Programozási technológia   
3. Beadandó

Név: Magyar Viktor

Neptun kód: O9WEJJ

Dátum: 2024.05.10.

Feladat száma: 3

[1. Feladat leírása: 3](#_Toc166242877)

[2. Megoldási terv 4](#_Toc166242878)

[2.1 Megoldás leírása 4](#_Toc166242879)

[2.2 Model komponens 4](#_Toc166242880)

[2.2.1 labirinth.model.gamecontrol package 4](#_Toc166242881)

[2.2.2 labirinth.model.map package 5](#_Toc166242882)

[2.2.3 labirinth.model.utilities package 6](#_Toc166242883)

[2.2.4 labirinth.model.entities package 6](#_Toc166242884)

[2.2.5 labirinth.model.gamestates package 7](#_Toc166242885)

[2.2.6 labirinth.model.resources package 7](#_Toc166242886)

[2.3 View komponens 7](#_Toc166242887)

[2.5.1 labirinth.view package 7](#_Toc166242888)

[2.5.2 labirinth.view.scorelist package 7](#_Toc166242889)

[2.5.3 labirinth.view.mainmenu package 8](#_Toc166242890)

[2.5.4 labirinth.view.startgamemenu package 9](#_Toc166242891)

[2.5.5 labirinth.view.game package 9](#_Toc166242892)

[2.5.6 labirinth.view.gameover package 10](#_Toc166242893)

[3. Esemény-eseménykezelő párok 11](#_Toc166242894)

[4. Pályát generáló algoritmus leírása 11](#_Toc166242895)

[5. Tesztelési Terv 11](#_Toc166242896)

# 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 képen szöveg, képernyőkép, szám, sor látható

Automatikusan generált leírás

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 A képen szöveg, Párhuzamos, szám, dokumentum látható

Automatikusan generált leírás

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

A képen szöveg, nyugta, diagram látható

Automatikusan generált leírás

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.

### 2.2.4 labirinth.model.entities package

A képen szöveg, nyugta, diagram, Párhuzamos látható

Automatikusan generált leírás

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 képen szöveg, képernyőkép, sor, diagram látható

Automatikusan generált leírás

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

A képen szöveg, nyugta, Betűtípus, sor látható

Automatikusan generált leírás

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

A képen szöveg, Betűtípus, nyugta, sor látható

Automatikusan generált leírásA képen szöveg, képernyőkép, Betűtípus, nyugta látható

Automatikusan generált leírás

### 2.5.2 labirinth.view.scorelist package

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

A MainWindow osztály a JFrame osztályból származik és innen indul el az alkalmazás. Ez egy Singleton osztály. A main metódusban a MainWindow.getInstance metódusát hívjuk csak meg. A megfelelő View osztályokat egy CardLayout-ba töltjük be a konstruktorban, illetve az onViewChanged metódus segítségével válthatunk a nézet-ek között. Ezt a metódust az ApplicationState.onStateInitialize metódusa hívja meg.

A MenuButton általánosan a menü gombokat reprezentáló osztály, ami egy IMenuOption-t kap meg és a doOption metódusát hívja meg a gomb megnyomásakor.

A View osztály egy generikus osztály, ami a különböző nézetek ősosztálya.

### 2.5.3 labirinth.view.mainmenu package

A képen szöveg, képernyőkép, nyugta, Betűtípus látható

Automatikusan generált leírás

A MainMenuView a View<InMainMenu> osztályából származik le, így az initialize egy InMainMenu példányt kap. Ez alapján lekéri tőle az IMenu attribútumát, majd MenuButton gombokat helyez el egymás alá középre.

### 2.5.4 labirinth.view.startgamemenu package

A képen szöveg, képernyőkép, sor, szám látható

Automatikusan generált leírás

A InitializeGameView a View<InInitializeGameMenu> osztályából származik le, így az initialize egy InInitializeGameMenu példányt kap. Ez alapján inicializálja a különböző vezérlőket, illetve a JTextFieldListener osztály segítségével a JTextField-ekhez hozzárendeli az InInitializeGameMenu különböző settereit. Tehát, amikor a text field-ek értéke változik, akkor a controller példány értéke is változik, ami kivételt dob, ha valamilyen input nem megfelelő. Ekkor piros keretet kap a text field.

### 2.5.5 labirinth.view.game package

A képen szöveg, nyugta, Párhuzamos, diagram látható

Automatikusan generált leírás

Ez a csomag tartalmazza a játékmenet nézetéhez szükséges osztályokat. A FieldButton reprezentálja a mezőket, ami egy field-et kap a konstruktorában és feliratkozik a FieldChangeListener.onSignalChanged eseményére. A signalMap attribútum segítségével változtatja át String típusúra az adott Signal objektumot és azt állítja be a saját szövegének, amikor változik a Signal attribútum. A PlayerPanel a játékos releváns adatait megjelenítő panel, a ThrowButton, pedig a pálya feletti gombokat reprezentálja, amit megnyomva a játékos ledobja az adott oszlopba a saját jelét. A TicTacToeGameView ezen osztályok segítségével készíti el a játék nézetét.

### 2.5.6 labirinth.view.gameover package

A képen szöveg, nyugta, képernyőkép, sor látható

Automatikusan generált leírás

# 3. Esemény-eseménykezelő párok

|  |  |
| --- | --- |
| **Esemény** | **Eseménykezelő** |
| A játékos az egyik oszlopban elhelyezi a saját jelét | ThrowListener |
| A játékos megnyom egy menü gombot | MenuButton |
| A játékos beír valamit az egyik szövegmezőbe | JTextFieldListener |
| Egy mező jele megváltozik a pályán | FieldChangeListener |
| Játékosok váltakozása a nézetben játékmenet közben | PlayerListener |
| Játék vége | MatchListener |
| A pályán egy oszlop betelik | MatchListener |
| A pályát alaphelyzetbe állítják | MatchListener |

# 4. Pályát generáló algoritmus leírása

# 5. Tesztelési Terv

|  |  |
| --- | --- |
| **Név** | **Lépések** |
| Játék kezdés teszt | * Indítsd el a programot * Nyomj a Start Game gombra * Nyomj a Start gombra   Elvárt működés:  A játékmenet elindul a player1 nevezetű játékos zöld színű, tehát ő a kezdő játékos, a tábla üres, illetve a tábla felett lévő gombok zöld kerettel rendelkeznek és megnyomhatóak. Bal felső saroknál két menüelem látszódik. |
| Játék vége teszt1 | * Indítsd el a programot * Nyomj a Start Game gombra * Nyomj a Start gombra * Nyomd meg az 1. oszlop gombját * Nyomd meg a 2. oszlop gombját * Nyomd meg a 2. oszlop gombját * Nyomd meg a 3. oszlop gombját * Nyomd meg a 3. oszlop gombját * Nyomd meg az 1. oszlop gombját * Nyomd meg a 3. oszlop gombját * Nyomd meg a 4. oszlop gombját * Nyomd meg a 4. oszlop gombját * Nyomd meg a 4. oszlop gombját * Nyomd meg a 4. oszlop gombját   Elvárt működés:  A lépések sikeresek és az utolsó lépés után jelzi a program a felhasználónak, hogy a player1 nevű játékos nyert 4 átlósan kirakott jellel. |
| Játék vége teszt2 | * Indítsd el a programot * Nyomj a Start Game gombra * Nyomj a Start gombra * Nyomd meg az 1. oszlop gombját * Nyomd meg a 1. oszlop gombját * Nyomd meg a 2. oszlop gombját * Nyomd meg a 2. oszlop gombját * Nyomd meg a 3. oszlop gombját * Nyomd meg az 3. oszlop gombját * Nyomd meg a 1. oszlop gombját * Nyomd meg a 4. oszlop gombját * Nyomd meg a 2. oszlop gombját * Nyomd meg a 4. oszlop gombját   Elvárt működés:  A lépések sikeresek és az utolsó lépés után jelzi a program a felhasználónak, hogy a player2 nevű játékos nyert 4 darab sorban kirakott jellel. |
| Játék vége teszt3 | * Indítsd el a programot * Nyomj a Start Game gombra * Nyomj a Start gombra * Nyomd meg az 1. oszlop gombját * Nyomd meg a 2. oszlop gombját * Nyomd meg a 1. oszlop gombját * Nyomd meg a 2. oszlop gombját * Nyomd meg a 1. oszlop gombját * Nyomd meg az 2. oszlop gombját * Nyomd meg a 1. oszlop gombját * Nyomd meg a 2. oszlop gombját * Nyomd meg a 3. oszlop gombját * Nyomd meg a 4. oszlop gombját * Nyomd meg a 3. oszlop gombját * Nyomd meg a 4. oszlop gombját * Nyomd meg a 3. oszlop gombját * Nyomd meg a 4. oszlop gombját * Nyomd meg a 3. oszlop gombját * Nyomd meg a 4. oszlop gombját   Elvárt működés:  A lépések sikeresek és az utolsó lépés után jelzi a program a felhasználónak, hogy a meccs döntetlen és látszódik, hogy a tábla teljesen betelt az X és az O jelekkel. |
| Kilépés a játékból teszt | * Indítsd el a programot * Nyomj az Exit Game gombra   Elvárt működés:  A program bezáródik, amikor megnyomják az Exit Game gombot. |
| Vissza a menübe teszt | * Indítsd el a programot * Nyomj a Start Game gombra * Nyomj a Start gombra * Nyomj a Menu gombra   Elvárt működés:  A játékmenetben a Menu gombra kattintva visszatér a program a főmenübe. |
| Vissza teszt | * Indítsd el a programot * Nyomj a Start Game gombra * Nyomj a Back gombra   Elvárt működés:  A meccs adatainak megadásakor a Back gombra kattintva visszatér a program a főmenübe. |
| Új játék kezdése teszt | * Indítsd el a programot * Nyomj a Start Game gombra * Nyomj a Start gombra * Nyomj a New Game gombra * Írd be a Column Number X Row Number mezőbe a 8x5 értéket * Nyomj a Start gombra   Elvárt működés:  Az első start gombra nyomáskor egy 4x4-es tábla jelenik meg, a második start gombra nyomáskor, pedig egy 8x5-ös tábla jelenik meg. |