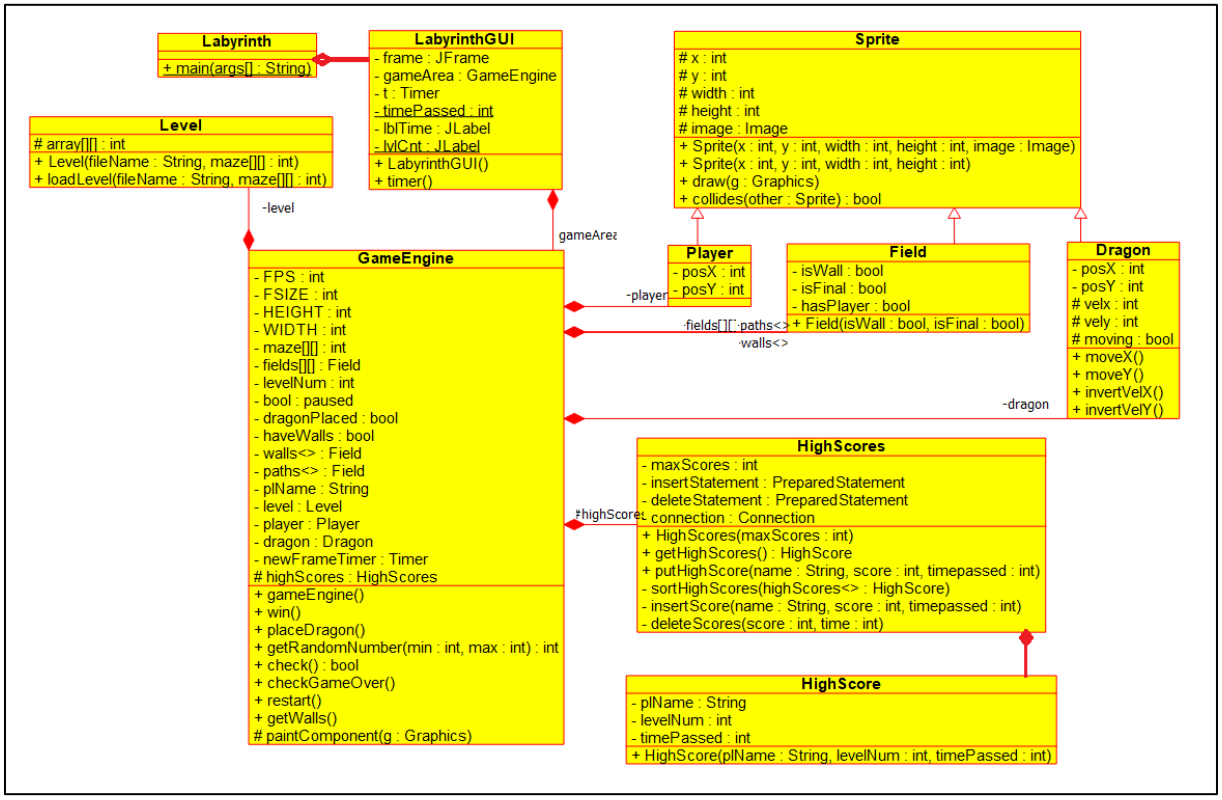
Programozási technológia – 3.beadandó

Schmidt Bálint Márk – FRJR89

Feladat:

3. Labirintus (Labyrinth) 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.



Programozási technológia – 3.beadandó

A program osztályai:

- Sprite – A Sprite ősosztály azon elemeket valósítja meg, melyeket meg kell rajzolni a programban. Adattagjai határozzák meg a pályán való elhelyezkedését és kinézetét.
    - Field – A Field osztály a labirintus mezőit valósítja meg. Három adattagja van, melyek logikai változók, azt mutatják meg, hogy a mező a cél-e, fal-e és hogy rajta van-e a játékos.
    - Player – A Player osztály a játékost valósítja meg.
    - Dragon – A Dragon osztály a labirintus sárkányát valósítja meg. A velx, vely adattagok a sárkány mozgásához szükségesek, a metódusokkal együtt.
- HighScore – A HighScore osztály egy eredményt valósít meg. A játékos neve, végigjátszott pályáinak száma és ideje tartozik az adattagjaihoz.
- HighScores – A HighScores osztály valósítja meg a program adatbázis kezelését. A maxScores adattag adja meg, hogy hány eredmény fér a toplistára.
- Level – A Level osztály felelős az adott pálya betöltéséért, az array adattagban pedig a falak kerülnek kiválasztásra.
- GameEngine – A GameEngine, ahogy az nevéből is adódik, a játék működését valósítja meg. Többek között itt történik a játékos mozgásának, sárkány mozgásának működtetése, a pálya sötétítése és a pályák, karakterek betöltése is.
- LabyrinthGUI – A LabyrinthGUI osztály felelős az ablak megjelenítésért, azon belül az idő, pályaszám, toplista és a menüpontok működéséért.

Esemény – Eseménykezelés:

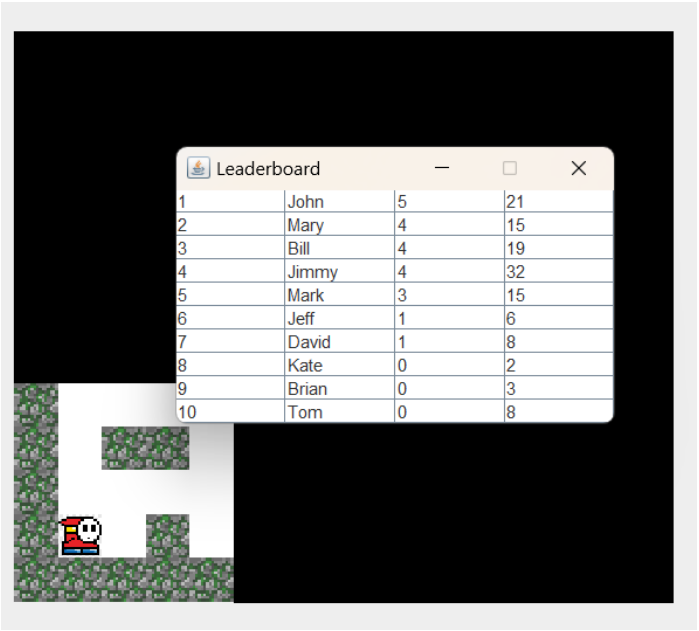
- Billentyűzet (Fel, Le, Balra, Jobbra nyilak, ESC gomb) – A GameEngine osztály konstruktorában az InputMap, ActionMap put(…) metódusaival vannak beállítva. A nyilak lenyomásakor a játékos az adott irányba mozdul, akkor, ha a mező nem fal. Ha a célra lép a játékos akkor az adott szintet teljesítette. Az ESC gomb lenyomása esetén a paused adattag igaz lesz, ekkor leáll az idő, a sárkány és a játékos se tud mozogni az ismételt lenyomásig.
- NewFrameListener – A sárkány mozgásának megvalósítója. Az adott FPS szerint 1000/FPS ms-ként hívódik meg, az algoritmus a dokumentáció későbbi részében olvasható.
- restartMenuItem listener – A játék ablak menüjében található Restart gomb eseménykezelője, amely újraindítja a játékot az első szinttől kezdve. Az idő is nullázódik.
- exitMenuItem listener – A játék ablak menüjében található Exit gomb eseménykezelője, amely megnyomása után a játék ablaka bezárul és leáll a program.
- lBoardItem listener – A játék ablak menüjében található Leaderboard gomb eseménykezelője, mely során megnyílik még egy ablak, mely az adatbázis alapján, megadja a játék toplistáját.

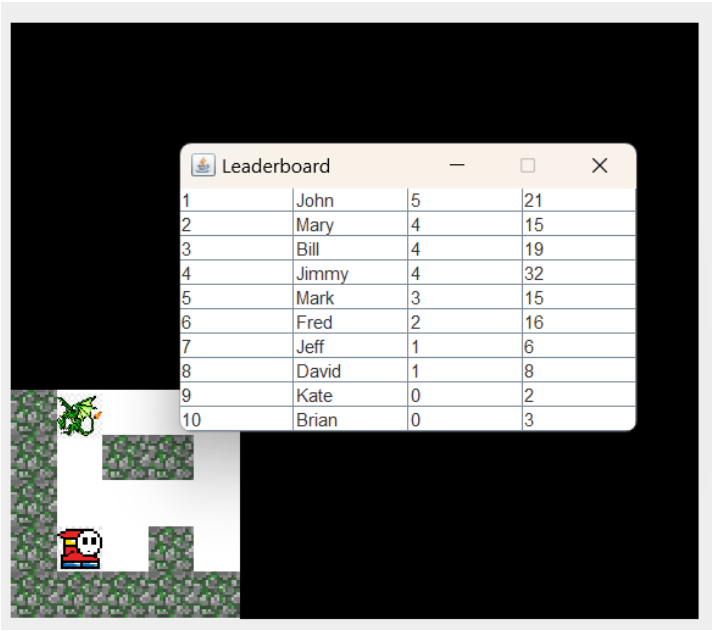
Mindegyik eseménykezelő az ActionListener interface megvalósítása.

Programozási technológia – 3.beadandó

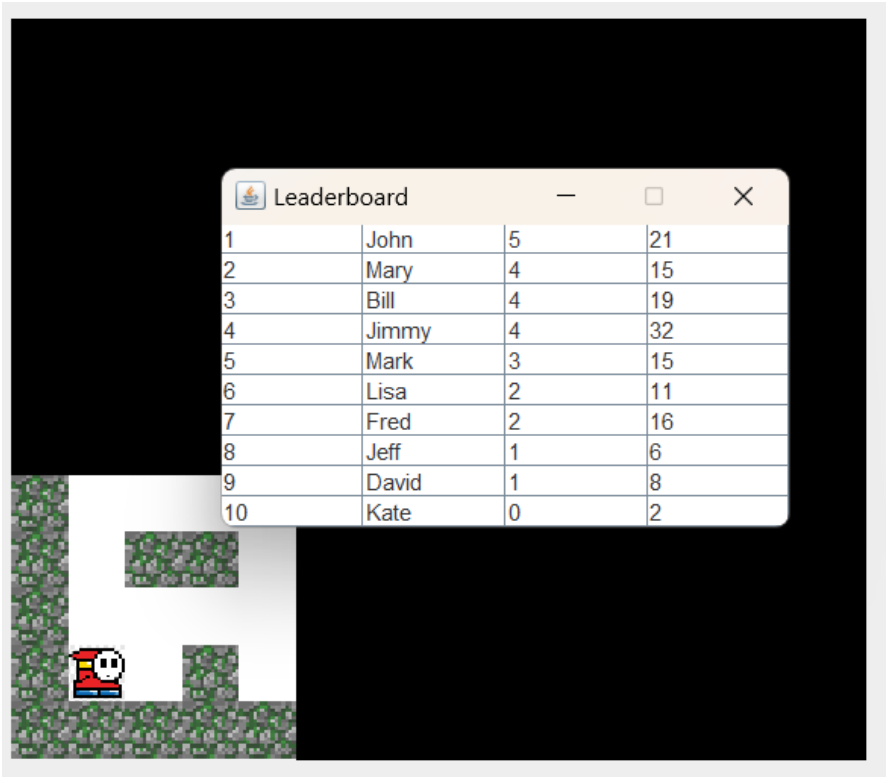
Tesztesetek:

A 10. helyezett kiesik a toplistáról miután egy másik játékos egy jobb eredményt ér el.



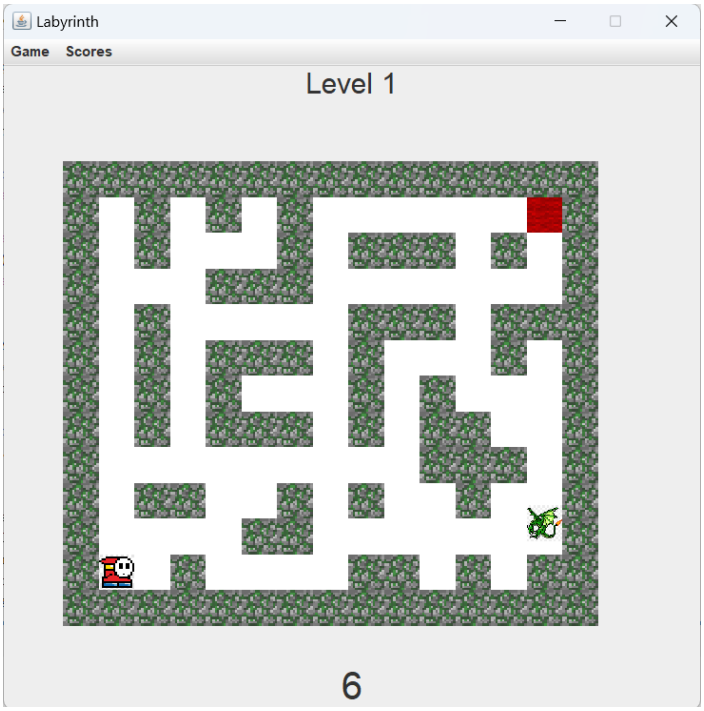


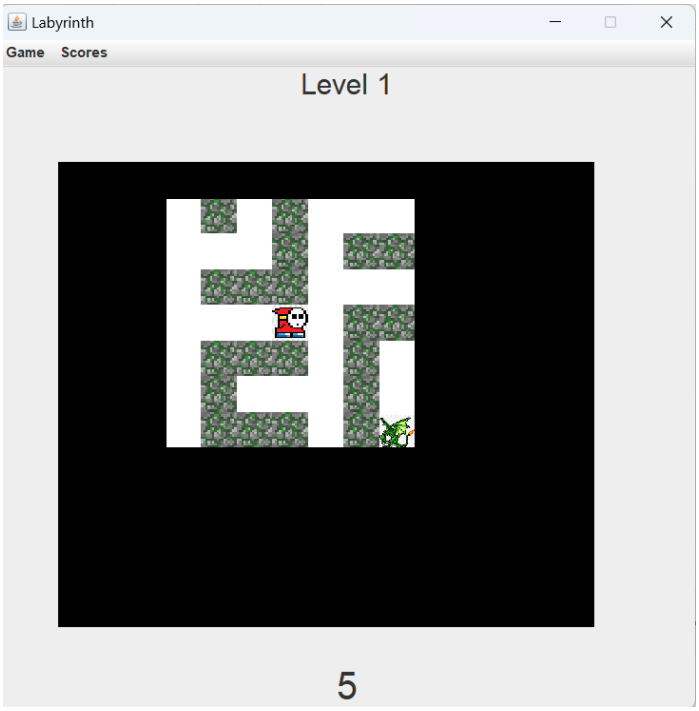
A listán előrébb helyezkedik az, aki ugyanazt a teljesítményt kevesebb idő alatt érte el.



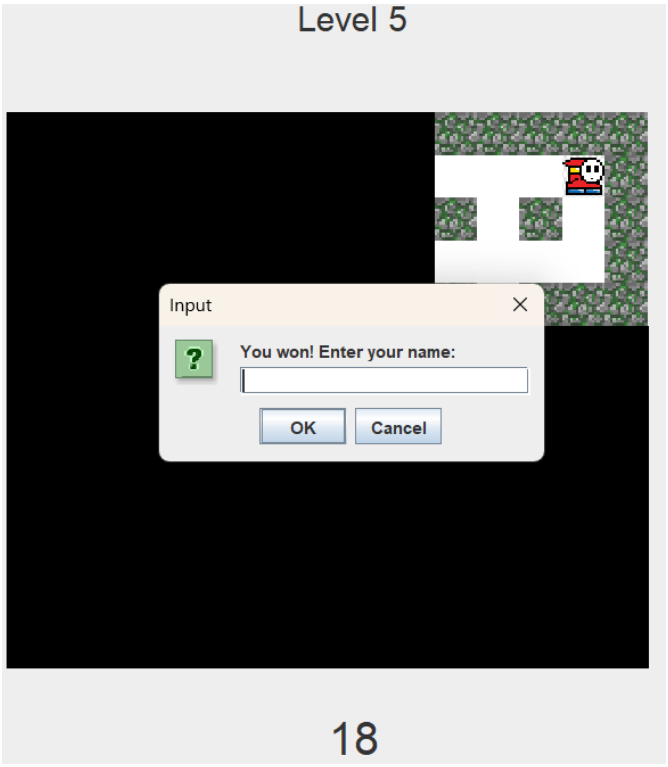
Programozási technológia – 3.beadandó

A játéktér sötétítés nélkül és sötétítéssel.





A játékos, ha az összes pályán végigért, akkor megadhatja a nevét és rákerül a toplistára. Ha a sárkány megöli, akkor is a neve megadása után kerülhet rá a toplistára, amennyiben jó eredményt ért el. Ezután újrakezdődik a játék.



Programozási technológia – 3.beadandó

Érdekesebb algoritmusok:

A labirintus és a játék megrajzolása és sötétítés:

A labirintus előre elkészített txt fájlokban van felépítve. 0,1,2 és 9 mezőkből áll (0 – út, 1 – fal, 2 – start, 9 – cél). A Level osztály loadLevel(..) algoritmusa a mezőket a maze tömbbe teszi. A paintComponent(..) metódus két ciklussal végig megy ezen tömbön és a benne lévő számok szerint generál Field példányokat. Ezek draw() metódusával meg is rajzolja őket, valamint a fields tömbbe beteszi ezeket az objektumokat. Ezután a játékos és a sárkány kerül megrajzolásra. Az első alkalommal a sárkány is elhelyezésre kerül és a falak is bekerülnek a walls ArrayListbe.

A sötétítés során egy egymásba ágyazott ciklus nézi a játékos pozícióját és ahhoz képest sötétíti a pályát, hogy a maze tömbben az indexek milyen távol vannak a játékos pozíciójától.

A sárkány elhelyezése:

A placeDragon() metódusban két ciklus végignézi a fields tömböt, hogy mely mezőkre teljesül, hogy nem fal és a játékos nem halna meg azonnal a sárkány elhelyezésekor. Ezeket a mezőket a paths Listhez adja hozzá. A paths Listet a Collections.shuffle(..) metódus megkeveri, majd a sárkány pozícióját a pályán a paths első elemére állítja az algoritmus. Ezután még generál egy random számot, amely 0 vagy 1. Egy switch vizsgálja ezt és a számtól függően indul majd el a sárkány egy irányba.

A sárkány mozgása:

A sárkány mozgásának kezdése az eseménykezelés résznél olvasható. Elhelyezés után elindul egy irányba. Ha nincs a játék megállítva (ESC gomb), akkor folyamatosan mozog ugyanabba az irányba. Ha falba ütközik (check() függvény ellenőrzi), akkor a mozgás irányától függően az eljárás visszateszi az előző olyan helyre a sárkányt, ahol még nincs fal, majd 0 és 3 között generálódik egy random szám. Egy switch vizsgálja ezt és a számtól függően a 4 irányból elindítja egybe a sárkányt. Mindezt addig, amíg ütközik a falba. Amerre először nem ütközik, abba az irányba indul el. Utána újrarajzolódik a pálya.

Sárkány és játékos ütközik-e?

Ezt a checkGameOver() függvény vizsgálja a sárkány minden egyes mozgása után, tehát a játék közben folyton. A függvény a játékos és a sárkány pozíciója alapján vizsgálja, hogy a játékos szomszédos mezőn áll-e vagy ha mozgásban van épp, akkor a Sprite ősosztály collides() metódusa is nézi az ütközést. Ha igaz, tehát a játékos meghal, akkor leáll a játék és a játékosnak a felugró ablakba kell a nevét megadnia. Ha eredménye kiemelkedő, akkor bekerül a top10-be. Ezután a játék elölről kezdődik.