HÁZI FELADAT

Programozás alapjai 3.

Akari-Light Up

Réti Ádám

AO7JX4

2022. november 29.

1. Specifikáció

Szabályok

A szabályok egyszerűek. A Light Up játékot egy téglalap/négyzetes alakú rácson játsszák. A rács fekete és fehér cellákat is tartalmaz. A cél az, hogy az izzókat úgy helyezzük el a rácson (minél kevesebbet), hogy minden fehér négyzetet megvilágítson (sárgává változtasson). Egy cellát egy izzó megvilágít, ha ugyanabban a sorban vagy oszlopban vannak, és ha nincs közte fekete cella. Ezenkívül egyetlen izzó sem világíthat meg egy másik izzót (azaz két izzó nem lehet azonos oszlopban/sorban). Néhány fekete cellában számok vannak. A fekete cellában lévő szám azt jelzi, hogy hány izzónak kell cellával szomszédosnak lennie. Ezek a számok 0, 1..4-ig mehetnek hiszen egy cellának csak 4 szomszédos cellája lehet (átlóban nem szomszédosak).

Terv

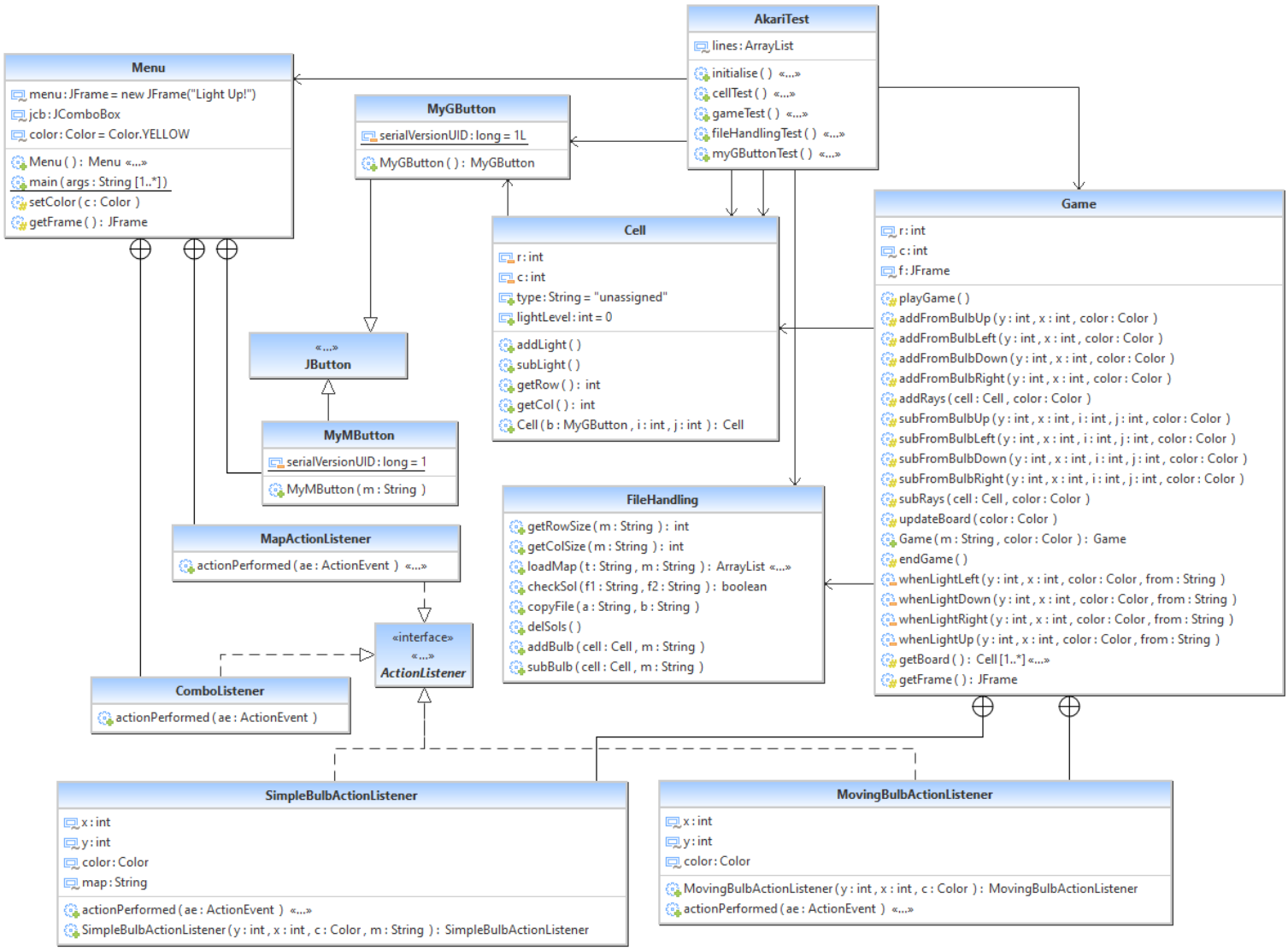
A program elindításakor a felhasználó egy menüben találja majd magát, ahol a pályakiválasztásért felelős gombokat fog látni (pl. Map1, Map2…). Értelemszerűen ezek gombok egyikére kattintva a felhasználó az egyik (én általam előre elkészített) pályát betöltheti, és megkezdheti a játékot. A pálya betöltése után, a felhasználó magát a négyzetrácsos pályát, egy JTextField-et, Back, Reset és egy Done! gombot fog látni. Izzók elhelyezésére az egyik fehér négyzetrácsra bal egérgombbal kattintva van lehetőség. Miután a felhasználó végzett a pálya megoldásával, az előbb említett Done! gomb megnyomásával ellenőrizheti, hogy helyesen oldotta-e meg a választott pályát, amelyre a JTextField kimenete fog választ adni (Congratulation! vagy Incorrect solution!). A Reset gombbal, pedig törölheti az eredeti megoldását és újrakezdheti a pálya megoldását (törli az elhelyezett izzókat). A Back gombbal, a felhasználó bármikor visszaléphet a pályaválasztás menüjébe (ezzel elveszik a választott pálya jelenlegi állása).

Implementáció

A terv alapján leírtakat a Java Swing, és annak beépített eszközeivel fogom megvalósítani. A pályakezelést, fájlból beolvasással fogom megvalósítani (ezek valószínűleg kézzel írt map.txt fájlok lesznek, amelyekben általam kigondolt számsorozatokkal fogom eltárolni az egyes pályákat, pl.: 5155 egy olyan sort hozna létre a négyzetrácson, amelyben az 5-ös fehér cellákat az 1-es pedig egy olyan fekete cellát jelöl, amiben egy 1-es van). Magát a négyzetrácsot, és a benne lévő cellákat pedig egy kétdimenziós tárolóban fogom tárolni (Valószínűleg 2D ArrayList).

***Megjegyzés**: Az eredeti játékot kiegészítettem egy prizmával és egy forgatható lámpával. Részletesebb leírást lásd később.*

2. Osztálydiagram



Az itt látható UML diagram az Eclipse Marketen belül található UML Lab Modelling IDE 1.30.0-s verziójával készült. A nested classok jelölésére az elkészült diagramot Photoshopban megszerkesztettem.

3. Megvalósitás

Összesen 11 osztályt használtam fel, melyek közül az egyik (AkariTest) a tesztelésért felel, és ezek közül 5 osztály pedig beépített osztályként (nested class) van megvalósítva (ezek nagyrésze ActionListener implementációk).

3.1. Főbb Osztályok bemutatása

3.1.1. Game osztály

Ez a játék menetéért felelős osztály

Attribútumok:

- **int r – Játék sorainak száma**
- **int c – Játék oszlopainak száma**
- **JFrame f – Játék megjelenítésért felelős keret**

Konstruktor:

public Game (String m, Color color) throws Exception

Inicializálja a FileHandling file-t. Ez alapján betölti a Menu általl átküldött pálya számát, majd annak sor/oszlophosszával feltölti a r és c értékeit. Létrehozza a JFrame f- hez illeszkedő további komponenseket. A pályát egy r*c-s GridLayout-ban hozza létre, amelyet saját Cell objektumokkal tölt fel. Minden MyGButton-nek saját típusa van, a FileHandling file által betöltött számsorozatok alapján.

Tagfüggvények:

- **getBoard (): Cell [] []** -- Konstruktor által létrehozott pályát adja vissza
- **getFrame (): JFrame** -- Konstruktor által létrehozott JFrame-et adja vissza

  - **addFromBulbUp (int, int, Color): void**
  - **addFromBulbLeft (int, int, Color): void**
  - **addFromBulbDown (int, int, Color): void**
  - **addFromBulbRight (int, int, Color): void**

  Ez a 4 függvény felel a fényszint növeléséért, CSAK az egyik irányba.

- **addRays (Cell, Color): void** -- Ez a függvény a fő fényszintnövelő (ezt használja a játékos által elhelyezett lámpa), meghívja mind a 4 irányra a fényszintnövelő függvényeket.

  - **subFromBulbUp (int, int,int,int, Color): void**
  - **subFromBulbLeft (int, int,int,int, Color): void**
  - **subFromBulbDown (int, int,int,int, Color): void**
  - **subFromBulbUpRight (int, int,int,int, Color): void**

  Ez a 4 függvény felel a fényszint csökkentéséért, CSAK az egyik irányba.

- **subRays (Cell, Color): void** -- Ez a függvény a fő fényszintcsökkentő (ezt használja a játékos által elvett lámpa), meghívja mind a 4 irányra a fényszintcsökkentő függvényeket.
  - **whenLightLeft (int,int, Color, String): void**
  - **whenLightDown (int,int, Color, String): void**
  - **whenLightRight (int,int, Color, String): void**
  - **whenLightUp (int,int, Color, String): void**

  Ez a 4 függvény mind privát, csak a MovingBulbActionListener használja őket, az egyes állapotokból, egy másik állapotba mozgáshoz (ezek állapotgép függvények).

- **updateBoard (Color): void** -- Az egyes cellák fényszintje alapján változtatja (jeleníti meg) a képernyőn megjelenített pályát
- **playGame (): void** -- A konstrukban hívódik meg, a létrehozott pályán létrejövő összes gombot engedélyezi.
- **endGame (): void** – A Done! gomb megnyomására hívodik meg, ha a játékos általi megoldás helyes. A pályán lévő gombokat kikapcsolja.

3.1.2. Cell osztály

Ennek az osztálynak az objektumai építik fel magát a megjelenített játéktáblát.

Attribútumok:

- **int r -- Ez adja meg hogy a játéktábla melyik sorában található meg az objektum.**
- **int c -- Ez adja meg hogy a játéktábla melyik oszlopában található meg az objektum.**
- **String type -- A cellának a típusa, amit a Game Class konstruktora rendel hozzá egy szám alapján, kezdetben „unassigned”.**
- **int lightLevel – A cella fényszintjét adja meg.**

Konstruktor:

public Cell (MyGButton b, int i, int j)

Inicializálja a button, r, és c-t a paraméterként átadott értékekkel.

Tagfüggvények:

- **getButton (): MyGButton** – Visszadja a cellának a button-ját
- **getType (): String** -- Visszaadja a cellának a típusát
- **setType (): void** -- Beállítja a cellának a típusát
- **setLightLevel (): void** – Beállítja a cellának a fényszintjét.
- **getLightLevel (): int** – Visszaadja a cellának a fényszintjét.
- **addLight (): void** – A cella fényszintjét növeli eggyel.
- **subLight (): void** – A cella fényszintjét csökkenti eggyel.
- **getRow (): int** – A cella elhelyezkedésének a sorát adja vissza.
- **getCol (): int** – A cella elhelyezkedésének az oszlopát adja vissza.

3.1.3. MyGButton osztály

Ez az osztály a JButton egy leszármazottja.

Attribútumok:

- **long serialVersionUDI =1L**

Konstruktor:

public MyGButton ()

Létrehoz egy olyan button-t, amelynek előre beállított tulajdonságai/kinézete van.

Tagfüggvények:

--

3.1.4. FileHandling osztály

Ez az osztály felel a játékon belüli fájlkezelésekért.

Attribútumok:

--

Konstruktor:
default

Tagfüggvények:

- **getRowSize (String): int** – Beolvas egy paraméterként átadott pályát, és visszaadja annak a sorainak a hosszát.
- **getColSize (String): int** – Beolvas egy paraméterként átadott pályát, és visszaadja annak az oszlopainak a hosszát.
- **loadMap (String, String): ArrayList<ArrayList<Character>>** -- Beolvas egy pályát, vagy annak egy megoldását egy 2D ArrayList-be és visszaadja azt.
- **checkSol (String, String): boolean** – A játékos általi megoldás helyességét ellenőrzi a megoldás fájljával (két fájlt soronként összehasonlít).
- **copyFile (String, String): void** – Ez a függvény lemásolja a kiválasztott pálya eredeti állapotát egy másik fájl-ba (fájlmásoló függvény).
- **addBulb (Cell, String): void** – Ez a függvény a copyFile által lemásolt fájlt módosítja, a paraméterként átadott Cella (ahová a játékos lámpát rakott) megegyező sor/oszlopa alapján helyez el egy 'L'-t a játékos megoldásának a fájljába (tehát egy fájlban jelzi, hogy lámpaelhelyezés történt).
- **subBulb (Cell, String): void** – Ez a függvény a copyFile által lemásolt fájlt módosítja, a paraméterként átadott Cella (ahová a játékos lámpát rakott) megegyező sor/oszlopa alapján helyez el egy '6'-t a játékos megoldásának a fájljába (tehát egy fájlban jelzi, hogy lámpaelvétel történt).
- **delSols (): void** – A Game Class konstruktora mindig futtatja, hogy egyszerre csak egy játékos általi megoldás fájl létezzen (törli az összes játékos általi megoldás fájlt).

3.1.5. Menu osztály

Ez az osztály felel a menü kezeléséért. Ez az objektum jön létre elsőként a main-ben.

Attribútumok:

- **JFrame menu – Ez a menünek a kerete**
- **JComboBox jcb – Ez a menü keretében lévő opcióválasztó JComboBox**
- **Color color – Ez a JComboBox által kiválasztott lámpa színe. Alapértelmezett esetben sárga.**

Konstruktor:
public Menu ()
Létrehoz egy menüt, amiben gombok találhatóak, meg egy JComboBox. Ez állítja még be a menünek a kinézetét is. Az egyes gombokra kattintva létrejön egy Game objektum, a kiválasztott pálya és szín alapján.

Tagfüggvények:

- **setColor (Color): void** – Beállítja a JComboBox által kiválasztott színt.
- **getFrame (): JFrame** – Visszaadja a menünek a keretét.

3.2. Beépített (nested) osztályok bemutatása

3.2.1. SimpleBulbActionListener

Ez az osztály a Game osztályon belül található, és implementálja az ActionListener interface-t. Ez felel a játékos lámpaelhelyezéséért.

Attribútumok:

- **int x – A játékos által választott cellának az oszlopa**
- **int y – A játékos által választott cellának a sora**
- **Color color – A játékos által választott lámpa színe**
- **String map – A játékos által választott pálya száma**

Konstruktor:

public SimpleBulbActionListener (int y, int x, Color c, String m)
Inicializálja az attribútumokat a paraméterként átadott értékekkel.

Tagfüggvények:

- **public void actionPerformed(ActionEvent ae)** – Ez a felüldefiniált függvény, ami a következőt hajtja végre: Ha a játékos egy üres cellára kattint, akkor egy lámpát helyez el, ha egy lámpára kattint, akkor elveszi azt a lámpát.

3.2.2. MovingBulbActionListener

Ez az osztály a Game osztályon belül található, és implementálja az ActionListener interface-t. Ez felel a játék által elhelyezett forgatható lámpa forgatásáért.

Attribútumok:

- **int x – A játékos által választott cellának az oszlopa**
- **int y – A játékos által választott cellának a sora**
- **Color color – A játékos által választott lámpa színe**

Konstruktor:

public MovingBulbActionListener (int y, int x, Color c)
Inicializálja az attribútumokat a paraméterként átadott értékekkel.

Tagfüggvények:

public void actionPerformed (ActionEvent ae) – Ez a felüldefiniált függvény, ami a következőt hajtja végre: Ha a játékos egy mozgatható lámpára kattint, az elfordul óramutató járásával ellentétesen 90 fokkal. Ha a cella, amerre a lámpa fordulna már alapból bevan világítva, akkor többször fordul el 90 fokkal, mindaddig, amíg egy olyan cellához ér, ami nincs mellette bevilágítva. Ha minden irányból megvan világítva a lámpa, akkor rákattintva nem történik semmi, az eredeti állapotában marad. Mindezt egy állapotgép hajtja végre (4*4 db állapot ez összesen).

3.2.3. MapActionListener

Ez az osztály a Menu osztályon belül található, és implementálja az ActionListener interface-t. Ez szabályozza a játékos pályaválasztását egy gombra kattintva.

Attribútumok:

--

Konstruktor:
default

Tagfüggvények:

public void actionPerformed (ActionEvent ae) – Ez a felüldefiniált függvény, ami a következőt hajtja végre: Ha a játékos egy „map button”-re kattint, akkor az ActionListener-hez tartozó ActionCommand alapján létrehoz egy megfelelő játékszámú játékot.

3.2.4. ComboListener

Ez az osztály a Menu osztályon belül található, és implementálja az ActionListener interface-t. Ez szabályozza a játékos, pályához tartozó lámpa színének állítását.

Attribútumok:

--

Konstruktor:
default

Tagfüggvények:

public void actionPerformed(ActionEvent ae) – Ez a felüldefiniált függvény, ami a következőt hajtja végre: Ha a játékos a JComboBox-ból kiválaszt egy színt, akkor amikor egy pálya betöltésére kattint, akkor azzal a színnel fog minden lámpa világitani.

3.2.5. MyMButton

Ez az osztály a JButton egy leszármazottja.

Attribútumok:

- **long serialVersionUDI =1L**

Konstruktor:
public MyGButton ()
Létrehoz egy olyan button-t, amelynek előre beállított tulajdonságai/kinézete van. Ez az osztály használja fel a MapActionListener osztályt.

Tagfüggvények:

--

3.3. Tesztelésért felelős osztály bemutatása

3.3.1. AkariTest osztály

Ez az osztály a felel a JUnit tesztesetekért.

Attribútumok:

- **ArrayList lines – Ebben tárolódik egy pálya felépítése karaktersorozatként**

Konstruktor:

default

Tagfüggvények:

`@BeforeEach`

- **public void initialise () throws Exception** – Ez tölti fel a más osztályból származó objektumokat, értékekkel.

`@Test`

- **public void cellTest ()** – Ez a teszteset teszteli a cellák getter függvényeinek a helyes működését.

`@Test`

- **public void fileHandlingTest () throws IOException** – Ez a teszteset teszteli a fájlkezelésért felelős függvények a helyes működését.

`@Test`

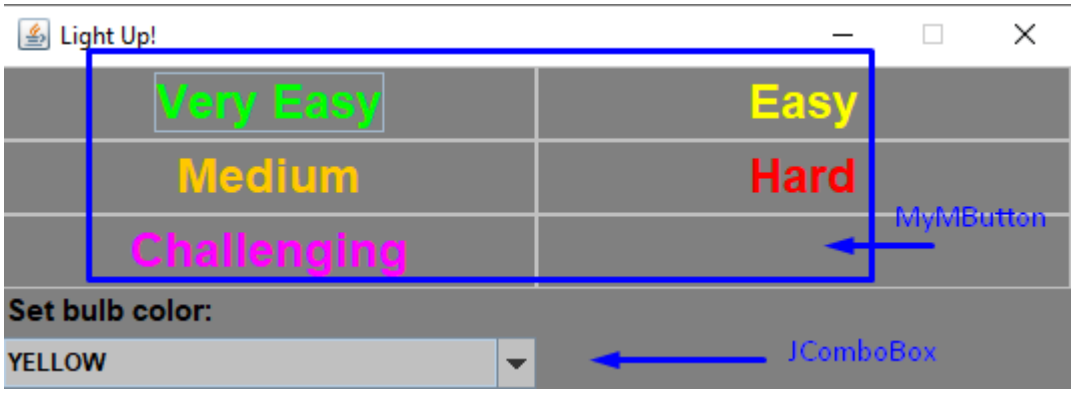
- **public void myGButtonTest** ( ) – Ez a teszteset teszteli a cellákban létrejövő MyGButton-ok helyes kinézetét.

`@Test`

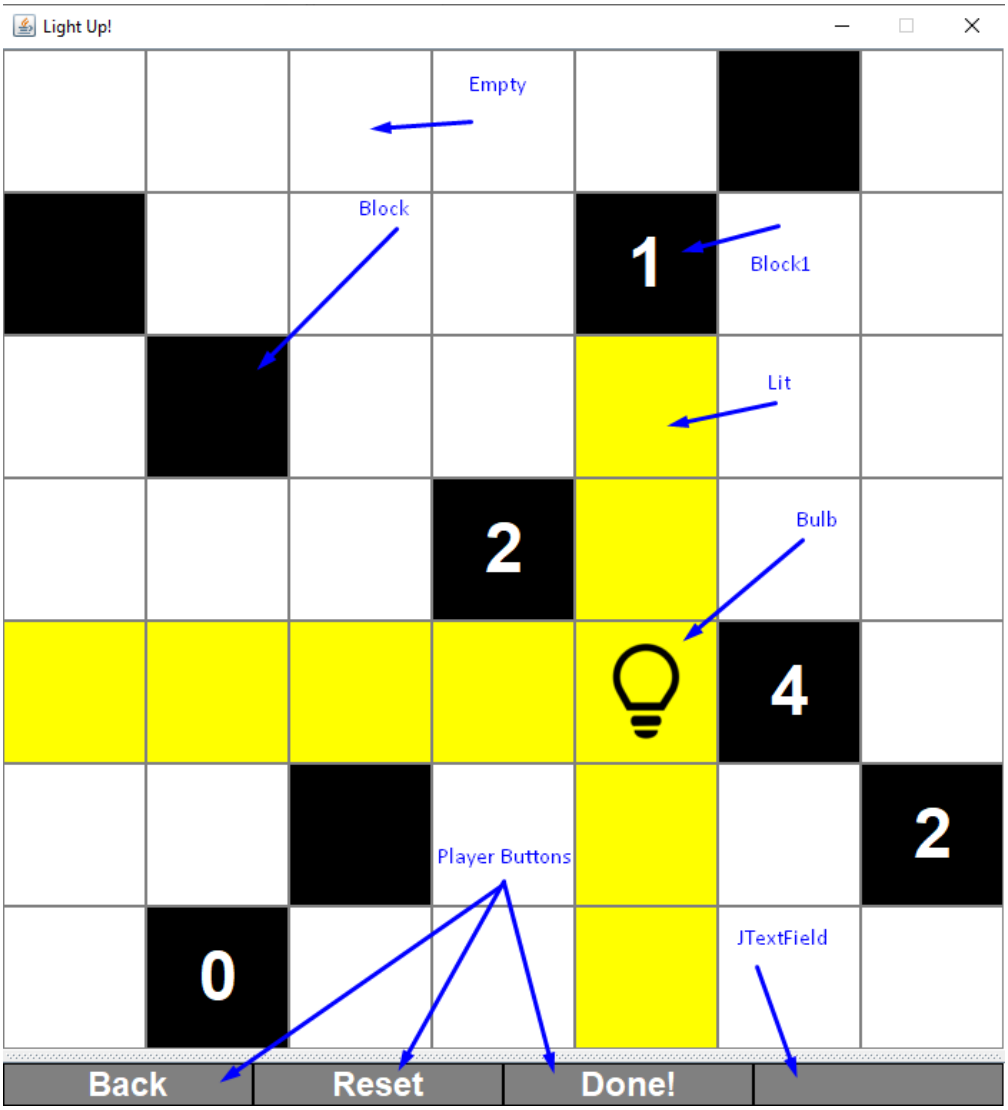
- **public void gameTest ()** – Ez a teszteset teszteli a játék menete közben történő fényváltozás helyességét (részleteseb leírás a kódban kikommentezve). Emellett még a játék indítása/befejezését is teszteli.

4. A játék elemeinek vizuális bemutatása

4.1. Menü kinézete



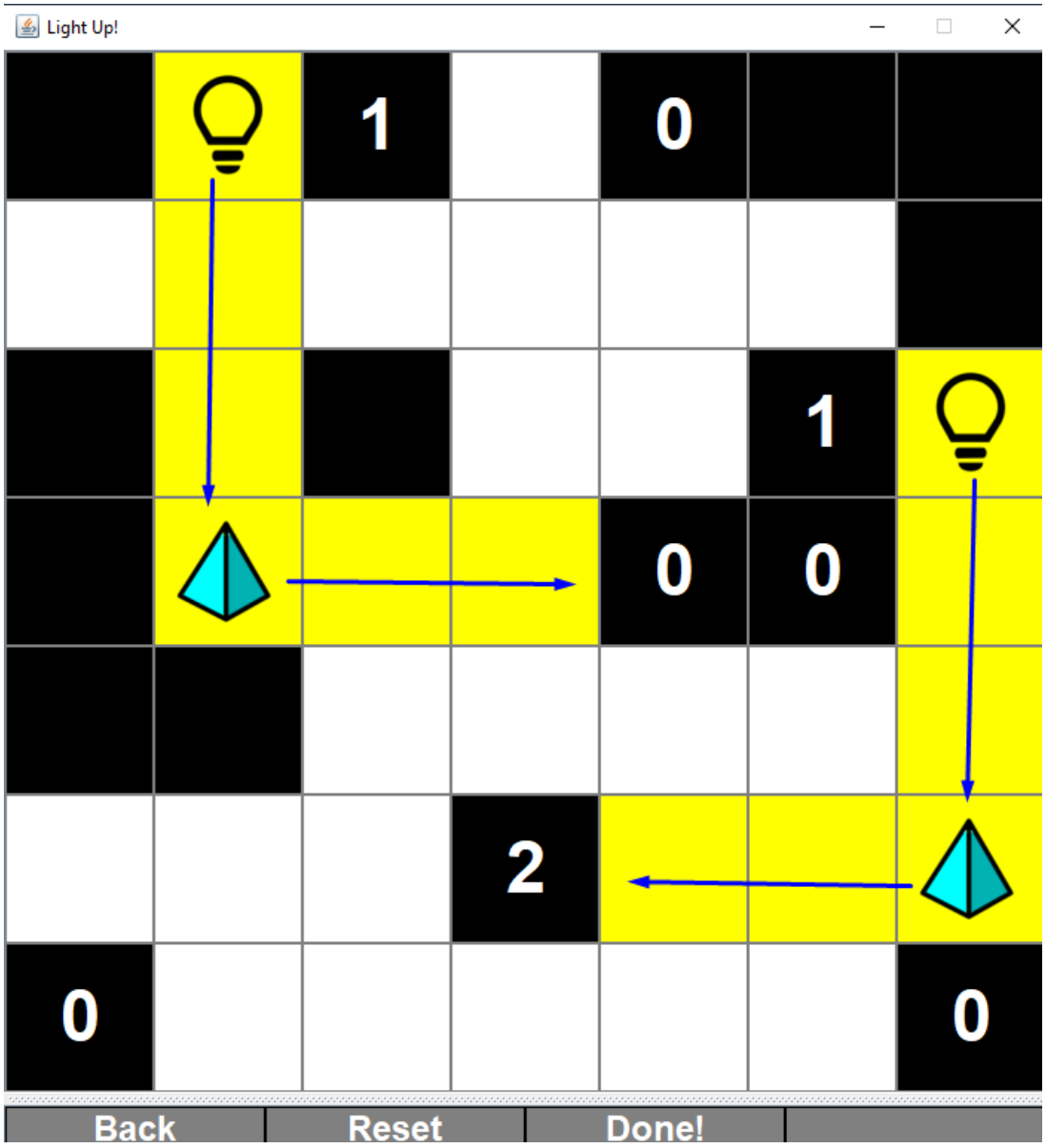
4.2. Pálya kinézete



4.3. Prizma bemutatása

Ez egy én általam kitalált játék elem az eredeti játékban nincs ilyen. Ha bárhonnan egy prizmát valamilyen lámpa megvilágit, akkor a prizma azt a fényt **minden irányba** megtöri (Dióhéjban: Ha egy prizmát fény ér, akkor az olyan mintha, a prizma helyére is rakott volna a játékos egy lámpát).

FONTOS! Az az eset nem került lekódolásra, amikor egy prizmából jövő fény, egy másik prizmába megy, ezért az a szabály, hogy két prizma nem lehet azonos oszlopban/sorban.

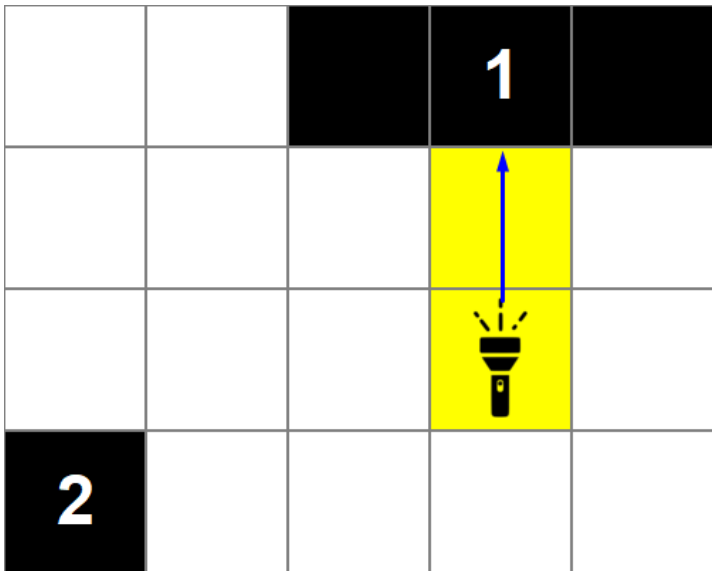


4.4. Mozgó lámpa bemutatása

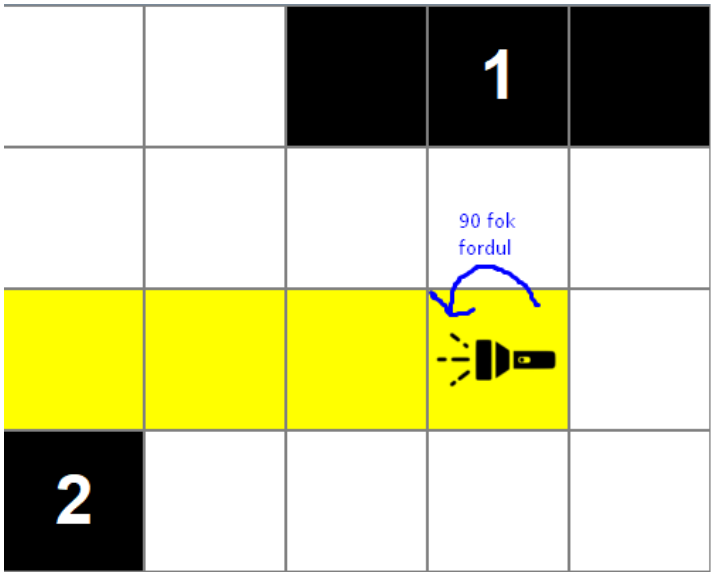
Ez egy én általam kitalált játék elem az eredeti játékban nincs ilyen. A MovingBulbActionListener osztály már ismertette a működését, itt csak vizuálisan mutatom be.

FONTOS! Ezeknek az állapotát a Done! gomb nem ellenőrzi, csak a pályák egyetlen helyes megoldásának megtalálásában segítenek ezek a lámpák.

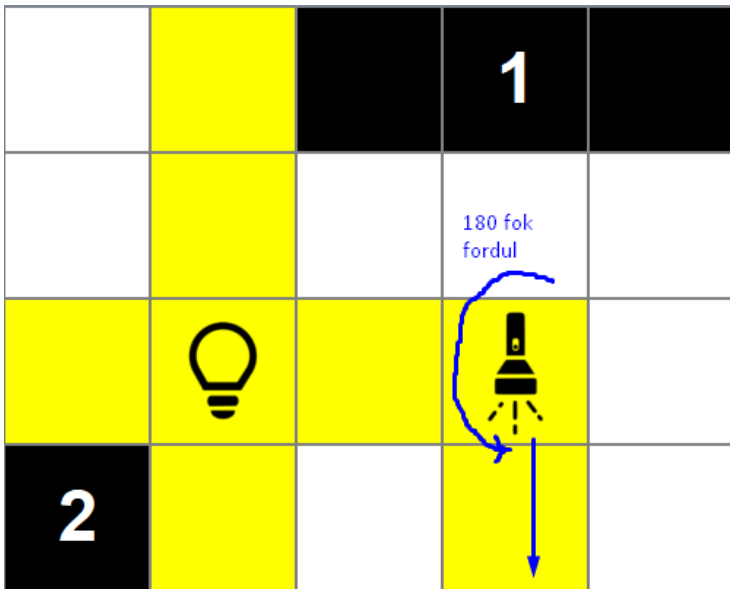
4.4.1. Alapállapot: fel



4.4.2. Kattintás után



4.4.3. Ha kattintás előtt fény világítja meg a mellette levő cellát



5. Felhasználói kézikönyv

A program futtatása során a felhasználó egy menüben találja magát, ahol gombokat láthat nehézségi szint szerint: Very Easy, Easy, Medium, Hard, Challenging. Ezek gombok egyikére kattintva a felhasználó betöltheti a nehézségi szinthez kapcsolódó pályát. Még mielőtt ezt megtenné egy lenyitható ablaknál (ami a menü bal alsó sarkában található) beállíthatja a lámpa színét: Sárga, Narancs, Piros, Zöld, Kék színűre.

A pályaválasztás után a program megjeleníti a kívánt pályát. A pályán lévő fehér négyzetekre kattintva elhelyezhet egy lámpát, ami minden irányba világit. Egy lámpára kattintva elveheti azt, ezzel megszűntetve az abból jövő fényt. A szabályokat lásd feljjebb! A pálya alatt a felhasználó még láthat három db gombot:

- Back – Ezzel a gombbal visszalehet lépni a menübe. VIGYÁZAT! A jelenlegi pálya állása elvész!
- Reset – Ezzel a gombbal újra lehet kezdeni a pályát.
- Done! – Ezzel a gombbal lehet ellenőrizni, hogy helyesen oldotta-e meg a felhasználó a feladványt. Ennek eredményét a Done! gombtól jobbra található részre fogja a program kiírni, ami lehet:
    - Correct Solution!
    - Incorrect Solution!

Ha a felhasználó helyesen oldotta meg a pályát, akkor a program már tovább nem engedélyezi a pálya módosítását. Ezután a felhasználó vagy kiléphet a programból a jobb felső sarokban található X-re kattintva, vagy visszaléphet a menübe a Back gomb lenyomásával, új pályát választva ott.