Eötvös Loránd Tudományegyetem
Informatikai Kar
Komputeralgebra Tanszék

Valószínűséget használó játékok

Burcsi Péter
docens
Komputeralgebra Tanszék

Bujtás Ferenc
nappali tagozat
programtervező informatikus szak

Budapest, 2018

Tartalom

Bevezetés **2**
Felhasználói dokumentáció **3**
Feladat leírása 3
Penney's Game leírása 3
Kígyók és Létrák leírása 4
Ki Nevet a Végén leírása 4
Futási környezet 6
A program telepítése/indítása 6
Főmenü 6
Penney's Game – Beállítások ablak 7
Penney's Game – Műveletek ablak 9
Penney's Game – Példa 10
Kígyók és Létrák – Beállítások ablak 12
Kígyók és Létrák – Műveletek ablak 13
Kígyók és Létrák – Példa 15
Ki Nevet a Végén – Beállítások ablak 16
Ki Nevet a Végén – Műveletek ablak 18
Ki Nevet a Végén – Példa 20
Hibalehetőségek 21
Fejlesztői dokumentáció **23**
Feladat leírása 23
Futási környezet 23
Megoldás 23
Programfelépítés 23
Algoritmus részlet 24
Matematikai háttér 26
Osztályok 28
MainMenu Form 28
PenneyOptions Form 29
PenneyMain Form 32
SnLOptions Form 36
SnLMain Form 39
KNaVOptions Form 43
KNaVMain Form 45
Penney Osztály 50
SnakesAndLadders Osztály 55
KiNevetAVegen Osztály 58
Dice Osztály 62
Graph Osztály 64
Markov Osztály 65
Tesztelés 67
Hiba üzenetek tesztelése 67
Penney's Game tesztek 69
Ki Nevet a Végén tesztek 71
Snakes and Ladders tesztek 74
Use Case teszteset 77
Irodalomjegyzék **80**

Bevezetés

A program célja egy valószínűséget használó játék elkészítése, lejátszása, és elemzése. A játék létrehozásához 3 különböző játékszabály lett megírva a programnak:

Penney's Game - Melyben két játékos játszik egymás ellen, a kockával dobott sorozatok megtippelésében.

Kígyók és Létrák - Melyben egy játékos próbál kockadobásokkal végigérni egy táblán, ahol a mezők között tetszőleges átmeneteket adhatunk meg.

Ki Nevet a Végén egy változata - Két játékos játszik egymás ellen, hogy a figuráikat célba juttassák, ebben a játékban a véletlenen kívül a játékosok döntése is szerepet játszik.

Ezen játékok konkrét paramétereit (tábla nagysága, használt kockák, stb.) meg lehet adni, majd az így létrehozott játékot lehet játszani (számítógép ellen), és valószínűségi szempontból elemezni a programmal.

Felhasználói dokumentáció

Feladat leírása

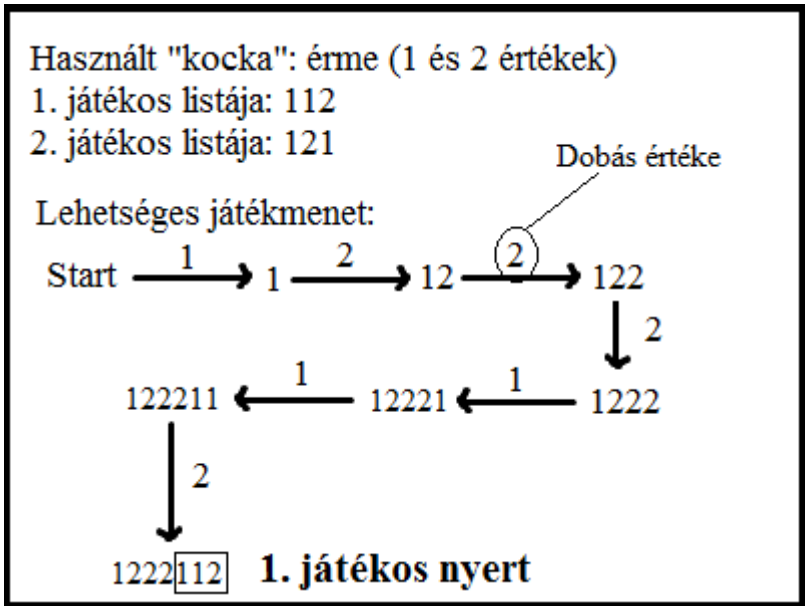
A programban 3 különböző, valószínűséget használó játék játékszabálya alapján hozható létre egy játék. Létrehozása után ez a játék a programmal lejátszható es elemezhető. A játékok ábrázolása irányított gráffal történik, a rajtuk végrehajtható elemzések: Monte Carlo szimuláció és a játékot Markov-láncként kezelve a transzformációs mátrixán elvégzett műveletekkel történő elemzések. (Ha jelenleg egy bizonyos állapotban vagyunk, X lépés után melyik állapotban mennyi eséllyel leszünk? Ha jelenleg egy bizonyos állapotban vagyunk, melyik végállapotban milyen valószínűséggel végzünk? És eközben a többi állapotot várhatóan hányszor érintjük?)

A programmal így játszható/elemezhető játékok leírása következik.

Penney's Game leírása

Ebben a játékban két játékos játszik. A játék dobásainál egy érmét (vagy kockát, vagy akár több kockát) használunk. A játék megkezdése előtt mindkét játékos megad egy listát, amely a lehetséges dobások eredményeinek egy ismétléses kombinációja (nem kötelező, hogy ez a két lista egyenlő hosszúságú legyen).

Példa:



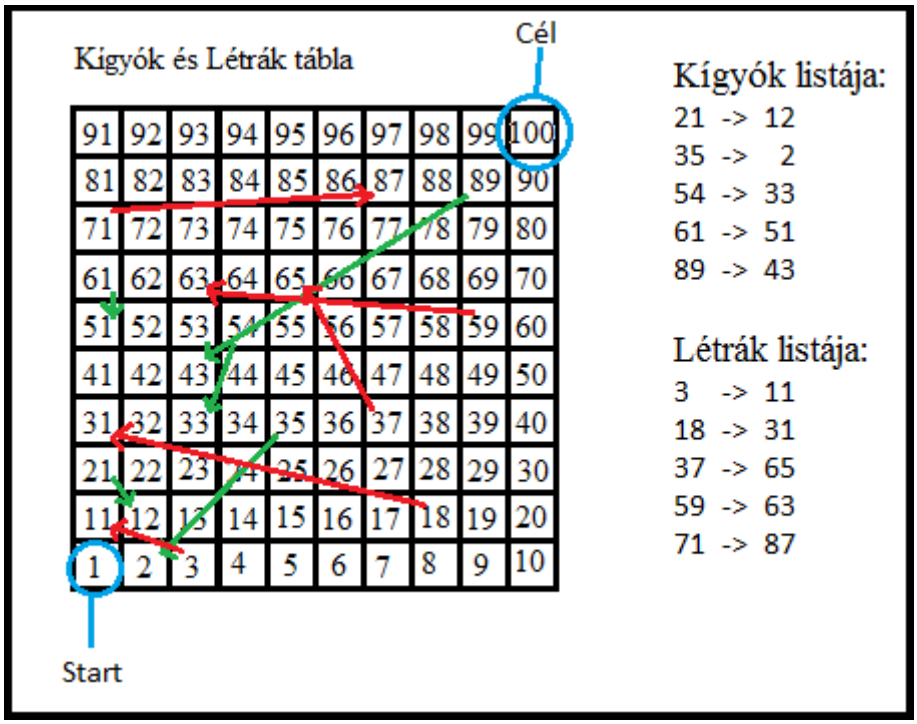
A játék minden lépése egy közös dobásból áll. **Egy n-hosszúságú listával rendelkező játékos akkor nyer, ha az utolsó n dobás eredményeinek a listája megegyezik az általa a játék előtt kiválasztott listával.**

Kígyók és Létrák leírása

Ebben a játékban egy játékos játszik. **Célunk, hogy a tábla elejéről a tábla végéig eljussunk.** Minden lépésnél dobunk egy kockával (vagy akár több kockával), majd a dobott értéknek megfelelő számú mezővel lépünk a táblán előre. Amennyiben a tábla végén túl lépnénk a dobásunkkal, a tábla végébe léphetünk.

A célba jutást nehezítik/könnyítik a táblán elhelyezett “kígyók” és “létrák”. A kígyók és létrák egy kezdő- és végpontból állnak. Amennyiben egy lépés végén a kezdőpozíciójukban vagyunk, a következő lépésünket a végpontjukból fogjuk megkezdeni. Kígyónak hívjuk őket, ha ezzel a tábla végétől távolabb kerültünk, létrának, ha ezzel közelebb kerültünk a tábla végéhez. Tábla elejéről és végéről nem indulhat kígyó és létra, egy mezőről mindig csak egy kígyó és létra indul, és ha indul egy mezőről létra vagy kígyó, akkor az a mező nem lehet végpontja egyik másik kígyónak vagy létrának.

Példa:

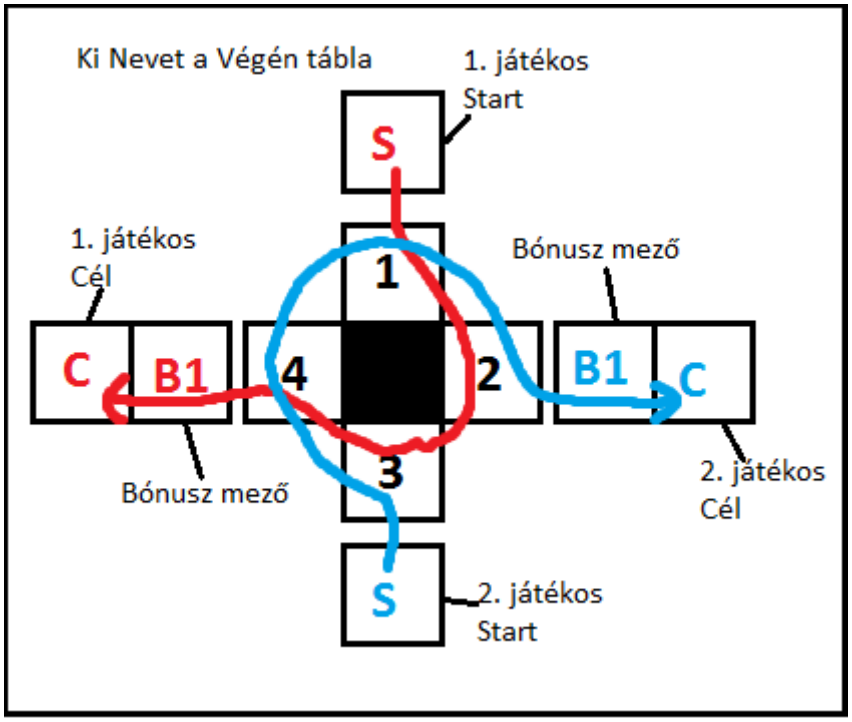


Ki Nevet a Végén leírása

Ebben a játékban két játékos játszik. **Célunk, hogy egy kezdő pozíciójukból az összes figurájukat eljuttassák a célba a másik játékos előtt.** A tábla kör alakú, a két játékos kezdő pozíciója egymással szemben van, minden figurájuk a táblán ugyanabban az

irányban halad. A céljukat a figurák úgy érik el, ha körbeérnek a pályán, majd végighaladnak a saját céljukhoz vezető úton. Amennyiben ezen a célon túllépnénk dobásunkkal, a célba léphetünk.

Példa:



Minden lépésnél a játékosok felváltva dobnak egy kockával (vagy akár több kockával), majd a dobott értéknek megfelelő számú mezővel lépnek a táblán előre egy általuk választott figurával. Azonban bizonyos szabályokat a lépéseinknek be kell tartani.

- Egy mezőn nem lehet egyszerre több figurája egy játékosnak (ez alól kivétel a kezdő és cél pozíció)
- Kezdő pozíciót csak akkor hagyhatjuk el figuránkkal, ha a dobásunkkal a lehető legnagyobb értéket dobtuk ki
- Amennyiben olyan mezőre lépnénk, ahol az ellenfél egy figurája áll, az ellenfél figurája visszakerül a kezdő pozíciójába

Taktikák:

A játék során az ellenfél (és az elemzések során a játékos) lépéseit előre megadott taktikákkal adjuk meg. A programnak 4 alapértelmezett taktika lett megírva. Ezek:

- **Taktika 1:** Megpróbál mindig a lehető legelőrébb lévő figurával lépni.
- **Taktika 2:** Megpróbál mindig a lehető leghátrébb lévő figurával lépni.

- **Taktika 3:** Mint Taktika 1, de prioritása van azoknak a lépéseknek, amivel az ellenfél egy figuráját ütjük.
- **Taktika 4:** Mint Taktika 2, de prioritása van azoknak a lépéseknek, amivel az ellenfél egy figuráját ütjük.

Futási környezet

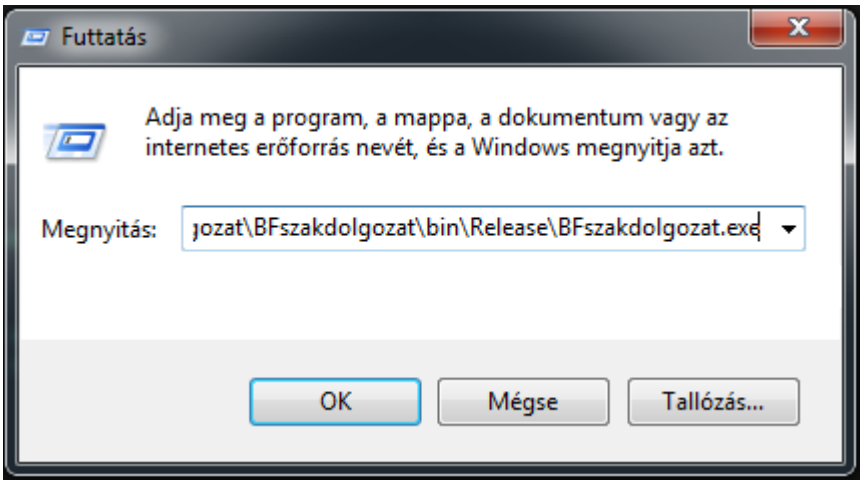
IBM PC, exe futtatására alkalmas operációs rendszer (pl. Windows XP).

A program telepítése\indítása

A program nem igényel exe fájlon kívül más fájlokat / telepítési folyamatot. Az exe file

(„A program mappájának az elérési útvonala”)\BFszakdolgozat\bin\Release\BFszakdolgozat.exe

néven található meg. A Start menü Futtatás menüpontjában a fenti fájl kiválasztásával (beírásával) indítható.

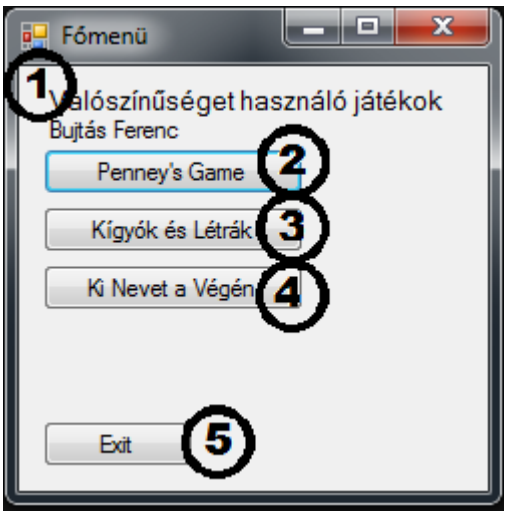


Főmenü

Leírás:

Ez az ablak nyílik meg a program elindításakor, itt található a program címe, innen lehet a választott játéknak megfelelő új ablakot megnyitni, és innen léphetünk ki a programból.

Ablak kinézete:



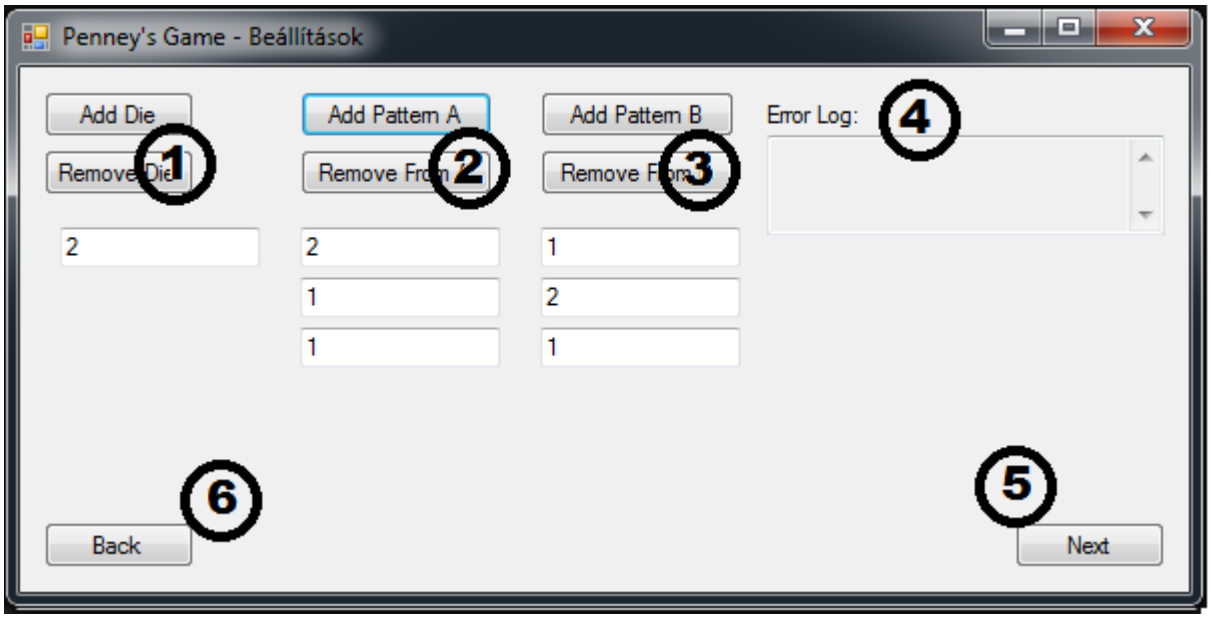
1) Az ablak bal felső sarkában található a program (és szakdolgozat címe: „**Valószínűséget használó játékok**” és a szerző neve: „**Bujtás Ferenc**”)
2) A **„Penney’s Game”** gomb megnyitja a **Penney’s Game – Beállítások ablak**ot, ahol megadhatjuk egy elemezni kívánt Penney’s Game játék adatait.
3) A **„Kígyók és Létrák”** gomb megnyitja a **Kígyók és Létrák – Beállítások ablak**ot, ahol megadhatjuk egy elemezni kívánt Kígyók és Létrák játék adatait.
4) A **„Ki Nevet a Végén”** gomb megnyitja a **Ki Nevet a Végén – Beállítások ablak**ot, ahol megadhatjuk egy elemezni kívánt Ki Nevet a Végén játék adatait.
5) Az **„Exit”** gombbal tudunk a programból kilépni.

Penney’s Game – Beállítások ablak

Leírás:

Ebben az ablakban adhatók meg a játszani/szimulálni/elemezni kívánt „Penney’s game” játéknak a tulajdonságai: játékhoz használt kockák és az játékosok által használt listák.

Ablak kinézete:



1) Az „**Add Die**” és a „**Remove Die**” feliratú gombok használhatók kockák hozzáadására és elvételére. Az így a gombok alatt megjelenő dobozok mindegyikét ki kell tölteni egy számmal, ami a használt dobókocka oldalszáma. – *Emlékeztető: Ahhoz, hogy egy figurát kihozzunk a kezdőpozícióból, az adott lépésben történt dobásunknak maximálisnak kellett lennie. (1 db 6-oldalú dobókocka esetén, ha 6-ot dobtunk)*
2) Az „**Add Pattern A**” és a „**Remove From A**” feliratú gombok használhatók az 1. (A) játékos által használt lista megadására. Az így a gombok alatt megjelenő dobozok mindegyikét ki kell tölteni egy számmal, ami a kiválasztott lehetséges dobásérték. – *Emlékeztető: A lista alulról felfelé olvasandó! (Tehát ha alulról felfelé lista 1,2 a lista, akkor az esetben nyerünk, ha az utolsó előtti dobás 1 volt, az utolsó dobás pedig 2)*
3) Az „**Add Pattern B**” és a „**Remove From B**” feliratú gombok használhatók az 2. (B) játékos által használt lista megadására. Az így a gombok alatt megjelenő dobozok mindegyikét ki kell tölteni egy számmal, ami a kiválasztott lehetséges dobásérték. – *Emlékeztető: A lista alulról felfelé olvasandó! (Tehát ha alulról felfelé lista 1,2 a lista, akkor az esetben nyerünk, ha az utolsó előtti dobás 1 volt, az utolsó dobás pedig 2)*
4) Az „**Error Log**” alatt található mezőbe írja ki a program a hibaüzeneteit.
5) Miután beállítottuk az adatokat, a **„Next” gombra** kattintva érhetjük el a **Penney’s Game - Műveletek** ablakot, ahol a játék/szimuláció/elemezés megtörténik. Amennyiben a megadott tulajdonságok nem helyesek, a program jelzi a hibát és nem lép tovább.

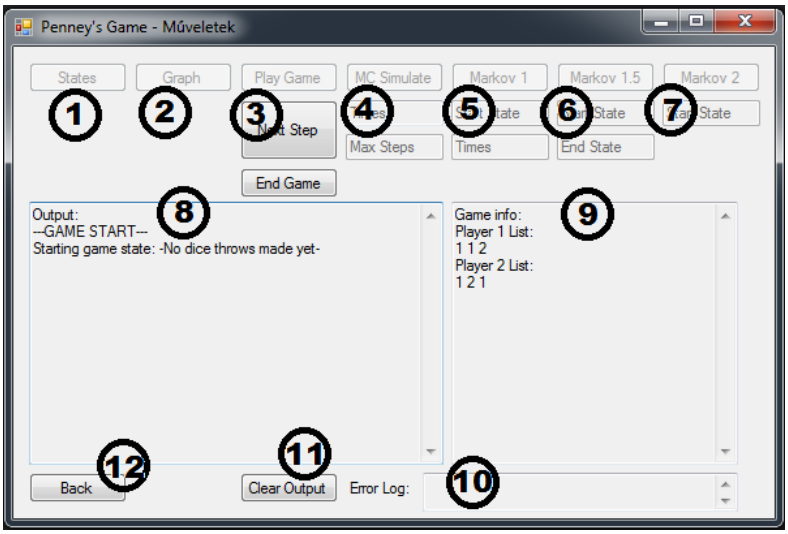
6) A **„Back" gomb**bal a **Főmenü ablak**ra tudunk visszatérni, de a program ebben az esetben az itt elvégzett módosításokat nem menti.

Penney's Game – Műveletek ablak

Leírás:

Ebben az ablakban végezhető a korábban megadott „Penney's game" játék játszása/szimulálása/elemzése. Az egyes műveletek a hozzájuk tartozó gombok lenyomásával futtathatók, amennyiben paraméterek szükségesek a műveltekhez, azokat a gombok alatti szövegdobozok helyes kitöltésével lehet megadni. Az eredmények a bal oldali (Output) dobozban jelennek meg, a játék adatai a jobb oldali (Game info) dobozban találhatók. A jobb alul található (Error Log) dobozba íródnak a program hibaüzenetei.

Ablak kinézete:



1) A **„States" gomb**bal megjeleníthető a játék összes elérhető játékállapota.
2) A **„Graph" gomb**bal megjeleníthető az ezen állapotok közötti átmenetek listája.
3) A **„Play" gomb**bal a játék lejátszását kezdhetjük meg, ehhez a „**Next Step**" és „**End Game**", gombokat megjeleníti, és amíg a játék tart, nem tudunk más műveletet elindítani.
4) Az **„MC Simulate" gomb**bal Monte Carlo szimuláció végezhető. A program a „**Times**" szövegdobozban megadott számú alkalommal lejátssza a játékot

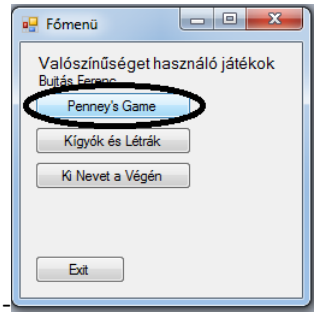
(Abbahagyva a játékot, ha „**Max Steps**” lépés alatt nem fejeződik be), majd a szimulációk eredményét visszaadja.

5) A **„Markov 1” gomb**bal megkapható, hogy a „**Start State**” számú játékállapotból indulva „**Times**” lépés elteltével mennyi az eséllyel leszünk az egyes játékállapotokban.
6) A **„Markov 1.5” gomb**bal megkapható, hogy a „**Start State**” számú játékállapotból indulva átlagosan hányszor fogunk az „**End State**” (***nem végállapot!***) játékállapoton áthaladni, amíg eljutunk egy végállapotba.
7) A **„Markov 2” gomb**bal megkapható, hogy a „**Start State**” számú játékállapotból indulva a játék végén mennyi az eséllyel leszünk az egyes végállapotokban.
8) Az „**Output:**” szövegdobozba írja ki a program a játék/szimuláció/elemzés eredményeit.
9) A „**Game Info:**” szövegdobozba írja ki a program az ablak megnyitásakor annak a játéknak a tulajdonságait, amin a műveleteinket végezzük.
10) Az „**Error Log**” alatt található mezőbe írja ki a program a hibaüzeneteit.
11) A **„Clear Output” gomb**bal törölhető az „**Output**” szövegdoboz jelenlegi tartalma.
12) A **„Back” gomb**bal a **Penney’s Game - Beállítások ablak**ra tudunk visszatérni, de a program ebben az esetben az itt elvégzett módosításokat nem menti.

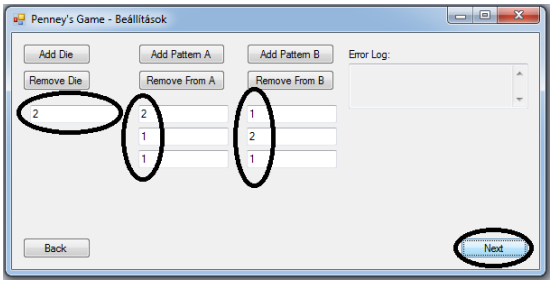
Penney’s Game – Példa

A Kígyók és Létrák leírásánál példaként bemutatott Penney’s Game játékról tudni szeretnénk, hogy pontosan mennyi eséllyel nyer melyik játékos.

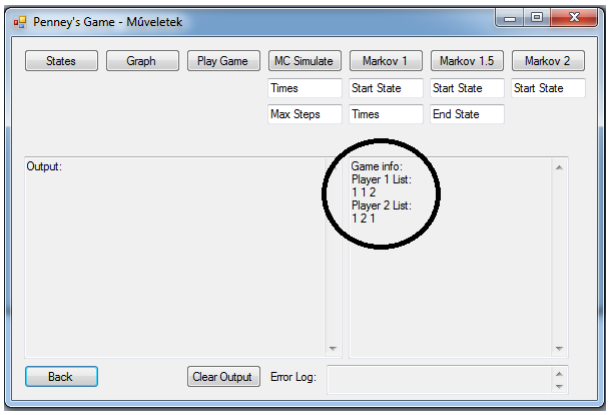
0. Elindítjuk a programot.
1. Kiválasztjuk a főmenüben a Penney’s Game játékot.



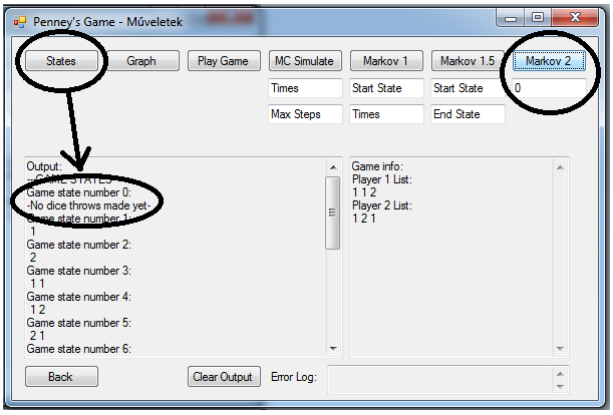
2. Megadjuk a játék paramétereit, majd a „Next” gombra kattintunk.



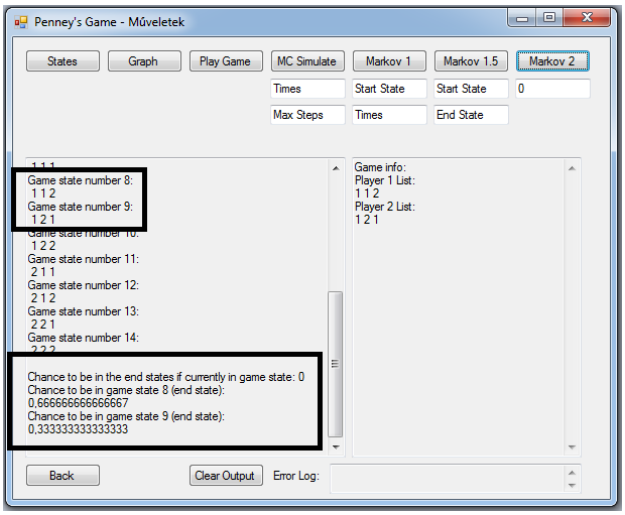
3. Ellenőrizzük a „Game info” dobozban, hogy a megfelelő adatokat adtunk meg.



4. Megadjuk a „Markov 2” elemzéshez szükséges paramétert (milyen eséllyel jutunk el az végállapotokba, ha a megadott állapotban vagyunk) – ehhez segítségnek használjuk a „States” gombot a kezdőállapot számának (0) megtalálására. Amikor ez megvan, a „Markov 2” gombra kattintunk.



5. Az „Output” dobozból leolvashatjuk az eredményt.

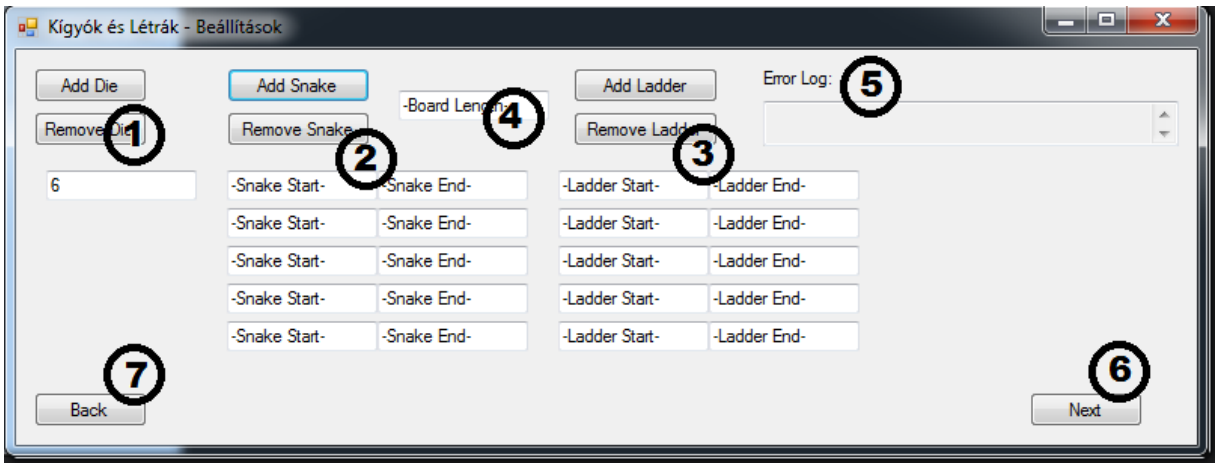


Kígyók és Létrák – Beállítások ablak

Leírás:

Ebben az ablakban adhatók meg a játszani/szimulálni/elemezni kívánt „Kígyók és létrák” játéknak a tulajdonságai: tábla mérete, játékhoz használt kockák és a táblán elhelyezkedő kígyók és létrák.

Ablak kinézete:



1) Az „**Add Die**” és a „**Remove Die**” feliratú gombok használhatók kockák hozzáadására és elvételére. Az így a gombok alatt megjelenő dobozok mindegyikét ki kell tölteni egy számmal, ami a használt dobókocka oldalszáma.
2) Az „**Add Snake**” és a „**Remove Snake**” feliratú gombok használhatók a „kígyók” megadására. Az így a gombok alatt megjelenő doboz párok mindegyikét ki kell tölteni egy számmal, ami az egyek kígyók kezdő- és vég pontjait jelölik. – *<u>Emlékeztető</u>: Az első és utolsó mezőkből nem indulhat kígyó vagy létra.*

Továbbá, ha egy mezőből csak egy kígyó vagy létra indulhat, és ha indul egy kígyó vagy létra a mezőből, akkor az a mező nem lehet más kígyó vagy létra végpontja.

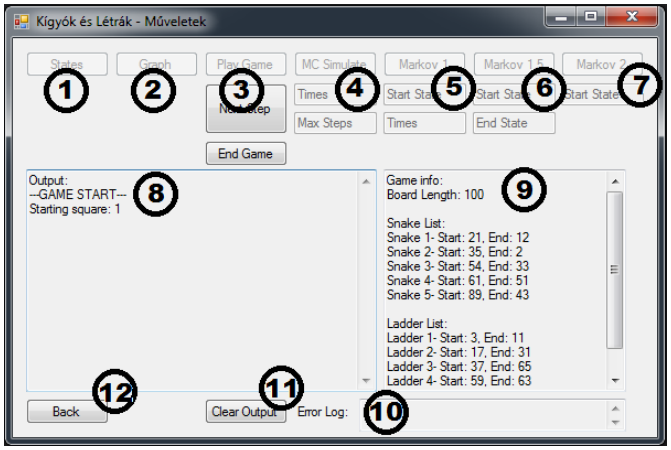
3) Az „**Add Ladder**” és a „**Remove Ladder**” feliratú gombok használhatók a „létrák” megadására. Az így a gombok alatt megjelenő doboz párok mindegyikét ki kell tölteni egy számmal, ami az egyek létrák kezdő- és vég pontjait jelölik. – *Emlékeztető: Az első és utolsó mezőkből nem indulhat kígyó vagy létra. Továbbá, ha egy mezőből csak egy kígyó vagy létra indulhat, és ha indul egy kígyó vagy létra a mezőből, akkor az a mező nem lehet más kígyó vagy létra végpontja.*
4) A kezdetben „**-Board Length-**” feliratú **szövegdoboz**ba adjuk meg hány mezőből áll a tábla, aminek az 1. mezőjéről indulunk.
5) Az „**Error Log**” alatt található mezőbe írja ki a program a hibaüzeneteit.
6) Miután beállítottuk az adatokat, a „**Next**” **gomb**ra kattintva érhetjük el a **Kígyók és Létrák - Műveletek ablak**ot, ahol a játék/szimuláció/elemezés megtörténik. Amennyiben a megadott tulajdonságok nem helyesek, a program jelzi a hibát és nem lép tovább.
7) A „**Back**” **gomb**bal a **Főmenü ablak**ra tudunk visszatérni, de a program ebben az esetben az itt elvégzett módosításokat nem menti.

Kígyók és Létrák – Műveletek ablak

Leírás:

Ebben az ablakban végezhető a korábban megadott „Kígyók és létrák”játék játszása/szimulálása/elemzése. Az egyes műveletek a hozzájuk tartozó gombok lenyomásával futtathatók, amennyiben paraméterek szükségesek a műveltekhez, azokat a gombok alatti szövegdobozok helyes kitöltésével lehet megadni. Az eredmények a bal oldali (Output) dobozban jelennek meg, a játék adatai a jobb oldali (Game info) dobozban találhatók. A jobb alul található (Error Log) dobozba íródnak a program hibaüzenetei.

Ablak kinézete:



1) A **„States" gomb**bal megjeleníthető a játék összes elérhető játékállapota.

2) A **„Graph" gomb**bal megjeleníthető az ezen állapotok közötti átmenetek listája.

3) A **„Play" gomb**bal a játék lejátszását kezdhetjük meg, ehhez a „**Next Step**" és „**End Game**", gombokat megjeleníti, és amíg a játék tart, nem tudunk más műveletet elindítani.

4) Az **„MC Simulate" gomb**bal Monte Carlo szimuláció végezhető. A program a „**Times**" szövegdobozban megadott számú alkalommal lejátssza a játékot (Abbahagyva a játékot, ha „**Max Steps**" lépés alatt nem fejeződik be), majd a szimulációk eredményét visszaadja.

5) A **„Markov 1" gomb**bal megkapható, hogy a „**Start State**" számú játékállapotból indulva „**Times**" lépés elteltével mennyi az eséllyel leszünk az egyes játékállapotokban.

6) A **„Markov 1.5" gomb**bal megkapható, hogy a „**Start State**" számú játékállapotból indulva átlagosan hányszor fogunk az „**End State**" (***nem végállapot!***) játékállapoton áthaladni, amíg eljutunk egy végállapotba.

7) A **„Markov 2" gomb**bal megkapható, hogy a „**Start State**" számú játékállapotból indulva a játék végén mennyi az eséllyel leszünk az egyes végállapotokban.

8) Az „**Output:**" szövegdobozba írja ki a program a játék/szimuláció/elemzés eredményeit.

9) A „**Game Info:**” szövegdobozba írja ki a program az ablak megnyitásakor annak a játéknak a tulajdonságait, amin a műveleteinket végezzük.

10) Az „**Error Log**” alatt található mezőbe írja ki a program a hibaüzeneteit.

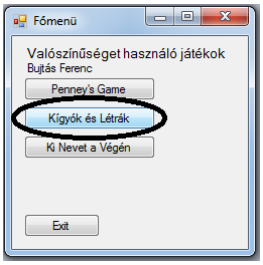
11) A **„Clear Output” gomb**bal törölhető az „**Output**” szövegdoboz jelenlegi tartalma.

12) A „Back” gombbal a **Kígyók és Létrák - Beállítások ablak**ra tudunk visszatérni, de a program ebben az esetben az itt elvégzett módosításokat nem menti.

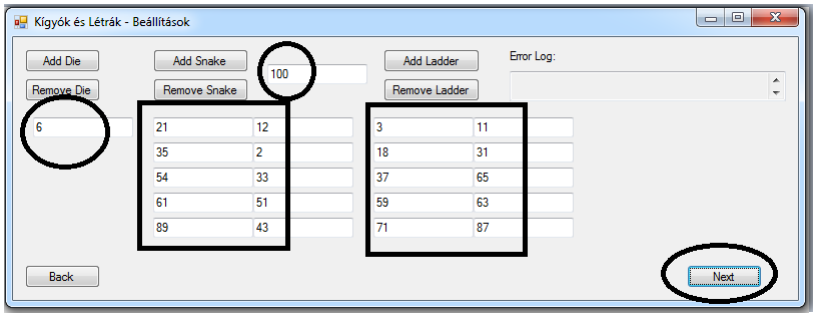
Kígyók és Létrák – Példa

A Kígyók és Létrák leírásánál példaként bemutatott Kígyók és Létrák játékot le szeretnénk játszani.

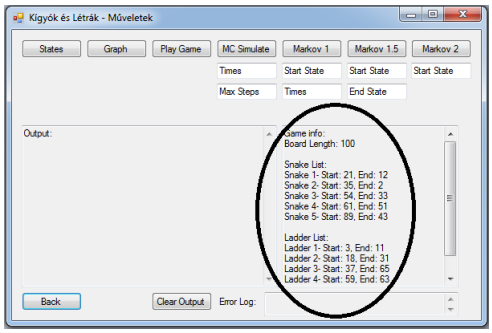
0. Elindítjuk a programot.
1. Kiválasztjuk a főmenüben a Kígyók és Létrák játékot.



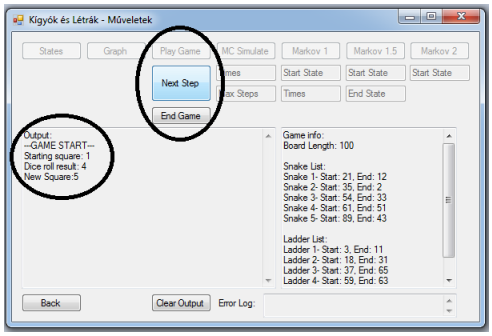
2. Megadjuk a játék paramétereit, majd a „Next” gombra kattintunk.



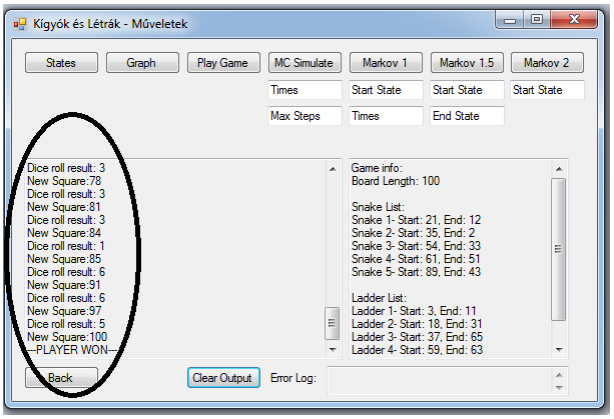
3. Ellenőrizzük a „Game info” dobozban, hogy a megfelelő adatokat adtunk meg.



4. A „Play Game” gomb megnyomásával elkezdjük a játék lejátszását. Minden új lépéshez (dobáshoz) a „Next Step” gombot nyomjuk meg. A játék menetét az „Output” dobozban tudjuk követni. Amennyiben félbe akarjuk szakítani a játékot, azt az „End Game” gombbal tehetjük meg.



5. Amikor a játék elérte a végét, a program automatikusan újra elrejti a „Next Step” és az „End Game” gombot, és engedélyezi a többi gomb használatát.



Ki Nevet a Végén – Beállítások ablak

Leírás:

Ebben az ablakban adhatók meg a játszani/szimulálni/elemezni kívánt kétszemélyes „Ki nevet a végén?” játéknak a tulajdonságai: tábla mérete és egyéb tulajdonságai,

játékosok bábuinak száma, játékhoz használt kockák, melyik játékos kezd és az ellenfél által használt stratégia.

Ablak kinézete:

1) Az „**Add Die**” és a „**Remove Die**” feliratú gombok használhatók kockák hozzáadására és elvételére. Az így a gombok alatt megjelenő dobozok mindegyikét ki kell tölteni egy számmal, ami a használt dobókocka oldalszáma. – *<u>Emlékeztető</u>: Ahhoz, hogy egy figurát kihozzunk a kezdőpozícióból, az adott lépésben történt dobásunknak maximálisnak kellett lennie. (1 db 6-oldalú dobókocka esetén, ha 6-ot dobtunk)*
2) A kezdetben „**-Board Length-**” feliratú **szövegdoboz**ba adjuk meg hány lépésből áll a kör alakú tábla. (Mindenképpen páros számot adjunk meg, különben nem lehetne a két játékos számára szimmetrikus!)
3) A kezdetben „**-Bonus Steps-**” feliratú **szövegdoboz**ba adjuk meg hány lépést kell tenni egy bábunak a célig, miután körbe ért a táblán. Ezek a lépések egy, a táblától elvezető úton történnek, itt már az ellenfél nem tudja a figurát leütni.
4) A kezdetben „**-Figures A-**” feliratú **szövegdoboz**ba adjuk meg hány figurával rendelkezik az 1. játékos (játszás esetén a felhasználó).
5) A kezdetben „**-Figures B-**” feliratú **szövegdoboz**ba adjuk meg hány figurával rendelkezik az 1. játékos (játszás esetén az ellenfél által irányított ellenfél).
6) A „**Player Starts**” melletti dobozt pipáljuk ki, ha azt szeretnénk, hogy az 1. játékos kezdjen, ellenkező esetben hagyjuk üresen.

7) Az ablak jobb felső sarkában található leugró ablakkal választhatjuk ki a **2. játékos által használt taktikát**. *(Taktikák leírása a Ki Nevet a Végén játék leírásánál található)*

8) Az „**Error Log**” mellett található mezőbe írja ki a program a hibaüzeneteit.

9) Miután beállítottuk az adatokat, a „**Next**” **gomb**ra kattintva érhetjük el a **Ki Nevet a Végén - Műveletek ablak**ot, ahol a játék/szimuláció/elemezés megtörténik. Amennyiben a megadott tulajdonságok nem helyesek, a program jelzi a hibát és nem lép tovább.

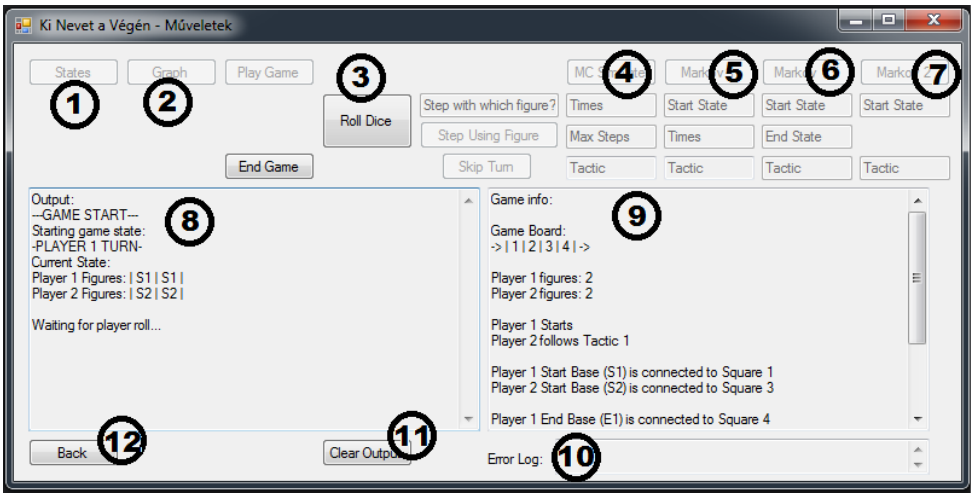
10) A „**Back**” **gomb**bal a **Főmenü ablak**ra tudunk visszatérni, de a program ebben az esetben az itt elvégzett módosításokat nem menti.

Ki Nevet a Végén – Műveletek ablak

Leírás:

Ebben az ablakban végezhető a korábban megadott „Ki nevet a végén?” játék játszása/szimulálása/elemzése. Az egyes műveletek a hozzájuk tartozó gombok lenyomásával futtathatók, amennyiben paraméterek szükségesek a műveltekhez, azokat a gombok alatti szövegdobozok helyes kitöltésével lehet megadni. Az eredmények a bal oldali (Output) dobozban jelennek meg, a játék adatai a jobb oldali (Game info) dobozban találhatók. A jobb alul található (Error Log) dobozba íródnak a program hibaüzenetei.

Ablak kinézete:



1) A **„States” gomb**bal megjeleníthető a játék összes elérhető játékállapota.

2) A **„Graph" gomb**bal megjeleníthető az ezen állapotok közötti átmenetek listája.

3) A **„Play" gomb**bal a játék lejátszását kezdhetjük meg, ehhez a „**Roll Dice**", „**Step Using Figure**", „**Skip Turn**", „**End Game**", gombokat és a „**Step using which figure?**" szövegdobozt megjeleníti, és amíg a játék tart, nem tudunk más műveletet elindítani.

4) Az **„MC Simulate" gomb**bal Monte Carlo szimuláció végezhető. A program a „**Times**" szövegdobozban megadott számú alkalommal lejátssza a játékot (Abbahagyva a játékot, ha „**Max Steps**" lépés alatt nem fejeződik be), majd a szimulációk eredményét visszaadja. Mivel a felhasználó (1. játékos) nem lépteti a figurákat, ezért a „**Tactic**" mezőben adjuk meg, melyik taktikát használja az 1. játékos (1,2,3,4 - *Taktikák leírása a Ki Nevet a Végén játék leírásánál találhat*)

5) A **„Markov 1" gomb**bal megkapható, hogy a „**Start State**" számú játékállapotból indulva „**Times**" lépés elteltével mennyi az eséllyel leszünk az egyes játékállapotokban. Mivel a felhasználó (1. játékos) nem lépteti a figurákat, ezért a „**Tactic**" mezőben adjuk meg, melyik taktikát használja az 1. játékos (1,2,3,4 - *Taktikák leírása a Ki Nevet a Végén játék leírásánál találhat*)

6) A **„Markov 1.5" gomb**bal megkapható, hogy a „**Start State**" számú játékállapotból indulva átlagosan hányszor fogunk az „**End State**" (***nem végállapot!***) játékállapoton áthaladni, amíg eljutunk egy végállapotba. Mivel a felhasználó (1. játékos) nem lépteti a figurákat, ezért a „**Tactic**" mezőben adjuk meg, melyik taktikát használja az 1. játékos (1,2,3,4 - *Taktikák leírása a Ki Nevet a Végén játék leírásánál találhat*)

7) A **„Markov 2" gomb**bal megkapható, hogy a „**Start State**" számú játékállapotból indulva a játék végén mennyi az eséllyel leszünk az egyes végállapotokban. Mivel a felhasználó (1. játékos) nem lépteti a figurákat, ezért a „**Tactic**" mezőben adjuk meg, melyik taktikát használja az 1. játékos (1,2,3,4 - *Taktikák leírása a Ki Nevet a Végén játék leírásánál találhat*)

8) Az „**Output:**" szövegdobozba írja ki a program a játék/szimuláció/elemzés eredményeit.

9) A „**Game Info:**" szövegdobozba írja ki a program az ablak megnyitásakor annak a játéknak a tulajdonságait, amin a műveleteinket végezzük.

10) Az „**Error Log**” alatt található mezőbe írja ki a program a hibaüzeneteit.

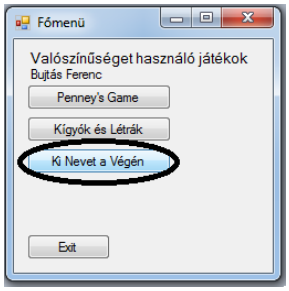
11) A **„Clear Output” gomb**bal törölhető az „**Output**” szövegdoboz jelenlegi tartalma.

12) A **„Back”** gombbal a **Ki Nevet a Végén - Beállítások ablak**ra tudunk visszatérni, de a program ebben az esetben az itt elvégzett módosításokat nem menti.

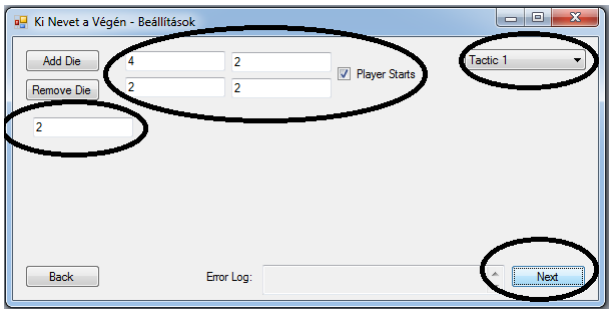
Ki Nevet a Végén – Példa

A Ki Nevet a Végén leírásánál példaként bemutatott Ki Nevet a Végén játékon szeretnénk Monte Carlo szimulációt folytatni.

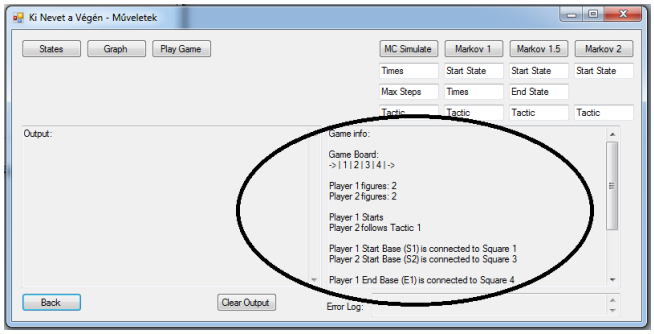
0. Elindítjuk a programot.
1. Kiválasztjuk a főmenüben a Ki Nevet a Végén játékot.



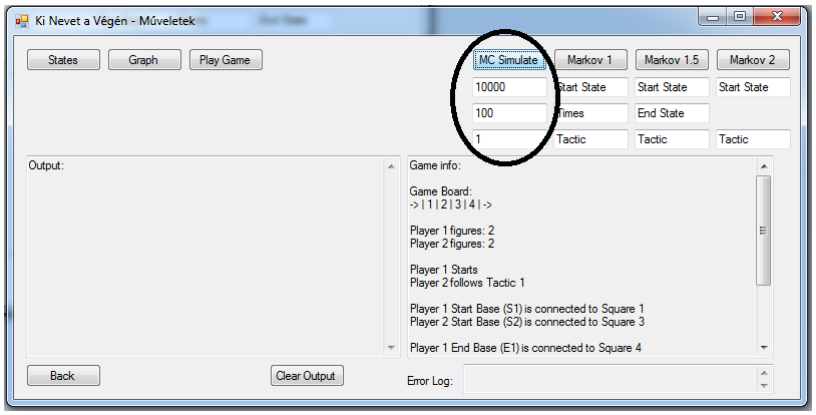
2. Megadjuk a játék paramétereit, majd a „Next” gombra kattintunk.



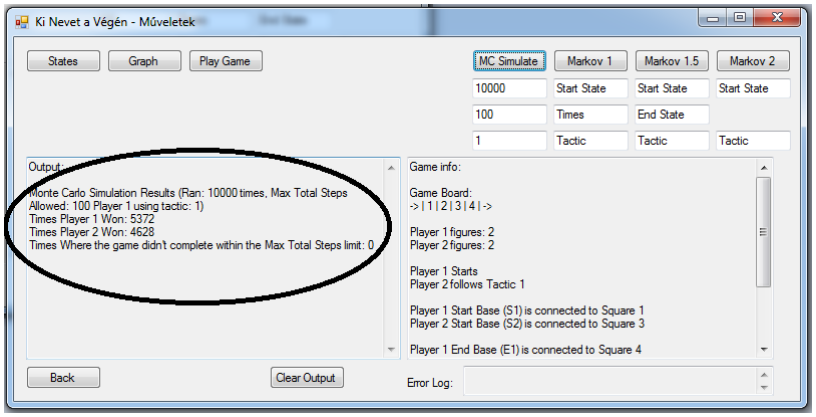
3. Ellenőrizzük a „Game info” dobozban, hogy a megfelelő adatokat adtunk meg.



4. Megadjuk a Monte Carlo szimulációhoz szükséges paramétereket. (10000-szer futtatjuk, 100 maximális lépésből állhat egy játék, az 1.játékos is az 1. számú taktikát alkalmazza) Amikor ez megvan, a „MC Simulate” gombra kattintunk.



5. Az „Output” dobozból leolvashatjuk az eredményt.



Hibalehetőségek

Hibaüzenettel járó hibák

A játékok és a műveletek paramétereire vannak megkötések. Amennyiben úgy próbálunk a programmal műveltet végrehajtani, hogy az általunk adott paraméterek ezeknek nem felelnek meg, akkor a kívánt művelet nem hajtódik végre, és a program az aktuális ablak hibaüzeneteket tartalmazó szövegdobozába írja bele a hiba leírását.

Hibaüzenettel nem járó hibák

Nem várt eredményhez vezethet a játék tulajdonságainak, vagy a műveletek paramétereinek rossz megadása – olvassa át a dokumentáció megfelelő részét, hogy megbizonyosodjon róla, hogy az adatokat helyesen adta meg!

Program működését akadályozó hibák

Mivel a program által használt memória véges, ezért ha a megadott paramétereink miatt a program több memóriát igényelne, mint amennyi a rendelkezésére áll, a program működése megakad. A programot ilyenkor újra kell indítani, és kisebb méretű paraméterekkel újra lehet próbálkozni. *(Megjegyzés: A játékok irányított gráfként tárolása miatt ez a hiba a vártnál kisebb értékekre is előfordulhat, különösen a Ki Nevet a Végén játéknál, a sok különböző eltárolandó játékállapot miatt.)*

Fejlesztői dokumentáció

Feladat leírása

A feladat egy olyan szoftver megvalósítása volt, amellyel három, valószínűséget használó játék (Penney's Game, Kígyók és Létrák, Ki Nevet a Végén) adható meg, játszható és elemezhető. A játékok irányított gráfként vannak ábrázolva és a játék játszása ezen a gráfon egy kezdőcsúcsból kiinduló, végcsúcsok egyikében végződő útnak felel meg. A Monte Carlo szimuláción kívül további elemzések valósulhatnak meg, amennyiben a játékot Markov láncként fogjuk fel. (*Részletek a Matematikai háttér részben*)

Futási környezet

IBM PC, exe futtatására alkalmas operációs rendszer.

C# nyelv, .NET framework `v4.0.30319`, Microsoft Visual Studio 2015 fejlesztői környezet.

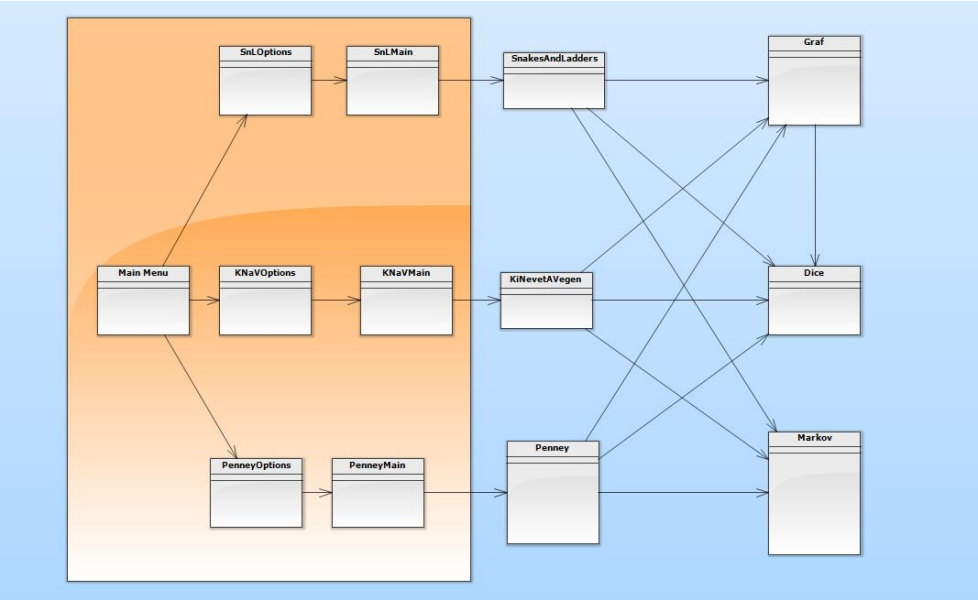
Megoldás

Ahogy azt a feladat leírásánál is említve volt, a program által kezelt játékok irányított gráfként vannak tárolva. Ezen a gráfon már a külön játékok szabájaitól függetlenül, hasonló módon tudunk műveleteket és elemzéseket végezni. (Eltérés ettől a Ki Nevet a Végén játszása – ilyenkor a felhasználónak van döntése abban, hogy egy lépésnél melyik bábuval próbál lépni.) Tehát a feladat egyrész abból áll, hogy az egyes játékokat irányított gráffá alakítsuk, másrészt pedig az ezen a gráfon végrehajtandó műveleteket kell implementálni.

Programfelépítés

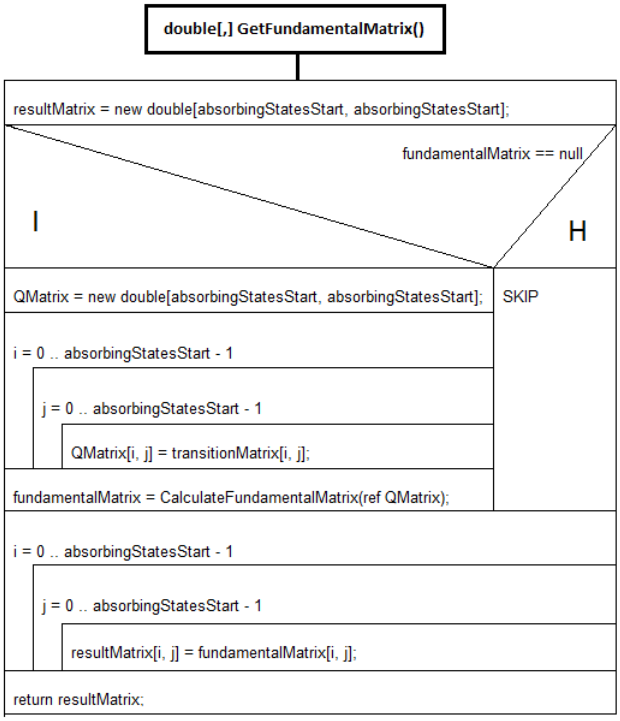
A program összesen 7 Windows Form-ból és további 6 osztályból áll: a **MainMenu** form-ról indul a program futása, játékonként 2-2 form (**Options** és **Main – Options**-ban adja meg a felhasználó a játék adatait, és **Main**-ben végezhet rajta műveleteket). Minden játékhoz ezen kívül tartozik egy azt leíró osztály (**SnakesAndLadders, KiNevetAVegen, Penney**) , aminek műveleteivel a játék és az elemzések megvalósíthatók. Ezekben a műveletekben segít a **Dice** osztály (amivel a kockadobások

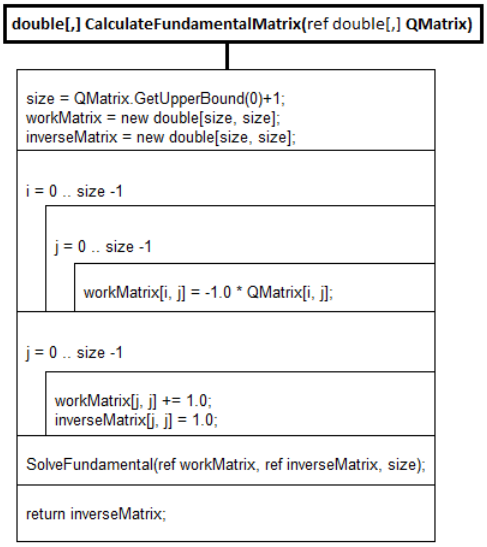
szimulálhatók), a **Graph** osztály (amivel a gráfon lehet lépéseket tenni), és a Markov osztály. (amivel a Markov lánc tranzíciós mátrixát lehet elemezni)

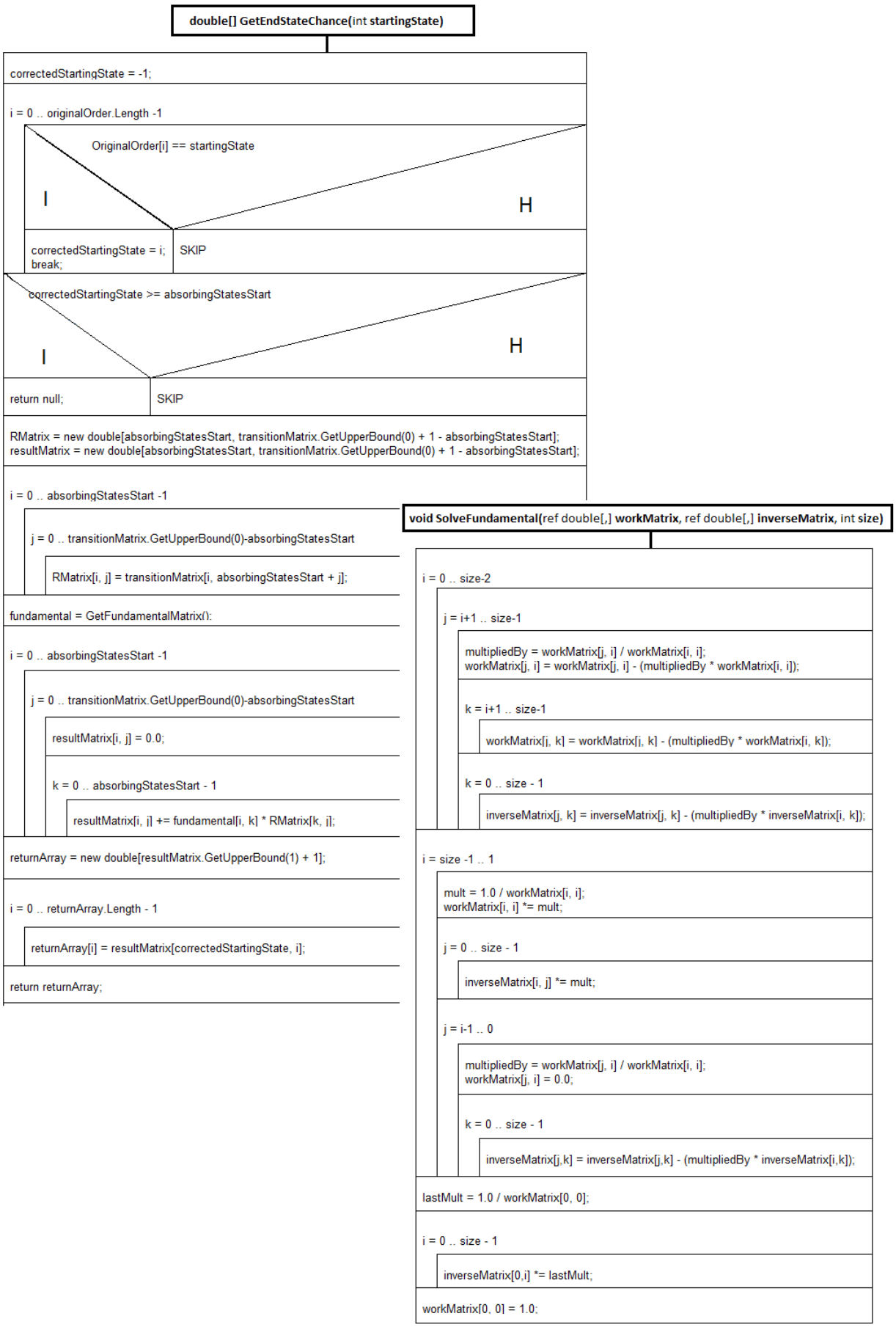


Algoritmus részlet

Ahogy az a Matematikai háttér részben írva, a **Markov** osztály felelős a játéknak megfelelő Markov lánc fundamentális mátrixának kiszámolásáért. A releváns függvények struktogramjai:






double[] GetEndStateChance(int startingState)
correctedStartingState = -1;
i = 0 .. originalOrder.Length -1
OriginalOrder[i] == startingState
I
H
correctedStartingState = i;
break;
SKIP
correctedStartingState >= absorbingStatesStart
I
H
return null;
SKIP
RMatrix = new double[absorbingStatesStart, transitionMatrix.GetUpperBound(0) + 1 - absorbingStatesStart];
resultMatrix = new double[absorbingStatesStart, transitionMatrix.GetUpperBound(0) + 1 - absorbingStatesStart];
i = 0 .. absorbingStatesStart -1
j = 0 .. transitionMatrix.GetUpperBound(0)-absorbingStatesStart
RMatrix[i, j] = transitionMatrix[i, absorbingStatesStart + j];
fundamental = GetFundamentalMatrix():
i = 0 .. absorbingStatesStart -1
j = 0 .. transitionMatrix.GetUpperBound(0)-absorbingStatesStart
resultMatrix[i, j] = 0.0;
k = 0 .. absorbingStatesStart - 1
resultMatrix[i, j] += fundamental[i, k] * RMatrix[k, j];
returnArray = new double[resultMatrix.GetUpperBound(1) + 1];
i = 0 .. returnArray.Length - 1
returnArray[i] = resultMatrix[correctedStartingState, i];
return returnArray;
void SolveFundamental(ref double[,] workMatrix, ref double[,] inverseMatrix, int size)
i = 0 .. size-2
j = i+1 .. size-1
multipliedBy = workMatrix[j, i] / workMatrix[i, i];
workMatrix[j, i] = workMatrix[j, i] - (multipliedBy * workMatrix[i, i]);
k = i+1 .. size-1
workMatrix[i, k] = workMatrix[i, k] - (multipliedBy * workMatrix[i, k]);
k = 0 .. size - 1
inverseMatrix[j, k] = inverseMatrix[j, k] - (multipliedBy * inverseMatrix[i, k]);
i = size -1 .. 1
mult = 1.0 / workMatrix[i, i];
workMatrix[i, i] *= mult;
j = 0 .. size - 1
inverseMatrix[i, j] *= mult;
j = i-1 .. 0
multipliedBy = workMatrix[j, i] / workMatrix[i, i];
workMatrix[j, i] = 0.0;
k = 0 .. size - 1
inverseMatrix[j,k] = inverseMatrix[j,k] - (multipliedBy * inverseMatrix[i,k]);
lastMult = 1.0 / workMatrix[0, 0];
i = 0 .. size - 1
inverseMatrix[0,i] *= lastMult;
workMatrix[0, 0] = 1.0;

Matematikai háttér

A program lényege, hogy miután a különböző játékokat irányított gráfként ábrázoltuk, ugyanolyan módon lehet rajtuk bejárásokat és elemzéseket végrehajtani. (Kivétel ez alól a Ki Nevet a Végén játszása, ugyanis a felhasználónak ekkor van döntése – melyik figrával lép.) Arról, hogy hogyan alakítjuk gráffá az egyes játékokat, az osztályok leírásánál olvashat. Ebben a részben a program által elvégzett műveleteket tárgyaljuk.

Játszás és Monte Carlo szimuáció

Ezeket együtt lehet kezelni, ugyanis (Ki Nevet a Végén kivételével) a Monte Carlo szimuláció lényegében nem különbözik a Játszás többszöri lefolytatásával. Az irányított gráfunk úgy lett elkészítve, hogy minden csúcsból vagy 0 (ha végállapot), vagy pontosan az összes különböző dobás-összértéknek megfelelő számú él vezet ki a megfelelő csúcsokhoz, a szükséges dobás szerint növekvő sorrendben. Egy lépésnél, miután dobtunk a kockákkal, könnyű megtalálni, hányas számú élen kell továbblépni. Startcsúcs meg van adva, végcsúcsba pedig akkor értünk, ha az adott csúcshoz nem tartozik él.

Markov láncok

A Markov láncok olyan diszkrét sztochasztikus folyamatok, amikre teljesül minden állapot és t=0,1,2.. mellett, hogy: $\mathbf{P(X_{t+1}=i_{t+1} \mid X_t=i_t, X_{t-1}=i_{t-1}, \dots, X_1=i_1, X_0=i_0) = P(X_{t+1}=i_{t+1} \mid X_t=i_t)}$

Azaz, a t+1. időpillanatban felvett állapot csak a t. időpillanatban felvett állapottól függ, nem függ a t-nél a korábban felvett állapotoktól.[1]

Belátható, hogy ez a feladat által kezelt 3 játékra teljesül. Az, hogy mi lesz a következő játékállapot egy bizonyos kockadobás esetén nem függ a korábbi játékhelyzetektől, csak a jelenlegi játékhelyzettől.

- Penney's game esetén ez a játékhelyzet az utolsó n dobás eredménye. (ahol n a játékosok lista hosszának maximuma)
- Kígyók és létrák esetén ez a játékhelyzet, a mező, amiről a lépést indítjuk.
- Ki Nevet a Végén estén az összes figura elhelyezkedése a táblán.

[1] Forrás: [1] Wayne L. Winston: Operations Research – Applications and Algorithms, 924.oldal

A műveleteinkhez a Markov lánc ún. átmeneti mátrixát (*transition probability matrix*) használjuk. Ha a játékállapotok száma m, akkor ez egy mXm-es mátrix, amiben az i. sor j. eleme tárolja a valószínűséget,hogy i-ből egy lépéssel j-be jutunk.[2] (A Markov osztály konstruktorában ezt a mátrixot rendezzük, hogy a játékot befejező végpozíciók (amik 1 valószínűséggel önmagukba mennek) sorai és oszlopai a mátrix végére kerüljenek.)

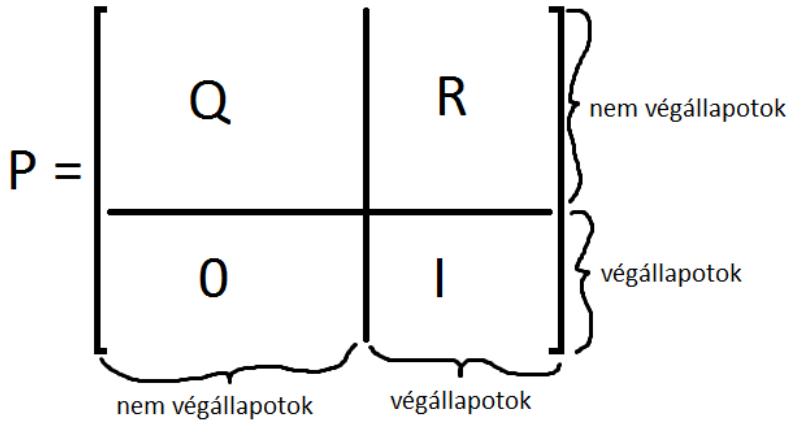
„Markov 1" elemzés

A legegyszerűbben lekódolható elemzés az n számú lépés után elfoglalt játékállapot valószínűsége. Teljes indukcióval belátható, hogy egy **P** mátrixú Markov láncban az esély, hogy i. pozícióból indulva n lépés után j-ben leszünk az egyenlő $\mathbf{P}^n$ i. sorának j. elemével.[3] Ehhez az elemzéshez csak a mátrixszorzás műveletet kell megírni, és azt megfelelő sokszor futtatni.

„Markov 1.5" és „Markov 2" elemzések

- **Markov 1.5 :** Bizonyos állapotból indulva a játék során egy megadott állapoton átlagosan várhatóan hányszor haladunk át, amíg végállapotba jutunk?
- **Markov 2:** Bizonyos állapotból indulva a mennyi az esély, hogy az egyes végállapotokba jutunnk?

Ez a két elemzés összefügg, ugyanis, ha az átmeneti mátrixot (amit korábban rendeztünk) a következő módon ábrázoljuk:



Akkor az $\mathbf{(I-Q)^{-1}}$*(Markov chain's fundamental matrix)* mátrix i. sorának j. eleme pont a Markov 1.5 elemzés megoldása, az $\mathbf{(I-Q)^{-1}R}$ mátrix i. sora pedig pont a Markov 2

[2] Forrás: [1] Wayne L. Winston: Operations Research – Applications and Algorithms, 925.oldal
[3] Forrás: [1] Wayne L. Winston: Operations Research – Applications and Algorithms, 928.oldal

elemzés megoldása.[4] Ennek a kiszámításához kellett átrendezni a mátrix sorait és oszlopait, ezen kívül a kellett függvény, ami kiszámítja (I-Q) inverzét. (A programban ehhez a Gauss-Jordan-eliminációt használtam.)

Osztályok

A program (ahogy az már a Programfelépítés pont alatt lett írva) 6 osztályból és 7 Windows formból áll. Itt következik ezeknek a részletes leírása.

MainMenu Form

Leírás:

A felhasználó ebben az ablakban választhatja ki melyik játékot kívánja játszani/szimulálni/elemezni, ennek megfelelő ablak nyílik meg. A felhasználó szintén innen tud kilépni a programból.

Változók:

- A **MainMenu** Form definiálásakor nem lett új változó megadva.

Elemek és a hozzájuk tartozó függvények:

- **titleLabel** („Valószínűséget használó játékok” szöveget tartalmazó felirat)
- **authorLabel** („Bujtás Ferenc” szöveget tartalmazó felirat)
- **PenneyButton**
    - **PenneyButton_Click** –Létrehozza és megjeleníti a **PenneyOptions** egy példányát, ezt az ablakot pedig elrejti.
- **SnLButton**
    - **SnLButton_Click** – Létrehozza és megjeleníti a **SnLOptions** egy példányát, ezt az ablakot pedig elrejti.
- **KNaVButton**
    - **KNaVButton_Click** – Létrehozza és megjeleníti a **KNaVOptions** egy példányát, ezt az ablakot pedig elrejti.
- **exitButton** (gomb „Exit” felirattal)

[4] Forrás: Wayne L. Winston: Operations Research – Applications and Algorithms, 943-945.oldal

- **exitButton_Click** – Bezárja ezt az ablakot, ezzel a program futását befejezve.

További függvények:

- A **MainMenu** Form definiálásakor nem lett új függvény megadva.

PenneyOptions Form

Leírás:

A felhasználó ebben az ablakban adja meg a **Penney** osztály inicializálásához szükséges adatokat: a játékhoz használt kockákat és az játékosok által használt listákat. Itt történik a **Penney** osztály bemenő adatainak az ellenőrzése, majd ezeket az adatokat (**diceArray, patternA, patternB**) adjuk tovább a **Penney Main** Form-nak, ahol a **Penney** osztály tényleges példányosítása és az azon elvégezhető műveletek történnek.

Változók:

- **originForm** (**MainMenu** egy példánya) – Az a Form amelyről ezt az ablakot elértük, a visszalépésnél ide kerülünk vissza.
- **diceArray** (integer tömb) – A felhasználó által szimulálni kívánt játékban használt kockák.
- **patternA** (integer tömb) – A felhasználó által szimulálni kívánt játékban az 1. (A) játékos által használt lista.
- **patternB** (integer tömb) – A felhasználó által szimulálni kívánt játékban a 2. (B) játékos által használt lista.
- **diceBoxes(**integer**)** – **diceArray** beviteléhez használt szövegmezők száma.
- **diceInputBoxes (**szövegdoboz elemekből álló lista**)** – **diceArray** beviteléhez használt szövegmezők.
- **patternABoxes(**integer**)** – **patternA** beviteléhez használt szövegmezők száma.
- **patternAInputBoxes (**szövegdoboz elemekből álló lista**)** – **patternA** beviteléhez használt szövegmezők.
- **patternBBoxes(**integer**)** –**patternB** beviteléhez használt szövegmezők száma.

- **patternBInputBoxes (**szövegdoboz elemekből álló lista**)** – **patternB** beviteléhez használt szövegmezők.

Konstruktor:

- **PenneyOptions**(MainMenu **origin**) – Inicializálja az osztályt, megadja az értéket az **originForm**-nak.

Elemek és a hozzájuk tartozó függvények:

- **backButton** (gomb „Back” felirattal)
    - **backButton_Click** – Bezárja ezt az ablakot és megjeleníti a korábban elrejtett, **originForm**-ban tárolt ablakot.
- **nextButton** (gomb „Next” felirattal)
    - **nextButton_Click** – Meghívja az **InputIsCorrect** függvényt, és amennyiben az igaz értéket ad vissza, elrejti ezt az ablakot és a megadott játék paraméterekkel létrehoz egy **PenneyMain Form**-ot.
- **addDiceButton** (gomb „Add Die” felirattal)
    - **addDiceButton_Click** –Az **AddNewDice()** függvénnyel létrehoz egy új elemet a **diceInputBoxes** listában.
- **deleteLastDice** (gomb „Remove Die” felirattal)
    - **deleteLastDice_Click** – Amennyiben pozitív a **diceBoxes** változó, csökkenti, és eltávolítja és törli a **diceInputBoxes** lista utolsó elemét.
- **addPatternAElementButton** (gomb „Add Pattern A” felirattal)
    - **addPatternAElementButton_Click** –Az **AddNewElementA ()** függvénnyel létrehoz egy új elemet a **patternAInputBoxes** listában.
- **deletePatternAElementButton** (gomb „Remove From A” felirattal)
    - **deletePatternAElementButton_Click** – Amennyiben pozitív a **patternABoxes** változó, csökkenti, és eltávolítja és törli a **patternAInputBoxes** lista utolsó elemét.
- **addPatternBElementButton** (gomb „Add Pattern B” felirattal)
    - **addPatternBElementButton_Click** –Az **AddNewElementB ()** függvénnyel létrehoz egy új elemet a **patternBInputBoxes** listában.
- **deletePatternBElementButton** (gomb „Remove From B” felirattal)

    - **deletePatternBElementButton_Click** – Amennyiben pozitív a **patternBBoxes** változó, csökkenti, és eltávolítja és törli a **patternBInputBoxes** lista utolsó elemét.
- **errorLabel** („Error Log:” szöveget tartalmazó felirat az **errorBox** felett)
- **errorBox** (Kezdetben üres szövegdoboz, amibe a hibaüzenetek kerülnek a **ShowErrorMessage** metódus futásakor)

További függvények:

- **TextBox AddNewDice()**
    - Létrehoz, és visszaad egy új szövegdobozt, aminek koordinátáit a **diceBoxes** változó segítségével határoz meg, majd növeli a **diceBoxes** változó értékét.
- **TextBox AddNewElementA()**
    - Létrehoz, és visszaad egy új szövegdobozt, aminek koordinátáit a **patternABoxes** változó segítségével határoz meg, majd növeli a **patternABoxes** változó értékét.
- **TextBox AddNewElementB()**
    - Létrehoz, és visszaad egy új szövegdobozt, aminek koordinátáit a **patternBBoxes** változó segítségével határoz meg, majd növeli a **patternBBoxes** változó értékét.
- **bool InputIsCorrect**()
    - Amennyiben helyesek az játéknak a **diceArray, patternA,** és **patternB**-ként megadni kívánt adatok, (amelyeket rend szerint a **diceInputBoxes, patternAInputBoxes,** és **patternBInputBoxes**-ben találhatók,) akkor feltölti ezeket a tömböket a megfelelő adatokkal, és igaz értéket ad vissza.
    - Ellenkező esetben az a **ShowErrorMessage** metódus segítségével jelzi a hibát, majd hamis értéket ad vissza.
- **void ShowErrorMessage**(String **errorDescription**)
    - A megadott hibaüzenetet jeleníti meg az **errorBox** szövegdobozban.

PenneyMain Form

Leírás:

Itt történik a **Penney** osztály tényleges példányosítása és a felhasználó itt hajthatja végre a játék elemzését.

Változók:

- **originForm** (**PenneyOptions** egy példánya) – Az a Form amelyről ezt az ablakot elértük, a visszalépésnél ide kerülünk vissza.
- **currentGame (Penney** egy példánya**) –** A műveletek végrehajtásához a konstruktor paraméterei alapján lérehozott **Penney** osztály példány.
- **patternA (**integer tömb**) –** Az 1. (A) játékos által használt lista.
- **patternB (**integer tömb**) –** Az 2. (B) játékos által használt lista.
- **possibleRolls(**integer**) –** A játékhoz megadott kocka kombinációval dobható különböző összegek száma.

Konstruktor:

- **PenneyMain** (PenneyOptions **origin**, ref int[] **diceArray**, ref int[] **patternA**, ref int[] **patternB**) – Megadja az értéket az **originForm**-nak, **patternA**-nak és **patternB**-nek. A megadott paraméterek alapján inicializálja a **currentGame Penney** játékot. Végül magát a form-ot inicializálja és kitölti az **infoTextBox**-ot.

Elemek és a hozzájuk tartozó függvények:

- **backButton** (gomb „Back” felirattal)
    - **backButton_Click** – Bezárja ezt az ablakot és megjeleníti a korábban elrejtett, **originForm**-ban tárolt ablakot.
- **outputTextBox** (szövegdoboz, kezdetben „Output:” tartalommal, amibe a műveltek eredményeit írjuk ki)
- **infoTextBox** (szövegdoboz, kezdetben „Game info:” tartalommal, amibe a konstruktor beleírja az aktuális játék részleteit)
- **showStatesButton** (gomb „States” felirattal)

    - **showStatesButton_Click** – Kiírja az összes lehetséges játékeset (a **currentGame gameStates** listájának felhasználásával) az **outputTextBox** szövegdobozba.
- **showGraphButton** (gomb „Graph” felirattal)
    - **showGraphButton_Click** – Kiírja az összes lehetséges átmenetet, amivel egy játékesetből egy másikba juthatunk (a **currentGame gameGraph** tömbjének felhasználásával) az **outputTextBox** szövegdobozba.
- **playGameButton** (gomb „Play Game” felirattal)
    - **playGameButton_Click** – Elindítja a játék játszását, a többi művelet gombját lezárja, és megjeleníti a játékhoz használt **stepButton** és **stopButton** gombokat.
- **stepButton** (gomb „Next Step” felirattal)
    - **stepButton_Click** – Játék játszásában tesz egy lépést, és annak részleteit kiírja az outputTextBox-ba.
- **stopButton** (gomb „End Game” felirattal)
    - **stopButton_Click** – Leállítja a játékot az **EndGame** metódus meghívásával.
- **mcSimulateButton** (gomb „MC Simulate” felirattal)
    - **mcSimulateButton_Click** – Meghívja a **SimulateCorrect** függvényt, és amennyiben igaz értéket kap vissza, Monte Carlo szimulációt hajt végre a függvény által értéket kapott paraméterekkel a **currentGame MonteCarloSimulation** függvényével.
- **mcSimulateTimesBox** (szövegdoboz „Times” kezdeti értékkel) – **SimulateIsCorrect** függvény használja
- **mcSimulateMaxStepsBox** (szövegdoboz „Max Steps” kezdeti értékkel) – **SimulateIsCorrect** függvény használja
- **markov1Button** (gomb „Markov 1” felirattal)
    - **markov1Button _Click** – Meghívja a **MarkovCorrect** függvényt, és amennyiben igaz értéket kap vissza, a kapott paraméterekkel a **currentGame Markov1** függvényével elemzi, hogy egy játékállapotból

kiindulva megadott számú lépés után mennyi eséllyel leszünk az egyes játékállapotokban.

- **markov1StartBox** (szövegdoboz „Start State” kezdeti értékkel) – **MarkovCorrect** függvény használja
- **markov1TimesBox** (szövegdoboz „Times” kezdeti értékkel) – **MarkovCorrect** függvény használja
- **markov1_5Button** (gomb „Markov 1.5” felirattal)
    - **markov1_5Button_Click** – Meghívja a **Markov1_5Correct** függvényt, és amennyiben igaz értéket kap vissza, a kapott paraméterekkel a **currentGame Markov1_5** függvényével elemzi, hogy egy játékállapotból kiindulva átlagosan hányszor fogunk áthaladni egy megadott játékállapoton (nem végállapot) amíg a játék befejeződik.
- **markov1_5StartBox** (szövegdoboz „Start State” kezdeti értékkel) – **Markov1_5Correct** függvény használja
- **markov1EndBox** (szövegdoboz „End State” kezdeti értékkel) – **Markov1_5Correct** függvény használja
- **markov2Button** (gomb „Markov 2” felirattal)
    - **markov2Button_Click** – Meghívja a **Markov2Correct** függvényt, és amennyiben igaz értéket kap vissza, a kapott paraméterekkel a **currentGame Markov2** függvényével elemzi, hogy egy játékállapotból kiindulva mennyi eséllyel fejezzük be a játékot az egyes végállapotokban.
- **markov2StartBox** (szövegdoboz „Start State” kezdeti értékkel) – **Markov2Correct** függvény használja
- **clearOutputButton** (gomb „Clear Output” felirattal)
    - **clearOutputButton_Click** – Visszaállítja az **outputTextBox** tartalmát a kezdeti értékére.
- **errorLabel** („Error Log:” szöveget tartalmazó felirat az **errorBox** mellett)
- **errorBox** (Kezdetben üres szövegdoboz, amibe a hibaüzenetek kerülnek a **ShowErrorMessage** metódus futásakor)

További függvények:

- **void GameEndedCheck**(ref int[] **currentState**)
    - Eldönti a jelenlegi játékállapotról, hogy a játék végállapota-e (a **currentGame EndGameCheck** függvényét használva), és amennyiben végállapot, automatikusan kiírja az **outputTextBox**-ba, hogy vége a játéknak és meghívja az **EndGame** metódust.
- **void EndGame**()
    - A játék játszásához használt gombokat (**stepButton, stopButton**) újra elrejti, és a többi művelet gombjait és szövegdobozait újra elérhetővé teszi.
- **bool SimulateCorrect**(ref int **times**, ref int **maxSteps**)
    - A paramétereknek értékül adja **mcSimulateTimesBox** és **mcSimulateMaxStepsBox** tartalmait és igaz értéket ad vissza, amennyiben azok megfelelnek a feladat követelményeinek. (amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)
- **bool MarkovCorrect**(ref int **start**, ref int **times**)
    - A paramétereknek értékül adja **markov1StartBox** és **markov1TimesBox** tartalmait és igaz értéket ad vissza, amennyiben azok megfelelnek a feladat követelményeinek. (amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)
- **bool Markov1_5Correct**(ref int **start**, ref int **end**)
    - A paramétereknek értékül adja **markov1_5StartBox** és **markov1_5EndBox** tartalmait és igaz értéket ad vissza, amennyiben azok megfelelnek a feladat követelményeinek. (amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)
- **bool Markov2Correct**(ref int **start**)
    - A paramétereknek értékül adja **markov2StartBox** tartalmát és igaz értéket ad vissza, amennyiben az megfelel a feladat követelményeinek.

(amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)

- **void ShowErrorMessage**(String **errorDescription**)
    - A megadott hibaüzenetet jeleníti meg az **errorBox** szövegdobozban.

SnLOptions Form

Leírás:

A felhasználó ebben az ablakban adja meg a **SnakesAndLadders** osztály inicializálásához szükséges adatokat: a játékhoz használt kockákat és az játékosok által használt listákat. Itt történik a **SnakesAndLadders** osztály bemenő adatainak az ellenőrzése, majd ezeket az adatokat (**diceArray, boardLength, snakes, ladders**) adjuk tovább a **SnLMain** Form-nak, ahol a **SnakesAndLadders** osztály tényleges példányosítása és az azon elvégezhető műveletek történnek.

Változók:

- **originForm** (**MainMenu** egy példánya) – Az a Form amelyről ezt az ablakot elértük, a visszalépésnél ide kerülünk vissza.
- **diceArray** (integer tömb) – A felhasználó által szimulálni kívánt játékban használt kockák.
- **boardLength** (integer) - A felhasználó által szimulálni kívánt játékban a tábla hossza.
- **snakes** (integer tömböket tartalmazó lista) – A felhasználó által szimulálni kívánt játékban a „kígyók” listája. A tömbök mind kételeműek, előbb a kiinduló, majd a végpont mezőszámát tárolják.
- **ladders** (integer tömböket tartalmazó lista) – A felhasználó által szimulálni kívánt játékban a „létrák” listája. A tömbök mind kételeműek, előbb a kiinduló, majd a végpont mezőszámát tárolják.
- **diceBoxes(**integer**)** – **diceArray** beviteléhez használt szövegmezők száma.
- **diceInputBoxes (**szövegdoboz elemekből álló lista**)** – **diceArray** beviteléhez használt szövegmezők.
- **snakeBoxes(**integer**)** – **snakes** beviteléhez használt szövegmező-párok száma.

- **snakeInputBoxes (**szövegdoboz elemekből álló lista**)** – **patternA** beviteléhez használt szövegmezők.
- **ladderBoxes(**integer**)** –**ladders** beviteléhez használt szövegmező-párok száma.
- **ladderInputBoxes (**szövegdoboz elemekből álló lista**)** – **patternB** beviteléhez használt szövegmezők.

Konstruktor:

- **SnLOptions**(MainMenu **origin**) – Inicializálja az osztályt, megadja az értéket az **originForm**-nak.

Elemek és a hozzájuk tartozó függvények:

- **backButton** (gomb „Back” felirattal)
    - **backButton_Click** – Bezárja ezt az ablakot és megjeleníti a korábban elrejtett, **originForm**-ban tárolt ablakot.
- **nextButton** (gomb „Next” felirattal)
    - **nextButton_Click** – Meghívja az **InputIsCorrect** függvényt, és amennyiben az igaz értéket ad vissza, elrejti ezt az ablakot és a megadott játék paraméterekkel létrehoz egy **SnLMain Form**-ot.
- **addDiceButton** (gomb „Add Die” felirattal)
    - **addDiceButton_Click** –Az **AddNewDice()** függvénnyel létrehoz egy új elemet a **diceInputBoxes** listában.
- **deleteLastDice** (gomb „Remove Die” felirattal)
    - **deleteLastDice_Click** – Amennyiben pozitív a **diceBoxes** változó, csökkenti, és eltávolítja és törli a **diceInputBoxes** lista utolsó elemét.
- **addSnakeButton** (gomb „Add Snake” felirattal)
    - **addSnakeButton_Click** –Az **AddNewSnakeStartBox ()** és **AddNewSnakeEndBox()** függvényekkel létrehoz két új elemet a **snakeInputBoxes** listában.
- **deleteLastSnakeButton** (gomb „Remove Snake” felirattal)
    - **deleteLastSnakeButton_Click** – Amennyiben pozitív a **snakeBoxes** változó, csökkenti, és eltávolítja és törli a **snakeInputBoxes** lista utolsó két elemét.

- **addLadderButton** (gomb „Add Ladder” felirattal)
    - **addLadderButton_Click** – Az **AddNewLadderStartBox ()** és **AddNewLadderEndBox()** függvényekkel létrehoz két új elemet a **ladderInputBoxes** listában.
- **deleteLastLadderButton** (gomb „Remove Ladder” felirattal)
    - **deleteLastLadderButton_Click** – Amennyiben pozitív a **ladderBoxes** változó, csökkentti, és eltávolítja és törli a **ladderInputBoxes** lista utolsó két elemét.
- **boardLengthBox** (szövegdoboz „-Board Length-” kezdeti értékkel)
- **errorLabel** („Error Log:” szöveget tartalmazó felirat az **errorBox** felett)
- **errorBox** (Kezdetben üres szövegdoboz, amibe a hibaüzenetek kerülnek a **ShowErrorMessage** metódus futásakor)

További függvények:

- **TextBox AddNewDice()**
    - Létrehoz, és visszaad egy új szövegdobozt, aminek koordinátáit a **diceBoxes** változó segítségével határoz meg, majd növeli a **diceBoxes** változó értékét.
- **TextBox AddNewSnakeStartBox ()**
    - Létrehoz, és visszaad egy új szövegdobozt, aminek koordinátáit a **snakeBoxes** változó segítségével határoz meg.
- **TextBox AddNewSnakeEndBox ()**
    - Létrehoz, és visszaad egy új szövegdobozt, aminek koordinátáit a **snakeBoxes** változó segítségével határoz meg, majd növeli a **snakeBoxes** változó értékét.
- **TextBox AddNewLadderStartBox ()**
    - Létrehoz, és visszaad egy új szövegdobozt, aminek koordinátáit a **ladderBoxes** változó segítségével határoz meg.
- **TextBox AddNewLadderEndBox ()**
    - Létrehoz, és visszaad egy új szövegdobozt, aminek koordinátáit a **ladderBoxes** változó segítségével határoz meg, majd növeli a **ladderBoxes** változó értékét.

- **bool InputIsCorrect**()
    - Amennyiben helyesek az játéknak a **diceArray, boardLength, snakes,** és **ladders**-ként megadni kívánt adatok, (amelyeket rend szerint a **diceInputBoxes, boardLengthBox, snakeInputBoxes,** és **ladderInputBoxes**-ben találhatók,) akkor feltölti ezeket a tömböket a megfelelő adatokkal, és igaz értéket ad vissza.
    - Ellenkező esetben az a **ShowErrorMessage** metódus segítségével jelzi a hibát, majd hamis értéket ad vissza.
- **void ShowErrorMessage**(String **errorDescription**)
    - A megadott hibaüzenetet jeleníti meg az **errorBox** szövegdobozban.

SnLMain Form

Leírás:

Itt történik a **SnakesAndLadders** osztály tényleges példányosítása és a felhasználó itt hajthatja végre a játék elemzését.

Változók:

- **originForm** (**SnLOptions** egy példánya) – Az a Form amelyről ezt az ablakot elértük, a visszalépésnél ide kerülünk vissza.
- **currentGame (SnakesAndLadders** egy példánya**) –** A műveletek végrehajtásához a konstruktor paraméterei alapján lérehozott **SnakesAndLadders** osztály példány.

Konstruktor:

- **SnLMain**(SnLOptions **origin**, ref int[] **diceArray**, int **boardLength**, ref List<int[]> **ladders**, ref List<int[]> **snakes**) – Inicializálja az osztályt, megadja az értéket az **originForm**-nak, és a paraméterek alapján inicializálja a **currentGame SnakesAndLadders** játékot. Végül magát a form-ot inicializálja és kitölti az **infoTextBox**-ot.

Elemek és a hozzájuk tartozó függvények:

- **backButton** (gomb „Back” felirattal)

    - **backButton_Click** – Bezárja ezt az ablakot és megjeleníti a korábban elrejtett, **originForm**-ban tárolt ablakot.
- **outputTextBox** (szövegdoboz, kezdetben „Output:” tartalommal, amibe a műveltek eredményeit írjuk ki)
- **infoTextBox** (szövegdoboz, kezdetben „Game info:” tartalommal, amibe a konstruktor beleírja az aktuális játék részleteit)
- **showStatesButton** (gomb „States” felirattal)
    - **showStatesButton_Click** – Kiírja az összes lehetséges játékeset (a **currentGame gameStates** listájának felhasználásával) az **outputTextBox** szövegdobozba.
- **showGraphButton** (gomb „Graph” felirattal)
    - **showGraphButton_Click** – Kiírja az összes lehetséges átmenetet, amivel egy játékesetből egy másikba juthatunk (a **currentGame gameGraph** tömbjének felhasználásával) az **outputTextBox** szövegdobozba.
- **playGameButton** (gomb „Play Game” felirattal)
    - **playGameButton_Click** – Elindítja a játék játszását, a többi művelet gombját lezárja, és megjeleníti a játékhoz használt **stepButton** és **stopButton** gombokat.
- **stepButton** (gomb „Next Step” felirattal)
    - **stepButton_Click** – Játék játszásában tesz egy lépést, és annak részleteit kiírja az outputTextBox-ba.
- **stopButton** (gomb „End Game” felirattal)
    - **stopButton_Click** – Leállítja a játékot az **EndGame** metódus meghívásával.
- **mcSimulateButton** (gomb „MC Simulate” felirattal)
    - **mcSimulateButton_Click** – Meghívja a **SimulateCorrect** függvényt, és amennyiben igaz értéket kap vissza, Monte Carlo szimulációt hajt végre a függvény által értéket kapott paraméterekkel a **currentGame MonteCarloSimulation** függvényével.
- **mcSimulateTimesBox** (szövegdoboz „Times” kezdeti értékkel) – **SimulateIsCorrect** függvény használja

- **mcSimulateMaxStepsBox** (szövegdoboz „Max Steps” kezdeti értékkel) – **SimulateIsCorrect** függvény használja
- **markov1Button** (gomb „Markov 1” felirattal)
    - **markov1Button_Click** – Meghívja a **MarkovCorrect** függvényt, és amennyiben igaz értéket kap vissza, a kapott paraméterekkel a **currentGame Markov1** függvényével elemzi, hogy egy játékállapotból kiindulva megadott számú lépés után mennyi eséllyel leszünk az egyes játékállapotokban.
- **markov1StartBox** (szövegdoboz „Start State” kezdeti értékkel) – **MarkovCorrect** függvény használja
- **markov1TimesBox** (szövegdoboz „Times” kezdeti értékkel) – **MarkovCorrect** függvény használja
- **markov1_5Button** (gomb „Markov 1.5” felirattal)
    - **markov1_5Button_Click** – Meghívja a **Markov1_5Correct** függvényt, és amennyiben igaz értéket kap vissza, a kapott paraméterekkel a **currentGame Markov1_5** függvényével elemzi, hogy egy játékállapotból kiindulva átlagosan hányszor fogunk áthaladni egy megadott játékállapoton (nem végállapot) amíg a játék befejeződik.
- **markov1_5StartBox** (szövegdoboz „Start State” kezdeti értékkel) – **Markov1_5Correct** függvény használja
- **markov1EndBox** (szövegdoboz „End State” kezdeti értékkel) – **Markov1_5Correct** függvény használja
- **markov2Button** (gomb „Markov 2” felirattal)
    - **markov2Button_Click** – Meghívja a **Markov2Correct** függvényt, és amennyiben igaz értéket kap vissza, a kapott paraméterekkel a **currentGame Markov2** függvényével elemzi, hogy egy játékállapotból kiindulva mennyi eséllyel fejezzük be a játékot az egyes végállapotokban.
- **markov2StartBox** (szövegdoboz „Start State” kezdeti értékkel) – **Markov2Correct** függvény használja
- **clearOutputButton** (gomb „Clear Output” felirattal)

- **clearOutputButton_Click** – Visszaállítja az **outputTextBox** tartalmát a kezdeti értékére.

- **errorLabel** („Error Log:” szöveget tartalmazó felirat az **errorBox** mellett)
- **errorBox** (Kezdetben üres szövegdoboz, amibe a hibaüzenetek kerülnek a **ShowErrorMessage** metódus futásakor)

További függvények:

- **void GameEndedCheck**(ref int[] **currentState**)
  - Eldönti a jelenlegi játékállapotról, hogy a játék végállapota-e (a **currentGame EndGameCheck** függvényét használva), és amennyiben végállapot, automatikusan kiírja az **outputTextBox**-ba, hogy vége a játéknak és meghívja az **EndGame** metódust.
- **void EndGame**()
  - A játék játszásához használt gombokat (**stepButton, stopButton**) újra elrejti, és a többi művelet gombjait és szövegdobozait újra elérhetővé teszi.
- **bool SimulateCorrect**(ref int **times**, ref int **maxSteps**)
  - A paramétereknek értékül adja **mcSimulateTimesBox** és **mcSimulateMaxStepsBox** tartalmait és igaz értéket ad vissza, amennyiben azok megfelelnek a feladat követelményeinek. (amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)
- **bool MarkovCorrect**(ref int **start**, ref int **times**)
  - A paramétereknek értékül adja **markov1StartBox** és **markov1TimesBox** tartalmait és igaz értéket ad vissza, amennyiben azok megfelelnek a feladat követelményeinek. (amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)
- **bool Markov1_5Correct**(ref int **start**, ref int **end**)
  - A paramétereknek értékül adja **markov1_5StartBox** és **markov1_5EndBox** tartalmait és igaz értéket ad vissza, amennyiben azok megfelelnek a feladat követelményeinek. (amennyiben nem,

hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)

- **bool Markov2Correct**(ref int **start**)
    - A paramétereknek értékül adja **markov2StartBox** tartalmát és igaz értéket ad vissza, amennyiben az megfelel a feladat követelményeinek. (amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)
- **void ShowErrorMessage**(String **errorDescription**)
    - A megadott hibaüzenetet jeleníti meg az **errorBox** szövegdobozban.

KNaVOptions Form

Leírás:

A felhasználó ebben az ablakban adja meg a **KiNevetAVegen** osztály inicializálásához szükséges adatokat: a játékhoz használt kockákat: tábla mérete, egyéb tulajdonságai, a játékosok figuráinak számát, ki kezd, és mi a 2. játékos által használt taktika. Itt történik a **KiNevetAVegen** osztály bemenő adatainak az ellenőrzése, ezeket az adatokat adjuk tovább a **KNaVMain** Form-nak, ahol a **KiNevetAVegen** osztály tényleges példányosítása és az azon elvégezhető műveletek történnek.

Változók:

- **originForm** (**MainMenu** egy példánya) – Az a Form amelyről ezt az ablakot elértük, a visszalépésnél ide kerülünk vissza.
- **boardLength(**integer**)** – A körpálya hossza (pozitív páros szám).
- **bonusSteps(**integer**)** – A körút megtétele után hátralévő lépések száma.
- **figuresA(**integer**)** – Az 1. (A) játékos figuráinak száma.
- **figuresB(**integer**)** – A 2. (B) játékos figuráinak száma.
- **diceArray** (integer tömb) – A felhasználó által szimulálni kívánt játékban használt kockák.
- **playerStarts(**boolean**)** – Igaz, ha az 1. (A) játékos kezd, Hamis, ha a 2. (B) játékos kezd.

- **inputEnemyTactic(**integer**)** – A 2. (B) játékos által használt taktika számkódja. (1,2, vagy 3)
- **diceBoxes(**integer**)** – **diceArray** beviteléhez használt szövegmezők száma.
- **diceInputBoxes (**szövegdoboz elemekből álló lista**)** – **diceArray** beviteléhez használt szövegmezők.

Konstruktor:

- **KNaVOptions**(MainMenu **origin**) – Inicializálja az osztályt, megadja az értéket az **originForm**-nak.

Elemek és a hozzájuk tartozó függvények:

- **backButton** (gomb „Back” felirattal)
    - **backButton_Click** – Bezárja ezt az ablakot és megjeleníti a korábban elrejtett, **originForm**-ban tárolt ablakot.
- **nextButton** (gomb „Next” felirattal)
    - **nextButton_Click** – Meghívja az **InputIsCorrect** függvényt, és amennyiben az igaz értéket ad vissza, elrejti ezt az ablakot és a megadott játék paraméterekkel létrehoz egy **KNaVMain Form**-ot.
- **addDiceButton** (gomb „Add Die” felirattal)
    - **addDiceButton_Click** –Az **AddNewDice()** függvénnyel létrehoz egy új elemet a **diceInputBoxes** listában.
- **deleteLastDice** (gomb „Remove Die” felirattal)
    - **deleteLastDice_Click** – Amennyiben pozitív a **diceBoxes** változó, csökkenti, és eltávolítja és törli a **diceInputBoxes** lista utolsó elemét.
- **boardLengthBox** (szövegdoboz „-Board Length-” kezdeti értékkel)
- **bonusStepsBox** (szövegdoboz „-Bonus Steps-” kezdeti értékkel)
- **figuresABox** (szövegdoboz „-Figures A-” kezdeti értékkel)
- **figuresBBox** (szövegdoboz „-Figures B-” kezdeti értékkel)
- **playerStartsCheckBox** („Player Starts” feliratú check box, alapértelmezetten hamis értékkel)
- **enemyTacticComboBox** („**Tactic 1**”, „**Tactic 2**”, „**Tactic 3**”, „**Tactic 4**” lehetséges értékekkel rendelkező combo box, alapértelmezetten „**Tactic 1**” értékkel)

- **errorLabel** („Error Log:” szöveget tartalmazó felirat az **errorBox** mellett)
- **errorBox** (Kezdetben üres szövegdoboz, amibe a hibaüzenetek kerülnek a **ShowErrorMessage** metódus futásakor)

További függvények:

- **TextBox AddNewDice()**
    - Létrehoz és visszaad egy új szövegdobozt, aminek koordinátáit a **diceBoxes** változó segítségével határoz meg, majd növeli a **diceBoxes** változó értékét.
- **bool InputIsCorrect**()
    - Amennyiben helyesek az játéknak a **boardLength**, **bonusSteps**, **figuresA**, **figuresB**, **diceArray**, **playerStarts**, és **inputEnemyTactic**-ként megadni kívánt adatok, (amelyeket rend szerint a **boardLengthBox, bonusStepsBox, figuresABox, figuresBBox, diceInputBoxes, playerStartsCheckBox,** és **enemyTacticComboBox**-ban találhatók,) akkor feltölti ezeket az elemeket a megfelelő adatokkal, és igaz értéket ad vissza.
    - Ellenkező esetben az a **ShowErrorMessage** metódus segítségével jelzi a hibát, majd hamis értéket ad vissza.
- **void ShowErrorMessage**(String **errorDescription**)
    - A megadott hibaüzenetet jeleníti meg az **errorBox** szövegdobozban.

KNaVMain Form

Leírás:

Itt történik a **KiNevetAVegen** osztály tényleges példányosítása és a felhasználó itt hajthatja végre a játék elemzését.

Változók:

- **originForm** (**KNaVOptions** egy példánya) – Az a Form amelyről ezt az ablakot elértük, a visszalépésnél ide kerülünk vissza.
- **figuresA (**integer**) –** Az 1. (A) játékos figuráinak száma.
- **figuresB (**integer**) –** Az 2. (B) játékos figuráinak száma.

- **boardLength (**integer**) –** A körpálya hossza (pozitív páros szám).
- **bonusSteps(**integer**) –** A körút megtétele után hátralévő lépések száma.
- **playerStarts(**boolean**) –** Igaz, ha az 1. (A) játékos kezd, Hamis, ha a 2. (B) játékos kezd.
- **currentRollResult(**integer**) –** A játék játszása során használt változó, amiben minden dobása után a felhasználó (1. játékos) által dobott eredményt tároljuk.

Konstruktor:

- **KNaVMain**(KNaVOptions **origin**, int **boardLength**, int **bonusSteps**, int **figuresA**, int **figuresB**, ref int[] **diceArray**, bool **playerStarts**, int **inputEnemyTactic**) – Megadja az értéket az **originForm**-nak, **figuresA**-nak, **figuresB**-nek, **boardLength**-nek, **bonusStepts**-nek, és a **playerStarts**-nak. A megadott paraméterek alapján inicializálja a **currentGame KiNevetAVegen** játékot. Végül magát a form-ot inicializálja és kitölti az **infoTextBox**-ot.

Elemek és a hozzájuk tartozó függvények:

- **backButton** (gomb „Back” felirattal)
    - **backButton_Click** – Bezárja ezt az ablakot és megjeleníti a korábban elrejtett, **originForm**-ban tárolt ablakot.
- **outputTextBox** (szövegdoboz, kezdetben „Output:” tartalommal, amibe a műveltek eredményeit írjuk ki)
- **infoTextBox** (szövegdoboz, kezdetben „Game info:” tartalommal, amibe a konstruktor beleírja az aktuális játék részleteit)
- **showStatesButton** (gomb „States” felirattal)
    - **showStatesButton_Click** – Kiírja az összes lehetséges játékeset (a **currentGame gameStates** listájának felhasználásával) az **outputTextBox** szövegdobozba.
- **showGraphButton** (gomb „Graph” felirattal)
    - **showGraphButton_Click** – Kiírja az összes lehetséges átmenetet, amivel egy játékesetből egy másikba juthatunk (a **currentGame**

playGameGraph tömbjének felhasználásával) az **outputTextBox** szövegdobozba.

- **playGameButton** (gomb „Play Game” felirattal)
    - **playGameButton_Click** – Elindítja a játék játszását, a többi művelet gombját lezárja, és megjeleníti a játékhoz használt **stepButton**, **stopButton, figureStepButton, skipTurnButton** gombokat és a **figureSelectBox** szövegdobozt. Elindítja a játékot a **playerStarts** értékének megfelelő módon.
- **stepButton** (gomb „Roll Dice” felirattal)
    - **stepButton_Click** – Játékos végrehajt egy dobást a **currentGame PlayerRoll** függvényével, ezt kiírja az **outputTextBox**-ba, majd elérhetővé teszi a **figureStepButton, figureSelectBox, és skipTurnButton** elemeket.
- **figureStepButton** (gomb „Step Using Figure” felirattal)
    - **figureStepButton_Click** – Megpróbálja léptetni a játékos által a **figureSelectBox**-ban megadott figurát (a dobás már korábban megtörtént). Amennyiben sikertelen a lépés, **ShowErrorMessage** metódus használatával hibát jelzi a hibát. Sikeres lépés végén meghívja az **enemyStep** metódust.
- **skipTurnButton** (gomb „Skip Turn” felirattal)
    - **skipTurnButton_Click** – Lépés nélkül véget vet a játékos a körének (**currentGame PlayerSkipTurn** metódusával). Majd meghívja az **enemyStep** metódust.
- **figureSelectBox** (szövegdoboz „Step with which figure?” kezdeti értékkel) – **figureStepButton_Click** használja
- **stopButton** (gomb „End Game” felirattal)
    - **stopButton_Click** – Leállítja a játékot az **EndGame** metódus meghívásával.
- **mcSimulateButton** (gomb „MC Simulate” felirattal)
    - **mcSimulateButton_Click** – Meghívja a **SimulateCorrect** függvényt, és amennyiben igaz értéket kap vissza, Monte Carlo szimulációt hajt végre

a függvény által értéket kapott paraméterekkel a **currentGame MonteCarloSimulation** függvényével.

- **mcSimulateTimesBox** (szövegdoboz „Times” kezdeti értékkel) – **SimulateIsCorrect** függvény használja
- **mcSimulateMaxStepsBox** (szövegdoboz „Max Steps” kezdeti értékkel) – **SimulateIsCorrect** függvény használja
- **mcSimulateTacticBox** (szövegdoboz „Tactic” kezdeti értékkel) – **SimulateIsCorrect** függvény használja
- **markov1Button** (gomb „Markov 1” felirattal)
    - **markov1Button_Click** – Meghívja a **MarkovCorrect** függvényt, és amennyiben igaz értéket kap vissza, a kapott paraméterekkel a **currentGame Markov1** függvényével elemzi, hogy egy játékállapotból kiindulva megadott számú lépés után mennyi eséllyel leszünk az egyes játékállapotokban.
- **markov1StartBox** (szövegdoboz „Start State” kezdeti értékkel) – **MarkovCorrect** függvény használja
- **markov1TimesBox** (szövegdoboz „Times” kezdeti értékkel) – **MarkovCorrect** függvény használja
- **markov1TacticBox** (szövegdoboz „Tactic” kezdeti értékkel) – **MarkovCorrect** függvény használja
- **markov1_5Button** (gomb „Markov 1.5” felirattal)
    - **markov1_5Button_Click** – Meghívja a **Markov1_5Correct** függvényt, és amennyiben igaz értéket kap vissza a kapott paraméterekkel a **currentGame Markov1_5** függvényével elemzi, hogy egy játékállapotból kiindulva átlagosan hányszor fogunk áthaladni egy megadott játékállapoton (nem végállapot) amíg a játék befejeződik.
- **markov1_5StartBox** (szövegdoboz „Start State” kezdeti értékkel) – **Markov1_5Correct** függvény használja
- **markov1_5EndBox** (szövegdoboz „End State” kezdeti értékkel) – **Markov1_5Correct** függvény használja

- **markov1_5TacticBox** (szövegdoboz „Tactic” kezdeti értékkel) – **Markov1_5Correct** függvény használja
- **markov2Button** (gomb „Markov 2” felirattal)
    - **markov2Button_Click** – Meghívja a **Markov2Correct** függvényt, és amennyiben igaz értéket kap vissza, a kapott paraméterekkel a **currentGame Markov2** függvényével elemzi, hogy egy játékállapotból kiindulva mennyi eséllyel fejezzük be a játékot az egyes végállapotokban.
- **markov2StartBox** (szövegdoboz „Start State” kezdeti értékkel) – **Markov2Correct** függvény használja
- **markov2TacticBox** (szövegdoboz „Tactic” kezdeti értékkel) – **Markov2Correct** függvény használja
- **clearOutputButton** (gomb „Clear Output” felirattal)
    - **clearOutputButton_Click** – Visszaállítja az **outputTextBox** tartalmát a kezdeti értékére.
- **errorLabel** („Error Log:” szöveget tartalmazó felirat az **errorBox** mellett)
- **errorBox** (Kezdetben üres szövegdoboz, amibe a hibaüzenetek kerülnek a **ShowErrorMessage** metódus futásakor)

További függvények:

- **void enemyStep()**
    - Végrehajtja az ellenfél (2. játékos) lépését a játék játszásánál, a **currentGame TakeStep** függvényét használva.
- **void GameEndedCheck**(ref int[] **currentState**)
    - Eldönti a jelenlegi játékállapotról, hogy a játék végállapota-e és amennyiben végállapot, automatikusan kiírja az **outputTextBox**-ba, hogy vége a játéknak és meghívja az **EndGame** metódust.
- **void EndGame**()
    - A játék játszásához használt gombokat (**stepButton, stopButton, figureStepButton, skipTurnButton**) és szövegdobozt (**figureSelectBox**) újra elrejti, majd a többi művelet gombjait és szövegdobozait újra elérhetővé teszi.

- **bool SimulateCorrect**(ref int **times**, ref int **maxSteps**, ref int **tactic**)
    - A paramétereknek értékül adja **mcSimulateTimesBox, mcSimulateMaxStepsBox,** és **mcSimulateTacticBox** tartalmait és igaz értéket ad vissza, amennyiben azok megfelelnek a feladat követelményeinek. (amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)
- **bool MarkovCorrect**(ref int **start**, ref int **times**, ref int **tactic**)
    - A paramétereknek értékül adja **markov1StartBox markov1TimesBox** és **markov1TacticBox** tartalmait és igaz értéket ad vissza, amennyiben azok megfelelnek a feladat követelményeinek. (amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)
- **bool Markov1_5Correct**(ref int **start**, ref int **end**, ref int **tactic**)
    - A paramétereknek értékül adja **markov1_5StartBox markov1_5EndBox,** és **markov1_5TacticBox** tartalmait és igaz értéket ad vissza, amennyiben azok megfelelnek a feladat követelményeinek. (amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)
- **bool Markov2Correct**(ref int **start**, ref int **tactic**)
    - A paramétereknek értékül adja **markov2StartBox** és **markov2TacticBox** tartalmait és igaz értéket ad vissza, amennyiben azok megfelelnek a feladat követelményeinek. (amennyiben nem, hibaüzenetet ír a **ShowErrorMessage** metódussal és hamis értéket ad vissza)
- **void AppendGameState(**int[] **gameState)**
    - A megadott játékállapotot kiírja az **outputTextBox** szövegdobozba.
- **void ShowErrorMessage**(String **errorDescription**)
    - A megadott hibaüzenetet jeleníti meg az **errorBox** szövegdobozban.

Penney Osztály

Leírás:

Egy lehetséges „Penney's game" szimulációjára és elemzésére szolgáló osztály.

A játék leírása: Van egy vagy több kockánk, ezekkel egy dobásnak N különböző kimenetele lehet. A két játékosnak van egy-egy (M_1, M_2 hosszúságú, ahol M_1=M_2 nem feltétlenül teljesül) listája, amely a lehetséges dobások sorozatából áll. A játékosok közösen dobnak a kockával, az játékos nyer, amelyiknek előbb egyezik meg a saját listája az utolsó M_1 vagy M_2 darab dobás eredményeinek a listájával. (Akkor egyeznek meg, ha az utolsó dobott szám megegyezik a lista utolsó (legbaloldaliabb értékével), az utolsó előtti dobásnál kapott szám megegyezik a lista utolsó előtti értékével, stb.)

A default konstruktor által létrehozott Penney's game: 1 db „2 oldalú kocka" (pénzérme) dobása, ahol a két játékos listája 112 és 121.

Változók:

- **dicePool** (**Dice** osztály egy példánya)
    - Játék által használt kockákat tartalmazó példány a **Dice** osztálynak.
- **gameStates** (integer tömböket tartalmazó lista)
    - Lehetséges játékállapotokat tartalmazó lista. A játékállapotokat 2 hosszú integer tömbökként ábrázoljuk.
    - A tömb 0. eleme az alapján osztályozza az állapotokat, hogy hány dobás történt eddig. (Amennyiben M vagy annál több dobás történt, akkor a tömb 0. elemének M az értéke (M:=max(M_1,M_2), amennyiben ennél kevesebb akkor a dobások számával egyezik meg a tömb 0. eleme)
    - A tömb 1. elemét egy tetszőleges játékállapothoz az alábbi módon kapjuk meg: Tekintsük a dobások eredményeinek listáját egy D-számrendszer beli (ahol D legyen az összes különböző összértékű dobás) szám helyiértékeinek listájával. (Ahol a lista helyiértékek csökkenő sorrendben van, tehát a lista végén szerepel az egyes helyiértéken szereplő szám) Ezt a számot 10-es számrendszerbe átváltva megkapjuk az állapotához tartozó tömb 1. elemének értékét.
- **gameStateBreakpoints** (integer tömb)
    - A **gameStates** tömböt megalkotó **CreateStates()** függvény úgy hozza létre a játékállapotokat, hogy azok a tömbjük 0. eleme szerint növekvő sorrendben vannak. Ennek a tömbnek az n. eleme azt tárolja, hogy a

gameStates lista hányadik elemétől kezdődik azok a játékállapotok, ahol a tömb 0. eleme egyenlő n.

- **gameGraph** (integer listákat tartalmazó tömb)
    - Tömb, amely az egyes játékállapotok közötti átmeneteket tárolja. Minden lista az egyik állapotból lehetséges összes átmenetet tárolja, az elérésükhöz szükséges dobás szerint növekvő sorrendben. A játékállapotokra a **gameStates** listán belüli sorszámukkal hivatkozunk. Tehát a **gameGraph** N. listájának első eleme a **gameStates** tömb N. játékállapotából a lehető legkisebb kockadobással elért játékállapot **gameStates**–beli indexe lesz, a lista második eleme a második lehető legkisebb dobással elért játékállapot **gameStates**-beli indexe lesz, stb.
    - Amennyiben egy játékállapot végállapot, a hozzá tartozó lista üres.
- **actualGraph** (**Graph** osztály egy példánya)
    - A játék szimulálásához használt példánya a **Graph** osztálynak.
- **actualMarkov** (**Markov** osztály egy példánya)
    - A játék elemzéséhez használt példánya a **Markov** osztálynak.
- **patternA** (integer tömb)
    - Az „A” játékos által használt lista. (A konstruktorban a kettő megadott lista közül az első)
- **patternACode** (integer)
    - A **patternA** beli listához tartozó játékállapot **gameStates**-beli tömbjének 1 helyén álló érték (kiszámításának módja feljebb, a **gameStates** leírásánál)
- **patternAModulo** (integer)
    - D^k (ahol D legyen az összes különböző összértékű dobás, k pedig a **patternA** hossza)
    - A játékállapot és a **patternACode** összehasonlításánál használjuk, ugyanis előfordulhat, hogy a játékállapot hosszabb listát ír le a **patternA**-nál, de a vége pont megegyezik a **patternA**-val (azért jöhet létre ilyen helyzet, mivel nem kötöttük ki, hogy **patternA** és **patternB** azonos hosszúak)

- **patternB** (integer tömb)
    - Az „B” játékos által használt lista. (A konstruktorban a kettő megadott lista közül az második)
- **patternBCode** (integer)
    - A **patternB** beli listához tartozó játékállapot **gameStates**-beli tömbjének 1 helyén álló érték (kiszámításának módja feljebb, a **gameStates** leírásánál)
- **patternBModulo** (integer)
    - D^k (ahol D legyen az összes különböző összértékű dobás, k pedig a **patternB** hossza)
    - A játékállapot és a **patternBCode** összehasonlításánál használjuk, ugyanis előfordulhat, hogy a játékállapot hosszabb listát ír le a **patternB**-nél, de a vége pont megegyezik a **patternB**-vel (azért jöhet létre ilyen helyzet, mivel nem kötöttük ki, hogy **patternA** és **patternB** azonos hosszúak)

Konstruktorok:

- **Penney()** – Default konstruktor, amely az osztály leírásában megadott default Penney’s game játékot hozza létre.
- **Penney(**ref int[]**diceArray**, ref int[] **inputPatternA**, ref int[] **inputPatternB)** – Alternatív konstruktor, amely lehetővé teszi tetszőleges kockák és játékosok által használt listák megadását.

Egyéb Metódusok:

- **void CreateStates(**int **currentLength**, int **maxLength**, int **possibleRolls)** – Rekurzív metódus, amit a konstruktorokban hívunk meg, **gameStates** értékeit állítja be.
- **bool isEndStateIndex(**int **stateIndex)** – Függvény, ami egy **gameStates**-beli indexre visszaadja, hogy végállapot-e.
- **void CreateGraph(**int **possibleRolls)** - Metódus, amit a konstruktorokban hívunk meg, **gameGraph** értékeit állítja be. (Amennyiben a **gameStates** már be van állítva.)

- **void CreateMarkov ()** – Metódus, ami ténylegesen létrehozza az elemző **Markov** osztály példányát.
- **void ResetGame()** – Metódus, amin keresztül a jelenleg **Graph** osztályon belül zajló szimulációt le tudjuk állítani.
- **int[] TakeStep()** – Függvény, ami lépteti a szimulációt, majd visszaadja a jelenlegi játékállapotot.
- **int EndGameCheck(**ref int[] **currentState)** – Függvény, ami egy lehetséges játékállapotról visszaadja, hogy végállapot-e, és ha végállapot, akkor melyik játékos nyert. (0: nem végállapot, 1: **patternA** nyert, 2: **patternB** nyert)
- **int LastRollResult()** – Függvény, ami visszaadja a **dicePool** legutolsó dobásának eredményét.
- **int[] MonteCarloSimulation(**int **numberOfTests**, int **maxSteps)** – Függvény, ami visszaadja a játék sokszori futtatásának eredményét egy 3 elemű tömbként.
    - **0. elem ->** hányszor volt, hogy a játék nem fejeződött be a maximális lépésszám elérése előtt.
    - **1. elem ->** hányszor volt, hogy a játékot a **patternA**-hez tartozó játékos nyerte.
    - **2. elem ->** hányszor volt, hogy a játékot a **patternB**-hez tartozó játékos nyerte.
- **double[] Markov1(**int **startingStateIndex**, int **steps)** – Függvény, ami létrehozza a játékhoz tartozó **Markov** osztály példányt, amennyiben az még nem létezett, majd **Markov** osztály **transitionMatrix** változójának másolatát (**steps**-1)-ik hatványára emeli. Ennek adja vissza a **startingStateIndex** állapotnak megfelelő sorát (*Emlékeztető: Markov létrehozásakor a sorok és oszlopok sorrendje változott!*)
- **double Markov1_5(**int **startingState**, int **otherState)** - Függvény, ami létrehozza a játékhoz tartozó **Markov** osztály példányt, amennyiben az még nem létezett, majd **Markov** osztály **GetFundamentalMatrix** által visszaadott mátrixának a **startingState** állapotnak megfelelő sorának az **otherState** állapotnak megfelelő elemét adja vissza. (*Emlékeztető: Markov létrehozásakor a sorok és oszlopok sorrendje változott!*)

- **double[] Markov2(**int **startingState**, int[] out **endStates)** - Függvény, ami létrehozza a játékhoz tartozó **Markov** osztály példányt, amennyiben az még nem létezett, majd **Markov** osztály **GetEndStateChance** által visszaadott tömböt adja vissza, az **endStates**-nek értékül adva a végállapotok eredeti indexeit. (*Emlékeztető: Markov létrehozásakor a sorok és oszlopok sorrendje változott!*)

SnakesAndLadders Osztály

Leírás:

Egy lehetséges „Kígyók és létrák” játék szimulációjára és elemzésére szolgáló osztály.

A játék leírása: Van egy táblánk, amiken a bábunk halad a kezdőpozíciótól a vége felé. Van egy vagy több kockánk, ezekkel minden lépésben dobunk, majd a kockákon dobott értékek összegének megfelelő számú mezőt lépünk a tábla vége felé. A táblán vannak kitüntetett mezők, ún. „kígyók” és „létrák”, amelyekre lépve automatikusan a jelenlegi mezőről egy, az adott kígyó vagy létra definiálásakor megadott, új mezőre helyezzük át a bábut. Kígyó az a speciális mező, amely ezzel a tábla végétől távolabb helyezi a bábut, és létra az a speciális mező, amelyik ezzel az áthelyezéssel a tábla végéhez közelebb helyezi a bábut. Egy mezőhöz mindig csak egy kígyó vagy létra tartozhat, és ha egy mező kígyó vagy létra, akkor egy kígyó vagy létra sem lehet, amely erre a mezőre helyezne bábut.

A default konstruktor által létrehozott Kígyók és létrák játék: 100 mezőből álló tábla, azon 5 kígyóval (21->12, 35->2, 54->33, 61->51, 89->43) és 5 létrával (3->11, 18->31, 37->65, 59->63, 71->87)

Változók:

- **dicePool** (Dice osztály egy példánya)
    - Játék által használt kockákat tartalmazó példány a **Dice** osztálynak.
- **gameStates** (integer tömböket tartalmazó lista)
    - Lehetséges játékállapotokat tartalmazó lista. A játékállapotokat 1 hosszú integer tömbökként ábrázoljuk. Különböző játékállapotnak

számít a tábla minden olyan mezője ahonnan egy lépésünk indulhat. (tehát minden olyan mező, ahonnan nem indul kígyó vagy létra)
    - A tömb egyedüli eleme azt tárolja, hogy a játékállapot a tábla melyik mezője. (0-s indexelést használva)
- **gameGraph** (integer listákat tartalmazó tömb)
    - Tömb, amely az egyes játékállapotok közötti átmeneteket tárolja. Minden lista az egyik állapotból lehetséges összes átmenetet tárolja, az elérésükhöz szükséges dobás szerint növekvő sorrendben. A játékállapotokra a **gameStates** listán belüli sorszámukkal hivatkozunk. Tehát a **gameGraph** N. listájának első eleme a **gameStates** tömb N. játékállapotából a lehető legkisebb kockadobással elért játékállapot **gameStates**–beli indexe lesz, a lista második eleme a második lehető legkisebb dobással elért játékállapot **gameStates**-beli indexe lesz, stb.
    - Amennyiben egy játékállapot végállapot, a hozzá tartozó lista üres.
- **actualGraph** (Graph osztály egy példánya)
    - A játék szimulálásához használt példánya a **Graph** osztálynak.
- **actualMarkov** (Markov osztály egy példánya)
    - A játék elemzéséhez használt példánya a **Markov** osztálynak.

Konstruktorok:

- **SnakesAndLadders()** – default konstruktor, amely az osztály leírásában megadott default Kígyók és Létrák játékot hozza létre.
- **SnakesAndLadders (**ref int[,] **diceArray**, int **boardLength**, ref List<int[]> **ladders**, ref List<int> **snakes)** – Alternatív konstruktor, amely lehetővé teszi tetszőleges kockák, játéktábla hossz, kígyók és létrák megadását. A konstruktor ezen kívül rendezi a **snakes** és **ladders** listákat kezdőpozíció szerint növekvő sorrendbe.

Egyéb Metódusok:

- **void CreateHelpArray(**ref int[] **helpArray**, ref List<int[]> **ladders**, ref List<int[]> **snakes)** – Segéd metódus, amit a konstruktorok használnak a lokális **helpArray** értékeinek beállítására. A **helpArray** a tábla összes mezőjéhez tárolja, hogy arra

a mezőre lépre, melyik mezőről indíthatjuk a következő lépést. (Ami persze megegyezik a mezővel, ha nem indul onnan kígyó vagy létra.)

- **void CreateStates(**ref int[] **helpArray)** - Metódus, amit a konstruktorokban hívunk meg, **gameStates** értékeit állítja be a korábban a beállított **helpArray** segítségével.
- **void GetState(**int **squareNumber)** – Segéd függvény, amit a **CreateGraph** metódus használ, megadja a paraméterként megadott mezőszámhoz tartozó **gameStates** beli indexet.
- **bool isEndStateIndex(**int **stateIndex)** – Függvény, ami egy **gameStates**-beli indexre visszaadja, hogy végállapot-e.
- **void CreateGraph(**ref int[] **helpArray**, int **possibleRolls)** - Metódus, amit a konstruktorokban hívunk meg, **gameGraph** értékeit állítja be. (Amennyiben a **gameStates** már be van állítva.)
- **void CreateMarkov()** - Metódus, ami ténylegesen létrehozza az elemző **Markov** osztály példányát.
- **void ResetGame()** - Metódus, amin keresztül a jelenleg **Graph** osztályon belül zajló szimulációt le tudjuk állítani.
- **void TakeStep()** - Függvény, ami lépteti a szimulációt, majd visszaadja a jelenlegi játékállapotot.
- **int EndGameCheck(**ref int[] **currentState)** – Függvény, ami egy lehetséges játékállapotról visszaadja, hogy végállapot-e. (0: nem végállapot, 1: végállapot)
- **int LastRollResult()** – Függvény, ami visszaadja a **dicePool** legutolsó dobásának eredményét.
- **void MonteCarloSimulation(**int **numberOfTests**, int **maxSteps)** - Függvény, ami visszaadja a játék sokszori futtatásának eredményét, egy **maxSteps+1** elemszámú tömbként, amelyben a tömb n. elemének értéke:
    - n=0 : Hány olyan eset volt, ahol a játék nem fejeződött be **maxSteps** lépésen belül.
    - n>0 : Hány olyan eset volt, ahol a játék pontosan n eset alatt fejeződött be.

- **double[] Markov1(**int **startingStateIndex**, int **steps)** - Függvény, ami létrehozza a játékhoz tartozó **Markov** osztály példányt, amennyiben az még nem létezett, majd **Markov** osztály **transitionMatrix** változójának másolatát (**steps**-1)-ik hatványára emeli. Ennek adja vissza a **startingStateIndex** állapotnak megfelelő sorát (*Emlékeztető: Markov létrehozásakor a sorok és oszlopok sorrendje változott!*)
- **double Markov1_5(**int **startingState**, int **otherState)** - Függvény, ami létrehozza a játékhoz tartozó **Markov** osztály példányt, amennyiben az még nem létezett, majd **Markov** osztály **GetFundamentalMatrix** által visszaadott mátrixának a **startingState** állapotnak megfelelő sorának az **otherState** állapotnak megfelelő elemét adja vissza. (*Emlékeztető: Markov létrehozásakor a sorok és oszlopok sorrendje változott!*)
- **double[] Markov2(**int **startingState**, int[] out **endStates)** - Függvény, ami létrehozza a játékhoz tartozó **Markov** osztály példányt, amennyiben az még nem létezett, majd **Markov** osztály **GetEndStateChance** által visszaadott tömböt adja vissza, az **endStates**-nek értékül adva a végállapotok eredeti indexeit. (*Emlékeztető: Markov létrehozásakor a sorok és oszlopok sorrendje változott!*)

KiNevetAVegen Osztály

Leírás:

Egy lehetséges „Ki Nevet a Végén?” játék szimulációjára és elemzésére szolgáló osztály.

A játék leírása: Ebben a játékban két játékos játszik egymás ellen. Minden játékosnak van valahány figurája. Felváltva dobnak kockával (kockákkal), céljuk az, hogy a kezdetben a bázisukban lévő figuráikat a saját célmezőjükbe eljuttassák.
A tábla kör alakú, a bázisaik átellenes mezőkhöz csatlakoznak (**boardLength** páros szám), mindketten egy irányba léphetnek csak ezen a táblán. A táblán körbeérve még meg kell tenniük **bonusLength** (legalább 1) mezőnyi lépést, hogy a saját céljukba érjenek.
Minden lépésben egy játékos a dobott értéknek megfelelő mezőnyit léptethet egy figuráján. Amennyiben egy figura a lépés során egy olyan mezőre kerülne, ahol az

ellenfél egy figurája áll, a lépés megtörténik, és az ellenfél figurája visszakerül a bázisába. Bázisból kihozni figurát csak akkor lehet, ha a kockával (vagy kockákkal) a lehető legnagyobb értéket dobtuk, ilyenkor a közvetlenül a bázishoz csatlakozó első mezőre lép a figura. Nem léphetünk figuránkkal egy olyan mezőre ahol már van egy figuránk (kivétel a célmező), és ha nincs szabályos lépésre lehetőségünk egy körben, akkor nem hajtunk végre lépést abban a körben.

A default konstruktor által létrehozott Ki Nevet a Végén? játéknál egy olyan kockát használunk ami egyenlő eséllyel ad 1-et és 2-t eredményül, a kör alakú tábla hossza 4, bónusz lépések száma 2, mindkét játékosnak 2 figurája van, a 2. játékos alapértelmezett taktikája az 1 számú taktika, és az 1. játékos kezdi a játékot.

Változók:

- **dicePool** (**Dice** osztály egy példánya)
    - Játék által használt kockákat tartalmazó példány a **Dice** osztálynak.
- **gameStates** (integer tömböket tartalmazó lista)
    - Lehetséges játékállapotokat tartalmazó lista. A játékállapotokat **figuresA**+**figuresB** hosszú integer tömbként ábrázoljuk, ahol:
    - **0-figuresA-1** elemek: Az 1. játékos figuráinak elhelyezkedésük a táblán, csökkenő sorrendben aszerint, hogy hány lépést tettek meg a kezdőállapotból a végállapot felé.
    - **figuresA – figuresA+figuresB-1** elemek: A 2. játékos figuráinak elhelyezkedésük a táblán, növekvő sorrendben aszerint, hogy hány lépést tettek meg a kezdőállapotból a végállapot felé.
- **playGameGraph** (integer listákat tartalmazó tömb)
    - A játék játszásához használt gráf. Az első felében (ami az 1. játékos lépéseihez tartozó játékállapotokat tartalmazza), minden lehetséges lépés el van tárolva, míg a 2. játékos lépései a megadott taktikának megfelelőek.
- **analysisGameGraph** (integer listákat tartalmazó tömb)
    - A játék szimulálásához és elemzéséhez használt gráf. Mindkét játékos lépései a megadott taktikáknak megfelelőek.
- **boardLength** (integer)

    - Megadott tábla hossz.
- **bonusSteps** (integer)
    - Tábla végének elérése után hány lépést kell még a figuráknak lépniük a célba beérés előtt.
- **enemyTactic** (integer)
    - A 2. játékos által használt taktika száma.
- **lastTacticUsed** (integer)
    - A legutóbb létrehozott **analysisGameGraph**-hoz használt 1. játékos taktika
- **playGraph** (**Graph** osztály egy példánya)
    - A játék játszásához használt példánya a **Graph** osztálynak.
- **analysisGraph** (**Graph** osztály egy példánya)
    - A játék Monte-Carlo szimulációjánál használt példánya a **Graph** osztálynak.
- **analysisMarkov** (**Markov** osztály egy példánya)
    - A játék elemzéséhez használt példánya a **Markov** osztálynak.

Konstruktorok:

- **KiNevetAVegen ()** – Default konstruktor, amely az osztály leírásában megadott default Ki Nevet a Végén játékot hozza létre.
- **KiNevetAVegen (**int **boardLength**, int **bonusSteps**, int **figuresA**, int **figuresB**, ref int[] **diceArray**, bool **playerStarts**, int **inputEnemyTactic)** – Alternatív konstruktor, amely lehetővé teszi tetszőleges kockák, … megadását.

Egyéb Metódusok:

- **void HelpCreateStatesLeft(**ref int[] **currentNewState**, int **currentFigNum)** – Rekurzív metódus, ami a gameStates elemeinek létrehozásánál használt.
- **void HelpCreateStatesRight(**ref int[] **currentNewState**, int **currentFigNum)** – Rekurzív metódus, ami a gameStates elemeinek létrehozásánál használt.
- **void CreateStates()** – Metódus, ami létrehozza a **gameStates** elemeit.
- **int SearchForState(**ref int[] **searchFor)** – Függvény, ami egy játékállás helyét keresi meg a gameStates listában.

- **int AttemptStepPlayer(**int **currentState**, int **figureAttempt**, int **rollResult**, bool **isPlayerA)** – Függvény, ami visszaadja annak a játékhelyzetnek az indexét, amelyet azzal érünk el, ha a megadott **currentState** állapotból **rollResult** dobás után a **figureAttempt** figurát próbáljuk léptetni. (-1-et ad vissza amennyiben ez a lépés nem lehetséges)
- **bool isEndStateIndex(**int **stateIndex)** – Függvény, ami egy **gameStates**-beli indexre visszaadja, hogy végállapot-e.
- **int GetGraphConnectionForTactic(**int **tactic**, ref List<int>[] **fullGraph**, int **currentStatePos**, int **rollResult)** - Függvény, ami visszaadja annak a játékhelyzetnek az indexét, amelyet azzal érünk el, ha a megadott **currentState** állapotból **rollResult** dobás után a **tactic** taktickát követve lépünk.
- **void CreatePlayGraph()** – Metódus, ami létrehozza a **playGraph** tömböt.
- **void CreateAnalysisGraph (**int **tacticUsed)** – Metódus, ami a megadott taktika (és a **playGraph**) segítségével létrehozza az **analysisGraph**-ot.
- **void CreateMarkov (**int **tacticUsed)** – Metódus, ami a megadott taktika és az **analysisGraph** segítségével létrehozza az **actualMarkov** elemet.
- **void ResetGame()** – Metódus, ami visszaállítja a kezdőállapotát a **playGraph**-nak.
- **int[] PlayRound(**int **playerChoiceFromLastRound)** - …
- **int[] TakeStep()** - Függvény, ami lépteti a szimulációt, majd visszaadja a jelenlegi játékállapotot.
- **int PlayerRoll()** - Függvény, ami visszaadja a **dicePool** egy dobását.
- **int LastRollResult()** – Függvény, ami visszaadja a **dicePool** legutolsó dobásának eredményét.
- **int[] MonteCarloSimulation(**int **numberOfTests**, int **maxSteps**, int **tacticUsed)** - Függvény, ami visszaadja a játék sokszori futtatásának eredményét egy 3 elemű tömbként.
    - **0. elem ->** hányszor volt, hogy a játék nem fejeződött be a maximális lépésszám elérése előtt.
    - **1. elem ->** hányszor volt, hogy a játékot az 1. játékos nyerte.
    - **2. elem ->** hányszor volt, hogy a játékot a 2. játékos nyerte.

- **double[] Markov1(**int **startingStateIndex**, int **steps**, int **tacticUsed)** - Függvény, ami létrehozza a játékhoz és a **tacticUsed**-hoz tartozó **Markov** osztály példányt, amennyiben az még nem létezett, majd **Markov** osztály **transitionMatrix** változójának másolatát (**steps**-1)-ik hatványára emeli. Ennek adja vissza a **startingStateIndex** állapotnak megfelelő sorát (*Emlékeztető: Markov létrehozásakor a sorok és oszlopok sorrendje változott!*)
- **double Markov1_5(**int **startingState**, int **otherState**, int **tacticUsed)** - Függvény, ami létrehozza a játékhoz és a **tacticUsed**-hoz tartozó **Markov** osztály példányt, amennyiben az még nem létezett, majd **Markov** osztály **GetFundamentalMatrix** által visszaadott mátrixának a **startingState** állapotnak megfelelő sorának az **otherState** állapotnak megfelelő elemét adja vissza. (*Emlékeztető: Markov létrehozásakor a sorok és oszlopok sorrendje változott!*)
- **double[] Markov2(**int **startingState**, int[] out **endStates**, int **tacticUsed)** - Függvény, ami létrehozza a játékhoz és a **tacticUsed**-hoz tartozó **Markov** osztály példányt, amennyiben az még nem létezett, majd **Markov** osztály **GetEndStateChance** által visszaadott tömböt adja vissza, az **endStates**-nek értékül adva a végállapotok eredeti indexeit. (*Emlékeztető: Markov létrehozásakor a sorok és oszlopok sorrendje változott!*)
- **void PlayerSkipTurn()** – Metódus, aminek segítségével a játék játszásánál átugorhatjuk a játékos lépését, a gráfon belüli pozíciót megfelelően állítva. (Amennyiben nincs a játékosnak valid lépése)

Dice Osztály

Leírás:

Osztály, amivel szimulálni tudjuk a játék során történő kockadobásokat. Egy dobás állhat tetszőleges számú kocka egyszerre történő dobásából, a kockák tetszőleges (akár egymástól különböző) N-oldalúak, ahol egy N-oldalú kocka dobásának eredménye egy pozitív szám 1-től N-ig, egyenlő valószínűséggel.

Változók:

- **diceArray** (integer tömb)

    - Az osztály megalkotásakor megadott tömb, amely az egy dobáshoz használt összes kocka lapszámát tárolja. Pl.: ha diceArray = { 6, 10 }, akkor egy dobásnál egyszerre dobunk egy 6 és 10 oldalú kockával (amelyek 1-től vannak számozva).
- **odds** (double tömb)
    - Tömb, amely olyan hosszú ahány különböző értéke lehet egy dobásnak, benne pedig a különböző dobás értékek valószínűségei vannak (dobás érték szerint növekvő sorrendben, 0 indextől kezdve) Az osztályon belül, a GetOdds() függvény által van kiszámolva.
- **rnd** (a C# Random osztályának példánya)
    - A random dobások eredményének kiszámolására használjuk.
- **lastRollResult** (integer)
    - A **Roll()** függvény által utoljára dobott érték

Konstruktorok:

- **Dice()** – Default konstruktor, egy egyszerű, egy 6-oldalú dobókockát használó osztályt hoz létre.
- **Dice(**ref int[] **inputArray)** – Alternatív konstruktor, ami lehetővé teszi tetszőleges kocka kombinációk használatát.

Egyéb Metódusok:

- **int Roll()** – Függvény, ami visszaadja egy véletlenszerű dobás értékét (az összes kockán dobott számok összegét).
- **int MinRoll()** – Függvény, ami visszaadja az osztályhoz tartozó kockákkal kidobható legkisebb értéket.
- **int MaxRoll()** – Függvény, ami visszaadja az osztályhoz tartozó kockákkal kidobható legnagyobb értéket.
- **double[] GetOdds()** – Függvény, ami visszaadja az **odds** tömböt, és amennyiben az **odds** tömb még nem volt létrehozva, azt létrehozza.
- **void CreateOdds(**ref int[] **copyArray**, int **currentLength**, int **maxLength**, ref int **times**, ref int[] **occured)** – Rekurzív segéd metódus az **odds** tömb kiszámításához a **GetOdds()** függvényen belül.

- **int GetLastRollResult()** – Visszaadja a **lastRollResult**-ban tárolt értéket

Graph Osztály

Leírás:

Osztály, amivel egy játék menetét tudjuk szimulálni. A játék lehetséges állásait a **gameStates** lista tartalmazza, a közöttük lévő átmeneteket pedig a **gameGraph** tömb tartalmazza.

Változók:

- **gameStates** (integer tömböket tartalmazó lista)
    - Lehetséges játékállapotokat tartalmazó lista. A játékállapotokat integer tömbökként ábrázoljuk, értelmezésük az éppen szimulált játéktól függ.
- **gameGraph** (integer listákat tartalmazó tömb)
    - Tömb, amely az egyes játékállapotok közötti átmeneteket tárolja. Minden lista az egyik állapotból lehetséges összes átmenetet tárolja, az elérésükhöz szükséges dobás szerint növekvő sorrendben. A játékállapotokra a **gameStates** listán belüli sorszámukkal hivatkozunk. Tehát a **gameGraph** N. listájának első eleme a **gameStates** tömb N. játékállapotából a lehető legkisebb kockadobással elért játékállapot **gameStates**–beli indexe lesz, a lista második eleme a második lehető legkisebb dobással elért játékállapot **gameStates**-beli indexe lesz, stb.
    - Amennyiben egy játékállapot végállapot, a hozzá tartozó lista üres.
- **gameStateLength** (integer)
    - Azt tárolja, hogy az aktuálisan szimulált játék állapotának tárolására hány integerből álló tömböt használunk.
- **currentPosIndex** (integer)
    - Annak a játékállapotnak a **gameStates**-beli indexét tárolja, ahol a szimuláció éppen tart.
- **startingPosIndex** (integer)
    - Annak a játékállapotnak a **gameStates**-beli indexét tárolja, amelyik a játék kezdőpozíciója.
- **dicePool** (**Dice** osztáy egy példánya)

- A játékhoz egy dobásnál használt kockákat tárolására és minden lépésnél a kockadobások eredményeinek kiszámítására szolgál.

Konstruktor:

- **Graph(**ref List<int[]> **inputGameStates**, ref List<int>[] **inputGraph**, int **inputStartingPosIndex**, ref Dice **inputDicePool)** – konstruktor, ami a megadott kezdőpozíció (**startingPosIndex**) és előre elkészített **gameGraph**, **gameState** és **Dice** segítségével létrehozza a **Graph** osztályt

Egyéb Metódusok:

- **void Reset()** – Metódus, ami visszaállítja a játék szimulációt a kezdőpozícióba.
- **int[] GetGameState()** – Függvény, ami visszaadja a játékállapotot, ahol a szimuláció jelenleg tart.
- **bool Step()** – Függvény, ami végrehajt egy dobást, és a szimulációt ennek megfelelően lépteti. Amennyiben rendben lezajlott a lépés, igazat ad vissza, amennyiben egy végállapotból próbáltunk ellépni, hamisat ad vissza, lépés nem történik.
- **void SetGameState(int** newStatePosIndex**)** – Ami egy megadott állásba állítja a gráfbejárást. *(Csak a Ki Nevet a Végén játszásához használjuk)*
- **int GetGameState()** – Ami visszaadja a gráf melyik csúcsában tart a bejárás. *(Csak a Ki Nevet a Végén játszásához használjuk)*

Markov Osztály[5]

Leírás:

Osztály, amivel egy Markov-láncként ábrázolt játékon tudunk elemzéseket elvégezni. A játék állapotaira csak számokként hivatkozunk. (aktuális osztály **gameStates** listájában lévő indexük) A játék állapotai közötti átmeneteket a **transitionMatrix** tárolja, ami konstruktorban rendezve van, hogy az átmeneti állapotokhoz tartozó sorok és oszlopok előre, az elnyelő állapotokhoz (végállapotok) tartozók pedig a végére kerülnek.

[5] Markov láncokról, és a rajtuk elvégzett műveletekről bővebben a dokumentáció **Matematikai Háttér** (*20. oldal*) pontjában olvashat részletesebben.

Változók:

- **transitionMatrix** (double értékeket tartalmazó kétdimenziós mátrix)
    - Markov lánc tranzíciós (átmeneti) mátrixa. A különböző játékállapotok közötti átmenetek valószínűségét tárolja. A sorok és oszlopok rendezve lettek, hogy az átmeneti játékállapotok (amelyek nem 1 valószínűséggel önmagukba mennek) az elejére, és a végállapotok (amelyek 1 valószínűséggel önmagukba mennek egy lépés eredményeként) pedig a végére kerülnek.
- **originalOrder** (integer tömb)
    - Tömb, amiben tároltuk, hogy a mi volt a **transitionMatrix** sorainak és oszlopainak sorrendje. (A tömb n-edik eleme tárolja, hogy a jelenleg n. sor/oszlop hányadik sor/oszlop volt az átrendezés előtt.)
- **absorbingStatesStart** (integer)
    - Tárolja, hogy a tömb hányadik indexű eleme az első végállapoté a **transitionMatrix**-ben.
- **fundamentalMatrix** (double értékeket tartalmazó kétdimenziós mátrix)
    - A Markov lánc fundamentális mátrixa (**(I-Q)**$^{-1}$). Mérete (átmenti állapotok száma X átmeneti állapotok száma)

Konstruktor:

- **Markov(**ref double[,] **inputTransitionMatrix)** – konstruktor, ami egy előre elkészített (de még nem rendezett) tranzíciós mátrix (i. sor j. eleme = mennyi a valószínűsége, hogy az i. helyzetből egy lépéssel a j. helyzetbe jutunk), felhasználásával létrehozza a **Markov** osztályt.

Statikus Metódus:

- **double[,] MultiplyMatrices(**ref double[,] **matrixA**, ref double[,] **matrixB)** – Két mátrix összeszorzására használt statikus függvény. (**matrixA** van a szorzás bal oldalán) – Ez a függvény van használva a **KiNevetAVegen**, **Penney**, és **SnakesAndLadders** osztályok **Markov1** műveleteihez.

Egyéb Metódusok:

- **double[] GetEndStateChance(**int **startingState)** – A **transitionMatrix** és a **fundamentalMatrix**, értékeinek a használatával (amiben a **GetFundamentalMatrix** függvény segít) kiszámolja azt, hogy ha a játékban éppen a paraméterben (eredeti indexeléssel) megadott játékállapotban (ami nem lehet végállapot) vagyunk, mennyi az esélye, hogy az egyes végállapotokban végezzük a játékot. (**(I-Q)$^{-1}$*R**) – Ez a függvény van használva a **KiNevetAVegen**, **Penney**, és **SnakesAndLadders** osztályok **Markov2**műveleihez.
- **double[,] GetFundamentalMatrix()** – Amennyiben még nem lett érték adva a **fundamentalMatrix**-nak, meghívja a **CalculateFundamentalMatrix**-ot és annak eredményét eltárolja a **fundamentalMatrix** változóban, majd visszaadja egy másolatát. Ha már van értéke, a függvény visszaadja egy másolatát a mátrixnak. – Ez a függvény van használva a **KiNevetAVegen**, **Penney**, és **SnakesAndLadders** osztályok **Markov1_5** műveleteihez.
- **double[,] CalculateFundamentalMatrix(**ref double[,] **QMatrix)** – Függvény, ami kiszámolja a Markov lánc fundamentális mátrixát ((**(I-Q)$^{-1}$**), ahol a **Q** a **transitonMatrix** legnagyobb, csak átmeneti állapotokat tartalmazó („bal felső”) részmátrixa. Az inverz kiszámítására a **SolveFundamental** metódust használja.
- **void SolveFundamental(**ref double[,] **workMatrix**, ref double[,] **inverseMatrix**, int **size)** – Segédmetódus, amit a **CalculateFundamentalMatrix** függvény használ az **(I-Q)** mátrix inverzének kiszámítására.

Tesztelés

Megjegyzés: *A program számábrázolása miatt a tesztekben várt értékek és a program által visszaadott értékek között lehetnek elhanyagolható különbségek.*

Hiba üzenetek tesztelése

Hibák a Penney's Game tesztek megadásánál

-Nem adunk meg kockákat / megadott kocka érték nem pozitív szám

-Nem adunk meg listát valamelyik játékosnak / megadott lista érték kockadobással nem elérhető szám

-Két játékos listája megegyezik / egyik lista vége megegyezik a másik teljes listájával (hiba - ekkor előfordulhatna az, hogy ugyanabban az állapotban egyszerre nyer a két játékos)

Hibák a Ki Nevet a Végén tesztek megadásánál

-Nem adunk meg kockákat / megadott kocka érték nem pozitív szám

-Nem adunk meg tábla hosszt / megadott tábla hossz nem pozitív szám

-Nem adunk meg célba a beéréshez szükséges plusz lépések számát / a megadott beéréshez szükséges plusz lépések száma nem pozitív szám

-Nem adunk meg figurák számát valamelyik játékosnak / megadott figura szám nem pozitív szám

Hibák a Snakes and Ladders tesztek megadásánál

-Nem adunk meg kockákat / megadott kocka érték nem pozitív szám

-Nem adunk meg tábla hosszt / megadott tábla hossz nem egy, 1-nél nagyobb pozitív szám

-A kígyóknak és létráknak megadott pontoknak tábla mezőinek kell lenniük, továbbá nem indulhatnak a tábla első és utolsó mezőjéről

-Megadott kígyók között van nem helyes (kezdőpont előrébb van mint a végpont)

-Megadott létrák között van nem helyes (végpont előrébb van mint a kezdőpont)

-Egy mezőről több mint egy kígyó vagy létra indul (hiba - egyébként nem lenne egyértelmű a következő mező)

-Egy mezőről, amiről indul kígyó vagy létra, egyben végpontja egy másik kígyónak vagy létrának (hiba - nem engedtük meg az ilyen felállást a játék leírásánál)

Hibák a "Játszás/Szimuláció/Elemzés" ablaknál

-Megpróbálunk úgy Monte Carlo Szimulációt indítani, hogy:

  -Nem adtunk meg hányszor futtassa le a tesztet / megadott szám nem pozitív

  -Nem adtunk meg maximális lépésszámot le a tesztet / megadott szám nem pozitív

  -Ki Nevet a Végén játék esetén nem adtunk meg a játékos által használt taktikát

-Megpróbálunk úgy "Markov 1" elemzést indítani, hogy:

  -Nem adtunk meg kezdő játékállapotot / nem létező játékállapotot adtunk meg

    -Nem adtunk meg hány lépéssel később vizsgáljuk / megadott szám nem pozitív

    -Ki Nevet a Végén játék esetén nem adtunk meg a játékos által használt taktikát

-Megpróbálunk úgy “Markov 2” elemzést indítani, hogy:

-Nem adtunk meg kezdő játékállapotot / nem létező játékállapotot, vagy végállapot adtunk meg

-Ki Nevet a Végén játék esetén nem adtunk meg a játékos által használt taktikát

Penney’s Game tesztek

Teszt 1:

Teszt 1 - Bemenet:

Kockák: [2] , Player A Pattern: [1,1] , Player B Pattern: [2,2]

Teszt 1 - „States” gomb várt eredménye:

Visszaadja a játékállapotokat, amik az összes 0,1,2 elemből álló dobáskombinációk (ahol a dobásértékek: 1,2).

Teszt 1 - „Graph” gomb várt eredménye:

Helyesen visszaadja a játékállapotok közötti átmeneteket.

Teszt 1 - „Play Game” gomb várt eredménye:

A játék lejátszása rendben megtörténik. (A játék lépései a gráf és a játékszabályoknak megfelelően történnek, a játék vége és a győztes rendben van meghatározva.)

Teszt 1 - „MC Simulate” gomb várt eredménye:

Mivel a két lista lényegében nem tér el egymástól (mivel 1-es és 2-es dobás ugyanolyan valószínűségű), ezért azt várjuk, hogy a Monte Carlo szimuláció során hasonló sokszor fog nyerni a két játékos.

Teszt 1 - „Markov 1” gomb várt eredménye:

Azt várjuk, hogy a kezdőállapotból egy lépés után 0.5 eséllyel leszünk az „1” és „2” állapotokban. Második lépés után Egyenlő 0.25 eséllyel leszünk „11”, „12”, „21”, „22” állapotokban. További lépések után egyenlő ütemben csökken „12” és „21” esetek valószínűsége, és növekszik „11” és „22” végállapotok valószínűsége.

Teszt 1 - „Markov 1.5” gomb várt eredménye:

Azt várjuk, hogy a kezdőpozícióból indulva az összes nem végállapotot átlagosan 0.5-ször fogjuk érinteni.

Teszt 1 - „Markov 2” gomb várt eredménye:

Azt várjuk, hogy 1/2 valószínűséggel jutunk olyan helyzetbe, ahol az 1. játékos nyert, és 1/2 valószínűséggel jutunk olyan helyzetbe, ahol a 2. játékos nyert.

Teszt 2:

Teszt 2 - Bemenet:

Kockák: [2] , Player A Pattern: [1,1,2] , Player B Pattern: [1,2,1]

Teszt 2 - „States” gomb várt eredménye:

Visszaadja a játékállapotokat, amik az összes 0,1,2,3 elemből álló dobáskombinációk (ahol a dobásértékek: 1,2).

Teszt 2 - „Graph” gomb várt eredménye:

Helyesen visszaadja a játékállapotok közötti átmeneteket.

Teszt 2 - „Play Game” gomb várt eredménye:

A játék lejátszása rendben megtörténik. (A játék lépései a gráf és a játékszabályoknak megfelelően történnek, a játék vége és a győztes rendben van meghatározva.)

Teszt 2 - „MC Simulate” gomb várt eredménye:

Azt várjuk, hogy a Monte Carlo szimuláció során az 1. játékos körülbelül 2-szer annyiszor fog nyerni, mint a 2. játékos.

Teszt 2 - „Markov 1” gomb várt eredménye:

Azt várjuk, hogy a lépések számának növelésével csökken az esélye annak, hogy nem végállapotban vagyunk, és a valószínűsége annak, hogy az 1. játékos nyerőállapotában (8) vagyunk 2/3-hoz tart. (Közben 2. játékos nyerőállapotában (9) a valószínűség 1/3-hoz tart.)

Teszt 2 - „Markov 1.5” gomb várt eredménye:

Azt várjuk, hogy a 7. állapotból már nem érintjük a többi (nem vég) állapotot.

Teszt 2 - „Markov 2” gomb várt eredménye:

Azt várjuk, hogy 2/3 valószínűséggel jutunk olyan helyzetbe, ahol az 1. játékos nyert, és 1/3 valószínűséggel jutunk olyan helyzetbe, ahol a 2. játékos nyert.

Teszt 3:

Teszt 3 - Bemenet:

Kockák: [4] , Player A Pattern: [1,2,3] , Player B Pattern: [4,3,2,1]

Teszt 3 - „States” gomb várt eredménye:

Visszaadja a játékállapotokat, amik az összes 0,1,2,3,4 elemből álló dobáskombinációk (ahol a dobásértékek: 1,2,3,4).

Teszt 3 - „Graph” gomb várt eredménye:

Helyesen visszaadja a játékállapotok közötti átmeneteket.

Teszt 3 - „Play Game” gomb várt eredménye:

A játék lejátszása rendben megtörténik. (A játék lépései a gráf és a játékszabályoknak megfelelően történnek, a játék vége és a győztes rendben van meghatározva.)

Teszt 3 - „MC Simulate” gomb várt eredménye:

Azt várjuk, hogy az 1. játékos 4:1 arányban fogja nyerni a játékokat. (Megfelelően nagy Max Steps érték mellett)

Teszt 3 - „Markov 1” gomb várt eredménye:

Azt várjuk, hogy a lépések számának növelésével csökken az esélye annak, hogy nem végállapotban vagyunk, és a két végállapot beli valószínűségek a Monte Carlo szimuláció alapján várt értékekhez tartanak.

Teszt 3 - „Markov 1.5” gomb várt eredménye:

Azt várjuk, hogy megfelelnek a kapott értékek megfelelnek a várt eredményeknek. (Pl: ugyanannyi alkalommal érintjük 3333 és 2222 eseteket)

Teszt 3 - „Markov 2” gomb várt eredménye:

Azt várjuk, hogy 4/5 valószínűséggel jutunk olyan helyzetbe, ahol az 1. játékos nyert, és 1/5 valószínűséggel jutunk olyan helyzetbe, ahol a 2. játékos nyert.

Ki Nevet a Végén tesztek

Teszt 1:

Teszt 1 - Bemenet:

Kockák: [1,1] , Táblahossz: 2 , Bónusz lépések: 1, Játékos A figurák: 1 , Játékos B figurák: 1, Kezdő játékos: *tetszőleges* , Taktika: *tetszőleges*

Teszt 1 - „States” gomb várt eredménye:

Visszaadja a játékállapotokat, amiknek első fele azok az állapotok, ahonnan az 1. játékosé a következő lépés, a második fele pedig ezek másolata, kivéve, hogy itt a 2. játékosé a következő lépés. Ezek az állapotok 2 számból állnak: az 1. szám azt tárolja,

hogy az 1. játékos egyetlen figurája hány lépést tett meg a cél felé, a 2. szám pedig a 2. játékos figurájáról tárolja ezt.

Teszt 1 - „Graph” gomb várt eredménye:

Helyesen visszaadja a játékállapotok közötti átmeneteket, amelyben az 1. játékos (gráf csúcsainak első fele) összes lehetséges lépése szerepel, míg a 2. játékos lehetséges lépései csak a kockadobásoktól függnek. – Mivel a „kockadobás” eredménye ebben a játékban minden esetben 2, a gráf ennek megfelelően épül fel.

Teszt 1 - „Play Game” gomb várt eredménye:

A játék lejátszása rendben megtörténik. (A játék lépései a gráf és a játékszabályoknak megfelelően történnek, a játék vége és a győztes rendben van meghatározva.)

Teszt 1 - „MC Simulate” gomb várt eredménye:

Mivel a kockadobás egyértelmű és csak 1-1 figurája van a játékosoknak, választott taktikáktól függetlenül annak kell teljesülnie, hogy a kezdő játékos nyer minden esetben.

Teszt 1 - „Markov 1” gomb várt eredménye:

A játék egyértelmű: a visszaadott eredményben 1 darab 1-esnek kell szerepelnie, a többi értéknek 0-nak kell lennie.

Teszt 1 - „Markov 1.5” gomb várt eredménye:

A játék egyértelmű: a visszaadott eredményben csak 1-eseknek és 0-knak szabad lennie.

Teszt 1 - „Markov 2” gomb várt eredménye:

A játék minden esetben a kezdő játékos győzelmével ér véget, ez az 1. játékos esetén a [3,1], a 2. játékos esetén az [1,3] játékállapot.

Teszt 2:

Teszt 2 - Bemenet:

Kockák: [2] , Táblahossz: 4 , Bónusz lépések: 2, Játékos A figurák: 2 , Játékos B figurák: 2, Kezdő játékos: *tetszőleges* , Taktika: 1

Teszt 2 - „States” gomb várt eredménye:

Visszaadja a játékállapotokat, amiknek első fele azok az állapotok, ahonnan az 1. játékosé a következő lépés, a második fele pedig ezek másolata, kivéve, hogy itt a 2.

játékosé a következő lépés. Ezek az állapotok 4 hosszúak, az első kettő szám azt tárolja, hogy az 1. játékos figurái hány lépést tett meg a cél felé (növekvő sorrendben), az utolsó kettő szám pedig azt tárolja, hogy a 2. játékos figurái hány lépést tett meg a cél felé. (csökkenő sorrendben)

Teszt 2 - „Graph" gomb várt eredménye:

Helyesen visszaadja a játékállapotok közötti átmeneteket, amelyben az 1. játékos (gráf csúcsainak első fele) összes lehetséges lépése szerepel, míg a 2. játékos lehetséges lépései csak a kockadobásoktól függnek.

Teszt 2 - „Play Game" gomb várt eredménye:

A játék lejátszása rendben megtörténik. (A játék lépései a gráf és a játékszabályoknak megfelelően történnek, a játék vége és a győztes rendben van meghatározva.)

Teszt 2 - „MC Simulate" gomb várt eredménye:

Azt várjuk, hogy a kezdő játékos (akármelyik legyen), mindig az esetek felénél többször fog nyerni.

Teszt 2 - „Markov 1" gomb várt eredménye:

A mátrix mérete miatt pontos várt eredményt nem tudunk mondani, de azt várjuk, hogy a lépések számának növelésével nő a végállapotban tartózkodás esélye.

Teszt 2 - „Markov 1.5" gomb várt eredménye:

A mátrix mérete miatt pontos várt eredményt nem tudunk mondani.

Teszt 2 - „Markov 2" gomb várt eredménye:

Azt várjuk, hogy összesen 0.5-nél nagyobb valószínűséggel jutunk olyan helyzetbe, ahol a kezdő játékos nyert, és 0.5-nél kisebb valószínűséggel jutunk olyan helyzetbe, ahol a nem kezdő játékos nyert.

Teszt 3:

Teszt 3 - Bemenet:

Kockák: [2] , Táblahossz: 4 , Bónusz lépések: 2, Játékos A figurák: 3 , Játékos B figurák: 2 , Kezdő játékos: A, Taktika: 3

Teszt 3 - „States" gomb várt eredménye:

Visszaadja a játékállapotokat, amiknek első fele azok az állapotok, ahonnan az 1. játékosé a következő lépés, a második fele pedig ezek másolata, kivéve, hogy itt a 2.

játékosé a következő lépés. Ezek az állapotok 5 hosszúak, az első három szám azt tárolja, hogy az 1. játékos figurái hány lépést tett meg a cél felé (növekvő sorrendben), az utolsó kettő szám pedig azt tárolja, hogy a 2. játékos figurái hány lépést tett meg a cél felé. (csökkenő sorrendben)

Teszt 3 - „Graph" gomb várt eredménye:

Helyesen visszaadja a játékállapotok közötti átmeneteket, amelyben az 1. játékos (gráf csúcsainak első fele) összes lehetséges lépése szerepel, míg a 2. játékos lehetséges lépései csak a kockadobásoktól függnek.

Teszt 3 - „Play Game" gomb várt eredménye:

A játék lejátszása rendben megtörténik. (A játék lépései a gráf és a játékszabályoknak megfelelően történnek, a játék vége és a győztes rendben van meghatározva.)

Teszt 3 - „MC Simulate" gomb várt eredménye:

Azt várjuk, hogy hiába kezd az 1. játékos, a plusz 1 figura miatt a 2. játékos fog az esetek többségében nyerni.

Teszt 3 - „Markov 1" gomb várt eredménye:

A mátrix mérete miatt pontos várt eredményt nem tudunk mondani, de azt várjuk, hogy a lépések számának növelésével nő a végállapotban tartózkodás esélye.

Teszt 3 - „Markov 1.5" gomb várt eredménye:

A mátrix mérete miatt pontos várt eredményt nem tudunk mondani.

Teszt 3 - „Markov 2" gomb várt eredménye:

Azt várjuk, hogy összesen nagyobb valószínűséggel jutunk olyan helyzetbe, ahol a 2. játékos nyert, mint az 1.

Snakes and Ladders tesztek

Teszt 1:

Teszt 1 - Bemenet:

Kockák: [2] , Táblahossz: 3 , Kígyók: *üres* , Létrák: *üres*

Teszt 1 - „States" gomb várt eredménye:

Visszaadja a játékállapotokat, amik a célállapot és az összes olyan mező, ahonnan egy lépés indulhat, (1,2,3 mezők).

Teszt 1 - „Graph” gomb várt eredménye:

Helyesen visszaadja a játékállapotok közötti átmeneteket.

Teszt 1 - „Play Game” gomb várt eredménye:

A játék lejátszása rendben megtörténik. (A játék lépései a gráf és a játékszabályoknak megfelelően történnek, a játék vége rendben van meghatározva.)

Teszt 1 - „MC Simulate” gomb várt eredménye:

Amennyiben a maximális lépésszámot 1-re állítjuk, azt kell, hogy tapasztaljuk, hogy az esetek felében fognak a szimulációink befejeződni. (1-es mezőről indulunk, és 3-as mezőre kell eljutni)

Teszt 1 - „Markov 1” gomb várt eredménye:

A kezdő állapotból kiindulva egy lépés után azt várjuk, hogy 0.5 az esély arra, hogy a tábla 2. mezőjén vagyunk és 0.5 az esély arra, hogy a 3. mezőn, két lépés után pedig 1 valószínűséggel vagyunk a 3. mezőn.

Teszt 1 - „Markov 1.5” gomb várt eredménye:

A kezdő állapotból kiindulva azt várjuk, hogy a tábla 2. mezőjét 0.5-ször érintjük átlagosan.

Teszt 1 - „Markov 2” gomb várt eredménye:

Minden esetben 1 valószínűséggel eljutunk az egyetlen célmezőbe.

Teszt 2:

Teszt 2 - Bemenet:

Kockák: [6] , Táblahossz: 100 , Kígyók: [[21,12], [35,2], [54,33], [61,51], [89,43]] , Létrák: [[3,11], [18,31], [37,65], [59,63], [71,87]]

Teszt 2 - „States” gomb várt eredménye:

Visszaadja a játékállapotokat, amik a célállapotot és az összes olyan mező, ahonnan egy lépés indulhat (Számok 1-től 100-ig, a kígyó vagy létra kiinduló mezőinek kivételével).

Teszt 2 - „Graph” gomb várt eredménye:

Helyesen visszaadja a játékállapotok közötti átmeneteket.

Teszt 2 - „Play Game” gomb várt eredménye:

A játék lejátszása rendben megtörténik. (A játék lépései a gráf és a játékszabályoknak megfelelően történnek, a játék vége rendben van meghatározva.)

Teszt 2 - „MC Simulate” gomb várt eredménye:

Azt várjuk, hogy az esetek leggyakrabban 25-35 lépésen belül fejeződnek be.

Teszt 2 - „Markov 1” gomb várt eredménye:

Azt várjuk, hogy a lépésszám növelésével fokozatosan csökken a valószínűség, hogy a tábla elején vagyunk, és növekszik a valószínűség, hogy a végállapotban vagyunk (29-ik lépés és 30-ik lépés között lépi át a valószínűség 0.5-öt.)

Teszt 2 - „Markov 1.5” gomb várt eredménye:

Azt várjuk, hogy azokat a mezőkön, amelyek egy létra vagy kígyó végpontjai, átlagosan többször fogjuk érinteni, mint más, körülöttük lévő mezőket.

Teszt 2 - „Markov 2” gomb várt eredménye:

Minden esetben 1 valószínűséggel eljutunk az egyetlen célmezőbe.

Teszt 3:

Teszt 3 - Bemenet:

Kockák: [6] , Táblahossz: 10 , Kígyók: [[2,1], [3,1], [4,1], [5,1]] , Létrák: [[6,9], [7,10]]

Teszt 3 - „States” gomb várt eredménye:

Visszaadja a játékállapotokat, amik a célállapotot és az összes olyan mező, ahonnan egy lépés indulhat (Számok 1-től 10-ig, a kígyó vagy létra kiinduló mezőinek kivételével).

Teszt 3 - „Graph” gomb várt eredménye:

Helyesen visszaadja a játékállapotok közötti átmeneteket.

Teszt 3 - „Play Game” gomb várt eredménye:

A játék lejátszása rendben megtörténik. (A játék lépései a gráf és a játékszabályoknak megfelelően történnek, a játék vége rendben van meghatározva.)

Teszt 3 - „MC Simulate” gomb várt eredménye:

Azt várjuk, hogy maximális lépésszámot 2-re állítva 4/9 részében fog a szimuláció végrehajtódni. (1/3 + 2/3 * 1/6), az esetek 5/9 részében nem ér célba 2 lépésben.

Teszt 3 - „Markov 1” gomb várt eredménye:

Azt várjuk, hogy a lépésszám növelésével csökkeni fog a valószínűsége, hogy az 1. mezőn vagyunk, és nőni fog a valószínűsége, hogy a 10. mezőben (végállapot) vagyunk. Ez a növekedés üteme a Monte Carlo szimuláció alapján várt

Teszt 3 - „Markov 1.5” gomb várt eredménye:

Azt várjuk, hogy a 8. mezőt egyszer sem érintjük, és a 9. mezőt 0.5-ször érintjük átlagosan.

Teszt 3 - „Markov 2” gomb várt eredménye:

Minden esetben 1 valószínűséggel eljutunk az egyetlen célmezőbe.

Use Case teszteset

Program indítása, egy Penney’s Game játék elemzése, egy Kígyók és Létrák játék játszása, majd a program bezárása.

1. Indítsuk el a programot.
2. A Főmenü ablakban nyomjuk meg a „Penney’s Game” gombot.
3. A megnyílt Penney’s Game – Beállítások ablakban nyomjuk meg egyszer az „Add Dice” gombot, és az így megjelenő dobozba írjuk bele a kocka értékét: 2
4. A Penney’s Game – Beállítások ablakban nyomjuk meg háromszor az „Add Pattern A” gombot, és az így megjelenő dobozokat **alulról felfelé** töltsük ki: 1, 1, 2 értékekkel.
5. A Penney’s Game – Beállítások ablakban nyomjuk meg háromszor az „Add Pattern B” gombot, és az így megjelenő dobozokat **alulról felfelé** töltsük ki: 2, 1, 1 értékekkel.
6. A Penney’s Game – Beállítások ablakban nyomjuk meg a „Next” gombot.
7. A megnyílt Penney’s Game – Műveletek ablakban a „Game info” leírásnál az 1. játékos listájánál „1 1 2”-nek, a 2. játékos listájánál „2 1 1”-nek kell szerepelnie.
8. A Penney’s Game – Műveletek ablakban az „MC Simulate” gombot nyomjuk meg, miután beállítottuk az alatta levő dobozokban a „Times” értéket 10000-re a „Max Steps” értéket pedig 100-ra.
9. A Penney’s Game – Műveletek ablakban az „Output” dobozban a szimuláció eredménye helyes. (Minden esetben végzett 100 lépésen belül, és a 2. játékos körülbelül 3-szor annyi játékot nyert, mint az 1. játékos)

10. A Penney’s Game – Műveletek ablakban a „Markov 2” gombot nyomjuk meg, miután beállítottuk az alatta levő dobozokban a „Start State” értéket 0-ra.
11. A Penney’s Game – Műveletek ablakban az „Output” dobozban az elemzés eredménye helyes. (A 8. játékállapotban 0.25, a 11. játékállapotban 0.75 eséllyel végzünk)
12. A játékállapotok leírása a „States” gomb lenyomása után megtalálható az „Output” dobozban.
13. A Penney’s Game – Műveletek ablakban nyomjuk meg a „Back” gombot.
14. Az újra megnyílt Penney’s Game – Beállítások ablakban nyomjuk meg a „Back” gombot.
15. A Főmenü ablakban nyomjuk meg a „Kígyók és Létrák” gombot.
16. A megnyílt Kígyók és Létrák – Beállítások ablakban nyomjuk meg kétszer az „Add Dice” gombot, és hagyjuk meg az így megjelenő dobozoknak az automatikusan megadott 6 értéket.
17. A Kígyók és Létrák – Beállítások ablakban adjuk meg a „-Board Length-” dobozban a táblahosszt: 25
18. A Kígyók és Létrák – Beállítások ablakban nyomjuk meg kétszer az „Add Snake” gombot, és az így megjelenő dobozokat töltsük ki: [10,2] és [20, 7] kígyókkal.
19. A Kígyók és Létrák – Beállítások ablakban nyomjuk meg kétszer az „Add Ladder” gombot, és az így megjelenő dobozokat töltsük ki: [12, 18] és [3, 11] létrákkal.
20. A Kígyók és Létrák – Beállítások ablakban nyomjuk meg a „Next” gombot.
21. A megnyílt Kígyók és Létrák – Műveletek ablakban a „Game info” leírásnál a kígyók listájánál [10,2] és [20,7] szerepel, a létrák listájánál pedig [3,11] és [12,18] szerepel.
22. A Kígyók és Létrák – Műveletek ablakban nyomjuk meg a „Play Game” gombot.
23. Meg kell, hogy jelenjen a kezdőállapot leírása az „Output” dobozban, a „Next Step” gomb és az „End Game” gomb. (A többi művelet gombjának pedig deaktiválódniuk kell.)
24. A „Next Step” gomb nyomásával lehet a játékot léptetni, az „End game” gombbal lehet megszakítani. Minden gombnyomás után a program az „Output” dobozba írja az eredményt.

25. A játék végeztével a Kígyók és Létrák – Műveletek ablakban nyomjuk meg a „Back” gombot.
26. Az újra megnyílt Kígyók és Létrák – Beállítások ablakban nyomjuk meg a „Back” gombot
27. Az újra megnyílt Főmenü ablakban nyomjuk meg a „Exit” gombot.
28. A program be lett zárva, teszteset vége.

Irodalomjegyzék

Játékok Markov-láncként elemzéséhez segítség:

[1] Wayne L. Winston: Operations Research – Applications and Algorithms, Cengage Learning, 2004, [1418], ISBN-13: 978-0-534-38058-8