HÁZI FELADAT

Programozói dokumentáció

Kovács Dávid
FYEAS5

2023. november 26.

TARTALOM

1. Feladat

A feladat a malom táblajáték megvalósítása grafikus felhasználói interfésszel, valamint a lejátszott játékok eredményéből egyszerű statisztikákat kiállítani Java programozási nyelven.

2. Pontosított feladatspecifikáció

A közismert játék célja az, hogy az ellenfél korongjait levegyük, vagy olyan helyzetbe kerüljünk, hogy az ellenfél képtelen legyen lépni. Ennek elsődleges módja a malmok kialakítása (függőleges vagy vízszintes sorokban három korong egymás mellett), ugyanis egy malom létrejöttével levehetünk egy korongot az ellenfél készletéből (abból, ami a táblán van). Ha valakinek csak két korongja marad, elvesztette a játékot, mert már nem tud malmot kialakítani.
A játék alapvetően három fázisból áll:

- a bábuk felrakása a táblára
- lépegetés a táblán
- ugrálás a táblán (a korongot bárhova lehet rakni).

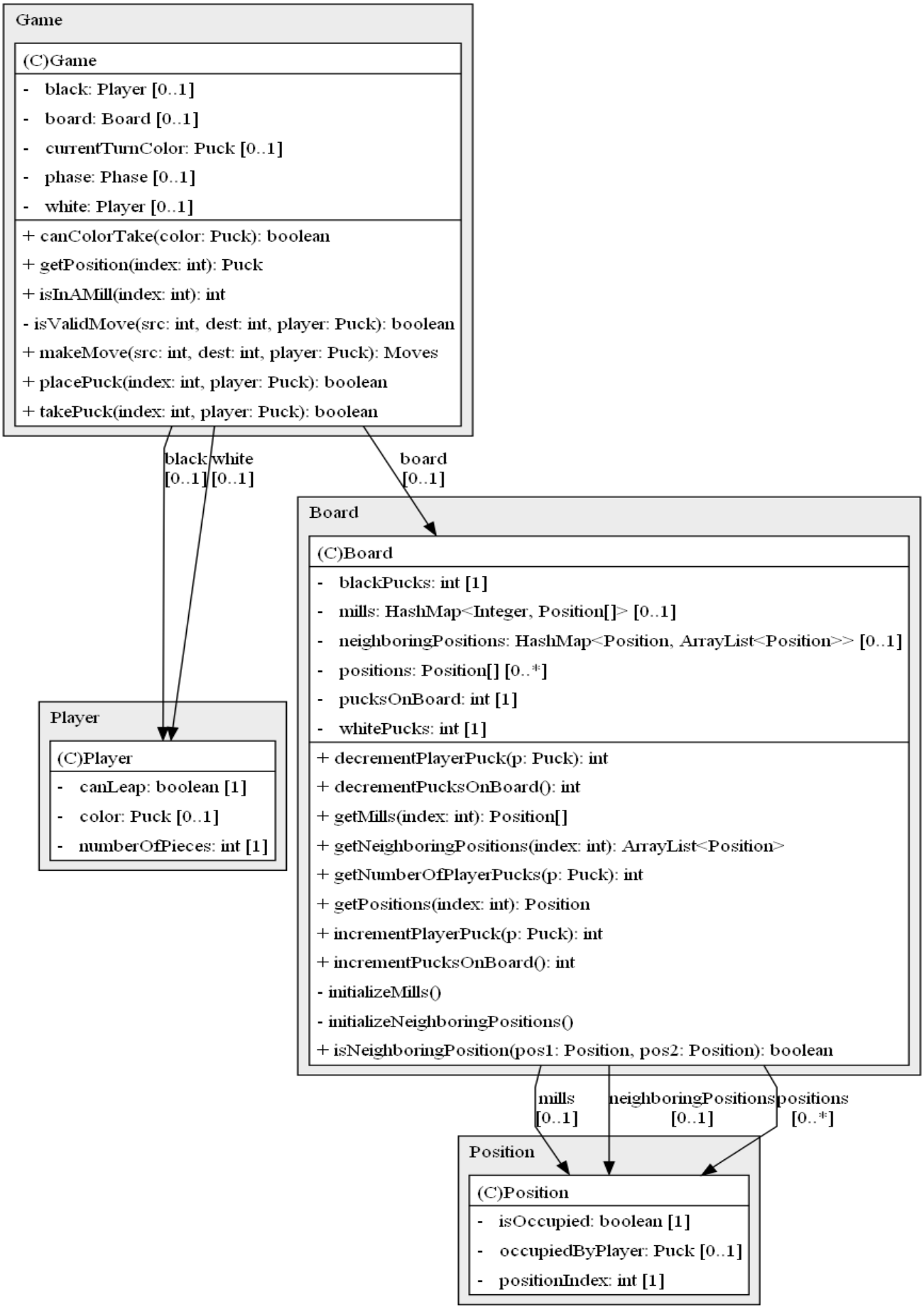
Az első fázisban az üres táblára a játékosok felváltva rakják fel a korongokat, mindig a világos kezd. Már a rakosgatásban is ki lehet rakni malmot. Ha valaki malmot rak ki, akkor levehet egy korongot az ellenfelétől, azt követően az ellenfele következik. Az ellenfél malmából nem vehetünk el korongot, kivéve, ha az ellenfél minden korongja malomban áll! Mindkét játékosnak 9-9 korongja van, ha ebből egyet már levettek, az természetesen már nincs játékban.
Ha az összes korongot felrakták a táblára, illetve már nem tudnak többet rakni, akkor következik a második fázis, a lépegetés. A korongok vízszintes és függőleges irányban mozdíthatók, átlósan nem. Olyan mezőre nem léphetünk, ahol már áll egy korong, és mindig csak a közvetlenül szomszédos csúcspontig léphetünk a vonal mentén. Ha valakinek már csak három korongja maradt, akkor azokkal szabadon ugrálhat a pálya bármely pontjára, és ugrálva is alakíthat ki malmot. A játék addig zajlik, míg valamelyik félnek 2 korongja marad a táblán, vagy nincs érvényes lépése.

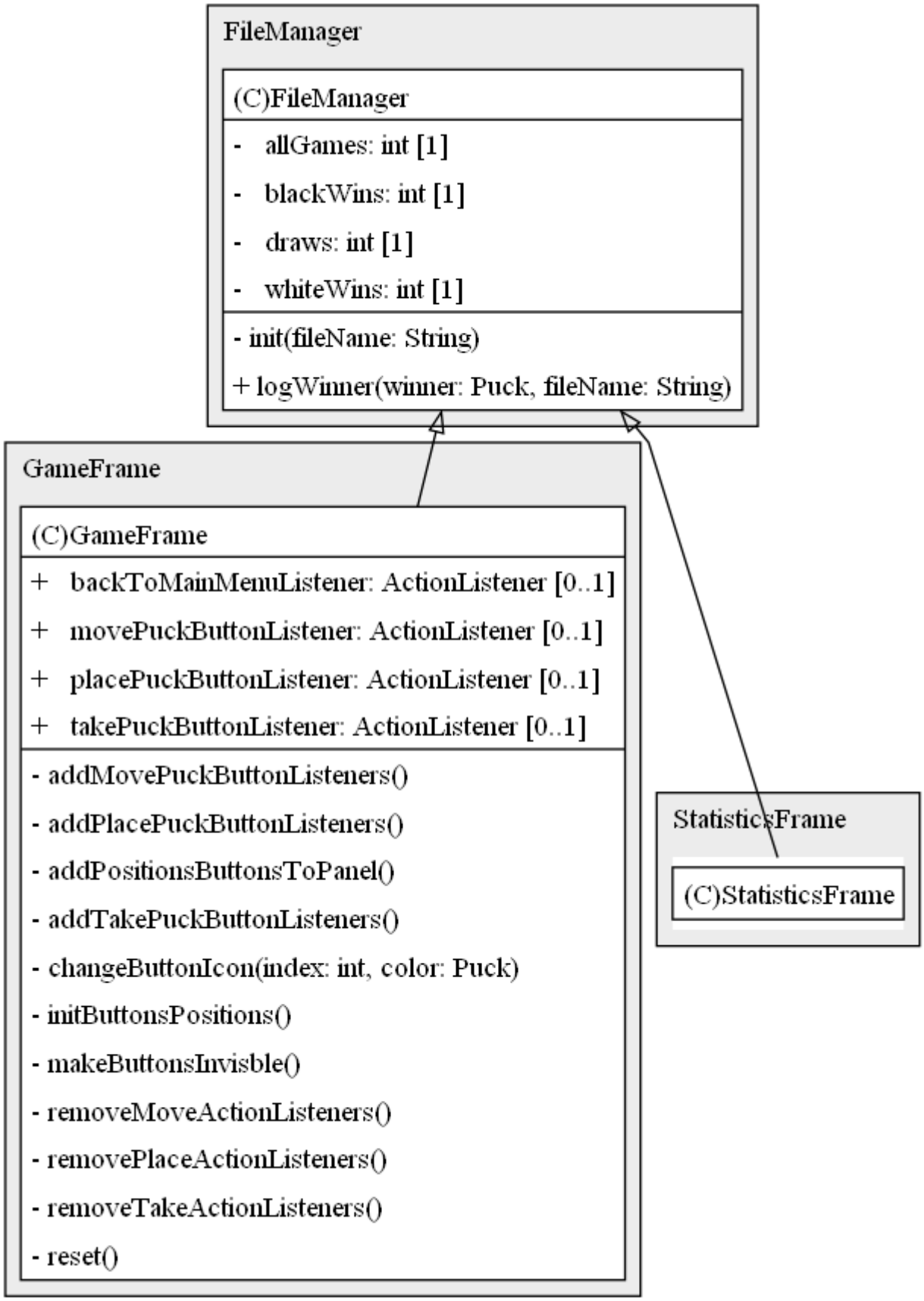
3. Terv

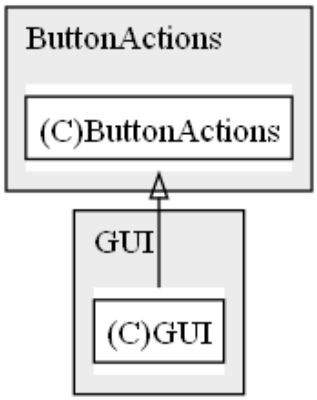
A feladatot lebontottam alapvető részekre, mint egy pozíció, tábla és maga a játék logikája. A grafikus felületek szintén részekre vannak bontva. Az említetteknek az együttes működése teszi lehetővé a program megvalósítását. **Figyelem,** a program a fájlok olvasás és írását a forráskódot tartalmazó mappából végzi el.

3.1. Objektum terv

A feladat megvalósítása számos osztályt vesz igénybe, lentebb találhatók az osztályok UML osztály diagramja, valamint az osztályok és további magyarázatot igénylő metódusaik leírása.


Game
(C)Game
- black: Player [0..1]
- board: Board [0..1]
- currentTurnColor: Puck [0..1]
- phase: Phase [0..1]
- white: Player [0..1]
+ canColorTake(color: Puck): boolean
+ getPosition(index: int): Puck
+ isInAMill(index: int): int
- isValidMove(src: int, dest: int, player: Puck): boolean
+ makeMove(src: int, dest: int, player: Puck): Moves
+ placePuck(index: int, player: Puck): boolean
+ takePuck(index: int, player: Puck): boolean
black, white
[0..1] [0..1]
board
[0..1]
Board
(C)Board
- blackPucks: int [1]
- mills: HashMap<Integer, Position[]> [0..1]
- neighboringPositions: HashMap<Position, ArrayList<Position>> [0..1]
- positions: Position[] [0..*]
- pucksOnBoard: int [1]
- whitePucks: int [1]
+ decrementPlayerPuck(p: Puck): int
+ decrementPucksOnBoard(): int
+ getMills(index: int): Position[]
+ getNeighboringPositions(index: int): ArrayList<Position>
+ getNumberOfPlayerPucks(p: Puck): int
+ getPositions(index: int): Position
+ incrementPlayerPuck(p: Puck): int
+ incrementPucksOnBoard(): int
- initializeMills()
- initializeNeighboringPositions()
+ isNeighboringPosition(pos1: Position, pos2: Position): boolean
Player
(C)Player
- canLeap: boolean [1]
- color: Puck [0..1]
- numberOfPieces: int [1]
mills
[0..1]
neighboringPositions
[0..1]
positions
[0..*]
Position
(C)Position
- isOccupied: boolean [1]
- occupiedByPlayer: Puck [0..1]
- positionIndex: int [1]


FileManager
(C)FileManager
- allGames: int [1]
- blackWins: int [1]
- draws: int [1]
- whiteWins: int [1]
- init(fileName: String)
+ logWinner(winner: Puck, fileName: String)
GameFrame
(C)GameFrame
+ backToMainMenuListener: ActionListener [0..1]
+ movePuckButtonListener: ActionListener [0..1]
+ placePuckButtonListener: ActionListener [0..1]
+ takePuckButtonListener: ActionListener [0..1]
- addMovePuckButtonListeners()
- addPlacePuckButtonListeners()
- addPositionsButtonsToPanel()
- addTakePuckButtonListeners()
- changeButtonIcon(index: int, color: Puck)
- initButtonsPositions()
- makeButtonsInvisble()
- removeMoveActionListeners()
- removePlaceActionListeners()
- removeTakeActionListeners()
- reset()
StatisticsFrame
(C)StatisticsFrame



4. Osztályok és algoritmusaik

4.1. GUI

A főmenüt jeleníti meg, ami a felhasználó kijelzőjének méretéhez illeszkedik. Tartalmaz egy label-t ami a nyertest fogja megjeleníteni egy játék befejezése után, valamint három gombot: Új játék, Statisztikák, Kilepés.

4.2. Player

Játékosokat reprezentáló osztály, nincs nagy jelentősége, mert a Game osztállyal jelentősen egyszerűbben kezelhetők a játékosok színei és, hogy hány korongjuk van a táblán.

- numberOfPieces Hány korongja van a játékosnak.
- color A játékos színe.
- canLeap A játékos tud-e nem szomszédos pozícióra lépni.

4.3. Position

- A malom tábláján egy helyet reprezentáló osztály.
- occupiedByPlayer A helyen levő játékos (fekete vagy fehér).
- isOccupied Szabad vagy foglalt a pozíció.
- positionIndex A hely indexe.
- neighboringPositions Egy pozíció szomszédjai legalább 2 és legfeljebb 4 szomszédja lehet egy helynek.

4.3.1. `public Position(int position)`

A pozíció osztály konstruktora, inicializáláskor mindegyik pozíció üres, azaz nem foglalt.

Parmétere

position	A pozíció indexe a táblán

4.4. Board

A játéktáblát reprezentálja, mivel a malom viszonylag "zárt" játék és a pozícióknak a szomszédossága, valamint, hogy malomba tartozásuk nem követ konkrét logikát, ezért célszerűnek találtam ezeket úgymond kézzel inicializálni.

4.4.1. private void initializeNeighboringPositions()

Egy HashMap-be hozzárendeli mindegyik pozícióhoz a vele szomszédos pozíciókat egy lista fromájában.

4.4.2. private void initializeMills()

16 darab malom létezik a játékban(4-4 a külső, középső és belső négyzeteken, valamint 4 a négyzeteket összekötő részeken), a függvény mindegyik malomhoz hozzárendeli, hogy milyen pozíciók vannak benne.

4.4.3. public boolean isNeighboringPosition(Position pos1, Position pos2)

Paraméterként átvett pozícióra megállapítsa, hogy szomszédos-e az aktuális pozícióval. Ha szomszédosok true-t térít vissza, ellenkező esetben false-ot.

4.5. Game

A játék logikájáért felelős osztály, tárolja a játéktáblát, játékosokat, a játék fázisát, az aktuális körben melyik játékos kell lépjen, valamint az, hogy hány korong van a táblán elhelyezve.

4.5.1. private boolean isValidMove(int src,int dest,Puck player)

Megvizsgálja, hogy az adott játékos által lehetséges-e az src indexű pozícióban lévő korongot elmozgatni a dest pozícióhoz. A mozgatás lehetséges, ha az src pozícióba a játékos korongja áll, a dest pozíció nem foglalt és szomszédos, valamint, ha már a játékosnak 3 korongja van és tud ugrálni a táblán, akkor a dest pozíció nem kell szomszédos legyen, csak szabad. Privát metódus, mert a lépést a makeMove függvény hajtja végre ténylegesen.

Paraméterei

src	Kezdő pozíció indexe.
dest	Az a pozíció indexe, hová a felhasználó mozgassa a korongját.
player	Melyik játékos kíséreli meg a lépést.

Visszatérít: Igaz, ha a játékos meg tudja tenni a lépést, hamis, ha nem.

4.5.2. public Moves makeMove(int src,int dest,Puck player)

Amennyiben a lépés megengedett végrehajtja, azaz a kezdő pozíció üres lesz és a célt pedig a paraméterként átvett színű játékos fogja birtokolni. Könnyebb tesztelés és hibakeresés céljából, paraméterként meg kell adni, hogy melyik játékos viszi véghez a lépést, a függvény, ha a lépés nem megengedett, akkor annak az okatát téríti vissza, ezek részletezve vannak a Moves enum-ben.

Paraméterei

src	Kezdő pozíció indexe.
dest	Az a pozíció indexe, hová a felhasználó mozgassa a korongját.
player	Melyik játékos kíséreli meg a lépést.

Visszatérít: A lépés minősítése.

4.5.3. public boolean placePuck(int index,Puck player)

Az adott indexű pozíciót az átvett játékos fogja birtokolni, amennyiben az üres. Mivel ez a legelső fázis, ha már a játékosok leraktak összes 18 korongot, akkor a lépegetés fázisra vált.
Paraméterei

index	A pozíció indexe ahová el szeretne a játékos elhelyezni egy korongot.
player	Melyik játékos kíséreli meg a lépést.

Viszatérít: Igaz, ha a művelet sikeres, hamis, ha az a pozíciót már birtokolják.

4.5.4. public int isInAMill(int index)

Visszatéríti, hogy hány malomban van az adott indexű pozícó.
Paramétere

index	A pozíció indexe

Visszatérít: Hány molmban van benne az adott indexű pozíció.

4.5.5. public boolean takePuck(int index,Puck player)

Az adott indexű pozíciót üresre állítsa, ha azt az ellenkező játékos birtokolja (egy korong levétele a tábláról).
Paraméterei

index	A pozíció indexe ahová el szeretne a játékos elhelyezni egy korongot.
player	Melyik játékos vesz le egy korongot.

Visszatérít: Igaz, ha le sikerült venni a korongot, hamis, ha nem.

4.5.6. public boolean canColorTake(Puck color)

A függvény megvizsgálja, hogy az adott játékos tud-e levenni az ellenfél korongjai közül. Más szavakkal, ha az ellenfélnek az összes korongja malomban van akkor nem lehet levenni közülük egyet sem.
Paramétere

color	A játékos, amiről meg kell vizsgálni, hogy le tud-e venni az ellenfél korongjai közül.

Visszatérít: Igaz, ha le tud venni, ellenkező esetben hamis.

4.5.7. public boolean isGameOver()

Megvizsgálja, hogy a játékosoknak, van-e érvényes lépésük, illetve legalább három korongjuk.
Visszatérít: Igaz, ha van érvényes lépés és legalább három korong, ellenkező esetben hamis.

4.6. GameFrame

Ez az osztály valósítsa meg a játék grafikus megjelenítését és lebonyolítását. A pozíciókon JButton gombok vannak, az ablak 1024*768-as méretű és nem lehet átméretezni, azért, hogy a gombok ne csusszannak el. A tábla a Frame háttere, amin elhelyezkedik egy átlátszó JPanel, ami gombokat tartalmazza. A gombok egy tömbben vannak tárolva.

4.6.1. public void reset()

Kezdőhelyzetbe hozza az ablakot, beleértve a játék logikáját is.

4.6.2. private void changeButtonIcon(int index,Puck color)

Az adott indexű gombnak egy képet állít be az alapján, hogy milyen korong szín lett átadva.
Paraméterei

index	A gomb indexe.
color	Meghatározza milyen képet kell beállítani.

4.6.3. public ActionListener placePuckButtonListener

A játék első fázisát bonyolítsa le, a korongok lerakását. Ha változott a játék fázis akkor az annak megfelelő ActionListener-t adja hozzá a gombokhoz. Amennyiben egy korong lerakása után malom keletkezett és a játékos aki lerakta a korongot le tud venni legalább egyet az ellenfél korongjai közül akkor a takePuckButtonListener-t adja hozzá a gombokhoz.

4.6.4. public ActionListener takePuckButtonListener

A lenyomott gombnak megfelelő korongot leveszi a tábláról, ha az lehetséges. Ha a művelettel a játék véget-ért akkor elmenti a nyertest a „winners.txt” fájlba, valamint visszatér a főmenübe és kii-rja a nyertes játékos színét, különben az aktuális fázis ActionListener-ét rendeli hozzá a gombokhoz.

4.6.5. public ActionListener movePuckButtonListener

A korongok mozgatását bonyolítsa le. Mivel egy korong mozgatásához két gomb lenyomása szükséges, ezért az első gomb indexét egy stack-be rakja, ez lesz az a korong, amit el szeretnénk elmozgatni, a második gombnyomás lesz az a pozíció ahová szeretnénk mozgatni a korongot. Ha érvényes a lépés a kezdőpont gombját láthatatlanná teszi és a végpontot, pedig annak a játékosnak a színére állítsa, aki megtette a lépést. Ha sikerül olyan helyzetet teremteni, ahol egyik játékosnak sincs érvényes lépése, akkor a játéknak vége és a metódus menti az eredményt a „winners.txt” fájlba.

4.7. StatisticsFrame

Egyszerű statisztikákat jelenít meg grafikus formában, a szövegeket JLabel-ek tartalmazzák, valamint tartalmaz egyetlen gombot, amivel vissza lehet térni a főmenübe.

4.8. FileManager

Egyszerű fájlkezelő osztály, ami egy adott fájlbó kinyeri, hogy melyik szín hányszor nyert, illetve a döntetlenek számát. A fájlokban mindegyik sorban egy szám kell legyen, ami lehet 0,1 és 2. A 0 egy fekete győzelem, 1 fehér győzelem és a 2-es a döntetlen.

4.8.1. public static void logWinner(Puck winner, String fileName)

Adott fájl végére ír egy számot az adott Puck szerint. Puck.BLACK = 0, Puck.WHITE = 1 Puck.NONE = 2.
Paraméterei:

winner	Melyik szín győzelmét vagy döntetlent kell bejegyezni.
fileName	A fájl neve.

4.9. Phase enum

A játék fázisai: PLACING - korongok lerakása, MOVING - korongok mozgatása szomszédos helyekre, LEAPING - korongok mozgatása nem szomszédos helyekre.

4.10. Moves enum

A lépéseket minősíti. INVALID_MOVE - nem megengedett lépés, VALID_MOVE - megengedett lépés, INVALID_SRC - nem megengedett kezdő pozíció, INVALID_DEST - nem megengedett végpozíció.

4.11. Puck enum

A korongok színei WHITE – fehér, BLACK – fekete, NONE – egyik sem azaz szabad a hely.

5. Tesztelés

Az osztályok metódusainak tesztelése JUnit-al történik, lényegtelennek találtam a nagyon egyszerű metódusok tesztelését, ezért azok ki vannak hagyva.

5.1. FileManager Test

A FileManager osztályt konstruktorának és fájlba írásának tesztjei. **FIGYELEM** tesztelés után a test.txt és test2.txt fájlok tartalmát törölni kell.

5.1.1. public static void tearDown()

Helyreállítsa a fájlokat, hogy ne akadályozzák a teszt ismételt fútatását.

5.2. BoardTest

A Board osztályt konstruktorának, `isNeighboringPosition()` és `setPuckToPosition()` metódusainak tesztjei.

5.3. GameTest

A Game osztály unit tesztjei.

5.3.1. public void millTest()

Egy malom alkotását teszteli a függvény. Lehelyezünk három fehér korongot, úgy, hogy malmot alkossanak. Az isInAMill() függvény 1-et kell visszatérítsen, bármely korongra, mert mindegyik egyetlen malomban van benne.

5.3.2. public void makeValidMoveTest()

Érvényes korong mozgatásának tesztelése.

5.3.3. public void makeInvalidSrcMoveTest()

Korong mozgatása érvénytelen kezdőpontról.

5.3.4. public void makeInvalidDstMoveTest()

Korong mozgatása érvénytelen végpontra.

5.3.5. public void makeInvalidMoveTest()

Érvénytelen mozgatása, a mozgatás akkor érvénytelen, ha a játékos nem tud ugrálni a táblán és nem szomszédos pozícióra lép.

5.3.6. public void canColorTakeFalseTest()

Ha az összes fehér korong malomban van, akkor a fekete játékos nem tud korongot levenni.

5.3.7. public void canColorTakeTrueTest()

Ha legalább egy fehér korong nincs malomban, akkor a fekete játékos tud levenni korongot.

5.3.8. public void takePuckFalseTest()

Ha az összes fehér korong malomban van, akkor a fekete játékos nem tud korongot levenni.

5.3.9. public void takePuckTrueTest()

Ha legalább egy fehér korong nincs malomban, akkor a fekete játékos tud levenni korongot.

6. Felhasználási útmutató

Egyszerű leírás a program felhasználásáról. A program használatához a Forráskód mappát kell megnyitni projektként egy preferált fejlesztőkörnyezetben, valamint a JUnit-ot hozzá kell adni a classpath-hez. Csak az említett lépések után lehet a programot és a teszteket futattni.

6.1. Főmenü

A program indítása után a főmenü tárul a felhasználó elé. A főmenüben található három gomb: Új játék, Statisztikák és Kilépés. Az alkalmazást a Kilépés vagy az ablak jobb felső sarkában lévő X megnyomásával leállítani. A főmenüben lehet navigálni a gombok között az egérrel, illetve a Tab billentyű lenyomásával, egy új játék indítása után már nem lehet Tab-al navigálni, csak egérrel.

6.2. Új játék

Egy új játék indításakor egy kisebb ablak tárul a felhasználó elé egy üres malomtáblával. A játék első fázisában korongokat a táblahelyeken elhelyezkedő kereteken belüli kattintással lehet elhelyezni. A két játékos egymás után felváltva helyez el korongokat a táblán az imént említett módon, ha már foglalt helyre próbál a felhasználó korongot elhelyezni semmi sem fog történni és újból próbálkozhat a korong elhelyezésével. Amennyiben a korongok elhelyezése közben az egyik fél egy malmot alakít ki, akkor szintén az ellenfél egyik korongjára kattintva leveheti azt, a felhasználó következmény nélkül kattinthat olyan táblahelyekre, amikben nincs korong, a saját korongjai vannak, ez ellenfél malomban lévő korongja van, ha az ellenfél összes korongja malomban van és a másik játékos malmot hozott létre, nem tud levenni az ellenfél korongjai közül és a következő lépés már az ellenfél fogja megtenni. Miután megtörtént a korongok elhelyezése, a mozgatás fázis következik. Egy korongot úgy lehet mozgatni, hogy az első kattintással kijelöljük a mozgatni kívánt korongot és a második kattintással azt a táblahelyet ahová szeretnénk mozgatni a kiválasztott korongot. Amennyiben rossz (olyan táblahely, amelyről az aktuális játékos nem tud érvényesen sehova sem lépni vagy a kijelölt koronggal nem lehet a célpontba lépni) kezdő vagy célpontot, akkor a játékos újra próbálkozhat. Minden sikeres korong mozgatás végleges nincs esély a mozgatás visszavételére. Fel szeretném hívni a **figyelmet**, arra, hogy egy korong mozgatását két kattintás viszi véghez, amiknek számít a sorrendje, kérem figyeljenek oda ennek a helyes használatára. Az említett lépések ismétlésével lehet a játékot lejátszani. Amennyiben a játék véget ért, automatikusan visszakerül a felhasználó a főmenüben, ahol az Új játék gomb felett megjelenik a győztes. Játék közben is van esély a főmenübe való visszatéréshez, a Vissza a főmenübe gomb lenyomásával, ebben az esetben a játék állása nem mentődik, az Új játék megnyomásával egy teljes új játékot kezd el a felhasználó.

6.3. Statisztikák

Ennek a gombnak a megnyomásával, megjelennek az eddig lejátszott játékok statisztikái, a következőket tartalmazza: az összes lejátszott játékok száma, fehér által nyert játékok száma és ezek hány százalékát képezik az összes játéknak, a feketéről ugyan ez, valamint a döntetlen játszmák száma. A Visszalépés a főmenübe gombbal, ahogy a neve is sugallja, vissza lehet térni a főmenübe.