

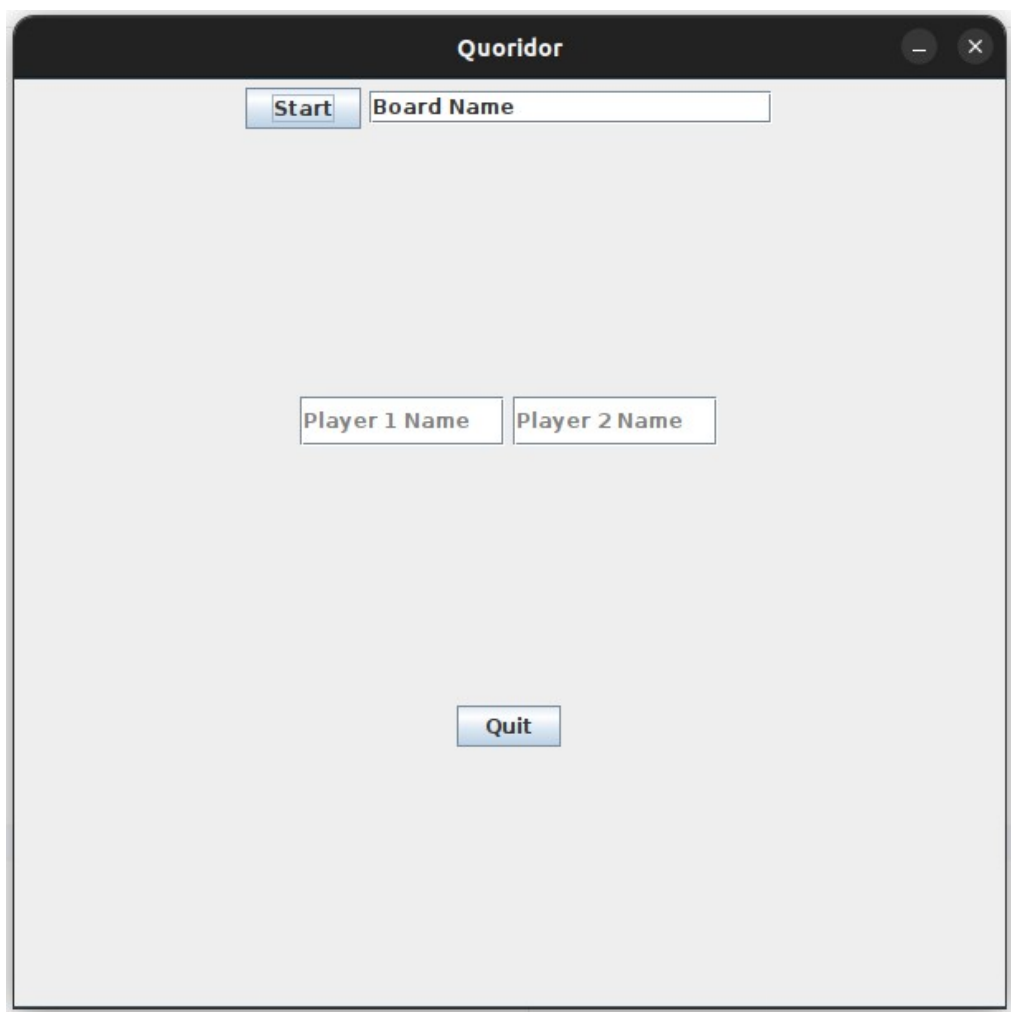
# Programozás alapjai 3. NHF Dokumentáció

Hubai Rajmund Szilveszter (CQKFI2)

A Quoridor nevezetű társasjátékot valósítottam meg Swing környezetben. A programban két játékos egymás ellen játszhat.

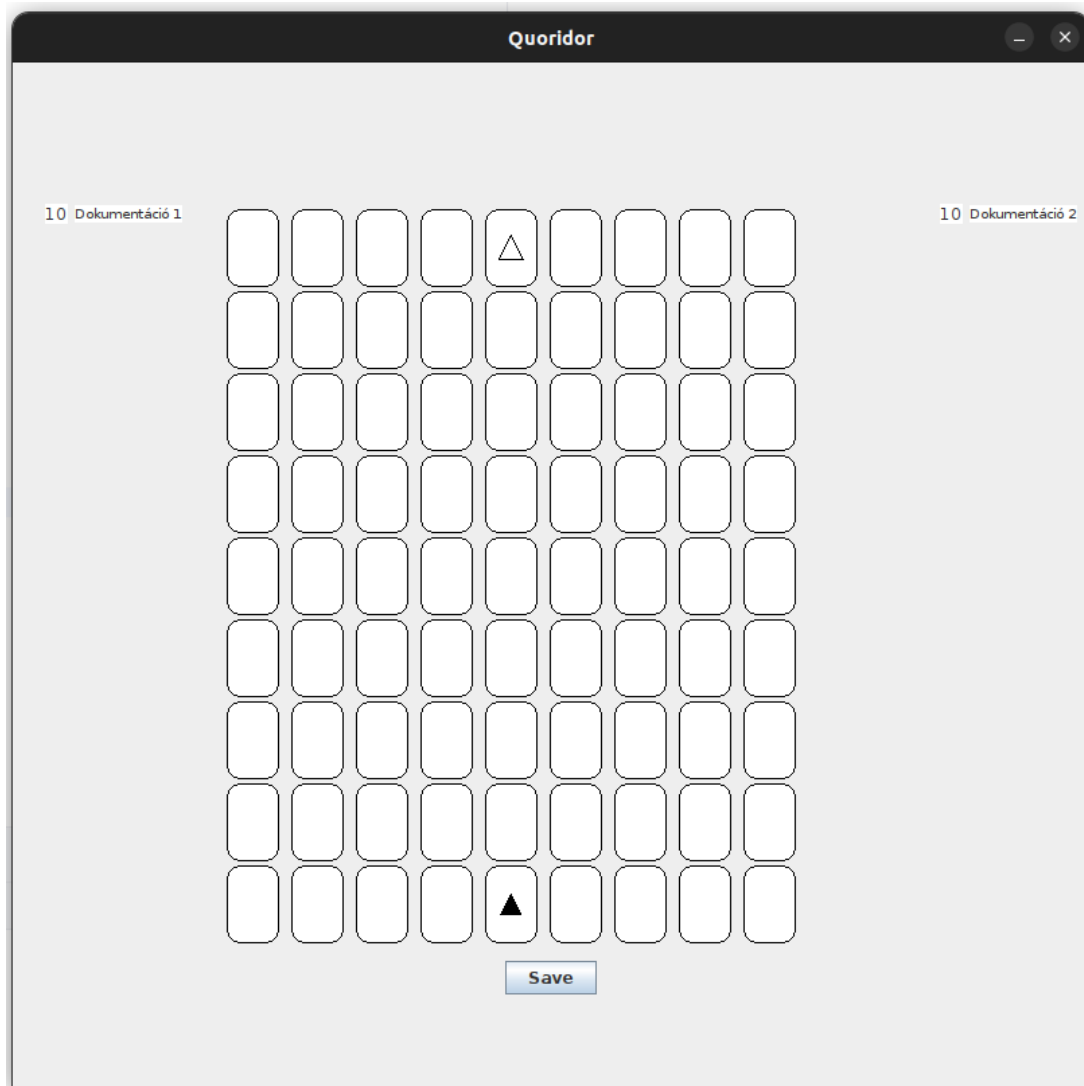
A játék célja, hogy mindkét játékosnak van 1-1 bábuja illetve 10-10 akadály, egymással szemben állnak a felek és mindkét félnek át kell jutnia pálya másik oldalára. Az nyer, aki először tud átélni, közben ellenfelüket akadályozni kell ebben, erre van a 10-10 akadály, teljesen azonban nem lehet bezárni - egy kivezető utat mindig szabadon kell hagyni. A soron lévő játékos választhat, hogy lép egyet a bábujaival, vagy elhelyez egy akadályt a táblán. Ha elfogyott az akadály lépni kell a bábuval. A bábuval egyszerre egyet lehet lépni bármelyik irányban, kivéve akadályokra. Ha két bábu szemben áll egymással úgy, hogy nincs közöttük akadály, a soron lévő játékos átugorhatja ellenfele bábuját.

A program indításakor egy kezdőképernyő fogad minket, melyen 2 gomb található: Start és Quit, illetve 3 beviteli pont az alább látható módon:

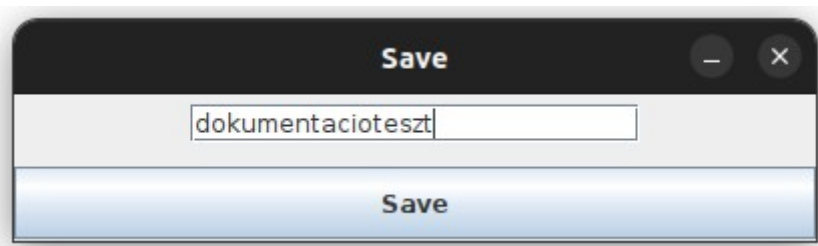


The screenshot shows a Java Swing window titled "Quoridor". Inside the window, there is a "Start" button at the top left. To its right is a text input field labeled "Board Name". Below these, there are two text input fields side-by-side, labeled "Player 1 Name" and "Player 2 Name". At the bottom center of the window is a "Quit" button. The window has a standard OS title bar with minimize, maximize, and close buttons.

A “Board Name” mezőben beírhatjuk a már elkezdett játékunkat és az a program ezt a táblát beolvassa és ugyanonnan lehet folytatni, ahol abba lett hagyva. A “Player 1 Name”, illetve a “Player 2 Name” mezőnél beállíthatjuk a két játékos nevét. Ha megnyomja a “Játék” gombot, illetve kiválasztotta, hogy mentett játékot szeretne folytatni vagy egy új játékot elkezdni, akkor ez a képernyő fog fogadni. A játékosokat a WASD illetve a nyilakkal lehet mozgatni, vízszintesen lerakni a blokkokat a bal klikk-kel, függőlegesen a jobb klikk lenyomásával lehet lerakni. Dupla kattintással felvehetőek az akadályok.

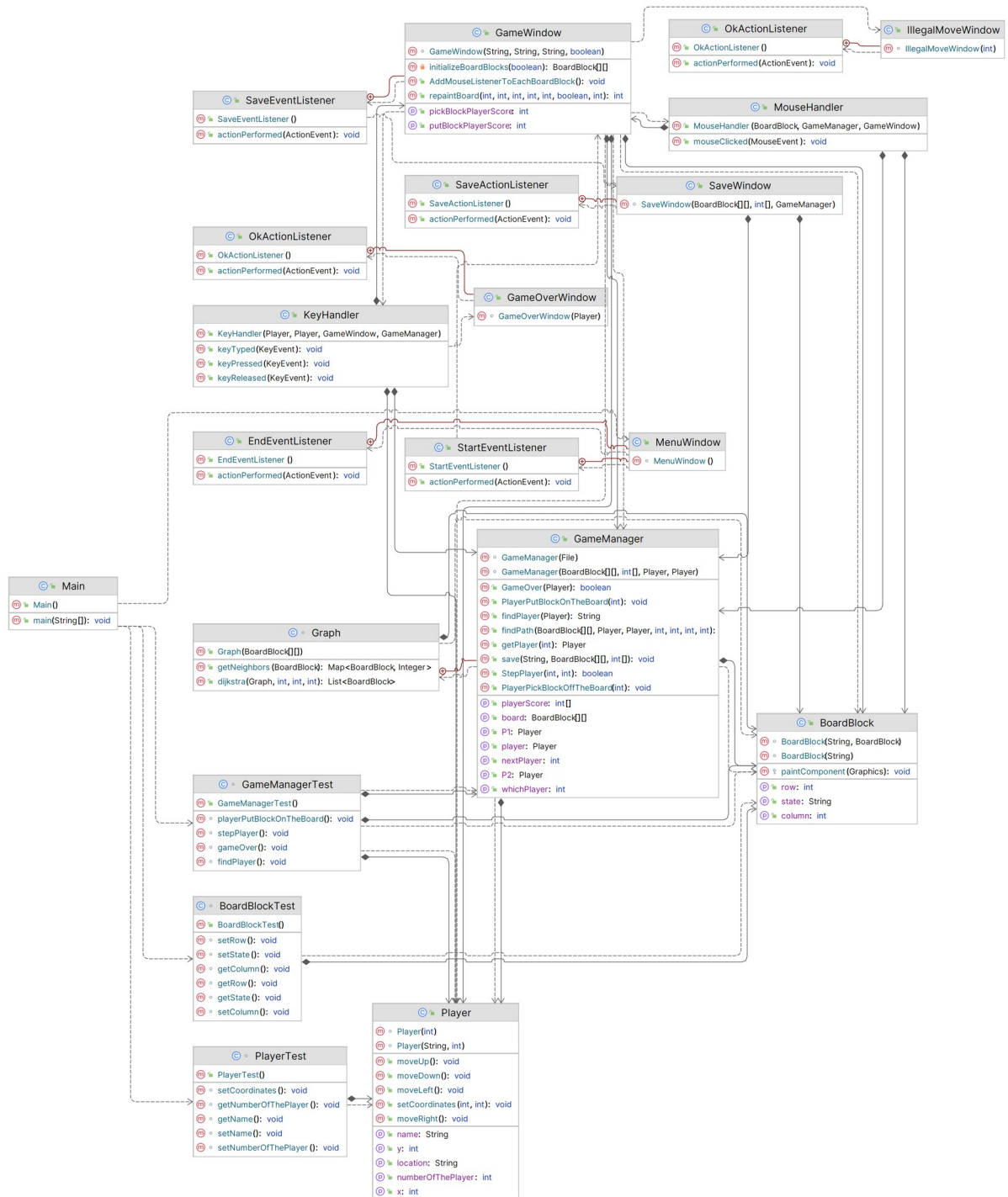


Alul található a “Save” gomb, amivel el lehet menteni a pályát, ekkor a felhasználótól kér egy szöveget, milyen néven legyen elmentve a fájl:



# Program működése

A program UML-diagramja:



## Adatstuktúrák

A játékhoz főként tömböket használtam a játéktér és a játékosok akadályainak számának tárolásához, mivel mindkettő adat fix mérettel rendelkezik. A játéktér egy 9x9-es, kétdimenziós mátrixtömbben van eltárolva BoardBlock-ok szerint.

A játék szabályzata szerint az ellenfél játékost nem lehet elkeríteni akadályokkal, hogy ellehetetlenítse a játékos eljutását a másik oldal felé, ezért az egész játéktérrel áthelyeztem egy HashMap kollekcióba egy BoardBlock és a hozzá tartozó szomszédai szerint, hogy ezen gráfalkotó algoritmust illetve legrövidebb útvonalkereső – Dijkstra – algoritmust futtassak.

### BoardBlock:

A BoardBlock osztálynak van állapota (state), sor- és oszloptulajdonsága (row, column), illetve grafikája, amit kirajzol a GameWindow.

### Player:

A Player osztálynak van neve (name), hanyadik játékos (numberOfThePlayer) illetve x és y koordinátái, ami a játéktéren belüli helyzetéhez kell.

### GameManager:

A GameManager osztálynak van játéktere (board), játékosok akadályainak számára alkalmas tömbje (playerScore), a játékos, ami éppen lépett (whichPlayer), a soron következő játékos (nextPlayer), illetve két játékos (P1, P2).

### Graph:

A Graph osztálynak két attribútuma van, az egyik egy blokk – szomszédság szerinti hashmap (graphMap), a másik a játéktér másolata (copyOfBoard).

A program mentésénél egy “.txt” kiterjesztésű file-ba menti az alább látható módon, így tárolja el a mezőbeli “értékeket” és az ellenfelek akadályainak számát:

```
10;9;Player 1 Name;Player 2 Name
E E E E P1 E E E E
E E E B B E E E E
...
```

- Az első sorban az első, illetve a második játékosnak a megmaradt akadályait és a játékosok neveit írja “;”-kel elválasztva
- “E”, mint Empty
- “P”, mint Player
- “B”, mint Block
- “P1”, mint első játékos
- “P2”, mint második játékos

## MenuWindow függvényei:

MenuWindow() :

- Ez a MenuWindow konstruktor. Lehelyezi a JButton gombokat, a JTextField szövegeket, amibe a felhasználótól bekéri a tábla és a játékosok neveit. A boardName, P1Name és a P2Name-be hozzáad egy-egy FocusListener, ami lehetővé teszi, hogy a JTextField-be placeholder kerüljön. A Start JButton-re egy StartEventListener()-t ad hozzá, amivel példányosítja a GameWindow osztályt azokkal a kritériumokkal, hogyha például a felhasználó nem írt be a boardName-hez szöveget, akkor úgy a GameWindow kezeli ezt.

StartEventListener.actionPerformed(ActionEvent e):

- Ez a "Start" JButton ActionListener osztályában lévő felülírt actionPerformed() metódus, ami lehetővé teszi a GameWindow példányosítását, illetve a MenuWindow bezárását.

EndEventListener.actionPerformed(ActionEvent e)

- Ez az "Quit" JButton ActionListener osztályában lévő felülírt actionPerformed() metódus, ami lehetővé teszi a program bezárását System.exit(0) függvény hívásával.

## GameWindow függvényei:

GameWindow():

-Ez a GameWindow konstruktor. Itt a MenuWindow-ban megadott paraméterek alapján meghívja a GameManager osztályt, ami felelős a játék működésért. E függvényben a GameManager segítségével legenerálja a pályát, illetve a játékosok neveit és pontszámait, hogy hány darab akadály áll rendelkezésükre.

private BoardBlock[][] initializeBoardBlocks(bool):

-Ezt a segédfüggvényt hívja meg a GameWindow konstruktor. Ebben inicializálja a játékteret, hozzáadja a JPanel-hez a blokkokat, illetve lehelyezi a játékosokat. Kétféle változat van, ha a menüben nem adott hozzá meglévő táblát, akkor felállít egy kezdetleges teret, ha hozzáadott, akkor lekéri a GameManager osztályból ezt a BoardBlock[][] attribútumot és aszerint tölti fel a JPanel-t. Mindkét esetben végig iterál a tömbön és a meglévő BoardBlock[][] mátrixszal tér vissza.

public void AddMouseListenerToEachBoardBlock():

-Ezt a segédfüggvényt szintén meghívja konstruktor. Itt hozzáadja az összes BoardBlockhoz a MouseListenereket, amikkel kattintható lesz a blokk.

Public int repaintBoard(int,int,int,int,int,bool,int):

-Ebben a metódusban, ha a felhasználó leütött egy billentyűt, akkor a játékterét újrarajzolja aszerint, hogy hogyan lépett a játékos. A metódus legelején kezeli azt, hogy kétszer akart-e lépni a játékos

illetve nem olyan helyre lépett a játékos, ahova lehet, ekkor létrehoz egy `IllegalMoveWindow` osztályt, ami kiírja a hibaüzenetet, hogy miért nem engedte lépni a játékost a függvény.

`SaveEventListener.actionPerformed(ActionEvent e):`

-Ez a metódus a "Save" gombhoz tartozik, ebben inicializál egy új `SaveWindow` osztályt, ami lehetővé teszi a pálya elmentését.

`public void setPutBlockPlayerScore(int):`

-Ez a metódus azt teszi lehetővé, ha valamelyik játékos letett egy akadályt, akkor csökkenti a játékos akadályainak számát.

`public void setPickBlockPlayerScore(int):`

-Ez a metódus hasonlóan működik, mint az előző, csak azzal a kivétellel, ha felvesz a játékos egy akadályt, akkor növeli eggyel az akadályainak számát.

## **IllegalMoveWindow függvényei:**

`IllegalMoveWindow(int):`

- Ez az `IllegalMoveWindow` konstruktor, itt adja hozzá a `JButton` gombot és a `JTextArea` szöveget. Bemeneti paraméterként egy számot kap, amivel eldönti, hogy milyen hiba adódott, milyen hibaüzenetet kell kiírnia. Ha 1-es, akkor az a hiba állt fenn, hogy kétszer egymás után akar lépni az egyik játékos, ha 2-es, akkor a játékos nem léphet arra a mezőre, ha 3-as, akkor nincs már több akadály a játékosnak, holott le akarja rakni, illetve ha 4-es számot kap, akkor amit beírt fájl nevet, nem található.

`OkActionListener.actionPerformed(ActionEvent):`

- Ez a függvény a `JButton`-nek van, ha rányom, akkor ez az ablak bezárul, ha 4-es hibát kapott az osztály inicializálásakor, akkor kilép a teljes programból.

## **GameOverWindow függvényei:**

`GameOverWindow(Player):`

- Ez a `GameOverWindow` konstruktor, az `IllegalMoveWindow`-hoz hasonlóan működik, csak a paraméterül kapott játékos nevét írja ki, hogy ő nyert.

## **SaveWindow függvényei:**

SaveWindow(BoardBlock[[]], int[], GameManager):

- Ez az osztály konstruktor, a fentiekhez hasonlóan működik, viszont itt van egy beviteli mező, ahova a felhasználó be tudja írni, hogy milyen néven szeretné elmenteni a megkezdett játékát.

## **GameManager függvényei:**

GameManager(BoardBlock[[]], int[], Player, Player):

- Ez a GameManager osztály egyik konstruktor, itt beállítja adattagjait a bemenő paraméterek szerint.

GameManager(File gameFile):

- Ez a másik konstruktor, itt paraméterként kap egy fájlt, és abból olvassa be az adatokat, a bemeneti fájl szerint állítja be adattagjait.

public void save(String, BoardBlock, int[]):

- Ez a függvény lehetővé teszi a mentést, ami a megadott fájlnévre kiírja a játék állását.

Ez az osztály fogja menedzselni a mentés és a betöltés funkciókat, hogy el tudjuk menteni, illetve be tudjuk tölteni a félbehagyott játékot.

public boolean StepPlayer(int,int):

- Ez a függvény vizsgálja meg a bemenő paraméterek szerint, hogy a játékos léphet-e arra a mezőre. Ha a két paraméter túlmegy a board tárolási kapacitásán, hamis értéket ad vissza, különben leellenőrzi, hogy a bemenő paraméterek szerint a board-ban található blokk állapota "E" vagy sem.

public boolean GameOver(Player):

- Ez a függvény ellenőrzi, hogy a két játékos x koordinátája egyenlő-e 0-val vagy 8-cal a paraméterül kapott játékos szerint.

public String findPlayer(Player):

- Ez a függvény csak a debuggoláshoz lett használva, megkeresi a paraméterül kapott játékos helyzetét és visszaadja koordinátáit (x,y) alakban.

public void PlayerPutBlockOnTheBoard(int):

- Ez a függvény a paraméterül kapott játékos számától függően változtatja a playerScore[] valamely értékét csökkenti 1-gyel, ha lerakott egy akadályt.

`public void PlayerPickBlockOffTheBoard(int):`

- Ez a függvény a paraméterül kapott játékos számától függően változtatja a `playerScore[]` valamely értékét növeli 1-gyel, ha lerakott egy akadályt.

`public boolean findPath(BoardBlock[][], Player, Player, int, int, int, int):`

- Ez a függvény teszi lehetővé, hogy ellenőrizze azt, hogy a játékos elérhet-e a neki megadott számú sorba, ha nem, akkor hamis, különben igaz értékkel tér vissza a függvény. Itt futtatja a dijkstra algoritmust mindkét játékosra.

## **Graph függvényei:**

`Graph(BoardBlock[][]):`

- Ez a Graph konstruktor, itt tölti fel a `graphMap` hashmap-et, megkeresi a blokkok szomszédait, amit a paraméterül kapott mátrixból nyer ki, illetve ezt a mátrixot lemásolja `copyOfBoard`-ba.

`public List<BoardBlock> dijkstra(Graph, int, int, int):`

- Ez a függvény teszi lehetővé a dijkstra algoritmus futtatását a gráfon. A dijkstra lefut, végül visszaadja a megkeresett legrövidebb útvonalat a játékos és a célsor felé. A `findPath()` metódusban ellenőrzi le ezt, hogy üres-e, azaz nem talált útvonalat.

## **KeyHandler függvényei:**

`KeyHandler(Player, Player, GameWindow, GameManager):`

- Ez a függvény a konstruktor, itt állítja be a saját adatait a paraméterül kapottak szerint.

`public void keyPressed(KeyEvent):`

- Ez a függvény kezeli a gombnyomásokat aszerint, hogy melyik játékos van éppen, végül a `GameWindow` osztállyal újrarajzoltatja a pályát az újonnan megadott paraméterek alapján.

## **MouseHandler függvényei:**

`MouseHandler(BoardBlock, GameManager, GameWindow):`

- Ez a függvény a konstruktor, itt állítja be a saját adatait a paraméterül kapottak szerint.

`public void mouseClicked(MouseEvent):`

- Ez a függvény a kattintás után teszi le a blokkokat, előtte ellenőrzi, hogy melyik játékos volt, illetve a blokk letétele után van-e útvonala a játékosoknak, ha nem, akkor az akadály lerakása nem történik meg.



## **Tesztelő osztályok függvényei:**

Minden tesztelő osztálynak van konstuktora, amivel a tesztelendő osztályokat (GameManager, Player, BoardBlock) meghívja, és a getter-setter metódusain kívül tesztel számos függvényt.