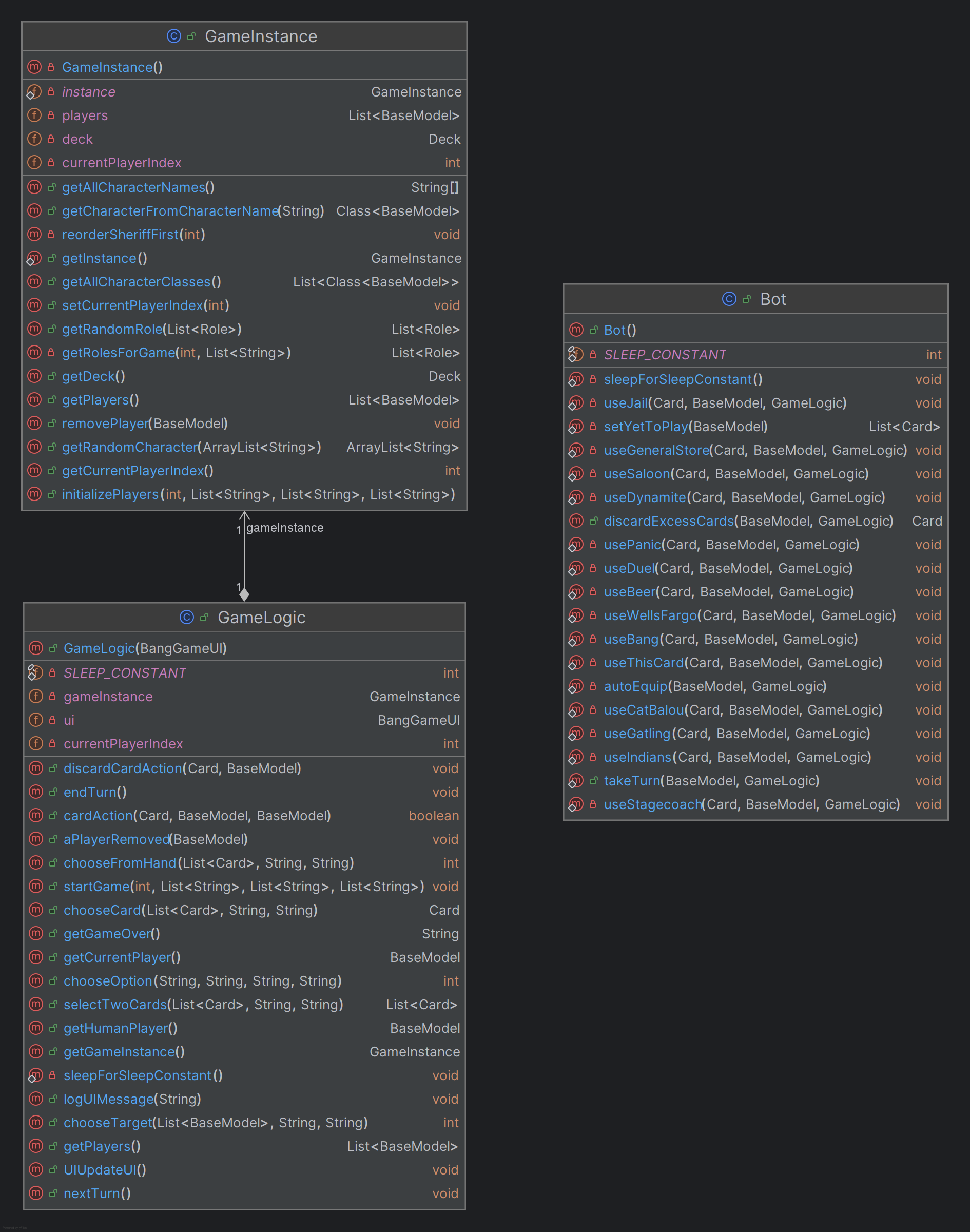
A UI struktúra kialakítása során több tudatos tervezési döntés született a jobb átláthatóság, karbantarthatóság és felhasználói élmény érdekében.

A MultiSelectCardLabel öröklődése a CardLabel osztályból lehetővé teszi a közös kártyamegjelenítési logika újrafelhasználását, miközben csak a többletfunkcionalitást (többes kiválasztás képessége) kellett implementálni a származtatott osztályban.

A BangGameUI és a különböző Label típusok ( CardLabel, MultiSelectCardLabel, HiddenCardLabel) közötti kompozíciós kapcsolat biztosítja, hogy ezek a vizuális elemek szorosan a fő játékfelülethez tartozzanak. A BangGameUI felelős ezen komponensek létrehozásáért és életciklusuk kezeléséért, ami egyértelműsíti az objektumok közötti hierarchiát és megkönnyíti az erőforrás-kezelést, mivel a "rész" objektumok a "teljes" objektummal együtt jönnek létre és szűnnek meg.

Az ImageUtils hasznossági osztály bevezetése központosítja a képkezelési feladatokat (betöltés, átméretezés stb.). Ezáltal elkerülhető a kódduplikáció a különböző UI komponensekben, amelyek képeket használnak, valamint egységes felületet biztosít a képekkel kapcsolatos műveletekhez, elősegítve a karbantarthatóságot.

A játékbeállítási folyamat ( MainMenuUI -> NewGameSettingUI -> PlayerSetupFrame) lépésekre bontása külön ablakokba/panelekbe tudatos döntés volt. Egyrészt felhasználóbarát, vezetett élményt nyújt a játékosnak a konfiguráció során. Másrészt megakadályozza, hogy a felhasználó egyszerre túl sok beállítási opcióval legyen terhelve. Harmadrészt pedig logikailag elkülöníti a játék előkészítésének feladatait a tényleges játékmenet megjelenítésétől, ami tisztább kódszerkezetet eredményez.



A csomag egyik kulcsfontosságú eleme a GameInstance osztály. Ennek az osztálynak a fő feladata, hogy a játék aktuális állapotának központi tárolójaként funkcionáljon. Itt tárolódnak a játékmenet szempontjából létfontosságú adatok, mint például a játékban részt vevő játékosok listája (players), a kártyapakli (deck) és az aktuálisan soron következő játékos indexe (currentPlayerIndex). Fontos megjegyezni, hogy a GameInstance Singleton tervezési mintát valósít meg, biztosítva ezzel, hogy az alkalmazásban mindig csak egyetlen példánya létezzen, így garantálva a játékállapot konzisztenciáját és egységes elérhetőségét.

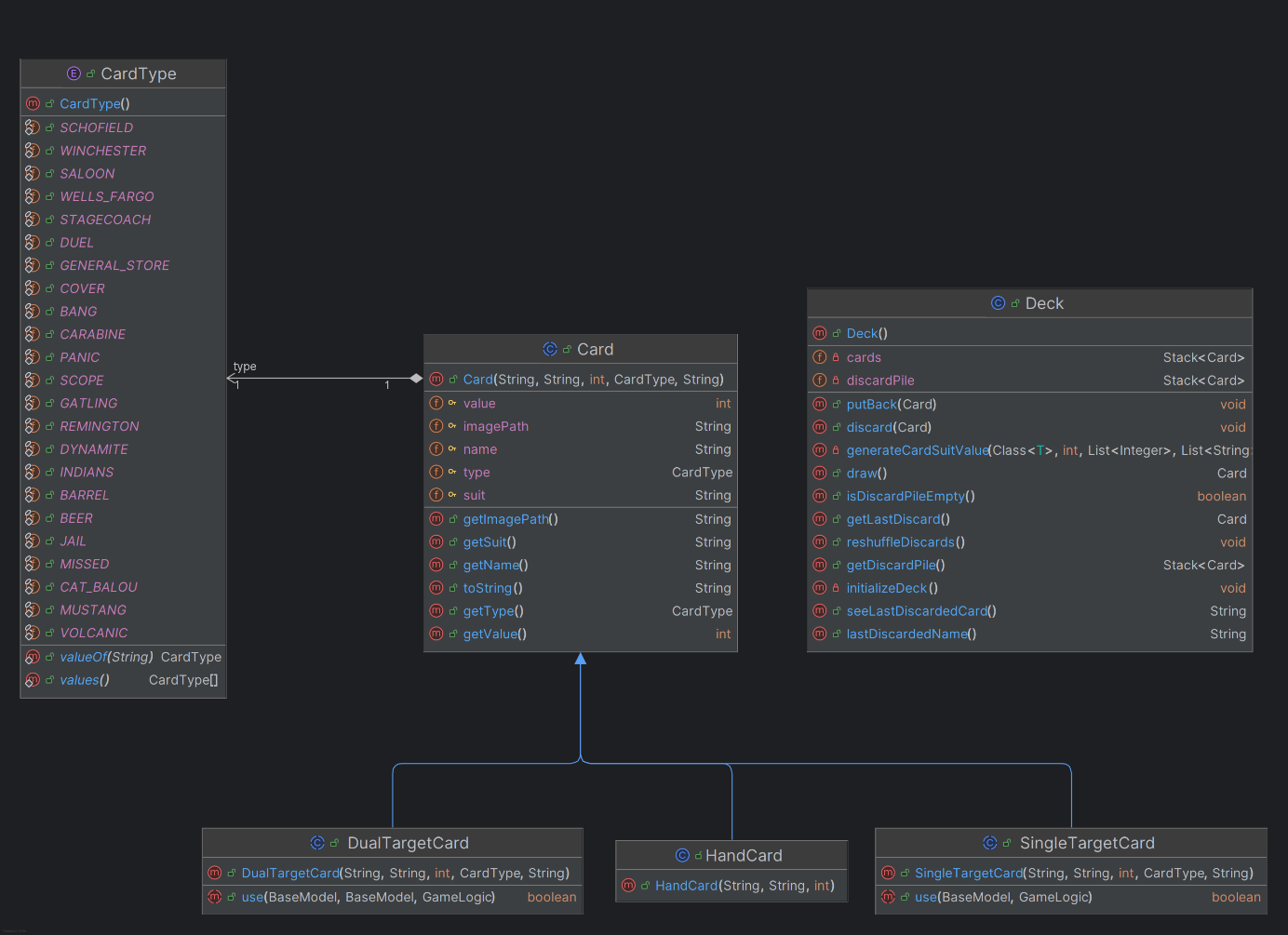
A csomag másik központi osztálya a GameLogic, amely az üzleti logika oroszlánrészét tartalmazza. Ez az osztály felelős a játékszabályok érvényesítéséért és a játékmenet dinamikájának irányításáért. Feladatai közé tartozik a játékosok akcióinak feldolgozása (pl. kártyahúzás, lap kijátszása), a körök léptetése, és a játék kimenetelének meghatározása (pl. győzelmi feltételek). A GameLogic szorosan együttműködik a GameInstance osztállyal az állapot lekérdezése és módosítása érdekében.

A harmadik osztály ebben a csomagban a Bot, amely a mesterséges intelligenciával vezérelt játékosok döntéshozatali logikáját foglalja magába. Amikor egy bot játékos kerül sorra, a GameLogic ennek az osztálynak a főmetódusait hívja meg, hogy meghatározza a bot lépéseit (milyen kártyát játsszon ki, kit célozzon meg stb.).

Az állapot (GameInstance) és a logika (GameLogic) szétválasztása egyértelműbbé teszi a rendszer felépítését. A GameInstance csak az adatokat tárolja, míg a GameLogic az ezeken az adatokon operáló szabályokat és folyamatokat tartalmazza. Ez a szétválasztás javítja a kód olvashatóságát, megkönnyíti a GameLogic tesztelését (akár különböző állapotokkal), és növeli a karbantarthatóságot.

A bot logikájának elkülönítése a Bot osztályba lehetővé teszi a mesterséges intelligencia viselkedésének egységbe zárását (enkapszuláció). Ez megkönnyíti a különböző nehézségű vagy stratégiájú botok implementálását (pl. további alosztályok létrehozásával vagy Strategy minta alkalmazásával) anélkül, hogy a GameLogic osztályt bonyolítanánk az AI specifikus részleteivel.

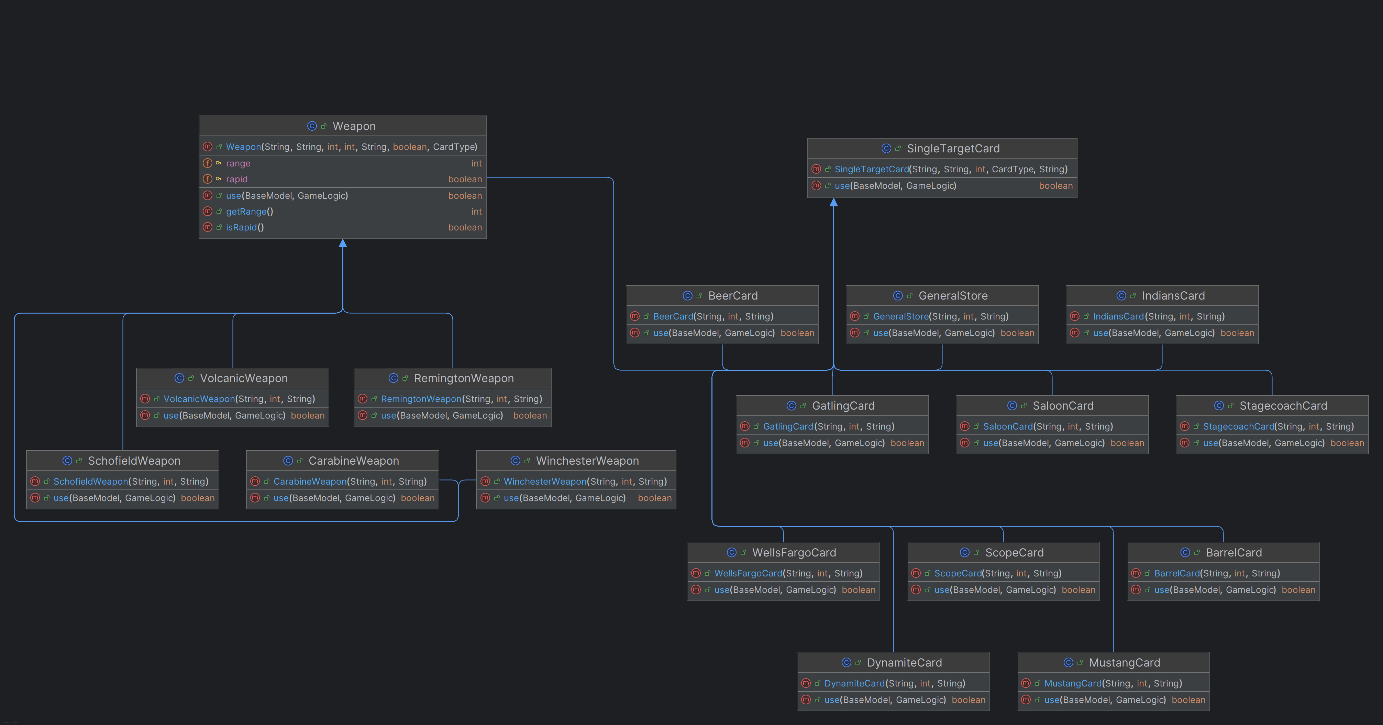
Az üzleti logika (bl) csomagba szervezés maga is egy tervezési döntés, amely a "Separation of Concerns" elvét követi, elkülönítve a játék mag logikáját a megjelenítési rétegtől (UI).



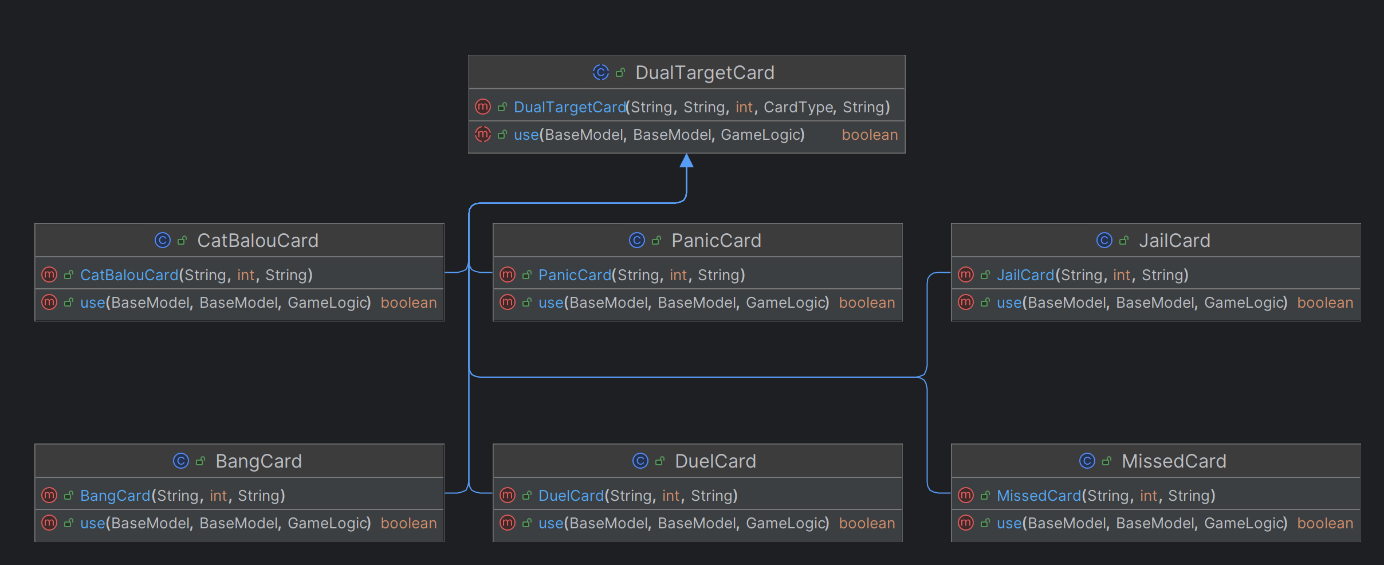
Ez az osztálydiagram a Bang! játék kártyáinak és a kártyapaklinak a modellezésére szolgáló osztályokat mutatja be, amelyek a model/cards csomagban helyezkednek el. Ez a struktúra definiálja a játékban használt kártyák alapvető tulajdonságait, típusait és a pakli kezelésének módját, alapot szolgáltatva ezzel a játék központi elemeinek.

A diagram középpontjában a Card osztály áll, amely egy absztrakt ősosztályként funkcionál minden egyes kártyatípus számára. Ez az osztály tartalmazza a kártyák közös jellemzőit (például név, szín), és itt kerül felhasználásra a CardType enum is, amely a kártyák típusának (pl. BANG, MISSED, BEER) kategorizálására szolgál. Ez az enum segíti a kártyák egyértelmű azonosítását és a rájuk vonatkozó szabályok alkalmazását a játéklogikában. A csomag másik fontos eleme a Deck osztály, amely a húzó- és dobópaklit reprezentálja. Felelőssége a Card objektumok létrehozása és a húzó- és dobópakli tárolása, valamint ezekkel kapcsolatos műveletek biztosítása, mint a keverés és a lapok húzása.

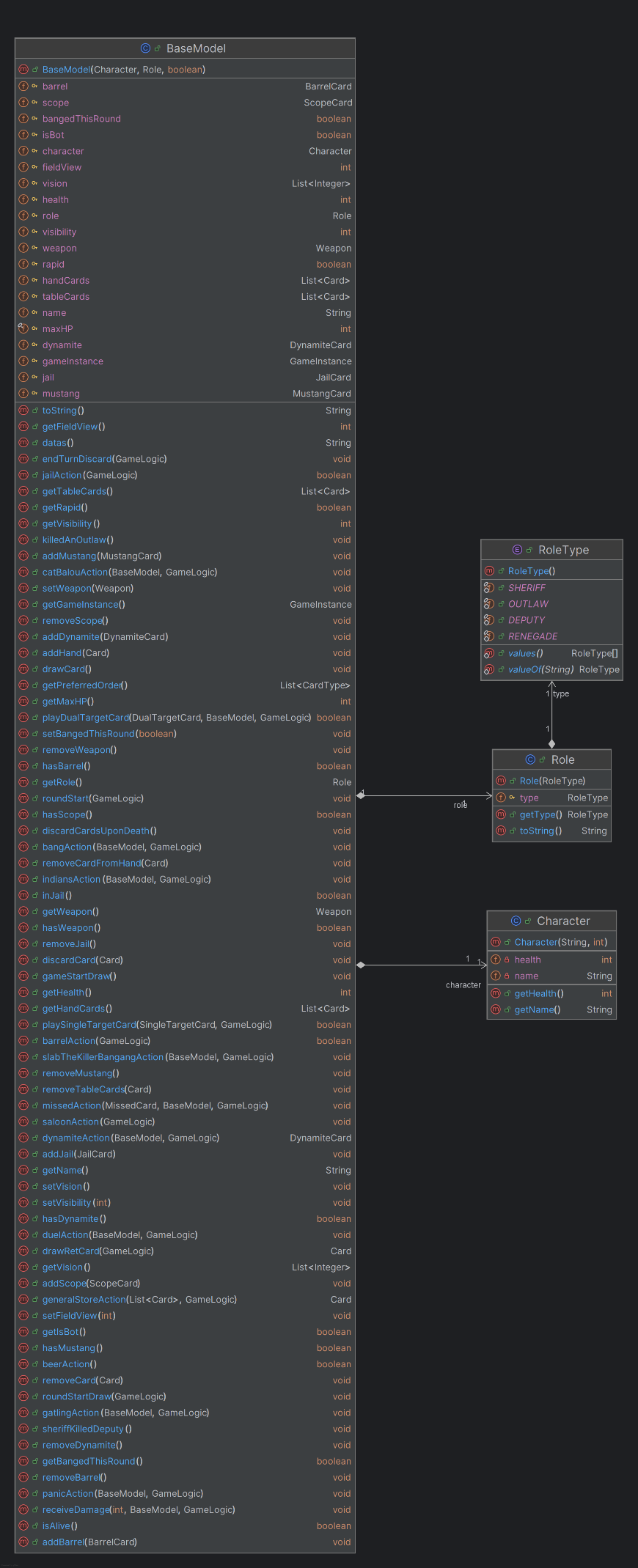
A Card osztályból több specifikusabb kártyatípus is származik öröklődés révén, amelyek a kártyák eltérő viselkedését vagy célzási mechanizmusát modellezik. A SingleTargetCard olyan kártyákat képvisel, amelyek kijátszásához egyetlen célpont (játékos) megadása szükséges. A DualTargetCard azokat a kártyákat modellezi, amelyek két célpontot igényelnek. A HandCard pedig azokat a kártyákat testesíti meg, amik betöltő szerepet töltenek be (pl. választani kell egy véletlenszerű kártyát az ellenfél kezéből). Ez az öröklődési hierarchia lehetővé teszi, hogy a közös kártyatulajdonságok az ősosztályban legyenek definiálva, míg a specifikus viselkedés vagy célzási logika a leszármazott osztályokban kapjon helyet. A Deck és a Card osztályok között aggregációs kapcsolat áll fenn, mivel a Deck tartalmazza és kezeli a Card példányok egy csoportját. Az öröklődési hierarchia alkalmazása a Card osztályból kiindulva lehetővé teszi a polimorfizmus kihasználását (a különböző kártyatípusok egységesen Card-ként kezelhetők, miközben a saját specifikus use(). Emellett elősegíti a kód újrafelhasználását (közös attribútumok az ősosztályban) és a rendszer bővíthetőségét új kártyatípusok egyszerű hozzáadásával. A CardType Enum használata típustiszta és jól olvasható módot biztosít a kártyák kategorizálására, ellentétben például szöveges konstansok használatával. Ez megkönnyíti a kártyatípus-alapú logikai elágazások (pl. if elágazások) implementálását a GameLogic-ban. A Deck osztály elkülönítése egységbe zárja a paklival kapcsolatos összes műveletet (keverés, húzás, üres pakli kezelése). Ez a felelősségi körök tiszta szétválasztását eredményezi, mivel a pakli kezelésének logikája nem keveredik a kártyák egyedi tulajdonságaival vagy a játék általános menetével. A model/cards csomag létrehozása logikailag csoportosítja a kártyák és a pakli reprezentációjához kapcsolódó osztályokat, javítva ezzel a projekt általános struktúráját és átláthatóságát.



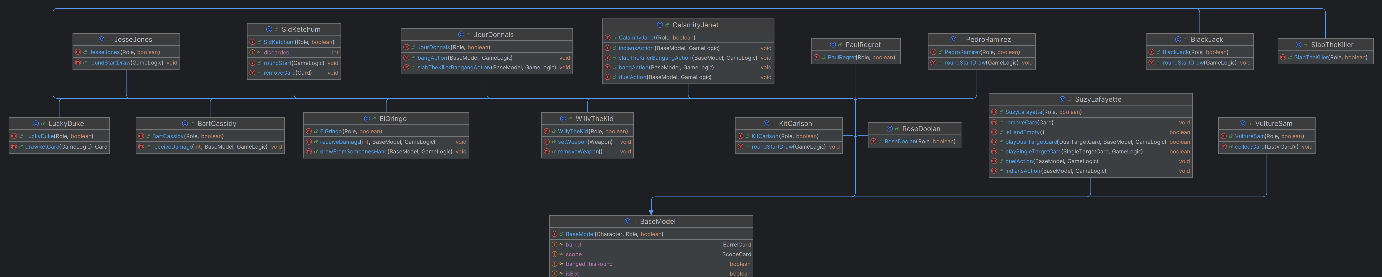
Ez a diagram a kártyamodell egy specifikus részletét mutatja be, kiemelve azokat a kártyatípusokat, amelyek a SingleTargetCard osztályból származnak, és amelyek kijátszásához vagy hatásának érvényesítéséhez egyetlen célpont szükséges.A diagramon látható, hogy a Weapon (Fegyver) osztály a SingleTargetCard egy specializációja. Ebből a Weapon osztályból további öt konkrét fegyverkártya-osztály öröklődik (például Schofield, Volcanic, stb.), amelyek a különböző lőtávolságú és képességű fegyvereket modellezik a játékban.A fegyvereken kívül további tizenegy különböző kártyaosztály (mint például a Sör vagy Indiánok!) is közvetlenül a SingleTargetCard osztályból származik. Ezek a kártyák testesítik meg azokat a különféle akciókat és hatásokat, amelyek egyetlen kiválasztott célpontra irányulnak vagy az összes játékosra.



Ez a diagram a kártyamodell azon részét ábrázolja, amely a DualTargetCard osztályból leszármazó kártyákat foglalja magában. Ezeket a kártyákat az jellemzi, hogy hatásuk érvényesítéséhez vagy kijátszásukhoz két célpont (ezek közül az egyik a lapot kijátszó játékos, másik a szó szerinti célpont) megjelölése szükséges. A diagram szerint hat különböző, konkrét kártyaosztály öröklődik közvetlenül a DualTargetCard ősosztályból. Mindegyik ilyen alosztály egyedi játékbeli akciót vagy hatást valósít meg, amelynek közös tulajdonsága a két célpontot igénylő mechanizmus. Az öröklődési struktúra itt is azt a célt szolgálja, hogy a két célpont kezelésével kapcsolatos közös logikát és adatokat a DualTargetCard bázisosztály tartalmazza.



Ez az osztálydiagram a model/utilities csomagban található alapvető osztályokat és típusokat mutatja be. Ezek az elemek kulcsfontosságúak a játékosok alapvető attribútumainak, mint a karakter és a szerepkör, egységes és strukturált reprezentációjához. A csomag központi eleme a BaseModel osztály, amely – ahogy a neve is sugallja – alapul szolgál a játék más, összetettebb entitásai számára, és kiemelt fontossággal bír a projekt egészében. A BaseModel egy alaposztály, amely az egyedi karakter osztályok ősosztályaként funkcionál. Elsődleges felelőssége, hogy közös struktúrát és alapvető funkcionalitást biztosítson a leszármazottak számára. A kompozíció alkalmazása a Character és Role objektumok BaseModel-en belüli tárolására (ahelyett, hogy például a karakter osztályok mindháromból öröklődne) természetesen modellezi a "van neki" (has-a) kapcsolatot. Egy játékosnak van egy karaktere és van egy szerepköre. Ez a megközelítés rugalmasabb és tisztább, mint a többszörös öröklődés, és jobban elválasztja az egyes koncepciókat. A Character osztály segíti a játékos által megszemélyesített konkrét karakter (pl. Bart Cassidy, Jesse Jones) ábrázolását. Ez az osztály tárolja a karakter nevét és maximális életerejét. A Role osztály a játékos titkos szerepkörét képviseli, amely alapvetően meghatározza a játékos győzelmi feltételeit. A Role osztály a RoleType enumot használja a konkrét szerepkör típusának tárolására. A RoleType egy enumeráció (enum), amely a játékban előforduló lehetséges szerepkörök zárt halmazát definiálja ( SHERIFF, DEPUTY, OUTLAW, RENEGADE). A RoleType Enum használata a szerepkörök definiálására növeli a típusbiztonságot, megakadályozza az elírásokat, és olvashatóbbá, könnyebben karbantarthatóvá teszi a kódot, összehasonlítva például sztringek vagy numerikus értékek használatával.



Ez az osztálydiagram a BaseModel osztályt és annak közvetlen leszármazottait, a játékban szereplő 16 konkrét karaktert mutatja be. A BaseModel itt mint alapvető osztály jelenik meg, amely meghatározza a karakterek közös struktúráját és alapvető tulajdonságait. A diagramon látható tizenhat specifikus osztály (például BartCassidy, SlabTheKiller, stb.) mind a BaseModel-ből öröklődik. Mindegyik ilyen osztály egy-egy egyedi karaktert testesít meg, definiálva annak kezdő életerejét és nevét, illetve implementálva a karakterre jellemző különleges képességeket, a BaseModel-ben definiált metódusok felülírásával. Ez az öröklődési hierarchia biztosítja, hogy minden karakter rendelkezzen a szükséges alapvető attribútumokkal, miközben lehetővé teszi az egyedi viselkedések és tulajdonságok egységbe zárását (enkapszulációját) az adott karakterhez tartozó osztályban. A struktúra így hatékonyan modellezi a Bang! játék változatos karaktereit.

## Megvalósítás

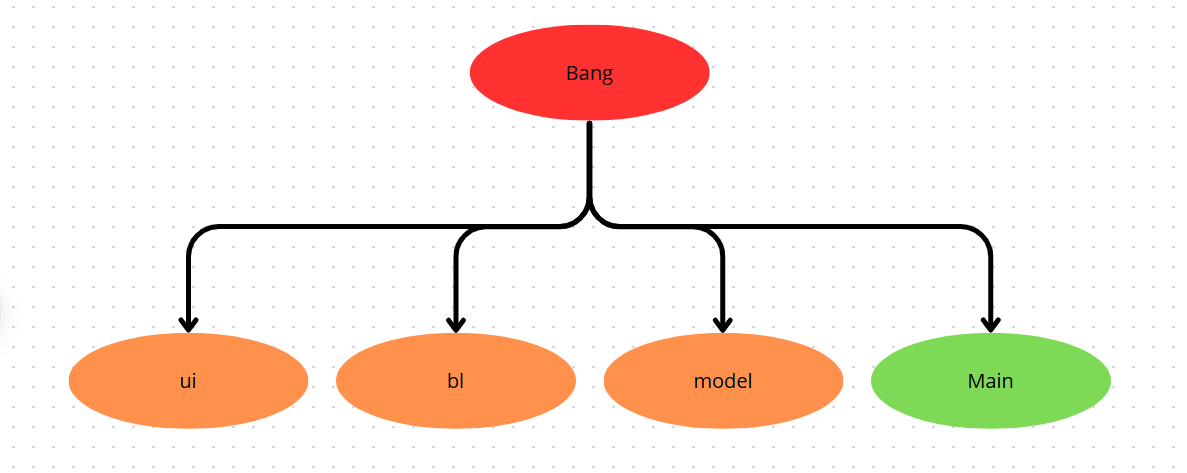
### Használt technológiák

A projekt megvalósítása során többféle technológia és eszköz került felhasználásra. Az alkalmazás alapját a Java 21 programozási nyelv adta, a fejlesztés pedig az IntelliJ IDEA Community Edition integrált fejlesztőkörnyezetben zajlott. A játék grafikus felhasználói felületének (GUI) kialakításához és a játékos interakcióinak kezeléséhez a beépített Java Swing könyvtárat alkalmaztam. A fejlesztési folyamat követhetőségének biztosítása és a verziók hatékony kezelése érdekében a Git verziókezelő rendszert használtam, a kód távoli tárolására és megosztására pedig a GitHub platformot vettem igénybe. Ez a megközelítés lehetővé tette, hogy rendszeres mentéseket készítsek, így a fejlesztési szakaszok könnyen nyomon követhetővé váltak, és szükség esetén lehetőség nyílt korábbi állapotokra való visszatérésre. A projekt dokumentációjának elkészítése során a diagramok vizualizációjára a PlantUML eszközt, míg az írásos dokumentum összeállítására a Microsoft Word szövegszerkesztőt alkalmaztam. Fontos megjegyezni, hogy a projekt nem használ külön build-rendszert (mint például Maven vagy Gradle), hanem közvetlenül IntelliJ projektként jött létre és futtatható. A Java Swing keretrendszer biztosítja a teljes grafikus felhasználói felületet (GUI), amelyen keresztül a játékos a játékot vezérli. A Git verziókezelővel a fejlesztési folyamat során rendszeres mentéseket készítettem, hogy a fejlesztési szakaszok könnyen nyomon követhetők legyenek, és szükség esetén korábbi állapotokra visszatérhessek.

## Projekt szerkezete, CI/CD

### Projekt szerkezete

A projekt egy egyszerű, IntelliJ IDEA által kezelt, tiszta Java projekt, amely nem használ külön buildrendszert (pl. Maven, Gradle). A projekt fájlszerkezete áttekinthető, a főbb csomagok logikailag jól elkülönítve tartalmazzák a különböző rétegekhez tartozó osztályokat:



* A banggame.model csomag tartalmazza a játék alapvető adatszerkezeteit: játékosok, kártyák, szerepkörök és kezeli az adott adatszerkezethoz vonatkozó szabályok zömét.
* A banggame.bl (business logic) csomag kezeli a játék inicializálását. Illetve, a játékmenet fő szabályait és folyamatát. Többek között a körök kezelését, kártyák húzását, és győzelmi feltétel esetleges teljesülését.
* A banggame.ui csomag tartalmazza a grafikus felhasználói felület összes elemét: panelek, ablakok, kezelőgombok.
* A Main.java osztályban történik a program inicializálása és a felhasználói felület elindítása.

### CI/CD folyamat

A projekt jelenlegi fejlesztési szakaszában nem került bevezetésre automatizált CI/CD (Continuous Integration/Continuous Deployment) rendszer. Ennek oka elsősorban a projekt jelenlegi mérete és az egyéni fejlesztői munkafolyamat, amely mellett az automatizálás előnyei még nem kerülnének teljes mértékben kihasználásra.

A fejlesztési workflow ennek megfelelően manuális lépésekből állt. A forráskód módosításai helyileg, az IntelliJ IDEA fejlesztőkörnyezetben történtek. A kód integritásának és a változások követhetőségének biztosítása érdekében a módosításokat rendszeresen mentettem a Git verziókezelő rendszer segítségével, majd a változtatásokat feltöltöttem a projekt GitHubon található távoli repositoryjába. A szoftver buildelése (fordítása) és futtatása is közvetlenül az IntelliJ IDEA beépített eszközeivel történt, külső build-automatizáló eszközök, mint a Maven vagy Gradle, használata nélkül. A funkcionalitás ellenőrzése, nagy mértékben manuálisan zajlott, elsősorban a grafikus felhasználói felületen keresztüli interakciókkal és a játékmenet kipróbálásával.

Bár jelenleg a folyamatok manuálisak, a projekt architektúrája és a verziókezelés használata megalapozza a jövőbeli automatizálási lehetőségeket. Amennyiben a projekt továbbfejlődik, vagy csapatmunka kerül bevezetésre, lehetőség nyílik CI/CD eszközök integrálására. Ilyen lehet például a GitHub Actions bevezetése, amely automatizálhatná a kód fordítását minden módosítás után, futtathatná az esetlegesen később hozzáadott egységteszteket (unit testeket), és kiépíthetne egy teljes CI/CD pipeline-t a fejlesztési folyamat felgyorsítása és a minőségbiztosítás javítása érdekében.

## Kódbázis felépítése, könyvtárszerkezet, forrásfájlok

A projekt forráskódja átlátható szerkezetben helyezkedik el, követve az MVC (Model-View-Controller) architektúra alapelveit. A csomagstruktúra célja, hogy elkülönítse a különböző funkcionális területeket: a megjelenítést, az üzleti logikát és az adatmodelleket.

### Főkategóriák

* banggame.model – Adatmodellek
* banggame.bl – Üzleti logika
* banggame.ui – Felhasználói felület
* Main.java – Belépési pont

#### banggame.model

Ez a csomag tartalmazza az összes osztályt, amelyek a játék adatainak tárolásáért felelősek. Itt találhatók a következők:

* BaseModel – Az összes játékos közös ősszülője. Tartalmazza az életerőt, kézben lévő kártyákat, asztali kártyákat stb.
* Character – Segíti a különböző karakterek leszármaztatását.
* BaseModel-ből származtatott osztályok – A 16 különféle karakter, mindegyik egyedi képességekkel.
* Card – Egy általános kártyát reprezentáló osztály.
* SingleTargetCard – A Card-ból származik, azon kártyák ősosztálya, melyeknek nem kell célpont a kijátszásukhoz.
* DualTargetCard – A Card-ból származik, azon kártyák ősosztálya, melyek egy bizonyos célpontra irányulnak.
* Card-ból származtatott osztályok – 22 egyedi Card osztály, különbözően felülírt függvényekkel.
* Role – Enum típusú osztály a szerepkörök kezelésére.
* RoleType – Enum típus a különböző szerepek definiálására (Sheriff, Outlaw, Renegade, Deputy).
* CardType – Enum típus a különböző kártyák definiálására (Bang!, Nem talált!, stb.).
* Deck – Ez az osztály generálja le a kártyapaklit és tartja számon a húzó- és dobópakli tartalmát.

Ezek az osztályok főként adatszerkezetek, getterekkel, setterekkel, és üzleti logikával.

#### banggame.bl

Az üzleti logika réteg, amely a játékmenet szabályait és folyamatait kezeli. A legfontosabb elemei:

* GameLogic – A játék fő vezérlője. Kezeli a köröket, a játékosok lépéseit, kártyák használatát, győzelem ellenőrzését.
* Bot – Az egyszerű mesterséges intelligencia, amely a botok körének automatikus lejátszását vezérli.
* GameInstance – Létrehozza a játékosokat és a kártypaklit. Illetve, a játék állapotát tárolja, aktív játékosok és a létrehozott Deck osztály.

Ez a réteg tartalmaz minden fontos szabályt és belső mechanizmust, amik a Bang! játékmenetét szimulálják.

#### banggame.ui

A felhasználói felületet megvalósító osztályok találhatóak itt, Swing alapokon. Főbb elemek:

* BangGameUI – A fő ablak, amely tartalmazza az összes vizuális elemet: játékos panelek, kártyák, logpanel, gombok.
* ImageUtils – Segít a felhasználói felület képeinek beállításában.
* MultiSelectCardLabel – Kezeli több kártya egyidejű kiválasztását.
* CardLabel, HiddenCardLabel – Különleges JLabel-ek, amelyek a kártyák megjelenítését (látható / rejtett módon) kezelik.
* MainMenuUI – A főmenü, ahol új játék indítható vagy a programból kiléphetünk.
* NewGameSettingsUI – Az új játék indításakor a játékosok szám választó ablak, ez hívja meg a PlayerSetupUI-t.
* PlayerSetupUI – Az új játék indításakor a karakter- és szerepkörválasztó ablak.
* GameOverDialog – A játék végén megadj a lehetőséget az új játék indítására.

Ezek az osztályok felelősek az eseménykezelésért is (pl. gombkattintásokra reagálás).

#### Main.java

A belépési pont, amely inicializálja az alkalmazást. Feladata:

* A program elindítása (main() metódus).
* A főmenü (MainMenuUI) megnyitása.

### Általános jellemzők

Kódolási stílus: A kód következetes névkonvenciókat követ (camelCase metódusok, PascalCase osztályok).

Egyszerűsített függőségek: A projekt kizárólag a Java SE beépített csomagjait használja (pl. javax.swing, java.util).

### Kód egyes elemeinek bemutatása

A projekt felépítése során végig törekedtem arra, hogy az egyes osztályok világosan elkülönüljenek egymástól, minden osztály jól körül határolható felelősségi körrel rendelkezzen. Ennek köszönhetően a kód átlátható marad, és bővítés, illetve karbantartás esetén is könnyen kezelhető.

A bl.GameInstance osztály az alkalmazás központi eleme, amely a Bang! játék egy adott futása során létrejövő egyetlen játékmenet teljes, aktuális állapotát tartalmazza és kezeli. Ez az osztály felelős azért, hogy egységes és konzisztens képet biztosítson a játék állapotáról, beleértve a résztvevő játékosokat (BaseModel leszármazott objektumok listájaként tárolva), a húzópaklit (Deck), valamint az információt arról, hogy éppen melyik játékos következik (currentPlayerIndex). Lényegében ez az osztály a játék "memóriája". Az osztály a Singleton tervezési mintát követi, amit a privát konstruktor és a statikus getInstance() metódus biztosít. Erre azért volt szükség, mert egy programfutás alatt csak egyetlen játékállapot létezhet, és a Singleton garantálja, hogy ebből az osztályból csupán egyetlen példány jöjjön létre. Ez biztosítja az állapot konzisztenciáját és könnyű, globális hozzáférést nyújt az üzleti logikai réteg (és közvetve más rétegek) számára anélkül, hogy az GameInstance objektumot explicit módon kellene továbbadni. Az osztály tartalmazza a játék inicializálásának logikáját is (initializePlayers), amely felelős a játékosok számának megfelelő karakterek és szerepkörök kiosztásáért (akár véletlenszerűen, akár a felhasználói felületről kapott beállítások alapján), valamint a játékosok listájának átrendezéséért, hogy a Seriff kezdjen. A GameInstance szorosan együttműködik a GameLogic osztállyal, amely innen olvassa ki az aktuális állapotot a döntéseihez és ide írja vissza a játékosok akcióiból vagy a játékmenetből következő állapotváltozásokat. A Bot osztály (gépi ellenfelek logikája) szintén felhasználja a GameInstance-ben tárolt adatokat a lépéseinek megtervezéséhez. Az osztály függ a model csomag több elemétől is, mint például a BaseModel, Role, RoleType, a konkrét karakterosztályok (model.utilities.characters.\*) és a Deck. A felhasználói felület (UI) réteg közvetlenül nem lép vele interakcióba, de az általa megjelenített adatok forrása végső soron a GameInstance-ben tárolt állapot.

A bl.GameLogic osztály a Bang! alkalmazás központi vezérlője, amely az üzleti logika magját képezi. Felelős a játékmenet levezényléséért, a játékszabályok betartatásáért, a játékosok akcióinak feldolgozásáért, a körök kezeléséért és a játék végállapotának figyeléséért. Ez az osztály közvetít a játék állapotát tároló GameInstance és a felhasználói interakciókat kezelő BangGameUI között.

A GameLogic egy központi orchestrátor osztályként működik, amely logikailag elkülönül a játék állapotától (GameInstance) és a megjelenítéstől (BangGameUI). Létrehozásakor megkapja a BangGameUI példányát, amelyen keresztül kommunikál a felhasználóval és frissíti a felületet. Nem követ szigorúan egyetlen komplex tervezési mintát, de a felelősségi körök szétválasztása egyértelmű. A játékmenet egyes pontjain (pl. dinamit, börtön) szekvenciálisan kezeli az eseményeket, és beépített késleltetést (sleepForSleepConstant) is alkalmazhat a jobb követhetőség érdekében.

A GameLogic alapvetően meghatározza a játék működését, szorosan együttműködve több más komponenssel:

* GameInstance: Folyamatosan használja a Singleton GameInstance-t az aktuális játékállapot (játékosok, pakli, soron következő játékos) lekérdezésére és módosítására.
* BangGameUI: Kommunikál a konstruktorban kapott UI példánnyal. Üzeneteket küld a felhasználónak (logUIMessage), frissíti a felületet (UIUpdateUI), és bekéri a felhasználói döntéseket (pl. kártya-, célpont-, opcióválasztás a choose\* metódusokon keresztül). A játék végét is itt jelzi ki (GameOverDialog).
* BaseModel (és leszármazottai): Meghívja a játékosokat reprezentáló BaseModel objektumok metódusait a körök különböző fázisaiban (pl. gameStartDraw, dynamiteAction, endTurnDiscard).
* Bot: Amikor egy mesterséges intelligencia által vezérelt játékos kerül sorra, a GameLogic delegálja a vezérlést a statikus Bot.takeTurn metódusnak.
* Card (és leszármazottai): Ismeri és kezeli a különböző kártyatípusokat (SingleTargetCard, DualTargetCard, stb.), például típusellenőrzéssel (instanceof) dönt a szükséges további interakciókról (pl. célpont kérése).

A Bang! játék logikus, jól strukturált csomagokra és osztályokra van felosztva, így átlátható mind a játék logikája, mind a felhasználói felület működése. A főbb csomagok közé tartozik a bl, model, ui, illetve néhány általános, a program egészére ható osztály is.

A bl csomag tartalmazza a játék fő mozgatórugóját, a GameLogic osztályt. Ez az osztály felel a játék állapotának kezeléséért, például a körök léptetéséért, a győztes meghatározásáért, , és a különféle események (például dinamit- vagy börtönhatás) lebonyolításáért. Itt található a játékosok listája, a jelenlegi játékos indexe, illetve maga A játékmenet is ennek az osztálynak a vezérlése alatt fut. A bl csomag tartalmazza még a GameInstance osztályt. Ez felel a játék felépítéséért. Létrehozza és tárolja a játékosokat a megadott paraméterek (konkrét vagy véletlenszerű karaktereket és szerepek) alapján. És tárolja a kártyapaklit. A botok működését a Bot osztály fogja össze. Ez az osztály tartalmazza a botok döntéshozatali logikáját, hogy körük során milyen lépéseket hajtsanak végre: milyen kártyát játsszanak ki, kit támadjanak meg, vagy éppen milyen fegyvert tegyenek le.

A model csomag a játék világát modellezi, ide tartoznak a karakterek, a szerepek, a különféle kártyák, valamint a játékosokat leíró adatszerkezetek is.

Az egyik központi osztály a BaseModel, amely egy játékost ír le, akár emberi, akár gépi ellenfélről van szó. Ez az osztály kezeli a játékos életerejét, kézben lévő és asztalon lévő, aktív, kártyáit. Illetve lebonyolítja a játékosra ható, a célpontra irányuló részét, a kijátszott kártyának. A játék során fontos szerepe van a karaktereknek is, így külön fájlokban található meg például Jesse Jones, Slab the Killer, vagy éppen Suzy Lafayette karakterének leírása. Ezek az osztályok a karakterek egyedi képességeit definiálják, és mind a BaseModel ősosztályból öröklődnek. Szintén a model csomagban kapott helyet a Deck osztály, amely a kártyapaklit és a dobott lapok halmazát kezeli. A Deck felelős a kártyák keveréséért, húzásáért, eldobásáért, valamint újrakeverésért, ha elfogyott a húzópakli.

A model csomagban még helyet kapott a Card osztály, amelyből minden kártya származik. Kettő fő részre oszlik, SingleTargetCard és DualTargetCard osztályokra. Fő különbség köztük, hogy a SingleTargetCard, aminek a neve először megtévesztőnek tűnhet, csak a kijátszó játékost kapja paraméterül, mert csak rajta, vagy azon keresztül hajtódik végre a kártya képessége. Ide tartozik a fegyver és a többi kék színű kártya zöme, illetve a barna színű kártyák körülbelül fele. A DualTargetCard paraméterül kapja a feladót és a célpontot, innen az osztály neve. Illetve még a Card osztályból öröklődik egy HandCard, amit akkor használ a program, hogyha szükség van egy filler (helykitöltő) kártyára. Külön érdemes megemlíteni a Role és Card enumerációkat, amelyek a szerepeket és a kártyák nevét tárolják szabványos formában.

A ui csomag tartalmazza a grafikus felületet kezelő osztályokat. A legfontosabb közülük a BangGameUI, amely az ablakot, a panelek elrendezését és a különböző események (gombnyomások, kártyaválasztások) kezelését végzi. Itt találhatók a játékos kezében lévő kártyák megjelenítéséért felelős CardLabel objektumok is, valamint olyan egyedi komponensek, mint a HiddenCardLabel, amely az ellenfelek kezében lévő, ismeretlen kártyák számát mutatja. Illetve a MainMenuUI, amelynél új játékot lehet indítani. A NewGameSettingsUI, ahol ki lehet választani a játékosok számát. A PlayerSetupFrame, ahol be lehet állítani a különöző játékosok karaktereit és szerepeit. És a GameOverDialog, ami egy győzelmi feltétel teljesülése után jelenik meg.

# Összefoglalás és további fejlesztési lehetőségek

# Irodalomjegyzék

# Melléklet

1. Néhány karakter speciális képessége ezt felülírja. Lásd: 2.2.4 Karakterek [↑](#footnote-ref-2)