Programozási technológia
3. Beadandó

Név: Magyar Viktor

Neptun kód: O9WEJJ

Dátum: 2024.05.10.

Feladat száma: 3

1. Feladat leírása:

Készítsünk programot, amellyel egy labirintusból való kijutást játszhatunk. A játékos a labirintus bal alsó sarkában kezd, és a feladata, hogy minél előbb eljusson a jobb felső sarokba úgy, hogy négy irányba (balra, jobbra, fel, vagy le) mozoghat, és elkerüli a labirintus sárkányát. Minden labirintusban van több kijutási útvonal. A sárkány egy véletlenszerű kezdőpozícióból indulva folyamatosan bolyong a pályán úgy, hogy elindul valamilyen irányba, és ha falnak ütközik, akkor elfordul egy véletlenszerűen kiválasztott másik irányba. Ha a sárkány a játékossal szomszédos területre jut, akkor a játékos meghal. Mivel azonban a labirintusban sötét van, a játékos mindig csak 3 sugarú körben látja a labirintus felépítését, távolabb nem. Tartsuk számon, hogy a játékos mennyi labirintuson keresztül jutott túl és amennyiben elveszti az életét, mentsük el az adatbázisba az eredményét. Egy menüpontban legyen lehetőségünk a 10 legjobb eredménnyel rendelkező játékost megtekinteni, az elért pontszámukkal, továbbá lehessen bármikor új játékot indítani egy másik menüből. Ügyeljünk arra, hogy a játékos, vagy a sárkány ne falon kezdjenek.

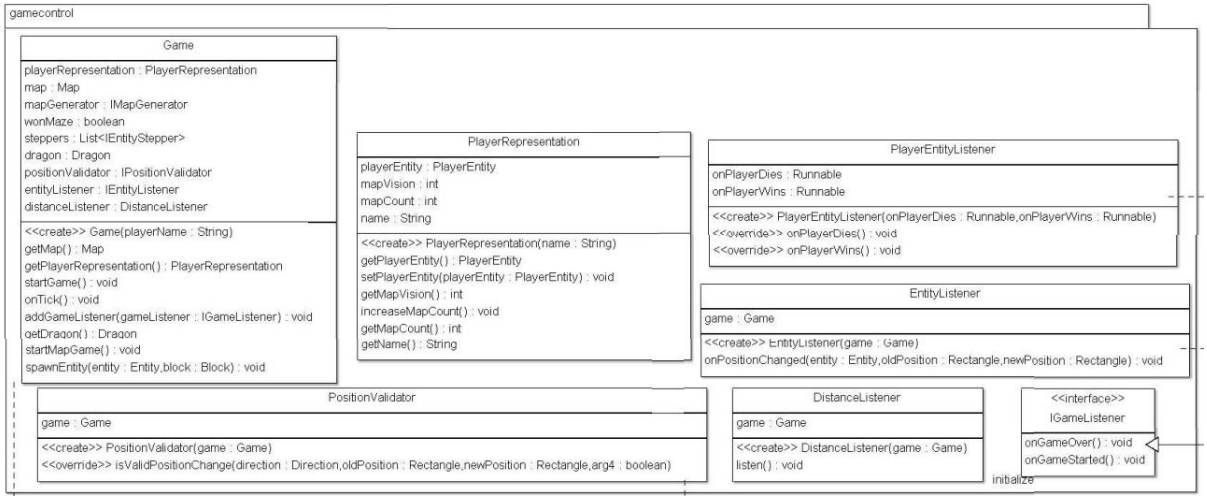
2. Megoldási terv

2.1 Megoldás leírása

A feladat megoldását az Model-View architektúra alapján készítettem el. Ami azt jelenti, hogy a program szétválasztható két komponensre, ami a Model, és a View. Részletesen bemutatom a következő fejezetekben, hogy mely package-ek, illetve osztályok teszik ki a két komponenst.

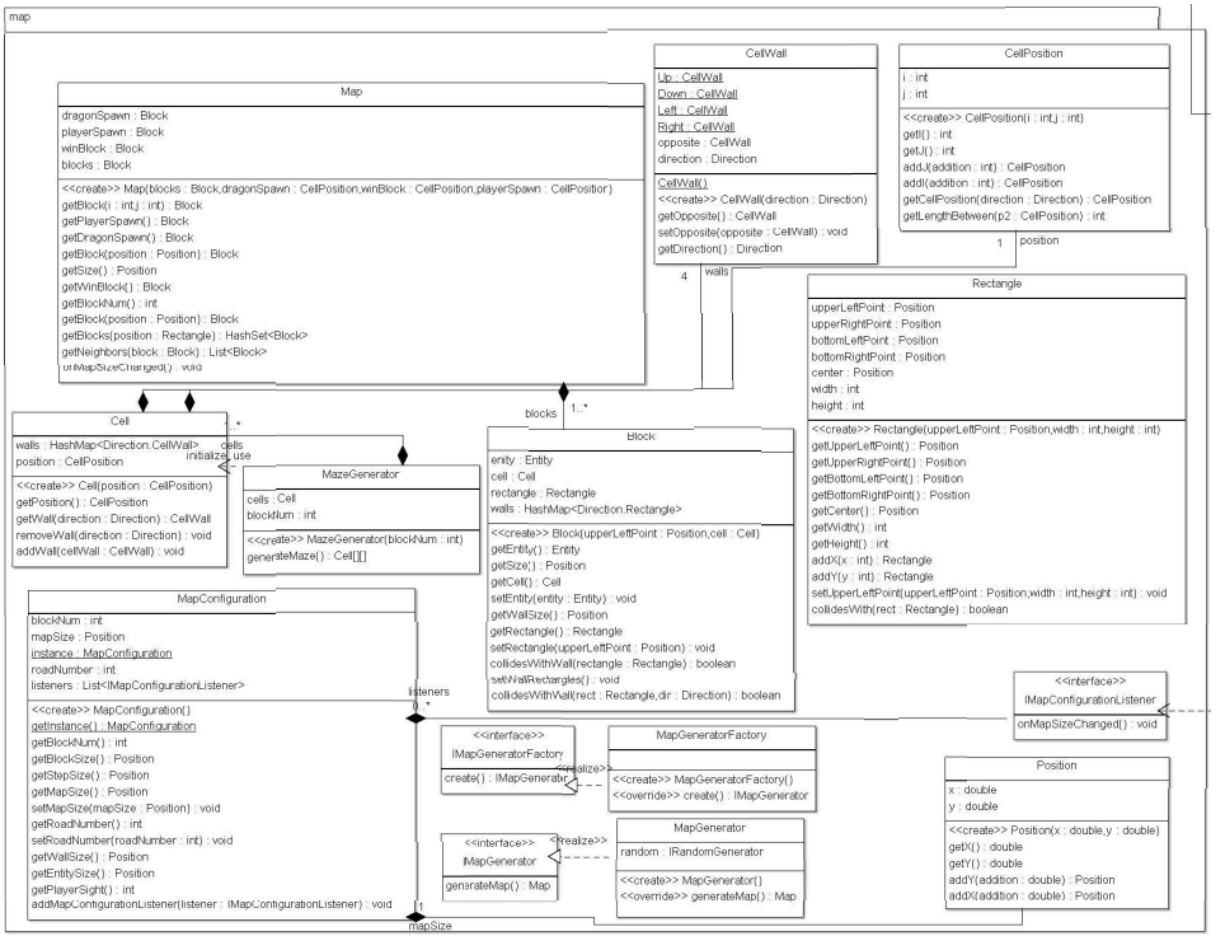
2.2 Model komponens

2.2.1 labirinth.model.gamecontrol package



A model komponens gamecontrol csomagja tartalmazza a játék menetének logikáját. A **Game** osztály objektuma tudja elindítani a játékot a **startGame** metódussal. Az **onTick** metódusa pedig tartalmazza a game loop belsejét, azonban ezt az **onTick** metódust **kívülről** kell meghívni az **időzítőnek**, amikor az intervalluma lejár. Az **IGameListener** interface segítségével kívűlről lehet reagálni a játék végére, illetve a játék kezdésére. A **Game.addGameListener** metódussal lehet regisztrálni a Game példányhoz egy ilyen interface implementációt. A többi getter metódusa a különböző példányok elérhetőségét biztosítja a View-nak. A **PlayerRepresentation** osztály egy játékost reprezentál, akinek van neve, egy PlayerEntity objektuma, teljesített pályák száma, illetve értéke, hogy milyen messze lát el a pályán. A **PlayerEntityListener** osztály implementálja az **IPlayerEntityListener** interfészt. Egy ilyen objektummal iratkozik fel a Game példány a játékos halálára, illetve arra, ha a játékos egy pályát teljesít. Ugyanígy az **EntityListener** osztály implementálja az **IEntityListener** interfészt és a Game osztály egy ilyen példánnyal reagál az entitások pozíció váltására.

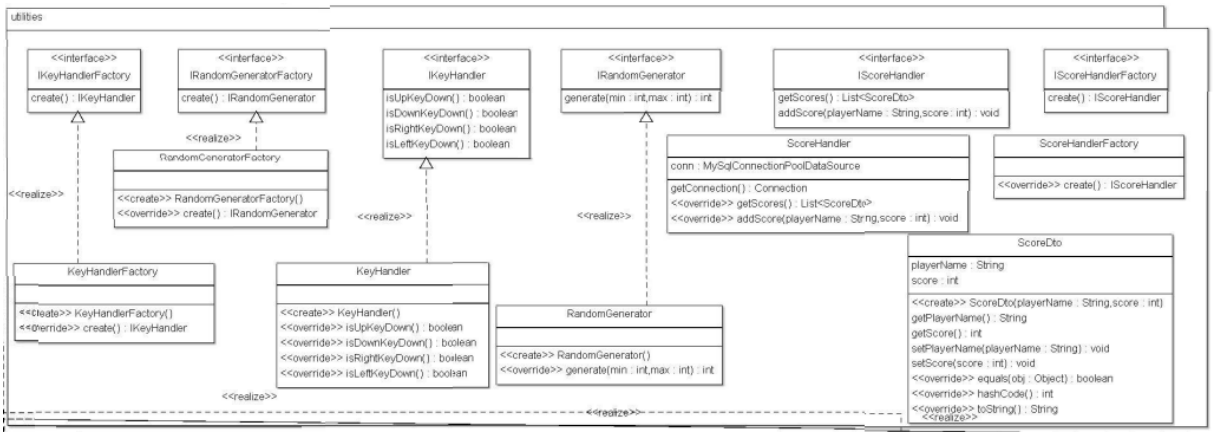
2.2.2 labirinth.model.map package



Ez a csomag felelős a pálya generálásáért, a pálya/térkép konfigurációjáért, illetve ebben van a térképet reprezentáló osztály. A **Map** osztály tartalmazza a pályának a Block példányait, ami egy nxn-es mátrix. Van egy játékos kezdőpontja, sárkány kezdőpontja, illetve egy olyan Block, amire, ha rálép a játékos teljesítette a pályát. A **Position** osztály példánya egy pozíciót reprezentál a pályán, méghozzá egy pixelnek a pozícióját. A **CellPosition** osztály példánya, pedig egy Cell vagy Block példánynak az indexeit reprezentálják. A **Rectangle** osztály egy négyzetet reprezentál a pályán. A collidesWith methódus segítségével ellenőrizzük, hogy egy másik négyzettel ütközik-e. A **CellWall** osztály egy adott irányban lévő falat reprezentáló osztály. A **Cell** osztály, pedig egy cellát reprezentál a mátrixban, aminek 4 irányban lehet fala, illetve van egy pozíciója. A **MazeGenerator** a cellák mátrixának gnerálásáért felelős osztály. Ez generálja le a pálya struktúráját, hogy mely celláknál hol vannak falak. A **MapGenerator** osztály, pedig a térkép generálásáért felelős osztály. Ez az osztály a MazeGenerator egy példánya által generált cellákból hoz létre egy Map példányt. Kijelöli a releváns Block-okat, ahol a sárkány, illetve a játékos kezd. A **MapConfiguration** osztály, pedig a pálya

konfigurációját tartalmazó osztály. Az **IMapConfigurationListener** interface segítségével fel lehet iratkozni az onMapSizeChanged metódussal a pálya méretének változására.

2.2.3 labirinth.model.utilities package



Az utilities csomag a különböző segéd osztályok csomagja. Ebben a csomagban nem összefüggő osztályok vannak, így felsorolásként mutatom be őket:

- **KeyHandler**: **IKeyHandler** interfészt megvalósító osztály. Ez a billentyűzet lenyomást figyelő osztály. Az osztály példányától az interfészen keresztül lehet lekérni az adott input-ot.
- **RandomGenerator**:**IRandomGenerator** interfészt megvalósító osztály. Véletlenszerű szám generálásáért felelős osztály.
- **ScoreHandler**: **IScoreHandler** interfészt megvalósító osztály. Az adatbázis kezeléséért felelős osztály. A getScores metódus lekéri az adatbázisban lévő adatokat. Az addScore metódus, pedig frissíti az adatbázist, tehát hozzáad egy új rekord-ot, vagy frissíti a már meglévő játékos eredményét, amennyiben ez szükséges.
- **ScoreDto**: Egy játékos elért eredményét reprezentáló osztály, aminek egy példánya egy (Data Transfer Object), azaz csak gettere és settere van, illetve nem tartalmaz konkrét logikát.

2.2.3.1 adatbázis előfeltételek

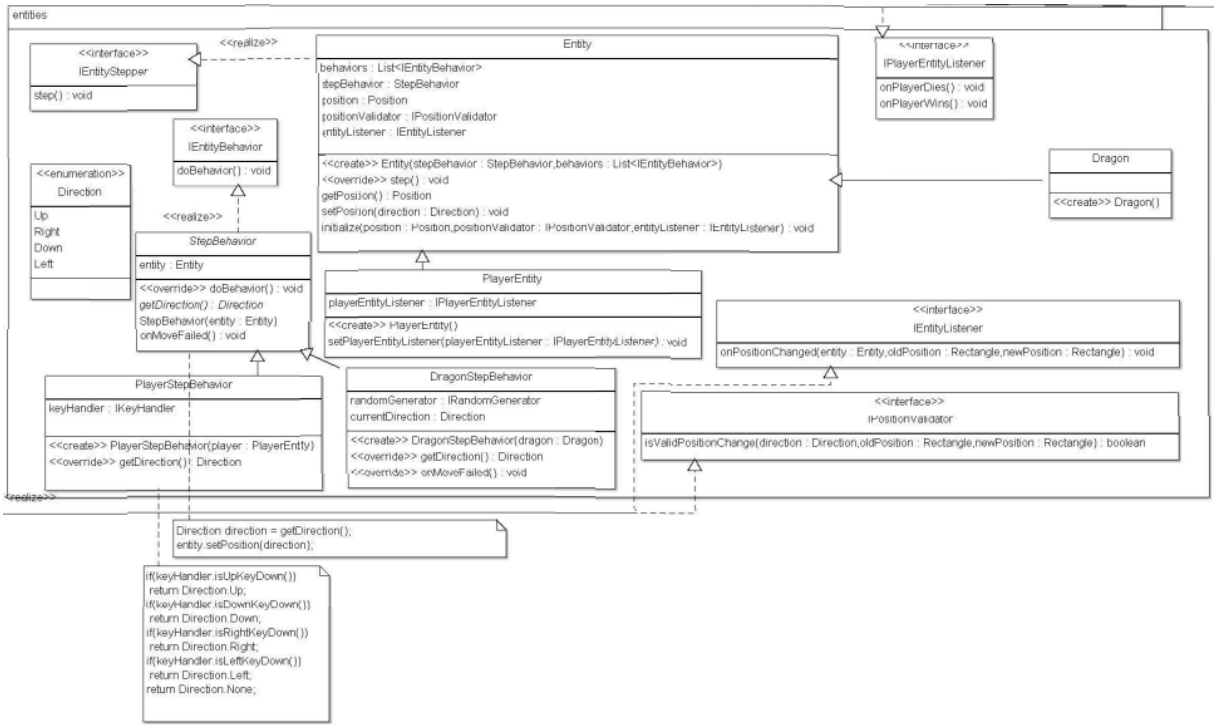
Ahhoz, hogy az eredmények elmentése, illetve megjelenítése sikeres legyen, ahhoz szükséges egy adatbázis, amihez csatlakozik a program. Az alábbi paraméterek kellenek az adatbázisnak, hogy a program működjön:

- Localhost-on 3306 porton legyen az adatbázis
- Az adatbázis neve legyen: „labirinth”
- Legyen egy „tanulo” nevű felhasználó „asdasd123” jelszóval az adatbázisnak

- Legyen a tanulo felhasználónak INSERT, UPDATE és SELECT joga.
- Tartalmazzon az adatbázis egy scores nevezetű táblát az alábbi tábla létrehozásával működik a program:

```
CREATE TABLE scores (
   ID int NOT NULL AUTO_INCREMENT,
   Name varchar(20) UNIQUE NOT NULL,
   MapCount int NOT NULL,
   PRIMARY KEY (ID)
);
```

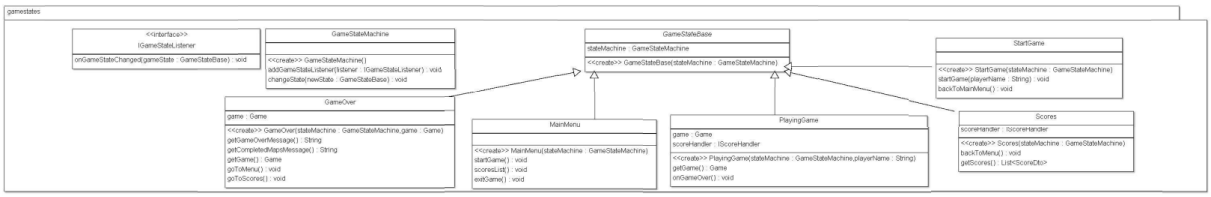
2.2.4 labirinth.model.entities package



Az entities csomag a pályán lévő „entitások” -hoz lévő osztályokat tartalmazza. Az **Entity** ősosztály az entitások alap funkcióit valósítja meg. (pl: pozíció változtatás) A **PlayerEntity** a játékos karakterét reprezentáló osztály a pályán, míg a **Dragon** osztály a sárkány karakterét reprezentáló osztály. A **StepBehavior** ősosztály segítségével a különböző lépés viselkedéseket lehet definiálni. A **PlayerStepBehavior** a játékos lépésének logikáját tartalmazza, a **DragonStepBehavior**, pedig a sárkány lépésének logikáját. Az **IEntityBehavior** interfész megvalósításával különböző viselkedéseket lehet hozzárendelni az adott entitásokhoz. A **Direction** enumeráció az irányokat reprezentálja. Az **IEntityStepper** interfész segítségével lehet léptetni az entitásokat, illetve Három interfész ír le különböző eseményeket:

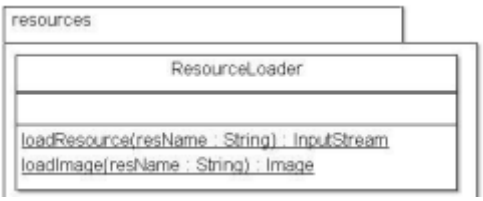
- **IPositionValidator**: Az Entity osztály tartalmazza ezt az interfészt implementáló objektumot. A setPosition metódus meghívásakor, akkor állítja be az új pozíciót az entitás magának, amennyiben a pozíció érvényes. Ez viszont már a játékmenetnek kell eldöntenie, nem pedig az entitásnak.
- **IEntityListener**: Ez egy tényleges esemény, amire fel lehet iratkozni. Akkor hívódik meg, amikor az új pozíció beállítása sikeres volt.
- **IPlayerEntityListener**: A játékos karakterének eseményei, amikor a játékos veszít, illetve amikor a játékos egy pályát teljesít, akkor hívódnak meg a megfelelő metódusok.

2.2.5 labirinth.model.gamestates package



A gamestates csomag az alkalmazás állapotait reprezentáló osztályokat tartalmazza, illetve az állapotgépet. Az adott állapotokban lévő publikus metódusokat használja a **View** komponens. Illetve az **GameStateMachine** osztály állapotátmenetet biztosít a különböző állapotok között.

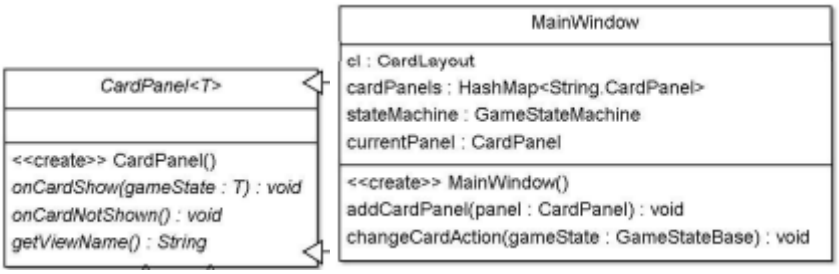
2.2.6 labirinth.resources package



Ebben a csomagban csak egy **ResourceLoader** osztály van, ami különböző resource-ok betöltését teszi lehetővé. A projektben csak a **loadImage** metódust használjuk a sprite-ok, illetve a pályaelemek betöltésére.

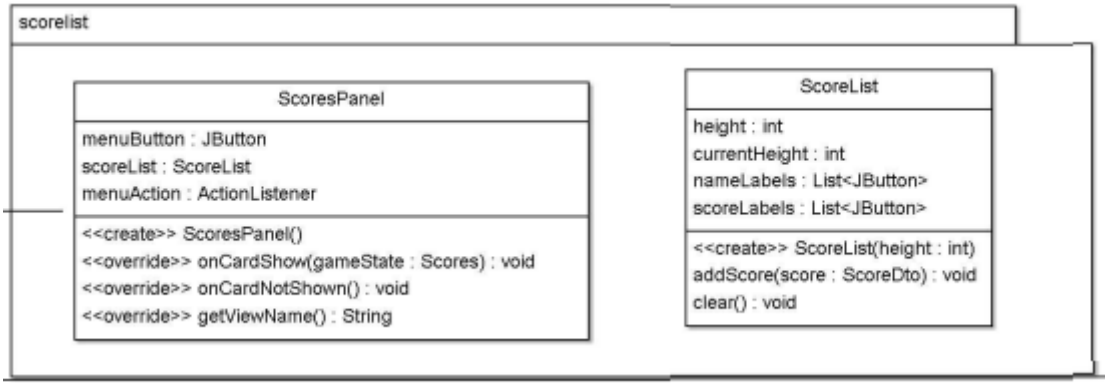
2.3 View komponens

2.5.1 labirinth.view package



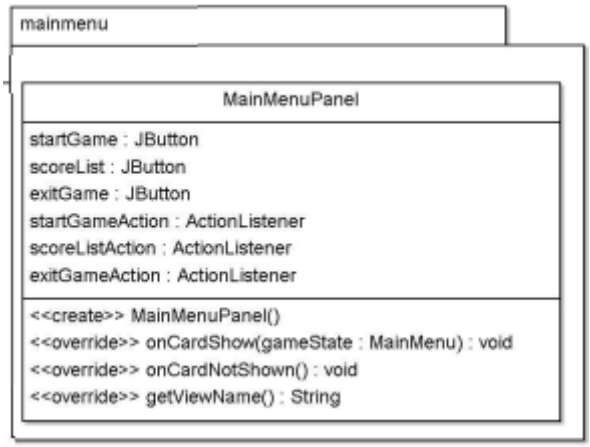
Ebben a csomagban vannak a különböző nézetek csomagjai, illetve a főablakot reprezentáló osztály a **MainWindow** osztály. A **CardPanel** osztály, pedig egy-egy „oldalt” reprezentál, ami egy generikus osztály és a **JPanel** osztályból származik. A **T** paraméterének egy **GameStateBase** osztályból származó típusnak kell lennie. A **MainWindow** osztály a **GameStateMachine** egy példánya segítségével, illetve a **CardPanel**-ek segítségével menedzseli azt, hogy melyik oldalt kell megjelenítenie. Megjelenítéskor meghívódik az adott **CardPanel** objektum **onCardShow** metódusa, illetve, amikor eltűnik az adott **CardPanel**, akkor az **onCardNotShown** metódus hívódik meg.

2.5.2 labirinth.view.scorelist package



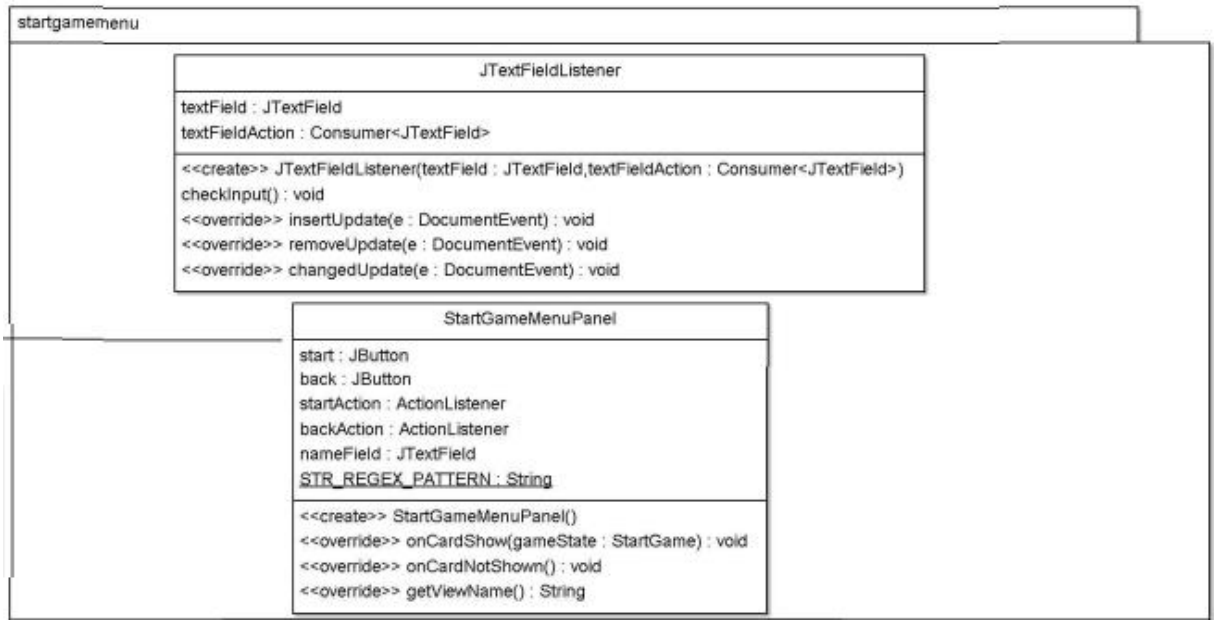
A **scorelist** csomag a játékosok eredményeinek oldalának megjelenítéséért felelős osztályokat tartalmazza. A **ScoreList** osztály egy **JPanel** osztályból származó osztály, ami egy táblázatban megjeleníti a játékosok által elért eredményeket. A **ScoresPanel** osztály, pedig a **CardPanel<Scores>** osztályból származik és az oldal megjelenítéséért, illetve az események kezeléséért felelős.

2.5.3 labirinth.view.mainmenu package



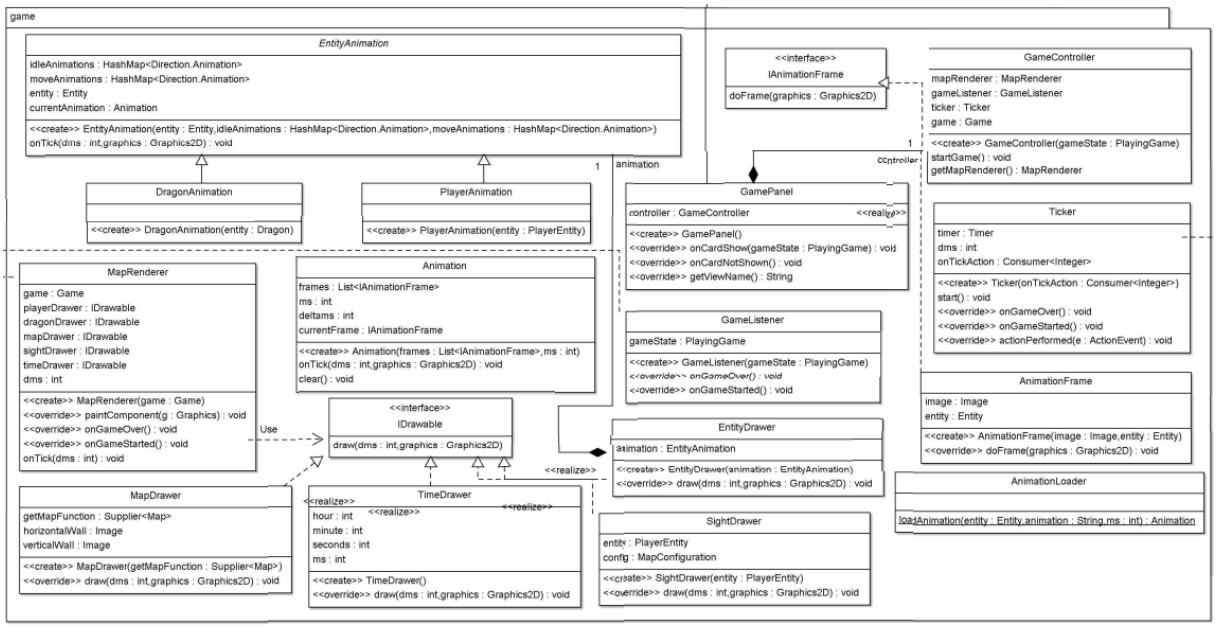
A **MainMenuPanel** osztály a **CardPanel<MainMenu>** osztályából származik le. Az osztály a főmenü megjelenítéséért, illetve a vezérlőelemek inicializálásáért felelős.

2.5.4 labirinth.view.startgamemenu package



A **startgamemenu** csomag tartalmazza a játék indítása előtti nézet osztályait, illetve eseménykezelőit. A **StartGameMenuPanel** osztály a **CardPanel<StartGame>** osztályából származik le. Az oldalon a gombok, illetve név bekéréséhez szükséges szövegmező inicializálásáért, visszaállításáért felelős osztály. A **JTextFieldListener** osztály, pedig egy **DocumentListener** interfészt implementál. Ennek segítségével tudunk reagálni a játékos nevének változására és amennyiben nem megfelelő a megadott név adott változtatásokat megcsinálni a grafikus felületen.

2.5.5 labirinth.view.game package



Ez a csomag tartalmazza a játékmenet nézetéhez szükséges osztályokat. Az **IDrawable** interfész egy interfészt biztosít a különböző rajzolható objektumokhoz. Az implementált **draw** metódusa rajzolja meg az adott dolgot a képernyőre a felhasználó számára. Ezek az osztályok implementálják:

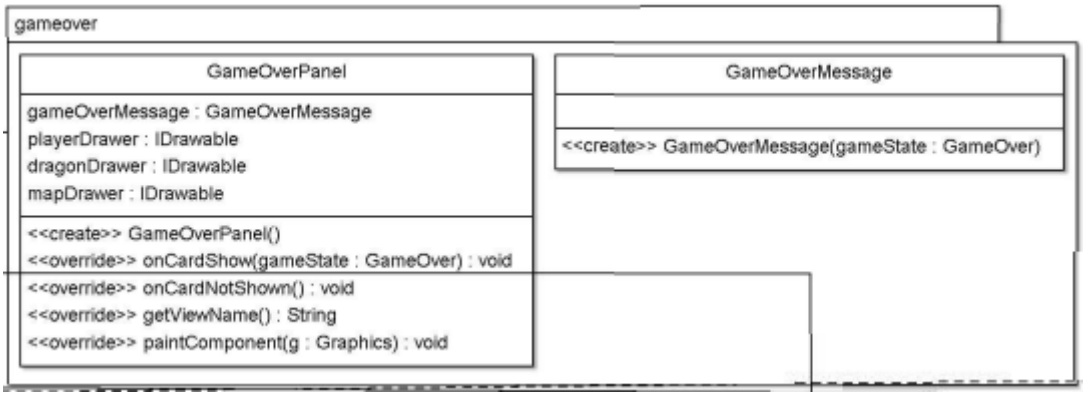
- **MapDrawer**: A pálya kirajzolásáért felelős osztály.
- **TimeDrawer**: A játék kezdése óta eltelt idő megjelenítéséért felelős osztály.
- **SightDrawer**: A játékos látókörét kirajzolásáért felelős osztály. Ez a már kirajzolt pályát feketére színezi, kivéve a játékost és a látókörét.
- **EntityDrawer**: Az entitások kirajzolásáért felelős osztály. Egy EntityAnimation objektumot kap, amit ki tud rajzolni.

A **MapRenderer** osztály felelős az egész játéknak a nézetének a megjelenítéséért. Ezt a fent leírt **Drawer** osztályok felhasználásának segítségével oldja meg. Ez az osztály implementálja az **IGameListener** interfészt is. Az **onTick** metódusának meghívásakor rajzolja újra a játékot az osztály.

A **Ticker** osztály felelős az időzítő működéséért. A számára megadott akciót fogja elvégezni minden frissítésnél. A **GameController** osztály, pedig a játék kontrollálásáért felelős osztály. Létrehoz egy **Ticker** példányt, aminek az akciójakor meghívja a **MapRenderer** példány és a **Game** példány **onTick** metódusát. startGame meghívásakor elindítja az időzítőt, illetve az **IGameListener** implementálásával, a játék végénél, pedig leállítja azt. Az **AnimationFrame**,

EntityAnimation, **PlayerAnimation** és **DragonAnimation** osztályok a különböző entitások animációjának kirajzolásához szükséges osztályok. Végül a **GamePanel** osztály, pedig a **CardPanel<PlayingGame>** osztályból származik le és a játék „oldalának” megjelenítéséért felelős osztály.

2.5.6 labirinth.view.gameover package



A **gameover** csomag tartalmazza a játék befejezésekor a nézetet reprezentáló osztályokat. A **GameOverPanel** osztály a **CardPanel<GameOver>** osztályból származik le és a teljes pálya kirajzolásához szükséges objektumokat tartalmazza, illetve a **GameOverMessage** osztály egy példányát, ami a játék végét jelzi egy üzenettel a felhasználónak.

3. Esemény-eseménykezelő párok

Esemény	**Eseménykezelő**
A játékost elkapja a sárkányt (életét veszti)	IPlayerEntityListener
A játékos teljesít egy pályát	IPlayerEntityListener
Egy entitás pozíciót vált a pályán	IEntityListener
A játéknak vége	IGameListener
A játék elkezdődött	IGameListener
A pálya mérete megváltozott	IMapConfigurationListener
Felhasználó megnyom egy billentyűt	KeyHandler (KeyListener-t implementálja)
Felhasználó beleír a játékos nevének a szövegmezőjébe	JTextFieldListener

4. Pályát generáló algoritmus leírása

A labirintus generálásának megoldását szeretném bemutatni. A megoldásomat kódban két részre osztottam:

4.1 Véletlenszerű labirintus generálása

```
// Recursive method to generate the maze
private void generateMazeRecursive(CellPosition cPos) {
        Direction[] dirs = Direction.values();
        Collections.shuffle(Arrays.asList(dirs)); // Shuffle the directions to randomize maze generation
        for (Direction dir : dirs) {
    // Get the position of the neighboring cell in the current direction
            CellPosition nPos = cPos.getCellPosition(dir);
    // Check if the neighboring cell position is valid and not visited
    if(nPos == null)
    {
        continue;
    }
            if (between(nPos.getI(), blockNum) && between(nPos.getJ(), blockNum)
                    && inInitialState(cells[nPos.getI()][nPos.getJ()])) {
        // If the neighboring cell is valid and in initial state, remove the wall between the current cell and the neighbori
        Cell cell = cells[cPos.getI()][cPos.getJ()];
                CellWall wall = cell.getWall(dir);
        cell.removeWall(dir);
                cells[nPos.getI()][nPos.getJ()].removeWall(wall.getOpposite().getDirection());
                generateMazeRecursive(nPos); // Recursively generate the maze starting from the neighboring cell
            }
        }
    }
```

Ahogy látható a kódban egy adott kezdőpontból kiinduló rekurzív metódus valósítja meg a labirintus létrehozását. Ez a rekurzív algoritmus annyit csinál, hogy:

- Az összes irányt összegyűjtjük és „összekeverjük”.
- Végig megyünk az „összekevert” irányokon:
    - Megnézzük, hogy az adott irányban van-e cella, ha nincs, akkor folytatjuk a következő iránnyal.
    - Ha az adott irányban van cella és még mind a 4 oldalán van fala, akkor töröljük a jelenlegi cellán az adott irányban lévő falat, illetve az adott irányban lévő cellán az ellenkező irányban lévő falat és abba az irányba megyünk tovább rekurzívan.

Ez az algoritmus, ahogyan látható, minden cellához maximum egyszer jut el, hiszen töröljük a falakat az adott cellán, illetve amerre megyünk tovább ott is törlünk egy falat, így ezekbe a cellákba többször már nem fogunk bele menni, hiszen már nincs 4 fala ezeknek a celláknak. Az is látható, hogy ez egy összefüggő gráfot fog eredményezni, amiben minden cellából el lehet jutni minden másik cellába pontosan egy útvonalon. Ez amiatt történik meg, mert végig iterálunk minden eljutott cellánál az összes irányba, így, ha valahol elakadt az algoritmus, akkor „visszafele” menet még találhat egy szabad cellát. Ez akkor nem történik meg, ha már minden cellába ellátogatott az algoritmusunk.

4.2 Több kijutási útvonal létrehozása

```
// Method to generate roads through the maze
private void generateRoads(int requiredCount)
{
    AtomicInteger count;
    do{
        count = new AtomicInteger(0);
        List<CellPosition> positions = new ArrayList<>();
        countRoadsBetween(count, new boolean[blockNum][blockNum], cells[blockNum - 1][0].getPosition(), cells[0][blockNum - 1].getP
        if(positions.isEmpty())
        {
            System.out.println("Empty");
            break;
        }
        Collections.shuffle(positions);
        CellPosition chosen = positions.get(0);
        removeRandomRoad(chosen);
    }while(count.get() < requiredCount);

}
```

Mivel a feladat leírásában benne van az, hogy több kijutási útvonal kell a labirintusban, ezért ezt egy külön metódusban oldottam meg, hiszen az előző algoritmus egy olyan labirintust hoz létre, ahol csak egy kijutási útvonal létezik. Ez a **generateRoads** metódus utakat generál a labirintusban addig amíg nincsen meg a szükséges útvonalak száma. A ciklus magja így néz ki:

- Meghívjuk a **countRoadsBetween** metódust, ami megszámolja rekurzívan, hogy mennyi út létezik a bal alsó saroktól a jobb felső sarkig, illetve a **positions** listába bele teszi, azokat a cellákat, ahol elakadt és elkezdett visszafele menni.
- Ezután egy véletlenszerű elemet kiveszünk a positions listából és a cellából törlünk véletlenszerűen egy falat a **removeRandomRoad** metódus segítségével.
- Ez addig folytatódik, amíg el nem érjük a megadott útvonalak számát, vagy a positions lista üres nem lesz. (Ekkor valószínűleg már nem marad fal)

5. Tesztelési Terv

Név	**Lépések**
Játék kezdés	• Indítsd el a programot • Nyomj a Start Game gombra • Írd be a szövegmezőbe, hogy „Player” • Nyomj a Start gombra Elvárt működés: A játékmenet elindul, a számláló a bal felső sarokban számolja az eltelt időt a játék kezdése óta, illetve.a játékos a bal alsó sarokban áll és a pályát látja 3 block sugarú körben.
Játék vége	• Indítsd el a programot • Nyomj a Start Game gombra • Írd be a szövegmezőbe, hogy „Player” • Nyomj a Start gombra • Keresd meg a sárkányt és menj a közelébe Elvárt működés: Miután a sárkány közelébe került a játékos (1 blocknyi távolságra), akkor a pálya teljes területe láthatóvá válik a számláló megáll, illetve középen megjelenik egy ablak, ami

	egy üzenetet hordoz a játék végéről, illetve két gombot, az egyikkel az eredmény listához tudsz navigálni, a másikkal, pedig a menühöz.
Eredmények megtekintése	• Indítsd el a programot • Nyomj a Scores gombra Elvárt működés: A Scores gombra kattintva megjelennek az adatbázisból lekért játékosok eredményei csökkenő sorrendben és maximum 10 darab. Amennyiben nem sikerült lekérdezni az adatbázisból az eredményeket a felhasználó számára jelzi ezt egy ablak, majd a Menu gombbal lehetséges vissza menni a menübe.
Játék kezdése sikertelen	• Indítsd el a programot • Nyomj a Start Game gombra • Írd be a szövegmezőbe, hogy „Player*” • Nyomj a Start gombra Elvárt működés: Amikor a Player után a csillagot beírja a felhasználó, akkor piros lesz a szövegmező kerete és a Start gomb szürke lesz. Amikor rányom a felhasználó a Start gombra, akkor nem történik semmi.
Kilépés a játékból	• Indítsd el a programot • Nyomj az Exit Game gombra Elvárt működés: A program bezáródik, amikor megnyomják az Exit Game gombot.
Vissza a menübe játék kezdése előtt	• Indítsd el a programot • Nyomj a Start Game gombra • Nyomj a Back gombra Elvárt működés: A játékmenetben a Back gombra kattintva visszatér a program a főmenübe.
Vissza a menübe játék befejezése után	• Indítsd el a programot • Nyomj a Start Game gombra • Írd be a szövegmezőbe, hogy „Player” • Nyomj a Start gombra • Keresd meg a sárkányt és menj a közelébe • Kattints a Menu gombra Elvárt működés: A játék befejezése után a Menu gombra kattintva visszatér a program a főmenübe.
Eredmények megtekintése a játék befejezése után	• Indítsd el a programot • Nyomj a Start Game gombra • Írd be a szövegmezőbe, hogy „Player” • Nyomj a Start gombra • Keresd meg a sárkányt és menj a közelébe • Kattints a Scores gombra Elvárt működés: A játék befejezése után a Scores gombra kattintva a program az eredmény listához navigál.

Falnak ütközés	• Indítsd el a programot • Nyomj a Start Game gombra • Írd be a szövegmezőbe, hogy „Player” • Nyomj a Start gombra • Nyomd le az A gombot ameddig falnak nem ütközik a játékos karakter. Elvárt működés: Az A betűt lenyomva a játékos karakter elindul balra, viszont falnak ütközéskor nem tud tovább haladni abba az irányba.
Mozgás	• Indítsd el a programot • Nyomj a Start Game gombra • Írd be a szövegmezőbe, hogy „Player” • Nyomj a Start gombra • Nyomd le a W betűt egy kevés időre • Nyomd le az A betűt egy kevés időre • Nyomd le a S betűt egy kevés időre • Nyomd le a D betűt egy kevés időre Elvárt működés: A W betűt lenyomva tartva a játékos karaktere elindul felfele, az A betűt lenyomva tartva a játékos karaktere elindul balra, az S betűt lenyomva tartva a játékos karaktere elindul lefele, a D betűt lenyomva tartva a játékos karaktere elindul jobbra.
Pálya teljesítése	• Indítsd el a programot • Nyomj a Start Game gombra • Írd be a szövegmezőbe, hogy „Player” • Nyomj a Start gombra • Elkerülve a sárkányt próbálj meg eljutni a jobb felső sarokba. • Ha a sárkány elkap, akkor **Menu** gombra kattintás és kezdd előről a lépéseket. Elvárt működés: Amikor a játékos karaktere eljut a jobb felső sarokba, akkor a program egy új pályát generál, a játékos karaktere lekerül a bal alsó sarokba és megint a jobb felső sarokba kell eljutnia.