**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Média- és Oktatásinformatika Tanszék**

**Labirintusverseny a mesterséges intelligenciával**

**Témavezető:**

Bende Imre   
tanársegéd, MA

**Szerző:**

Lukács Dávid István

Programtervező informatikus BSc.

Budapest, 2022

Tartalomjegyzék

[1. Bevezetés 3](#_Toc102732434)

[2. Felhasználói dokumentáció 4](#_Toc102732435)

[2.1 A program követelményei hardveres és szoftveres téren 4](#_Toc102732436)

[2.2 A Java telepítése 5](#_Toc102732437)

[2.3 A program futtatása 6](#_Toc102732438)

[2.4 Új felhasználó regisztrálása 7](#_Toc102732439)

[2.5 A játékszoftver funkciói 7](#_Toc102732440)

[**2.5.1 A pályakészítő funkció** 8](#_Toc102732441)

[2.5.2 Online játék 10](#_Toc102732442)

[2.5.3 Offline játék 13](#_Toc102732443)

[2.5.4 Toplista 15](#_Toc102732444)

[2.5.5 Saját labirintusok menüpontja 15](#_Toc102732445)

[2.6 Segítség funkciók a játékban 17](#_Toc102732446)

[3. Fejlesztői dokumentáció 18](#_Toc102732447)

[3.1 A program főbb technikai információi 18](#_Toc102732448)

[3.2 A program rétegei 19](#_Toc102732449)

[3.3 A program használati eset diagramja 20](#_Toc102732450)

[3.4 A program csomagjai 21](#_Toc102732451)

[3.4.1 a modell csomag 21](#_Toc102732452)

[3.4.1.2 A sárkány osztálya, és Trémaux algoritmusa 22](#_Toc102732453)

[3.4.1.2 A Labirintus osztály 26](#_Toc102732454)

[3.4.1.3 A játékos osztálya 28](#_Toc102732455)

[3.4.2 A nézet csomag 29](#_Toc102732456)

[3.4.2.1 a ResourceLoader osztály 30](#_Toc102732457)

[3.4.2.2 a GUIWindow osztály 31](#_Toc102732458)

[4. Program készítése 31](#_Toc102732459)

[5. Szakdolgozat felépítése 31](#_Toc102732460)

# Bevezetés

A szakdolgozatom témájának kiválasztása, ezáltal alapvető koncepciója az volt, hogy nem tartottam túl valóságos, gyakran előforduló esetnek azt, hogy labirintusba kerüljek valaha, de mindig úgy éreztem, hogy kicsit tartok, félek a megoldandó problémától, hogy nem lennék képes csak a józan eszemre és szerencsémre hivatkozva kijutni. Ez mellé jött az a tulajdonságom, hogy sokkal könnyebben szokott egy új dolog megtanulása, elsajátítása menni akkor, ha valaki azt a cselekvést éppen előttem végzi, és mindig érdeklődéssel töltött el az, ahogy próbáltam rájönni a cselekvés közben hozott döntések miértjére. Ezt a gondolatot és fejlődési vágyat ragadtam meg. Tehát a programom szeretné úgy szórakoztatni a felhasználóját, hogy gondolkodásra, gyorsaságra, döntéshozatalra készteti, illetve ad is neki egy példát, akitől akár tanulhat is, hiszen kijutása során egy ellenfélt állít vele szembe, aki közben mesterséges intelligenciát, gráfkereső és labirintus generáló algoritmusokat használ ellene. Azonban ezt még kevésnek tartottam, szerettem volna azt, hogyha a felhasználók egymással is interakcióba léphetnek valamilyen módon, de azt éreztem, hogy túl sok terhelést jelentene minden éppen online felhasználót megjeleníteni a labirintusban. A felhasználói élmény, és a szociális jelleg növelése érdekében egy közös adatbázisban láttam a megoldást, ahova a felhasználók kedvük szerint készíthetnek labirintusokat, ezáltal adva egy végtelenségig szórakoztató, el nem laposodó jelleget a játéknak, hisz bármely pillanatban bővülhet az adatbázis néhány új labirintussal, amivel aztán a többi felhasználó, vagy akár saját maga is küzdhet. A szociális jelleg egyfajta kompetitív tulajdonságot is hoz magával, ugyanis a játék bár elméletileg a végtelenségig mehet, a közben érkező folyamatos nehezítések, amely leginkább a mesterséges intelligencia fokozatos gyorsítását jelenti, egy idő után lehetetlenné teszik a túl jó pontszám elérését. Kicsit elvesz a kompetitív jellegből viszont a sok véletlenszerűség, hogy véletlenszerű pályát kapunk, ami abból kiindulva, hogy a labirintus kezdő és végpontja között nincs meghatározva sem minimális, sem átlagos távolság, végül változó nehézséget jelent a kijutást illetően. A nyelvek, környezetek választásánál az volt a szempontom, hogy az itteni tanulmányaim során megismert technológiákat, algoritmusokat vegyítsem olyan újdonságokkal, amik a témából kifolyólag érdekelnek engem.

# Felhasználói dokumentáció

Az alapvető probléma, hogy nem volt még szerencsém olyan játékszoftverhez, amelyben volt lehetőség labirintusok létrehozására, illetve a kijutáson kívül valamilyen extra nehézség is került a játékmenetbe. Jelen esetben, a program futása során egyfajta mesterséges intelligenciát személyesít meg, ami az egyik hatékonyabb labirintus útkereső algoritmust használja, ezzel próbálkozik az emberi versenytársa szerencséje, gyorsasága és tudása ellen. A probléma másik fele, hogy bár mindenképp szerettem volna a játékot még ennél is izgalmasabbá tenni azzal, hogy a felhasználókat az egymással történő versengésre, akadályoztatásra buzdítom, nem szerettem volna kiszúrni azon játékosokkal, akik infrastruktúrájuk, vagy pillanatnyi helyzetük miatt nem rendelkeznek aktív internetkapcsolattal. Így a játék rendelkezik egy offline móddal is, ami az eddigi tapasztalataim miatt kicsit nehezebbre is sikerült, mint az online mód, tekintettel arra, hogy az online módban a játékosok hajlamosak egyszerűbb labirintust csinálni, mint egy, a bonyolultságra összpontosító algoritmus.

2.1 A program követelményei hardveres és szoftveres téren  
A program futtatásához egy Windows 10 rendszerű, 64-bites processzoros számítógép szükséges. A pontos, minimum követelmények:

* Processzor: Legalább 1 gigahertz (GHz) órajelű processzor.
* RAM: 2 gigabyte (GB).
* Tárhely: Az operációs rendszeren, és a futtatókörnyezeten felül 2 GB szabad tárhely szükséges.
* Grafikus kártya: DirectX 9 kompatibilis grafikus kártya, legalább WDDM 1.0 illesztőprogrammal.
* Kijelző: Legalább 1280 x 720 pixel felbontású
* Operációs rendszer: A játék Windows és Ubuntu alatt is tud futni, a megfelelő játékélményhez a Windows 10/11 operációs rendszer ajánlott.
* Szoftver: A szoftver futtatásához egy legalább Java 17 Runtime verziót futtatni képes Java verzió szükséges.

## 2.2 A Java telepítése

A játékhoz szükséges java verzió (Windows és Linux esetén) a következő weboldalról érhető el: <https://www.oracle.com/java/technologies/downloads/#jdk18-windows> . Windows esetén a weblapot megnyitva a Windows fülön az „X64 Installer” sorban szereplő linken lehet letölteni a telepítő fájlt.  
A képen szöveg látható

Automatikusan generált leírás

1. ábra : a letöltési link lokációja a weblapon.

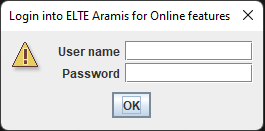
Példaképpen, a Java 18 telepítője 420 megabyte (MB) helyet igényel a merevlemezünkön. A telepítő megnyitásánál, ha szeretnénk eltérni az alapvető telepítési helytől, válasszuk ki a számunkra szimpatikus útvonalat, majd a tovább gombbal fel is települ a futtatási környezet. A képen szöveg látható

Automatikusan generált leírás

2. ábra : a Java futtatási környezet telepítő ablaka.

## 2.3 A program futtatása

A Java futtatási környezet feltelepülése után már csak a játékszoftver futtatása maradt hátra. Ezt a csatolmányban mellékelt tömörített állomány kicsomagolásával, majd a kicsomagolt verzióban, a főkönyvtárban elhelyezett „Labyrinth Adventure.jar” fájl megnyitásával érhetjük el.

A futtatás során (több képernyős rendszerek esetén az operációs rendszerünkben megadott alapértelmezett képernyőn) megnyílik egy kis ablak: 

3. ábra: Bejelentkező képernyő a program megnyitásakor

Ebben az ablakban szükséges az ELTE Aramis adatbázisához rendelkezésünkre álló felhasználónév-jelszó párt beírnunk, amivel később elérjük saját labirintusainkat, vagy megtekinthetjük, esetleg megdönthetjük rekordunkat. Amennyiben ilyennel nem rendelkezünk, a következő felhasználók valamelyikével szintén be tudunk lépni:

* Adminisztrátori felhasználó:
  + felhasználónév: IDU27K
  + jelszó: almafa
* Felhasználói jogokkal rendelkező felhasználók:
  + felhasználónév: HF3LBG
  + jelszó: HF3LBG
  + felhasználónév: RNYR2F
  + jelszó: RNYR2F
  + felhasználónév: GF4SDH
  + jelszó: GF4SDH

Ha nem rendelkezünk, vagy nem szeretnénk az imént megadottfelhasználók valamelyikét használni, a bemeneti mezőket hagyhatjuk üresen is, ezzel viszont csak az offline funkciók maradnak használhatóak. Ha ezt a döntést meghoztuk, az „OK” feliratú gombra kattintva a főmenübe jutunk.

2.4 Új felhasználó regisztrálása

Ha szeretnénk egy új felhasználót regisztrálni, azt úgy tehetjük meg, ha egy e-mailt írunk a szerver üzemeltetőjének és adminisztrátorának, a [nikovits@inf.elte.hu](mailto:nikovits@inf.elte.hu) címre.  
A levélben szerepeljen, hogy ezzel a játékkal szeretnénk játszani, és az is, hogy milyen felhasználónevet szeretnénk. Jelszót nem szükséges az e-mailbe írni, egy generált jelszót fog adni az adminisztrátor a válaszlevélben.  
A kért felhasználó létrehozása akár több ideig is eltarthat, addig is használhatjuk a feljebb megadottakat.

## **2.5 A játékszoftver funkciói**

A képen szöveg, áramkör, elektronika látható

Automatikusan generált leírás

4. ábra: A főmenü kinézete, ha nem elérhetőek az online funkciók

Sikertelen bejelentkezés, vagy szándékosan üresen hagyott bejelentkezési adatok esetén azt tapasztalhatjuk, hogy csak a „Play Offline” és „Exit Game” feliratú gombok elérhetőek. Amennyiben ez nem az elvárt eredmény, ellenőrizzük internetkapcsolatunkat és a program újra futtatásával, majd gondosan ellenőrzött bejelentkezési adatok megadásával próbálkozzunk ismét. Ha viszont a bejelentkezés sikeres, a következő főmenü jelenik meg:

A képen szöveg látható

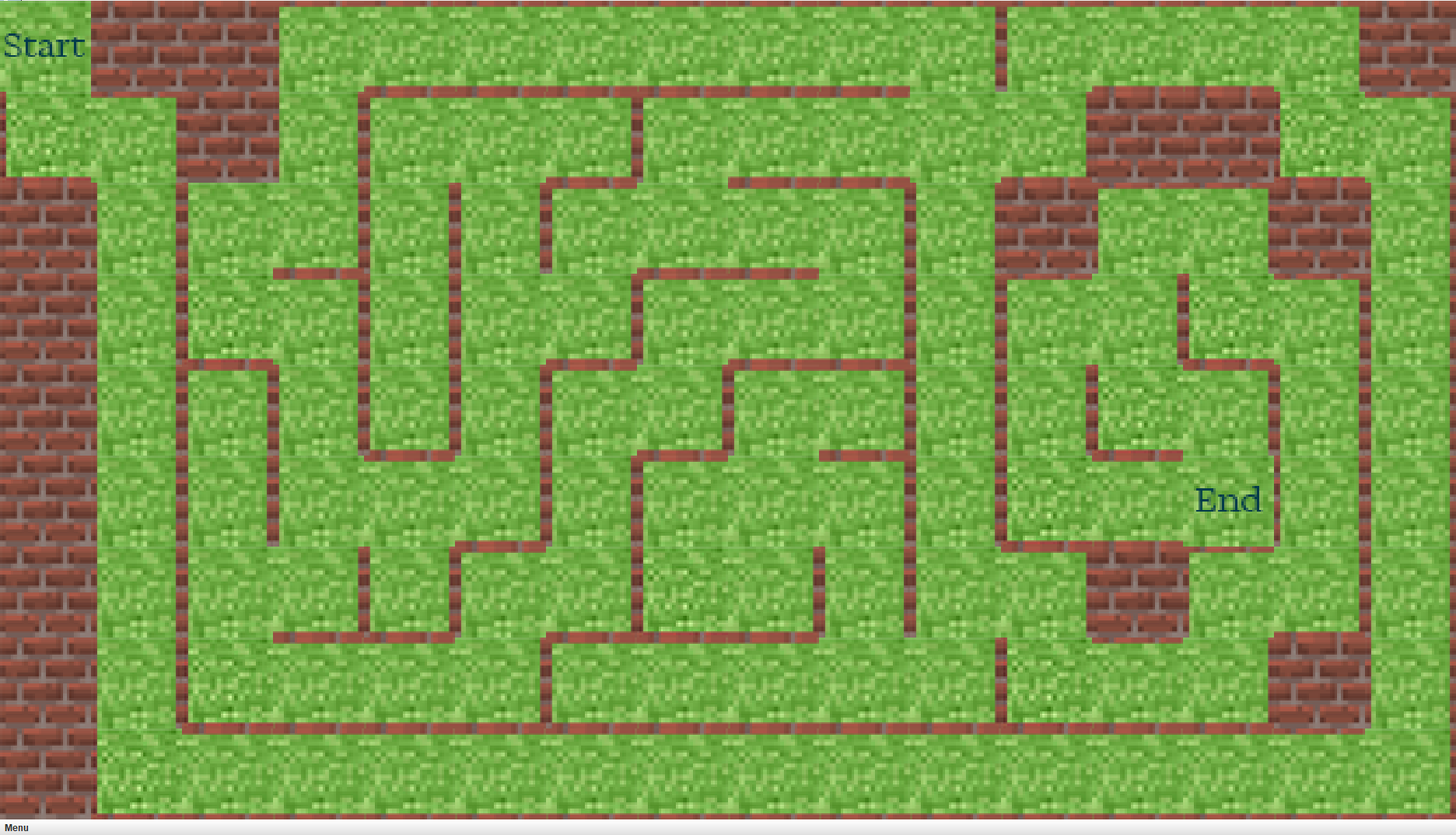
Automatikusan generált leírás

5. ábra: A főmenü kinézete, sikeres bejelentkezés, ezáltal elérhető online funkciók mellett

2.5.1 A pályakészítő funkció

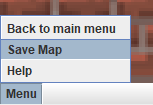
A főmenüből az elérhető menüpontok, és funkciójuk balról jobbra:  
„Map Builder” (csak online elérhető): Ezen menüpont alatt van lehetőségünk a pályakészítő funkciót elérni.

A pályakészítőben egy labirintust úgy tudunk megrajzolni, először az egérmutatót az általunk kiválasztott kezdőcellára helyezzük, majd a bal egérgombot lenyomva, és a kurzort valamilyen irányba húzva elkezdjük kialakítani a labirintusunkat. A labirintusunk kezdőhelye egy „Start” felirattal lesz jelölve, a labirintusból kivezető utat pedig teljesen a saját elképzelésünkre bízhatjuk. A végpontot bármikor változtathatjuk úgy, hogy a jobb egérgombbal a megfelelő cellára kattintunk. Ha nem tettünk ilyet, és úgy próbáljuk elmenteni a labirintust, akkor automatikusan a legutolsó olyan cellát jelöli ki, amelynél utat csináltunk.

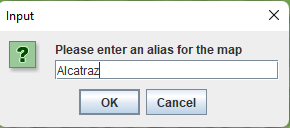
Arra viszont figyelnünk kell, hogy ha a labirintus kezdőpontjából a végpontjába nem vezet bejárható út (például úgy, hogy a labirintusunk végpontja és kezdőpontja közti részt minden esetben legalább egy fal elválasztja), akkor egy későbbi pontban az elmentést, az adatbázisba feltöltést a program nem fogja engedni. Egy példa egy elkészített labirintusról:  


6. ábra: Egy mentésre kész labirintus

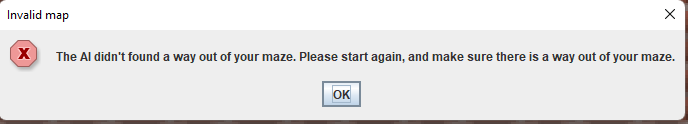
Ha elégedettek vagyunk az elkészített labirintussal, a képernyő bal alsó sarkában látható „Menu” feliratú gombra kattintva, majd a „Save Map” feliratú gombra kattintva a játék arra fog kérni bennünket, adjunk egy becenevet a labirintusunknak, amiről később majd felismerjük. Természetesen, ha mégsem szeretnénk elmenteni a labirintusunkat, a „Back to main menu” gombot megnyomva ismét a főmenüben találjuk magunkat.



7. ábra: Képernyő bal alsó sarkában található "Menu" gomb, amiből aztán a "Save Map" menüpont lesz elérhető.



8. ábra: Becenév adása a labirintusnak

Ha az OK gombra kattintottunk, a szoftver a háttérben ellenőrzi, hogy ténylegesen van-e kiút a labirintusból, amit készítettünk, és ha talált, azt úgy jelzi, hogy ki lép az aktuális ablakból. Ha viszont olyan labirintust szeretnénk elmenteni, ami nem rendelkezik kiúttal, a következő hibaüzenet fogad minket:   
  


9. ábra: Hibaüzenet, ha olyan labirintust készítenénk, amiből nincs kiút

Ha viszont nem kaptunk hibaüzenetet, és semmilyen más internetkapcsolati probléma nem avatkozott közbe, a „My Maps” menüpont alatt megtekinthető az új labirintusunk, de ezt a menüpontot még részletesen bemutatok.

### 2.5.2 Online játék

Ez a funkció csak aktív internetkapcsolat mellett elérhető. Ha a „Play Online” marad. Így folytatódik ez mindaddig, amíg vagy mi, vagy a sárkány nem talál ki a labirintusból.

Ez a játékmód annyiban jelent többet az offline, azaz internetkapcsolat nélkül is játszható módnál, hogy ebben az esetben csak olyan labirintusok jönnek elő, amelyeket egy valós felhasználó készített. Ha esetleg ismerős számunkra az egyik labirintus, érdemes megnézni a lenti sávban a készítő nevét, hisz könnyen előfordulhat, hogy pont mi készítettük.

menüpontot választjuk, egy nagyrészt felfedezetlen labirintusban találjuk magunkat egy sárkánnyal. A bal alsó sarokban látjuk a készítő felhasználó nevét, illetve kis várakozás esetén azt is megfigyelhetjük, hogy a sárkány tétovázás nélkül elkezdte a maga kis útját járni. Ebben a játékban ő lesz az ellenfelünk, aki folyamatosan azon lesz, hogy az algoritmusból könnyebben kijusson, mint mi. Hogy ezt hogyan teszi pontosan, ha érdekel bennünket, bármikor követhetjük útján, és tanulhatjuk el tőle a taktikáját, viszont az egyáltalán nem biztos, hogy jó döntés azokat a lépéseket követni, amiket a sárkánymeghoz.  
  
De ha elindulunk például saját utunkon, amit a billentyűzeten található négy nyíllal jelzett billentyű segítségével tehetünk meg, elkezdhetjük bejárni a labirintust, és ahogy odaérünk egy eddig még felfedezetlen részhez, úgy minden irányban felfedeződnek az ezelőtt még számunkra ismeretlen folyosók. Ezek később sem felejtődnek el, tehát ha már tovább haladunk, és egy másik folyosóba megyünk, ahol a falak rendszere miatt egyébként nem látnánk át az első, eredeti folyosóra, attól még az ugyanúgy látható



10. ábra: Online játékmenet a sárkánnyal és a karakterrel

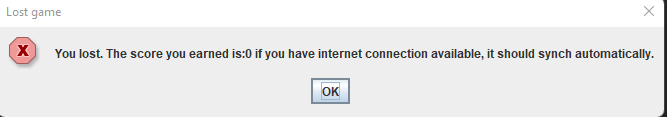
Abban az esetben, hogyha megtaláltuk az „End” feliratú mezőt a labirintusban, amely a kiutat szimbolizálja, érdemes minél hamarabb oda sietnünk, és várnunk a következő pálya betöltését.

Bizonyára feltűnhetett az első lejátszott alkalom után, hogy a sárkány jelentősen lassabban mozgott, mint amilyen gyorsan a kis kék karakterünk mozogni képes. Itt jön a képbe a pontszám jelentősége. A pontszám egyenlő a már, sárkánynál gyorsabban megoldott labirintusok számával. Így kezdeti állapotban ez a szám 0, emiatt a sárkány az alapvető, leglassabb sebességgel fog a kiút felé haladni.  
  
Így hogyha kijutottunk egy labirintusból, a feladat végül is nem változna, egy részt kivéve: A sárkány a pontszámunk növekedésével egyre jobban, és jobban fog próbálkozni, hogy emberi ellenfelén felülkerekedjen. Ezt szimplán úgy teszi, hogy labirintusonként növekszik egy picit a sebessége.

A pontszámunk, az adott labirintusban eltöltött másodpercek száma és a labirintus készítője mindig megjelenik az alsó menüsávban, így egy hosszabb, akár 10 labirintusos játékmenet után is mindig tisztában lehetünk azzal, hogy mennyi szükséges az éppen aktuális rekordunk megdöntéséhez.



11. ábra: Menüsor, ahol látható a labirintus készítője, a pontszám és az idő

Ha azonban sem a szerencsénk, sem a bevetett tudásunk nem hozta meg a tőle várt győzelmet, egy felugró ablak jelzi, hogy a sárkány nálunk gyorsabban kijutott. Ekkor, amennyiben sikerült a rekordunkat megdönteni, az új pontszámunk a felhővel szinkronizálódik, majd a játékmenet végezetéül ismét a főmenüben találjuk magunkat.  
  


12. ábra: A játék végét jelző üzenet

A játékot nehezíti, hogy tekintettel arra, hogy ez mégis csak egy kompetitív, szellemi jelenlétet igénylő játék, szünet tartására nincsen lehetőség. Ezzel elkerüljük az olyan „csalásokat”, ha például a szüneteltetett játék közben gondolkozna a játékos azon, hogy merre érdemes tovább indulnia, vagy tapasztalt játékosok esetében azt, hogy a lementett, lefényképezett labirintusok közül melyik az amelyikből aktuálisan ki kellene jutniuk. Ha viszont úgy érezzük, hogy a pályának csak nagyon lassan találnánk meg a kiútját, esetleg belefáradtunk az adott labirintus megoldásába, az alulról elérhető menüből bármikor kérhetünk egy új labirintust, viszont ezzel feláldozzuk a jelenlegi pontszámunkat. Természetesen ugyanígy a „Back to main menu” gombbal a főmenübe is visszajuthatunk, a játék további funkcióinak megismeréséhez.

### 2.5.3 Offline játék

Ha azonban nem vagyunk abban a kényelmes helyzetben, hogy aktív internetkapcsolattal rendelkezünk, például vonaton, repülőn, hajón ülünk, akkor is tartogat számunkra ez a játékszoftver, még ha csak csökkentett funkcionalitással is.

A játékmenet nagyon hasonló lesz ebben az esetben is, mint hogyha internetkapcsolattal rendelkeznénk: Ebben az esetben is a sárkány lesz az ellenfelünk, aki most is gyorsul annak függvényében, hogy hány labirintusból jutottunk ki egyhuzamban gyorsabban, mint bináris ellenfelünk. A különbség a pályák nehézségében és jellegében lesz, mivel nem tudunk ebben az esetben az adatbázisból adatot beszerezni. A játékprogram egy beépített algoritmussal rendelkezik, amely képes komplex labirintusokat készíteni, és pontosan ez kerül használatba az offline játékmód esetén. Ez az algoritmus hajlamos nagyon véletlenszerű, emiatt akár nagyon könnyű vagy nagyon nehéz labirintusokat is generálni, így alapvetően offline üzemmódban átlagosan kicsit nehezebb a kijutás, ezáltal a pontszerzés is.

A képen térkép látható

Automatikusan generált leírás

13. ábra: Egy algoritmus által generált labirintus

Természetesen a labirintus mérete a nehézség körülbelüli szabályozása érdekében mindig ugyanakkora, és az online, játékosok által pályák is lehetnek az elvártnál sokkal nagyobb erőfeszítést igénylőek. A játék viszont ebben a formájában ténylegesen gyakorlási célra szolgál, hiszen habár számolja az elért pontszámot a játék, ezt a játékmenet végén, azaz egy olyan esetben amikor a sárkány felülkerekedik rajtunk, nincs lehetőség a felhőbe menteni.

### 2.5.4 Toplista

Ahogy véget ér az első néhány lejátszott kör, egyre kíváncsibbak lehetünk arra, vajon hasonló érdeklődésű embertársaink hogyan teljesítettek az útvesztőkből való kijutásban. Ez a menüpont pontosan erre a kérdésre adja meg a választ. A menüben rákattintva egy kis ablakban nyílik meg a toplista, melyben az oszlopok szerint balról jobbra haladva egyenként a sorszámot, a hozzá tartozó felhasználót, és az általa elért legmagasabb pontszámát láthatjuk. Kilépni a jobb felső sarokban található „X” szimbólumra kattintással tudunk. Előfordulhat, hogy kis időbe telik, míg a frissen elért rekord eredményünk ide is bekerül, az is megeshet, hogy csak a következő játékindításnál lesz elérhető.

A képen asztal látható

Automatikusan generált leírás

14. ábra: A toplista

### 2.5.5 Saját labirintusok menüpontja

Ez a menüpont akkor lesz a felhasználó számára érdekes, hogyha a pályakészítő funkcióban már eljutott egy remekmű elkészítéséig. Azért, mert itt kilistázásra kerül az adott, éppen bejelentkező felhasználóhoz tartozó összes, az adatbázisban aktuálisan szereplő labirintus. Mivel itt is szükséges az adatbázissal adatot cserélni, hogy biztosan naprakész információt láthasson a játékos, elengedhetetlen az aktív internetkapcsolat.

Ha a menüpontra kattintunk tehát, egy, a toplistához nagyon hasonló táblázat fogad bennünket, benne az összes elkészített pályánkkal. A pályáknak látható a szöveges reprezentációja, illetve a felismerhetőség céljából az a becenév is, amit az elkészítésekor adtunk neki,

A képen asztal látható

Automatikusan generált leírás

Ebben a funkcióban van lehetőségünk visszavonni egy elkészített pályánkat. Lehet, hogy többet nem szeretnénk a készítők körébe tartozni, vagy szimplán túl könnyűnek, esetleg túl nehéznek ítéljük meg valamelyik elkészített pályát. Ha bármilyen itt leírt vagy nem leírt okból törölnénk, azt úgy tehetjük meg, hogy először kiválasztjuk azt a sort, amelyik a törlendő pálya adatait tartalmazza. A kijelölést kék színnel fogja a szoftver jelölni, majd ha biztosak vagyunk a döntésünkben, a sorok felett található „Delete Map” feliratra kattintva törölhetjük az adott labirintust az adatbázisból. A sikeres törlést egy felugró ablakkal jelzi a szoftver. Előfordulhat, hogy kis idő eltelik mire a labirintusaink közül tényleg eltűnik a frissen kitörölt pálya. A legszélsőségesebb esetben itt is a következő futtatásig bezárólag garantált, hogy a pálya eltűnik a játszható labirintusok sorából.  
  
2.5.6 Kilépés a játékból

Hogyha úgy döntöttünk, hogy elég volt a játékból, vagy abban az esetben, ha az internetkapcsolatunk elérhetősége változik, ez a funkció lehet a segítségünkre. A menüből utolsó gombként elérhető „Exit Game” megnyomásával, aktív internetkapcsolat esetén bontja a kapcsolatot a szerverrel a játék, amivel jelzi, hogy az általunk létrehozott új pályák, illetve esetlegesen megdöntött rekord egy ideig véglegesnek tekinthető. Ha offline állapotban vagyunk, de időközben lett internetkapcsolatunk, akkor egy gyors újraindítással, és bejelentkezéssel hamar elérhetővé tehetjük az online funkciókat.

## 2.6 Segítség funkciók a játékban

A pályakészítőben, illetve az online és offline játékmenetben az alul lévő Menu gombbal elérhető a „Help” alatt egy kis súgó, amivel az adott játékmenet, funkció világossá válik a kezdő játékosok számára.

# Fejlesztői dokumentáció

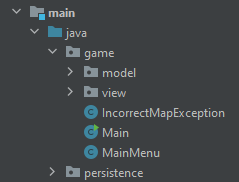
## 3.1 A program főbb technikai információi

Ez a program Java 17-es nyelvi szinten íródott, amely az úgynevezett Gradle projektépítő eszközt használja. A futtatáshoz, fejlesztéshez szükséges főbb függőségei:

* A Gradle projektépítő szoftver, amely néhány jelenlegi függőséget, és a jövőbeli függőségek tervezésében játszik jelentős szerepet.
* Bármely 17-es, vagy újabb verziójú Java Development Kit (szintén elérhető a <https://www.oracle.com/java/technologies/downloads/#jdk18-windows> weblapon).
* A betöltendő projekt egy JetBrains IntelliJ IDE ( Integrated Development Environment, azaz integrált fejlesztői környezet) projekt, amelyben minden be van konfigurálva a továbbfejlesztés, futtatás, tesztelés megkönnyítéséhez, így ezt a fejlesztői eszközt ajánlom.  
  A program fontosabb, nem triviális függvényei Javadoc-kal vannak dokumentálva, és magyarázva.
* Java Secure Channel 0.1.55-ös verzió, a porttovábbításhoz, és az ELTE Caesar szerveréhez való csatlakozáshoz.
* Oracle Java Database Connector 11-es verzió, az ELTE Aramis Oracle SQL szerverhez történő csatlakozáshoz, az ott történő lekérdezések, frissítések elvégzéséhez szükséges.
* JUnit 4.13.2 verziójú egységteszt-keretrendszer, ami a jelenlegi tesztkonfiguráció futtatásához szükséges.
* Hamcrest 1.3 verziójú keretrendszer, amely a JUnit használatához szükséges.

## 3.2 A program rétegei

A program összessége, a programtervezési mintának megfelelően csomagokra van bontva, így is kihasználva az objektum elvű programozásnak ezen részéből eredő láthatósági szabályok előnyeit.



15. ábra: A program csomagjainak struktúrája

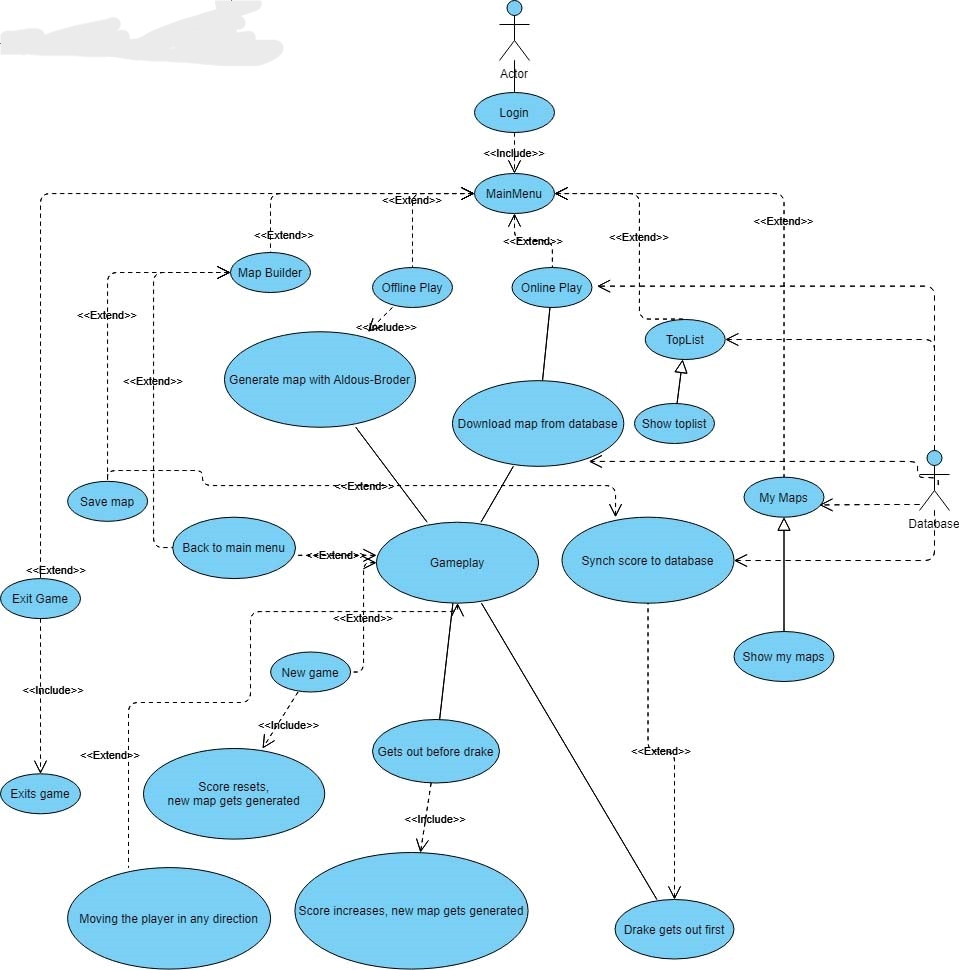
A program alapvetően a modell-nézet-vezérlő programtervezési mintát használja, így alapvetően a rétegek tekintetében igyekszik a felhasználói felületet, és az azon megjelenítendő, ábrázolandó adatot különválasztani. Ez azért fontos, hogy a felhasználói felület az adatkezelést ne befolyásolja, és az adatok átszervezhetők legyenek a felhasználói felület változtatása nélkül.

Tulajdonképpen esetünkben a vezérlőnek a főmenü tekinthető, hiszen ez az osztály hívja meg az osztályoknak az éppen használt funkcióhoz leginkább alkalmazkodó formáját. Ebben vannak azok a konstruktor illetve metódushívások, amik aztán a komponensek további funkcionalitását elindítják, ezzel az egész programból egy összefüggő egészet alkotva.

A modell rétegben helyezkedik el minden olyan entitás vagy objektum, amelyet később majd a nézeti rétegben megjelenítünk. Ide tartoznak a Dragon, a Cell, a Player, és a Labyrinth osztályok. Ebben a csomagban van leimplementálva többek között a sárkány döntései mögött álló mesterséges intelligencia, a cellák esetében a falakhoz, illetve a cella labirintusban elfoglalt helyéhez tartozó adattagok és lekérdező illetve beállító metódusai, vagy a játékoshoz tartozó pixel koordináták, illetve a mozgásának kezeléséhez szükséges egyéb változók.

A nézeti réteg lényegében az összes, főmenüből elérhető ablakot külön osztályban tartalmazza, egy olyan plusz kitétellel, hogy a játék komplexebb részein, például az offline vagy online játékban, a plusz izoláció érdekében további osztályok bevezetésére került sor. Az öröklődés lehető legjobb kihasználása érdekében létrehozásra került egy absztrakt osztály, illetve hogy a játékban használt grafikus elemek egy helyen kerüljenek betöltésre és kezelésre, egy statikus osztály is.

## 3.3 A program használati eset diagramja



16. ábra A program használati eset diagramja

A használati eset diagram segítségével megérthető, hogy az egyes szereplők, esetünkben a játékos milyen típusú interakciókat tud végrehajtani, és annak a jelzésére, megjegyzésére is adódik lehetőség, hogy ha ehhez van valamilyen függőség, ami által elérhetővé válik, a speciális esetünkben néhány funkció esetén az adatbázis az ilyen.

## 3.4 A program csomagjai

### 3.4.1 a modell csomag



17. ábra A modell csomag osztályainak UML-diagramja

#### 3.4.1.1 a cella osztály

Ahogy a fenti ábrán is látható, a legfontosabb, legkomplexebb osztály a Cell, azaz a cellák megvalósítása. Ebben az osztályban a falak egyenként egy boolean típusú változóval állíthatóak be, illetve a cellában tároljuk azt is, hogy a játékmenet során látható-e a játékos számára, hogy a labirintuson belül milyen pozícióban van, illetve található néhány segédváltozó is a sárkány útkereső algoritmusához.

Mivel a cella esetében nagyon sok féle állapot lehetséges (például csak bal oldalon rendelkezik fallal, vagy alul-felül, vagy akár mind a négy oldalról blokkolt a bejutás az adott cellába) ezért a selectImage metódussal a cella a megfelelő paraméterű képet juttatja vissza a nézet csomagból érdeklődő osztályoknak.

A sárkány útkereső algoritmusához, a hasBeenVisitedByDragon és az IsDeadEnd nevű változók tartoznak, ezen értékeket a bejárás során változtatja, ezzel elősegítve, hogy ugyanazon útra ne menjen rá kettőnél többször.

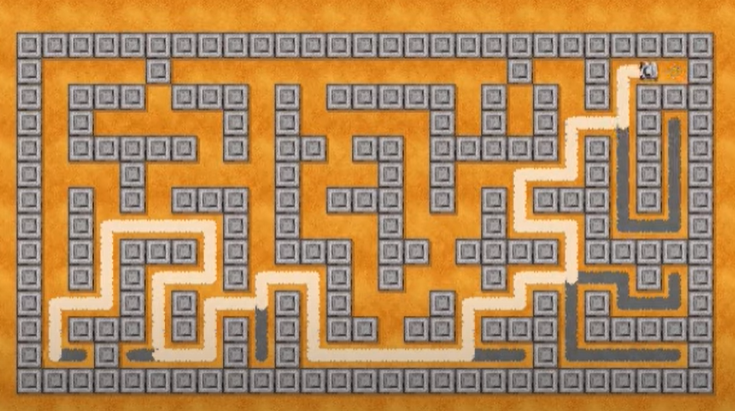
Szintén itt van megoldva az isVisibleForPlayer boolean segédváltozóval az, hogy amikor a játék elkezdődik, a játékos számára nem láthatóak azon részei a labirintusnak, ahol még egyáltalán nem járt, ezzel kicsit realisztikusabbá és nehezebbé téve az útkeresést.

### 3.4.1.2 A sárkány osztálya, és Trémaux algoritmusa

A sárkány személyesíti meg a mesterséges intelligenciát a játékszoftverben, így ebben az osztályban, az alapvető pozícióval kapcsolatos dolgokat kivéve leginkább a Trémaux útkereső algoritmus implementálása található meg. Az útkeresés, ezáltal a felhasználóval való versenyzés a következő, egyszerű szabályok követésével magyarázható:

* Minden utat, ahol járunk, megjelölünk. A megjelölés módja bár esetünkben segédváltozó, a valóságban fontos az is, hogy egyértelműen el lehessen dönteni a jelölés alapján, hányszor jártunk arra.
* Olyan útra, ami már kétszeres jelöléssel (vagy esetünkben az isDeadEnd változó true állapotával) rendelkezik, tilos harmadjára is lépni.
* Ha egy kereszteződésbe (olyan cella, amit legalább 3 különböző irányba lehet elhagyni) érünk, és a kereszteződésben nem jártunk még egyik úton sem (kivéve azt, amelyikből a kereszteződésbe érkeztünk), válasszunk véletlenszerűen egy utat a még nem felfedezettek közül, és haladjunk tovább.
* Ha olyan kereszteződésbe értünk, ahol már jártunk egyszer ( tehát azon az úton kívül, amin érkeztünk, létezik legalább egy olyan másik út, ami már legalább egyszer meg lett jelölve), akkor válasszunk egy olyan utat, ahol a lehető legkevesebb jelölés van eddig (lehetőleg 0, különben 1.)
* Ha olyan kereszteződésbe értünk, ahol már minden útnál két jelölést találunk, kivéve azt amiből a kereszteződésbe érkeztünk, forduljunk vissza, és jelöljük meg a kereszteződést zsákutcaként.

Ez az algoritmus egyébként a 19. századból származik, és körülbelül száz évvel később generalizálták az algoritmust, és a mélységi keresés nevet kapta.



18. ábra A Trémaux algoritmus. Szürke vonal jelöli a már kétszer, fehér vonal jelöli a csak egyszer megjelölt utat.

Tehát így a sárkány egy úton maximum kétszer fog végig haladni, ezzel pedig jóval hatékonyabban, gyorsabban tud adott esetben kijutni a labirintusból, mint a játékos.

Az algoritmus úgy került implementálásra, hogy az egy jelölést a hasBeenVisitedByDragon true értéke jelentette, a két jelölést, mivel ez magával hozza azt is, hogy arra az útra többet nem szabad lépni, a DeadEnd változóval helyettesítettem. Az implementációban a kisebb komponenseket a doTremauxPathfinding függvény fogja össze, ami igaz értékkel tér vissza akkor, ha megtalálta az utat, különben hamissal. Az algoritmusban a ciklus addig fut, amíg nem találtuk meg a végső cellát, vagy amíg az egyik belső függvénye nem mondja azt, hogy nem tud már hova menni, ezzel egyértelműsítve azt, hogy nincs kiút a labirintusból.  
A ciklus egyszerűen működik mindaddig, amíg nem érkezünk el egy kereszteződéshez, ahol a leírt szabályokat kell majd alkalmazni. Tehát amíg nincs kereszteződés, addig triviális a sárkány számára, hogy mi a következő lépés, így ezt a goUntilJunction függvényben egyetlen lépésben meg tudja tenni.

Ez a függvény folyamatosan keresi a következő irányt, és többször felmerülhet a túlindexelés problémája, így erre egy isValidMove függvényt implementáltam, ami azt vizsgálja, kiindexelnénk-e a tömbből, hogyha a paraméterben megadott irányba mennénk. Az irányok jobbra, balra, fel és le lehetnek, ezeket egy enumerációban, a Direction osztályban egyértelműsítettem.  
Hogyha az isValidMove igazzal tér vissza, akkor tudhatjuk, hogy lekérhetjük a tömbből azt a szomszédot, amerre az adott iterációban épp mozogni szeretnénk. Ha lekértük a szomszédos cellát, megvizsgáljuk, hogy mozoghatunk-e abba az irányba egyáltalán (akadályozza-e fal a haladást), és hogy ha igen, akkor azt hogy hány jelölés van ott.  
Első körben mindig azt az irányt keressük, ami még felfedezetlen, ezzel elkerüljük azt hogy véletlenszerűen visszaforduljon a sárkány egy kanyargósabb folyosó közepén.  
Ha viszont ilyen utat nem talált, akkor valószínűleg abban a helyzetben vagyunk, hogy már visszafele jövünk ezen az úton, így ebben az esetben még egyszer megnézzük, hogy melyik irányba szabályos a mozgás, és a második esetben már csak az lesz a kitétel, hogy kétszer jelölt utat ne válasszunk. Ha ezek után sem sikerült olyan irányt választani, amely bármelyik feltételnek megfelel, akkor hamissal tér vissza a goUntilJunction függvény, ezzel jelezve az őt hívó függvénynek, hogy nincs kiút, ne próbálkozzon tovább. Ez az eset viszont ritkán fordul elő, így nézzük mi történik ha bármelyik körben talál egy jó irányt.

Az új irány kiválasztása után megpróbáljuk változtatni a sárkány helyét, ezt a selectNewCell metódussal tesszük, ami egy irányt kap paraméterül, méghozzá azt, amit az előbb a szabályok alapján a legjobb iránynak véltünk. ez a függvény aztán, ha ebben az esetben sem indexelnénk ki a tömbből, akkor a tryGoingToNeighbourCell függvényt hívja, ami végül meg is változtatja a sárkány aktuális pozícióját, ha azt a falak szerkezete, vagy a dupla jelölés nem gátolja. Abban az esetben, ha sikerült a helyváltoztatás, a képbe jön a késleltetés: mivel ez a kis szoftver egy szempillantás alatt lefut minden modern számítógépen, nem lenne túl fair a játékossal szemben, ha nem fognánk vissza a képességeit. Így az adott szálat, amin az algoritmus fut, bizonyos időre elaltatjuk, és ez a bizonyos idő a nézetben meghatározott paramétertől fog függeni, hiszen minél több ponttal rendelkezik a játékos, az algoritmus annál kevésbé van visszafogva. Tulajdonképpen ez volt egy iterációja a GoUntilJunction függvénynek, és ezt mindaddig csinálja, míg nem fut bele egy kereszteződésbe.

Kereszteződés esetén a fő függvénybe megyünk vissza, ahol a program fel fogja ismerni, hogy meg kell hoznunk az optimális döntést, azaz hogy melyik irányba megyünk tovább. Hogyha minden másik út, ahova mehetnénk, már kétszeres jelöléssel rendelkezik, könnyű a dolgunk, mert csak vissza kell fordulnunk. Ha nem ilyen egyszerű a dolgunk, a makeChoiceInJunction függvényt hívjuk meg, hogy elemezze ki a helyzetet.

a makeChoiceInJunction függvény a lehetséges irányokat véletlenszerűen próbálgatja, és keresi közülük azt, amelyik a legkevesebb jelöléssel rendelkezik. Ha csak egyetlen utat talált, azt jelenti hogy zsákutcában vagyunk, így meg is jelöli azt a területet, de mindenképpen visszatér egy iránnyal, amit aztán a fő hívó függvényünk megkap. A fő függvény ezután újra mozdulni próbál, a megállapított legjobb irányba, és ezzel végig is értünk egy iterációján a fő függvénynek, hiszen elértünk egy kereszteződésig, hoztunk egy döntést a következő irányról, majd meg is tettük a lépést abba az irányba. A következő iterációban újra vizsgálni fogjuk, hogy megint kereszteződésbe kerültünk-e, és így megy ez egészen addig, amíg nem találjuk meg a kiutat, vagy rá nem jövünk arra, hogy valójában nem is elérhető számunkra a kiút. Alább látható a fő hívó függvény struktúrája.

A képen szöveg látható

Automatikusan generált leírás

19. ábra A Trémaux algoritmus fő függvénye

### 3.4.1.2 A Labirintus osztály

A labirintus osztály felelős azért, hogy előteremtse a játékos számára a labirintust vagy egy adatbázisból letöltött adat segítségével, vagy pedig az Aldous-Broder algoritmussal, valamint azért, hogy a labirintusnak csak egy kezdete és egy kiútja legyen.  
  
Az Aldous-Broder algoritmust a játék offline állapotában hívjuk meg. Ez az algoritmus egy olyan komplex labirintust készít, amelynek minden pontjából minden pontjába el lehet jutni. Komplex labirintusnak hívjuk azokat a labirintusokat, amelyek kiútja nem található meg az úgynevezett Wall-Follower algoritmussal, amely annyiból áll, hogy végig a menetirányhoz képest jobb oldali falat követjük. Azzal a példával is szoktak élni, hogy ha a labirintus falai olyan anyagból lennének, ami vezeti az áramot, akkor ha egy szimpla labirintusban áramot vezetnénk a falba, az végighaladna a labirintus összes falán, de a komplex útvesztő esetében nem.

Az algoritmus, ami a labirintust generálja, egész egyszerű, és a véletlenszerűsége elősegíti a játék újrajátszhatóságát és változatosságát. Ez az algoritmus is igényli azt hogy a cellákat valamilyen módon meg tudjuk jelölni a generálás során, de a jelenlegi implementációban ezt úgy oldottam meg, hogy ha a cellába valamelyik oldalról be lehet lépni, akkor arra mondhatjuk azt, hogy már legalább egyszer meglátogattuk. Az algoritmus tehát a következő lépésekből áll:

1. Válasszunk egy véletlenszerű cellát, és jelöljük meg, mint már látogatott cella.
2. Csináljuk a következőt, amíg van olyan cella ami még nem volt feldolgozva:
   1. Válasszunk véletlenszerűen egy szomszédos cellát.
   2. Hogyha a választott szomszédos cella még nem volt látogatva:
      1. Tüntessük el a falat az előző és a jelenlegi cella között
      2. Jelöljük meg a jelenlegi cellát látogatottként.
   3. Jelöljük meg a választott szomszédot látogatottként.

A képen szöveg látható

Automatikusan generált leírás

20. ábra az Aldous-Broder algoritmus a kódban.

A labirintusnak ez az algoritmusa tehát két nagyon fontos függvénnyel rendelkezik: A szomszédra átváltó metódus, esetünkben a moveToAdjacentCell, és a generálás végét vizsgáló függvény, az isEndOfGeneration. Előbbi véletlenszerűen kiválaszt egy szomszédos cellát, ami ha még nem volt látogatva, meghívja a makeWayToAdjacentCell metódust, átadva neki azt hogy melyik irányba lévő szomszédjához csináljon utat. Utóbbi szimplán azt vizsgálja, hogy a tömbünk minden cellájához lett-e csinálva legalább egy út, ezzel biztosítva azt, hogy amikor kezdeti és végpontot választunk a labirintusnak, biztosan legyen a kettő között út.  
Így a cellák látogatottsági állapota a Cell osztályban található GetHasBeenSelected metódussal érhető el, ami egyszerűen megnézi, hogy minden fala fent van-e.

Viszont mindezt csak akkor használjuk, ha a konstruktor ennek megfelelően van felparaméterezve, különben csak feldolgozzuk a labirintus szöveges reprezentációját, és annak megfelelően állítjuk be a cellák falait. A labirintus szöveges reprezentációja a következőképpen néz ki:

* A sorrendje azonos a tömbben eltárolt cellák sorrendjével, így a tömbbel együtt iterálható a szöveg is a labirintus betöltésekor. A cellák információi szóköz karakterrel vannak elválasztva, és a cellák állapota a következőképpen derül ki:
  + Minden cella állapotát 4, vagy 5 karakter reprezentálja.
  + Az első 4 karakter vagy 1, vagy 0 értékkel rendelkezik, ezzel reprezentálva azt, hogy az adott irányból kell e fal a cellához, vagy sem. Az irányok a következő sorrendben vannak: Fel, le, balra, jobbra.
  + Az 5. karakter, ha van, akkor egy „s”, vagy egy „e” betű. Ezekből pályánként csak egy van a szövegben, hiszen a labirintusoknak csak egy kezdőpontja van.Előbbi a kezdő cellát, utóbbi a kiutat jelzi.

Tehát ha a cella szöveges reprezentációja „1101s” akkor egy olyan kezdőcellát kell létrehozni, amelybe csak a bal oldaláról lehet belépni, vagy ha „0000” lenne, akkor egy olyan celláról van szó, amely minden oldalról nyitott.

Így már minden részinformáció ismert ahoz, hogy a konstruktor paraméterezhetőségéről beszéljünk. A két paraméter, amit fogad, a boolean típusú isMapGenerationNeeded, és a mapData. az isMapGenerationNeeded értéke attól függ, hogy szeretnénk-e az Aldous-Broder algoritmust futtatni, míg a mapData egy szöveges típusú paraméter, és nem üres akkor, hogyha van olyan adat amit be szeretnénk tölteni. Így egyetlen konstruktor lesz használható a később bemutatandó pályaépítő, az offline és online játékokhoz, ezzel elősegítve a kód karbantarthatóságát.

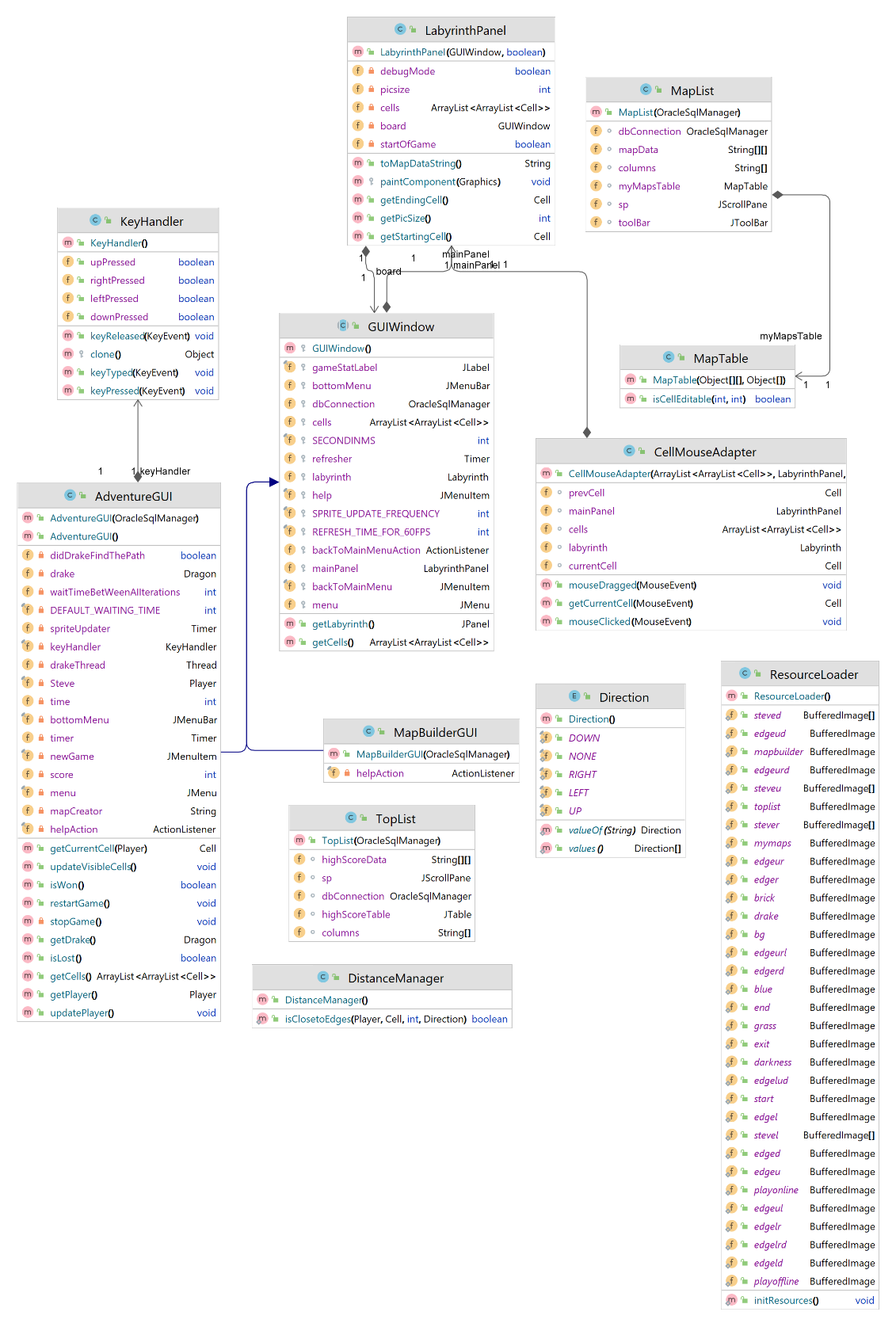
### 3.4.1.3 A játékos osztálya

Ez az osztály felelős azért, hogy a labirintusban egy bizonyos pozícióban elhelyezkedő játékos állapota tisztán elérhető legyen a nézet csomag számára, így tartalmazza azt, hogy a játékos milyen irányba néz, éppen mozog-e, melyik cellában, illetve azon belül is hogy hányadik pixelnél jár éppen. Mivel a játékos az egyetlen jelenlegi osztály, melynek futásidőben gyakran változik a reprezentálásra használt képe, ezt az osztályban egy külön tömbben, a myLooks-ban tárolom. Ez a paraméter aztán folyamatosan változik a példány állapotának függvényében, amikor a nézet csomag valamely komponense meghívja az updateLook függvényt.  
A képen szöveg látható

Automatikusan generált leírás21. ábra az updateLook függvény, amely a példány állapotának vizuális reprezentációjáért felel.

Ez a függvény a később bemutatandó, vizuális kellékek elérhetőségéért felelős osztály adattagjai közül választja ki megfelelőt.

### 3.4.2 A nézet csomag

Ez a csomag felelős mindenért, ami a képernyőn vizuálisan megjelenik, hiszen a fő feladata, hogy az imént bemutatott, modell csomagban helyet foglaló osztályokból példányosított objektumokat az állapotuk szerint megjelenítse. Itt van a pályakészítő, azaz MapBuilderGUI komponens, az offline és online játékért egyaránt szolgáló AdventureGUI, vagy a toplista és a labirintusok megtekinthetőségéért, adminisztrálhatóságáért felelős MapList osztály is. A csomag UML osztálydiagramja a következőképpen fest:

22. ábra a nézet csomag UML osztálydiagramja

### 3.4.2.1 a ResourceLoader osztály

Azért ezzel az osztállyal kezdem a csomag bemutatást, mert ennek az osztálynak az adattagjait később minden másik osztály használni fogja, és esszenciális hogy az adattagjai inicializálva és elérhetőek legyenek a többi osztály számára. Az alapvető ötlet az volt az osztály mögött, hogy a képek betöltése és memóriában tartása alkothatna egy külön statikus osztályt, amivel biztosítom azt, hogy csak egyszer lesznek a memóriába betöltve, de mégis mindig elérhető mindegyik másik osztály számára. Így az adattagjai mind kép formátumúak, és az initResources metódust a játék elején meghívva megbizonyosodhatunk arról, hogy ezek a fájlok végig a memóriában maradnak, és bármelyik használandó funkció esetén rövid idő alatt elérhetővé válik a játékszoftver számára.

### 3.4.2.2 a GUIWindow osztály

Ez az absztrakt osztály egy jó alapként szolgált arra, hogy a többi osztályban ennek a használatával a futásidejű polimorfizmus könnyen megvalósítható legyen, hiszen így elég csak ezt az osztályt használni az AdventureGUI és a MapBuilderGUI helyett, majd amikor ténylegesen valami osztályspecifikus adatot szeretnénk megjeleníteni, például majd az AdventureGUI esetében a játékost, vagy a sárkányt, akkor azt könnyebben meg lehet oldani. Az osztály konstruktora is használható arra, hogy az ebből származtatott osztályokban közös, alapértelmezett adatokat már előre be lehessen állítani, így aztán a későbbi osztályokban ez könnyen meghívható az ősosztály konstruktorának hívásával.

### 3.4.2.3 a pályakészítő, és a CellMouseAdapter osztálya

A pályakészítő, azaz a MapBuilderGUI osztálya a GUIWindow-hoz képest nagyon kevés extra információt tartalmaz, hiszen tulajdonképpen annyira van szükség, hogy megjelenítsük a cellákat, és különböző egérműveletekre változtassuk a kiválasztott cellák állapotát. Az alapvető ablak felépítése egy JFrame-ből, egy JPanel-ből és egy JMenu-ből áll, amit aztán később feltöltünk plusz adatokkal, például a segítség vagy a pálya elmentése funkciókat előidéző gombokkal. Az imént bemutatott labirintus adattagot így úgy paraméterezzük fel, hogy sem az Aldous-Broder, sem a pályabetöltős részt nem alkalmazza, tehát egy falakkal teli tömböt kapunk a végén. A labirintusra egy CellMouseAdapter figyelő lett beállítva, amely az egérmutató pixel koordinátái alapján kiszámolja, hogy melyik cellára kattintottunk, vagy kattintás közbeni húzás vagy angol nevén „drag” esemény esetén megadja azt, hogy melyik celláról milyen irányba húztuk el az egérkurzort. Így, ha rendelkezésre áll a jelenlegi cella, és az irány, hogy milyen irányba kell mozogni, akkor onnantól már meg tudja hívni a labirintus makeWayToAdjacentCell metódusát a kiszámolt iránnyal, és így meg is valósult a pályakészítő fő része.

### 3.4.2.4 A játékmenet osztálya, az AdventureGUI

Ez az egyetlen osztály a csomagon belül amely az összes modellben definiált és dokumentált osztályt példányosítja, és használja: A háttérben lévő labirintusra a játékost és a sárkányt is egyaránt felrajzolja, miközben a fut a sárkány algoritmusa, másodpercenként nagyjából hatvanszor újrarajzoljuk a panelt, számoljuk hogy mennyi ideje próbálunk az adott labirintusból kijutni,és gondoskodik a sárkány folyamatos gyorsításáról. Emellett fontos kiemelendő rész, hogy a két konstruktora van definiálva, ebből az egyik egy OracleSQLManager típusú paramétert vár, amivel ez a konstruktor lesz felelős az online játékért, míg a másik az offline módot aktiválja.

Az osztályban 3 időzített metódus, és egy plusz Thread fut:

* a timer nevezetű időzítő, amely egy egyszerű 1 másodperces, azaz 1000 milliszekundumos számláló, amibe belekerült az eltöltött idő számolása, és annak az ellenőrzése, hogy a játék véget ért-e.
  + A játékok aktuális eredményének ( tehát hogy nyertünk, esetleg vesztettünk már) eldöntésére az isLost és isWon metódusok szolgálnak: előbbi abban az esetben igaz, ha a sárkány metódusa végigfutott, és ezáltal a didDrakeFindPath változó igaz értéket kapott, utóbbi pedig akkor lesz igaz, hogyha a játékos jelenlegi cellája, azaz pozíciója megegyezik az éppen generált labirintus kiútjával. Ezeket a kondíciókat a játék másodpercenként nézi.
* a refresher időzítő, amely egy 15 milliszekundumos késleltetéssel rendelkezik, ezzel elérve a nagyjából 60-szoros frissítést, amely egy egészséges terhelés melletti optimális, kellően reszponzív és folyamatos képet biztosít. A következő metódusokat hívja:
  + A labirintus panelnak a teljes újrarajzolását, amelyben implementálva van a játékos és a sárkány kirajzolása is
  + A látható cellák ellenőrzését és frissítését, ezzel elősegítve, hogy tényleg akkor, abban a pillanatban lesznek láthatóak az újonnan felfedezett cellák, amikor a játékos odaér.
* a spriteUpdater időzítő, 100 milliszekundumonként kerül meghívásra az egyetlen hozzárendelt függvény:
  + A Player osztály updateLook metódusát hívja, ami aztán beállítja magának az iránynak megfelelő, illetve a lépés előidézése érdekében folyamatosan váltakozó vizualizációt. Ez azért pont ide került, mert a többi időzítőben a két metódushívás között eltelő idő vagy túl sok, vagy éppen túl kevés ahhoz hogy a lépegető, reszponzívan irányt váltó játékos látványát előteremtsük.
* a drakeThread szál, mely folyamatosan fut, de a sárkány paraméterében megadjuk a waitTimeBetweenAIIterations paramétert, amelyet a sárkány később a szál altatására fog használni, így olyan tempót adva az algoritmus egyes lépéseinek, amellyel az ember is tud versenyezni. Ez a paraméter aztán természetesen a további pályák elérésével folyamatosan csökken, egészen pontosan alkalmanként az előző érték 0.95-szörösére.

A játékok újraindítására a restartGame metódus szolgál, amely megállítja az imént felsorolt időzített funkciókat, megszakítja a drakeThread szálat, új labirintust tölt vagy generál a konfigurációtól függően, ezeket újra elhelyezi az ablakon belül, majd ha elkészítette az eredeti ablakot ismét, akkor újra elindítja az időzített metódusokat és a sárkány szálát.

Arra az esetre, ha a játék a játékos kérésére, vagy egyéb okok miatt véget ér, a stopGame metódus a legalkalmasabb. Ez csak leállít minden párhuzamosított folyamatot, nullázza az időt és a pontszámot, majd bezárja az ablakot. Természetesen, ha az elveszített játék végett kerül meghívásra, előtte megtörténik a pontszám szinkronizálása, ha az adatbázis kapcsolat (még mindig) elérhető.

### 3.4.2.5. DistanceManager, távolság számoló osztály

Egy egyszerű statikus metódus, amely a játékos mozgása közben segít távolságot számolni a kirajzolt játékos képe és a cellák szélei között. Ha a cella adott oldalán fal található, akkor ez az osztály, de leginkább egyetlen metódusa, az isClosetoEdges lesz felelős azért, hogy eldöntse, mehet-e még az adott irányba a játékos vagy sem. A játékos képének négy sarkának koordinátáit figyeli, és számolja ki a cella falai közötti távolságot. Egy felbontás alapján skálázódó, sizeMultiplier-re keresztelt segédváltozó dönti el azt, hogy hány pixel lesz az a távolság, aminél közelebb már nem lehet az adott cella adott széléhez a játékos.

### 3.4.2.6. KeyHandler, a játékos mozgásáért felelős osztály

Ebben az osztályban van implementálva az, hogy éppen melyik billentyű van lenyomva, és irányonként egy segédváltozóval el van intézve az, hogy a lenyomott billentyűk folyamatos mozgást idézzenek elő. Tehát ha például folyamatosan fölfele szeretnénk menni, akkor az upPressed nevű segédváltozó igaz értéket fog kapni, és mindaddig tartja ezt az értéket, amíg nem engedjük fel a felfele nyilat, vagy a W billentyűt. Amint megtörténik a felengedés, újra hamis értéket fog kapni, és így az AdventureGUI-ban a mozgató metódus mindig egy éppen aktuális állapotot fog kapni, és ezáltal folyamatos lesz a játékos mozgása.

### 3.4.2.6. A labirintust vizualizáló metódus, azaz a LabyrinthPanel

Ez a panel rajzolja ki a legtöbb elemét a játéknak, például a játékost, a sárkányt vagy a cellákat. Az AdventureGUI csak tartalmazza ezt az ablakot, ehhez még hozzákerül a menü, ami már annak az osztálynak a felelőssége. Itt a két legfontosabb metódus az a konstruktor, ahol az AdventureGUI ősosztálya van átadva, hogy aztán ez az osztály alkalmas legyen a MapBuilderGUI és az AdventureGUI kezelésére is.

a paintComponent metódus az ősosztályának egy felülbírált függvénye, amelyben először meghatározásra kerül az, hogy pontosan mekkora pixelméretet kell cellánként alkalmazni ahhoz, hogy minden kép pont kiférjen az ablakra. Ezután a cellákat lentről fölfelé, balról jobbra egyenként kirajzoljuk, úgy, hogy egymással ne érintkezzenek. Eztán futásidejű polimorfizmus használatával megnézzük, hogy a board adattag milyen típusú (vagy AdventureGUI típusú, vagy nem), és ha AdventureGUI, akkor ezáltal megbizonyosodhatunk arról, hogy a játékos és sárkány is elérhető, így azok pozícióit is lekéri és kirajzolja.

Valamint mivel itt derül ki, hogy pontosan melyik pixelre kell a játékost kirajzolni a játék elején, hiszen ez minden esetben függ a képernyő aktuális felbontásától, ezért egy startOfGame segédváltozóval egy egyszeri pixelbeállítást végzünk, hogy aztán ahhoz a pixelhez képest tudjon relatív mozogni a játékos. A sárkány kirajzolása a cellánkénti kirajzolásnál egy elágazás formájában lett implementálva. Így végül ez a metódus van hatvanszor meghívva másodpercenként.

### 3.4.2.7 a saját pályák listája, a MapList osztály

Ez az osztály a JTable Swing komponensből van származtatva, és tulajdonképpen egyedül a konstruktorban össze is áll a komponens, feltöltve adatokkal. A konstruktornak át kell adni az adatbázis kapcsolatot jelképező OracleSqlManager példányt, és abból egy lekérdezés segítségével egy String mátrixot kapunk vissza, benne a szükséges adattal. Ezt aztán a

# 4.Program készítése

A program elkészítése tűnik a legkönnyebb feladatnak, de nem az. Ha egy jó dolgozatot szeretnél, akkor igényesen, jól megtervezett, átgondolt alkalmazást kell készítened. A készítésnél sokan nem mernek olyan nyelvekhez, eszközökhöz nyúlni, amit nem ismernek. Mai világban inkább vannak mobil, vagy webes alkalmazások, mint asztali alkalmazás. Nem kell megijedni ezektől. Programozó vagy. Egy programozó bármilyen nyelven tud programozni. Csak rá kell szánni az időt, hogy megismerd a nyelvet. A programodban legyen help, ami segíti a felhasználót az alkalmazásod használatában.

# Szakdolgozat felépítése

A szokásos kérdés hogy épüljön fel a szakdolgozat. Most leírom, hogy mi az, ami célszerű egy szakdolgozatban.

Sablon

Kezdjük azzal, hogy milyen sablont használj a dolgozat elkészítéséhez. Ha Word-ben írod a dolgozatodat, akkor használhatod ezt a fájlt [2], amit most olvasol. Persze olvasd el a formai követelményeket, hogy minden teljesüljön (1,5-es sorköz), illetve a megfelelő részeket a borítón írd át. A Latex-ben írnád dolgozatodat, akkor használhatod Cserép Máté sablonját [3].

Fejezetek

A szakdolgozat alap fejezetei: a bevezetés, felhasználói dokumentáció és a fejlesztői dokumentáció. Ezen kívül célszerű még írni, egy összefoglalást és egy további tervek részt.

Témabejelentő: A témabejelentő kérvényt csatolni kell a dolgozat mellé. Ezt a témabejelentőt a neptunból kell letölteni és a feltöltendő „.zip” csomagba betenni!

Tartalomjegyzék: A dolgozat elején legyen egy tartalomjegyzék, ahol a főbb fejezetek látszódnak. Célszerű kerülni a túl „mély” fejezet szinteket, pl: 1.2.4.2.5.3-as fejezet. Ehhez persze mindenképp szükséges, hogy oldalszámozva legyen a dolgozat.

Köszönetnyilvánítás: Ha esetleg pályázati támogatást kaptál a dolgozatodhoz, akkor azt se felejtsd el megköszönni.

Bevezetés: Itt le kell írni, miről is fog szólni a szakdolgozatot, leírod mi a motiváció. Itt kell felkelteni az olvasó érdeklődését, hogy el akarja olvasni a dolgozatodat. Itt nem kell belemenni a részletekbe, itt lehet felületesen fogalmazni, hogy egy egyszerű halandó is megértse, miről szól a dolgozat. Itt veted fel a problémát, amivel elkezdtél foglalkozni. Bemutatod, hogyan akarod ezt megoldani. Lényegében egy előzetes áttekintést adsz a dolgozatról.

Felhasználói dokumentáció: Itt be kell mutatnod az alkalmazásod, olyan szinten, hogy a felhasználó használni tudja. Itt lehet berakni képernyőképeket, amelyek segítik a program használatát. Itt le kell írnod milyen gépigény szükséges, hogyan kell telepíteni / elindítani. Itt gondolkodj úgy, hogy ha te lennél a felhasználó, akkor tudnád-e használni ezek alapján a programodat. Itt lehet use case-eket használni, hogy bemutasd az egyes funkciókat.

Fejlesztői dokumentáció: Itt egy másik szemszögből kell bemutatnod a programodat. Itt egy olyan fejlesztő vagy egy cégnél, akinek azt mondják, hogy fejlesszen hozzá ehhez a programhoz újabb funkciókat. Ekkor csak a fejlesztői dokumentáció áll rendelkezésedre, hogy megértsd, melyik modul mit csinál, és hogy hova kell nyúlnod, hogy meg tudd valósítani, amit kértek. Itt mutasd be a programod rétegeit, adatbázist, táblákat, osztályokat, modulokat, fontosabb függvényeket, algoritmusokat, felhasználói eseteket, hálózati kommunikációt, fejlesztői környezetet. Itt szerepeljenek osztálydiagramok, UML ábrák. Itt lehet beszélni az üzemeltetésről is. Milyen időzített folyamatok vannak, hol van a logolás, használ-e felhő rendszereket, stb. Ebbe a fejezetbe tartozik a tesztelés is. Minden osztályodra legyen tesztelés is. Egység tesztek, felületi tesztek, integrációs tesztek stb. Ha nem tudsz automatikus tesztet írni a programodra, akkor csinálj teszt jegyzőkönyvet. Ebben leírod mi a funkció, amit tesztelsz és annak mi az elvárt kimenete, de nagyon sok tesztelő eszköz elérhető manapság. Olyanok is vannak, amelyek felületet képesek tesztelni.

Összefoglalás: Nagyon hasznos tud lenni, ha egy oldalban össze tudod foglalni, hogy miről is szól a dolgozatod. Ez annyiban másabb, mint a bevezetés, hogy itt feltételezheted, hogy már elolvasták a dolgozatodat. Itt újra mutasd be a problémát, amivel foglalkoztál, mutasd be mit és hogyan használtál és végül mutasd be, hogy mire jutottál.

További fejlesztési lehetőségek: Egy program sose készül el, csak jön a határidő. Ebbe a fejezetbe írd le, milyen további funkciókat tudsz elképzelni az alkalmazásodhoz. Mit lehetne még hozzáfejleszteni, mi az, amit esetleg máshogy is meg lehetett volna valósítani.

Formai segítségek

A hivatalos formai követelményeket ellenőrizd le a kari honlapon [4], de a fontosabbakat összegyűjtöttem neked. Az elte hivatalos címerét pedig innen tudod letölteni [5] .

Követelmények:

A szakdolgozatot kemény kötésben kell leadni, 1 példányban.

Fedőlapjának színe fekete, aranyszínű feliratokkal.

Lap: A4-es méret, színe fehér

Betűméret: 12 pont

Sorok: sorkizárt igazítás, 1,5-es sortávolság

Margó:

belső: 3,5 cm

külső: 2,5 cm

alsó: 2,5 cm

felső: 2,5 cm

Oldalszám: az oldalszámozást a tartalomjegyzéktől kezdve az irodalomjegyzékkel bezárólag folyamatosan kell végezni. Jelölése arab számokkal történik (a tartalomjegyzék oldalainak az oldalszámát nem szokás feltüntetni).

A dolgozat fő fejezetei (1. szintű címsorok) új oldalon kezdődjenek.

Képek: Ha képet raksz a dolgozatodba, akkor annak mindig legyen képaláírása és sorszáma. Ha raksz be képet, akkor a szövegben hivatkozz is rá. Az 1. ábrán láthatjuk például a Galaxis útikalauz stopposoknak című könyv bortóját. A könyvből elhíresült mondatot („Ne ess pánikba!”) Elon Musk is felhasználta, amikor egy Teslát lőtt ki az űrbe. Ezt láthatjuk a 2. ábrán. Másik oldalról a képaláírás olyan legyen, hogy meg tudjuk állapítani, mi van a képen és ne kelljen a szövegben keresgélni.

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/hu/thumb/3/32/Galaxis_%C3%BAtikalauz_gy%C5%B1jtem%C3%A9nyes.jpg/200px-Galaxis_%C3%BAtikalauz_gy%C5%B1jtem%C3%A9nyes.jpg  23. ábra Galaxis útikalauz stopposoknak könyv[[1]](#footnote-1) | https://upload.wikimedia.org/wikipedia/commons/1/17/Elon_Musk%27s_Tesla_Roadster_%2840110298232%29.jpg  24. ábra Elon Musk Teslája az űrben[[2]](#footnote-2) |
|  |  |

Hivatkozások: Ha felhasználsz bármilyen forrást akkor azt mindig hivatkoznod kell. Ez fontos, hogy semmilyen plágium ne merüljön fel. Erről egy papírt is ki kell töltened, hogy büntetőjogi felelősséged van ezzel kapcsolatban. Célszerű a szövegben is berakni egy hivatkozást, hogy kapcsolható legyen forrásjegyzékhez. Az előbbi képeket például a wikipédián találtam [6].

DVD melléklet

A DVD mellékletre mindent tegyél rá, hisz úgy is elfér. Legyen rajta a forráskód, legyen rajta futtatható formában az alkalmazásod, ha lehetséges. Legyen rajta a dolgozat .pdf formátumban, ill. a leadáshoz szükséges papírok is rákerülhetnek. Ha használtál külső csomagokat, adatokat, akkor azokat is tedd rá.

Leadás

Az ELTE IK-n megszünt a papír alapú szakdolgozat leadás! A szakdolgozatodat a neptunba kell feltölteni ”.zip” formátumba.

A becsomagolt állományban a következőknek kell hogy szerepeljenek [7]:

Témabejelentő, amit a neptunból töltesz le

Szakdolgozatod .pdf formátumban

A programod forráskódja

EDIT Hallgatói dolgozat űrlap

EDIT - Elhelyezési megállapodás - Hallgatói dolgozat

Szakdolgozat/Diplomamunka leadási és eredetiség nyilatkozat

EDIT - Hallgatói dolgozat titkosítása (ha szükséges)

Védés

Az elkészült dolgozatodat meg is kell védened. A védésre 10-15 perced van. Ennyi idő alatt kell bemutatnod az alkalmazásod működését. Szakdolgozatnál nem prezentációt várunk el, hanem működő, jól kinéző, használható alkalmazást. Ha van az alkalmazásodban olyan rész, amelyet nem tudsz megmutatni működés közben, vagy épp a nagy eredmény a háttérben futó motor működése, akkor készíts róla prezentációt. Ekkor is célszerű minimális mennyiséget csinálni, max. 3-4 dia. A védésre célszerű egy jól végiggondolt, jól felépített és begyakorolt bemutatással jönni. Ha jó a védésed, az akár a dolgozatod jegyén is tud javítani. A védésre mindig legyen egy ’B’ terv. Az ördög nem alszik, és ami el tud romlani, az el is fog [8]. Éppen nem lesz internet a védés alatt, nem elérhető a külső forrás, amit használsz, stb. Csinálj egy videót, ahol működik a rendszer, vagy készíts diasort a funkciókról. Ha az alkalmazás védés során nem működik, vagy hibát dob, akkor sajnos nem fogják elfogadni a dolgozatot. A védésen a forráskódot is megnézzük, így az légyen megfelelően kommentezve. Védés előtt próbáld ki, hogy működik-e az alkalmazásod az egyetemi gépeken, mert ha nem akkor hozz laptopot és mutasd be azon. Ha laptopon szeretnéd bemutatni, akkor mindenképp jelezd a bizottság elnökének emailben.

Értékelés

A szakdolgozatra 3 jegyet fogsz kapni. Egyet a dolgozatra, egyet a védésedre, egyet pedig a húzott tételre. A három átlaga lesz a végső jegyed, emiatt is fontos, hogy a dolgozatod rendben legyen, és jól meg tudd védeni. A dolgozatodat a témavezetőd is értékelni fogja. Ennek az értékelésnek a pontrendszere elérhető a kari honlapon [9].

Tipikus hibák

Rossz a dolgozat felépítése. Ha több részből tevődik össze a dolgozatod, akkor előbb kezd kisebb elemek bemutatásával, majd az abból épülő nagyobbakkal, és így tovább. Ha lapozgatni kell a dolgozatban, mert olyat említesz, ami később lesz bevezetve, akkor az nem jó.

Sok kép, kevés szöveg, vagy fordítva. A dolgozatba lehet rakni képet, de minden képhez tartozzon szöveg is. Ha csak bemásolod a képernyőmentéseket és nem írsz róluk, akkor az olvasó nem biztos, hogy tudja értelmezni azokat. Ha viszont csak szöveget írsz és nem ábrázolod képekkel, akkor az megint nem segíti az olvasót.

A dolgozat nem éri el a 40 oldalt. Ha kevesebb oldalad van, akkor valami még kimaradt, vagy nem elég részletesen írtad le.

A témavezető nem látja a végső verziót. Mindig küld el neki mielőtt bekötteted a dolgozatot, hogy hátha észrevesz még hibát benne.

Védés közben derül ki valami hiba a programban. Ez a legrosszabb forgatókönyv. A védésre legyen egy begyakorolt bemutatód, hogy mikor mit fogsz bemutatni. Ezt gyakorold be nyugodtan otthon, nézd meg hogy beleférsz-e az időbe, és hogy minden működik. Olyat ne mutass, amiben nem vagy 100%-ig biztos!

Frissítve: 2022-04-08

Forrásjegyzet  
<https://en.wikipedia.org/wiki/Use_case_diagram>  
<https://hu.wikipedia.org/wiki/Unified_Modeling_Language>  
<https://en.wikipedia.org/wiki/Maze-solving_algorithm>  
<https://www.youtube.com/watch?v=gVSEJdSQZVQ>  
<https://en.wikipedia.org/wiki/Maze_generation_algorithm>

[1] Modell-nézet-vezérlő, Wikipédia   
https://hu.wikipedia.org/wiki/Modell-n%C3%A9zet-vez%C3%A9rl%C5%91

|  |  |
| --- | --- |
| [1] | „ELTE EDIT - IK Szakdolgozatok,” [Online]. Available: http://edit.elte.hu/xmlui/handle/10831/27. [Hozzáférés dátuma: 09 07 2019]. |
| [2] | G. Gergő, „Szakdolgozat Word sablon,” 2019. [Online]. Available: http://ggombos.web.elte.hu/szakdoli/szakdoli\_how\_to.docx. [Hozzáférés dátuma: 19 11 2019]. |
| [3] | C. Máté, „ELTE szakdolgozat template (Github),” [Online]. Available: https://github.com/mcserep/elteikthesis. [Hozzáférés dátuma: 10 07 2019]. |
| [4] | „BSc szakdolgozat követelmények,” [Online]. Available: https://www.inf.elte.hu/dstore/document/257/PTI\_BSc\_szakdoli\_uj.pdf. [Hozzáférés dátuma: 19 11 2019]. |
| [5] | „ELTE címer,” [Online]. Available: https://www.inf.elte.hu/dstore/document/203/cimerek.doc. [Hozzáférés dátuma: 19 11 2019]. |
| [6] | „Galaxis útikalauz stopposoknak - wikipédia,” [Online]. Available: https://hu.wikipedia.org/wiki/Galaxis\_%C3%BAtikalauz\_stopposoknak. [Hozzáférés dátuma: 09 07 2019]. |
| [7] | „Szakdolgozat leadáshoz szükséges dokumentumok,” [Online]. Available: https://www.inf.elte.hu/content/adatlapok-formanyomtatvanyok.t.1052?m=129. [Hozzáférés dátuma: 19 11 2019]. |
| [8] | „Murphy törvénye - wikipédia,” [Online]. Available: https://unciklopedia.org/wiki/Murphy\_t%C3%B6rv%C3%A9nyei. [Hozzáférés dátuma: 09 07 2019]. |
| [9] | „ BSc pontrendszer,” [Online]. Available: https://www.inf.elte.hu/dstore/document/1667/Bsc\_b%C3%ADr%C3%A1lat.docx. [Hozzáférés dátuma: 19 11 2019]. |

1. [https://hu.wikipedia.org/wiki/Galaxis\_%C3%BAtikalauz\_stopposoknak#/media/F%C3%A1jl:Galaxis\_%C3%BAtikalauz\_gy%C5%B1jtem%C3%A9nyes.jpg](https://hu.wikipedia.org/wiki/Galaxis_%C3%BAtikalauz_stopposoknak%23/media/F%C3%A1jl:Galaxis_%C3%BAtikalauz_gy%C5%B1jtem%C3%A9nyes.jpg) [↑](#footnote-ref-1)
2. <https://hu.wikipedia.org/wiki/Galaxis_%C3%BAtikalauz_stopposoknak#/media/F%C3%A1jl:Elon_Musk's_Tesla_Roadster_(40110298232).jpg> [↑](#footnote-ref-2)