HÁZI FELADAT

Programozás alapjai 2.

Tervezés

Kovács Donát

BYVO9O

2023. április 15.

Tartalom

1. Feladat

Kiterjesztett kő-papír-olló játék

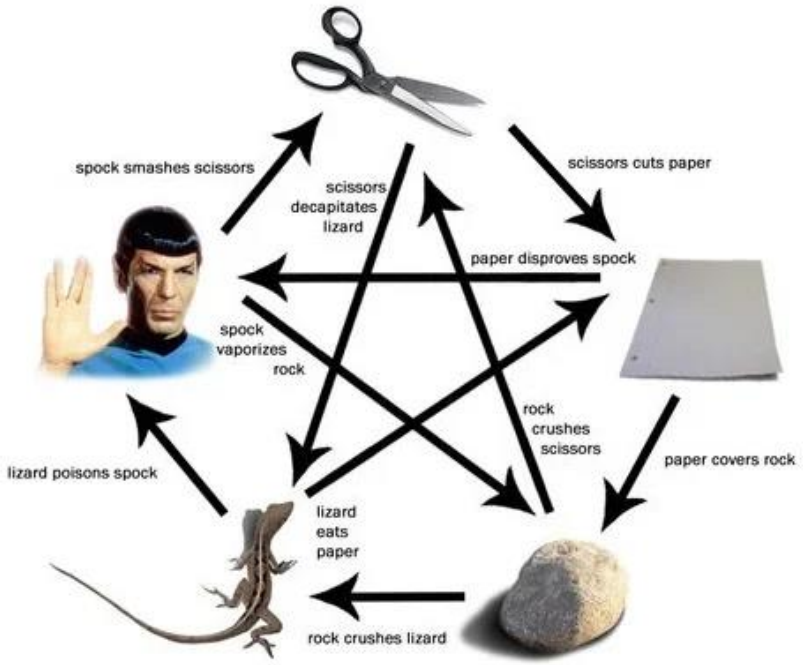
Tervezzen objektummodellt a kő-papír-olló játék modellezéséhez! Célunk, hogy a különböző stratégiával játszó játékosokat összesorsolva megállapítsuk a legjobb stratégiát, ha van ilyen. A modellben legyenek "Játékos" objektumok, melyek egy "Napló" objektum felügyeletével játszanak. Ez utóbbi gyűjti a statisztikát. Demonstrálja a működést külön modulként fordított tesztprogrammal! A játék állását nem kell grafikusan megjeleníteni, elegendő csak karakteresen, a legegyszerűbb formában! A megoldáshoz **ne** használjon STL tárolót!

2. Feladatspecifikáció

2.1 Játékszabályok

A kő-papír-ollót két játékos játssza. A játékosok egyszerre választanak ki egy-egy alakzatot (kő, papír vagy olló). A játék célja, hogy olyan alakzatot válasszunk, amely kiüti a másik játékos által választottat. A kő kicsorbítja az ollót, az olló elvágja a papírt, a papír becsomagolja a követ. Ha mindketten ugyanazt választották, akkor a kör eredménye döntetlen.

A játék kiterjesztett változata a kő-papír-olló-gyík-Spock, melynek lehetséges győzelmi kimenetei a következőek:



A játéknak akkor van vége, ha az előre megbeszélt mennyiségű győzelmet valaki eléri.

2.2 Játékosok

A játékot két játékos játszhatja, amelyek közül bármely lehet ember vagy számítógép. A számítógép esetében több stratégiát (pl.: random, az ellenfél előző lépéseit felhasználó, stb.) lehet kiválasztani. Alapértelmezetten ember játszik a gép random stratégiája ellen.

2.3 Menü

A program angol nyelven és karakteres módban működik.

1. Play

2. Options

    1. Game Type – kő-papír-olló vagy kő-papír-olló-gyík-Spock
    (alapértelmezett: kő-papír-olló)

    2. Game Mode – Step by step, Simulation vagy Tournament
    (alapértelmezett: Step by step)

    3. Game length – meddig tartson egy játék
    (alapértelmezett: 20 győzelemig)

    4. Player 1 Name: Player 1 – első játékos nevének beállítása, nincs megjelenítve tournament módban (alapértelmezett: Player 1)

    5. Player 1 Type: Human – első játékos típusának beállítása, nincs megjelenítve tournament módban

    6. Player 2 Name: Player 2 – második játékos nevének beállítása, nincs megjelenítve tournament módban (alapértelmezett: Player 2)

    7. Player 2 Type: Computer Strategy 1 – második játékos típusának beállítása, nincs megjelenítve tournament módban

    0. Main Menü – visszalépés a főmenübe

3. Rules – játékszabályok megjelenítése

0. Exit – kilépés a játékból

2.4 Játékmenet

Step by step

Ebben a játékmódban egy játékot lehet egyesével léptetve végigjátszani. Ez az egyetlen mód, ahol játszhat ember a számítógép ellen. A program minden lépésnél bekéri a játékostól az alakzat számát. Ha pedig számítógép játszik, akkor az Enter megnyomására történik a lépés.

Simulation

Ebben a játékmódban lehet léptetés nélkül, automatikusan végigfuttatni egy teljes játékot. A szimulációban csak számítógép játszhat számítógép ellen.

Tournament

Ebben a játékmódban lehet léptetés nélkül szimulálni különböző számítógép stratégiák teljes bajnokságát és erről egy statisztikát megjeleníteni.

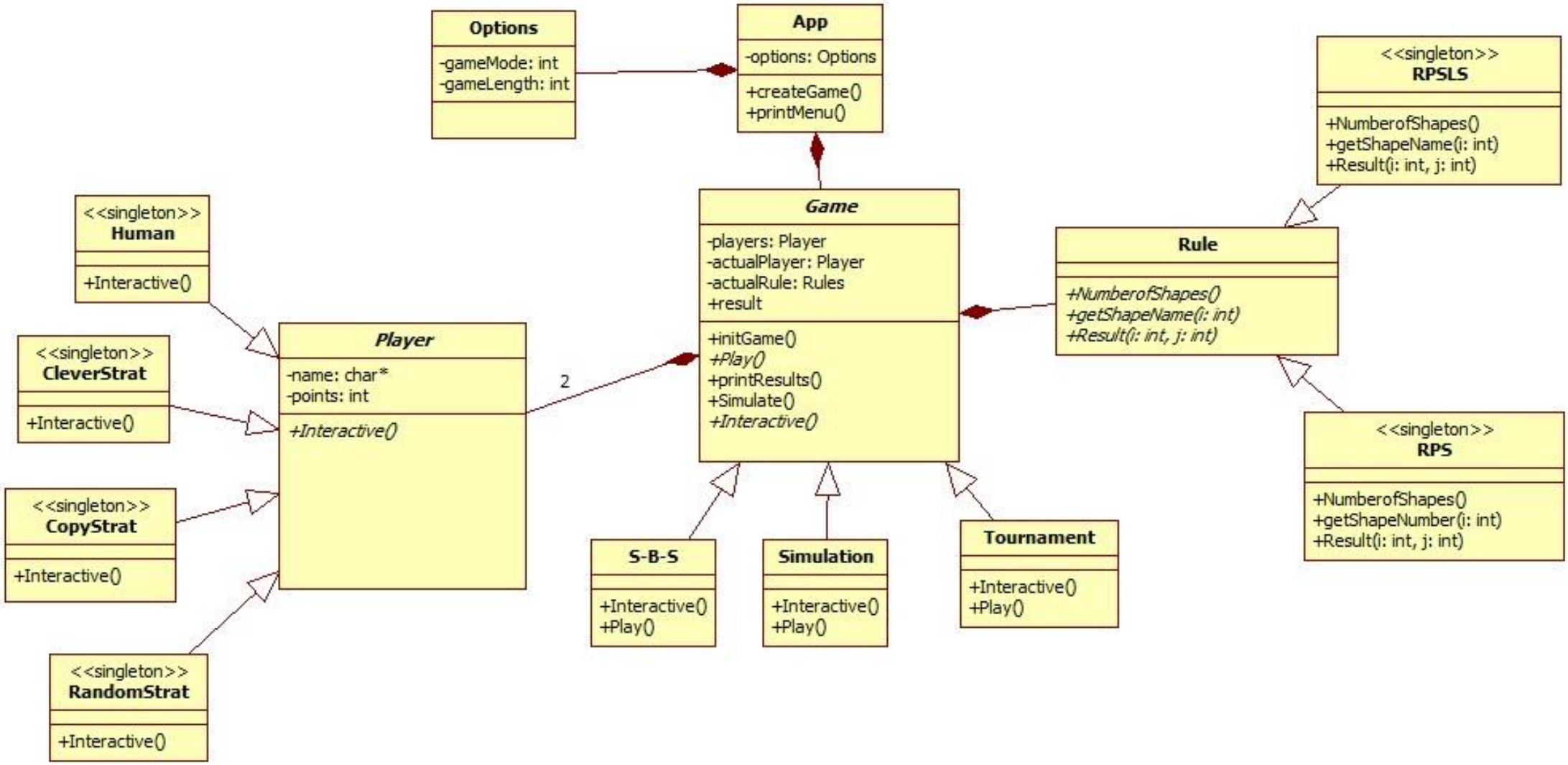
A játékmenet ha sok döntetlen miatt eléri a 100. játszmát, akkor ott vége lesz és az addig több győzelmet szerzett játékos nyeri a menetet. Ha viszont ugyanannyi győzelmük van, akkor a kimenetel döntetlen lesz.

A program a későbbiekben újabb algoritmusokkal tetszőlegesen bővíthető.

3. Terv

A feladat egy program objektum orientált tervezését mutatja be UML diagramban.

A program az alábbi osztályokat, azok kapcsolatát és attribútumait tartalmazza:


Options
-gameMode: int
-gameLength: int
App
-options: Options
+createGame()
+printMenu()
<<singleton>>
RPSLS
+NumberofShapes()
+getShapeName(i: int)
+Result(i: int, j: int)
<<singleton>>
Human
+Interactive()
Game
-players: Player
-actualPlayer: Player
-actualRule: Rules
+result
+initGame()
+Play()
+printResults()
+Simulate()
+Interactive()
Rule
+NumberofShapes()
+getShapeName(i: int)
+Result(i: int, j: int)
<<singleton>>
CleverStrat
+Interactive()
Player
-name: char*
-points: int
+Interactive()
2
<<singleton>>
RPS
+NumberofShapes()
+getShapeNumber(i: int)
+Result(i: int, j: int)
<<singleton>>
CopyStrat
+Interactive()
S-B-S
+Interactive()
+Play()
Simulation
+Interactive()
+Play()
Tournament
+Interactive()
+Play()
<<singleton>>
RandomStrat
+Interactive()

4. Algoritmusok

A feladatban lévő három stratégia algoritmusainak leírása:

Random Strategy:

Random számot generál 0-tól 2-ig vagy 0-tól 4-ig (a játéktípustól függően) és azt a számot adja vissza az algoritmus, amely egy alakzatnak a számát jelöli.

Copy Strategy:

Első lépésnél random generál számot 0-tól 2-ig vagy 0-tól 4-ig (a játéktípustól függően) és azt a számot adja vissza, majd utána az algoritmus minden lépésnél eltárolja az ellenfele alakzatának a számát és a következő lépésben azt adja vissza.

Clever Strategy:

Első lépésnél random generál számot 0-tól 2-ig vagy 0-tól 4-ig (a játéktípustól függően) és azt a számot adja vissza, majd a második lépéstől kezdve utolsó lépéskombináció gyakoriságának a kiszámolásával adja vissza azt az alakzatot, amely az ellenfél legnagyobb valószínűséggel bekövetkező lépését legyőzi.

A tournamentben lévő játékosok teljesítményét pontozó algoritmus leírása:

Az algoritmus egy szimulált játékmenet után a győztesnek 2 a vesztesnek 0 pontot ad. Döntetlen kimenetel esetén mindkét játékos 1-1 pontot kap. Így a bajnokság végén a pontok által statisztikát kapunk az adott stratégiák nyerési valószínűségéről.

5. Megjegyzés a tervhez

A játék tartalmaz tesztelést, melyet a main függvényben lehet definiálni makróval (#define TEST_MODE). A tesztesetekre van egy TEST makró, amely egy adott feltétel esetén megvizsgálja, hogy ugyanazt a kimenetet kapta-e. Ha igen, akkor egy „OK”-t, ha nem, akkor pedig „ERROR”-t, a függvény nevét és a meghívás sorát írja ki. Ha nincsen definiálva, akkor a játék indul el tesztelés nélkül. Amely így néz ki:

```
#define TEST_MODE
#ifdef TEST_MODE
#include <cstring>


#define TEST(condition) \
    if(condition){ \
        std::cout << "OK" << std::endl; \
    } \
    else { \
        std::cout << "ERROR checking " << #condition << " in line " << __LINE__ << std::endl; \
    }
```

6. Megvalósítás

A feladat megoldása 14 osztály és egy tesztprogram elkészítését igényelte. Az osztályok végleges interfésze enyhén eltér a tervezési lépésben meghatározott módtól. A tervezési résztől abban tér el, hogy lettek főként set és get, de további újabb függvényeket is létrehoztam, melyekre csak ahogy fejlesztettem a programot lett szükség. A tesztelés menete ugyanaz maradt, mint a Skeletonban létrehozott tesztprogram. Továbbá a feladatom nem tartalmaz STL könyvtárból származó eszközöket.

Új játékos vagy szabály hozzáadása esetén a base osztályból új származtatott osztályként kell létrehozni azt, majd override-olni a megfelelő virtuális függvényeket. Továbbá App osztálynak a CreateGame függvényében kell hozzáadni (AddPlayer / AddRule) az új játékost vagy szabályt és onnantól kezdve automatikus a megjelenítése és a használata.

7. Tesztelés

A teszteléshez a korábban is említett saját teszt makrót használtam. Továbbá teszteltem, hogy visszaad-e random egész számot valamely stratégia. A tesztelések során az alábbi eseteket vettem figyelembe:

```
void Test(){
    //Rules test
    RPS rps;
    RPSLS rpsls;
    TEST(rps.NumberofShapes() == 3);
    TEST(rpsls.NumberofShapes() == 5);
    TEST(strcmp(rps.GetShapeName( i: 1), "Rock") == 0);
    TEST(strcmp(rpsls.GetShapeName( i: 5), "Spock") == 0);
    TEST(rps.Result( i: 1,  j: 1) == 0);
    TEST(rps.Result( i: 1,  j: 3) == 1);
    TEST(rpsls.Result( i: 5,  j: 4) == -1);
    TEST(rps.WhoBeats( shape: 1) == 2);
    TEST(strcmp(rps.GetRuleName(), "Rock-Paper-Scissors") == 0);
    TEST(strcmp(rpsls.GetRuleName(), "Rock-Paper-Scissors-Lizard-Spock") == 0);
```

```
//App test
App app;
TEST(app.GetNumberOfPlayers() == 0);


//Player test
//Human human;
//human.GetNextShape(rps);

//Random test
RandomStrat random;
int rand1 = random.GetNextShape( &: rps);
std::cout << rand1 << std::endl;
int rand2 = random.GetNextShape( &: rpsls);
std::cout << rand2 << std::endl;
```

7.1 Memóriakezelés tesztelése

A memóriakezelést a laborgyakorlatokon is használt MEMTRACE modullal oldottam meg. A fordítási egységben include-oltam a memtrace.h állományt a standard fejlécállományok után. Memóriakezelési hibát a futtatások során nem tapasztaltam.

8. Az elkészített osztálysablonok bemutatása

Rules, RPS és RPSLS osztályok (rules.cpp és rules.h):

Rules osztály:

Tagfüggvények:

- `virtual int NumberofShapes() = 0:` Virtuális tagfüggvény, visszaadja a formák számát.
- `virtual const char* GetShapeName(int i) = 0:` Virtuális tagfüggvény, visszaadja a megadott indexű forma nevét.
- `virtual int Result(int i, int j) = 0:` Virtuális tagfüggvény, visszaadja az i és j formák eredményét.
- `virtual char* GetRuleName() = 0:` Virtuális tagfüggvény, visszaadja a szabály nevét.
- `virtual int WhoBeats(int shape) = 0:` Virtuális tagfüggvény, visszaadja, hogy melyik forma győzi le a megadott formát.

RPS osztály:

Tagfüggvények:

- `int NumberofShapes()`: A formák számát adja vissza (3).
- `const char* GetShapeName(int i)`: Visszaadja a megadott indexű forma nevét.
- `int Result(int i, int j)`: Visszaadja az i és j formák eredményét a következő táblázat alapján:
  ```
  { 0, -1, 1 },
  { 1, 0, -1 },
  { -1, 1, 0 }
  ```
- `char* GetRuleName()`: Visszaadja a szabály nevét ("Rock-Paper-Scissors").
- `int WhoBeats(int shape)`: Visszaadja, hogy melyik forma győzi le a megadott formát.

RPSLS osztály:

Tagfüggvények:

- `int NumberofShapes()`: A formák számát adja vissza (5).
- `const char* GetShapeName(int i)`: Visszaadja a megadott indexű forma nevét.
- `int Result(int i, int j)`: Visszaadja az i és j formák eredményét a következő táblázat alapján:
  ```
  { 0, -1, 1, 1, -1 },
  { 1, 0, -1, -1, 1 },
  { -1, 1, 0, 1, -1 },
  { -1, 1, -1, 0, 1 },
  { 1, -1, 1, -1, 0 }
  ```
- `char* GetRuleName()`: Visszaadja a szabály nevét ("Rock-Paper-Scissors-Lizard-Spock").
- `int WhoBeats(int shape)`: Visszaadja, hogy melyik forma győzi le a megadott formát a következő szabályok szerint:
  - `Rock`: Paper vagy Spock
  - `Paper`: Scissors vagy Lizard
  - `Scissors`: Rock vagy Spock
  - `Lizard`: Rock vagy Scissors
  - `Spock`: Paper vagy Lizard

App osztály (options.cpp és options.h):

Az általad megadott kódban található osztályok és azok változói és tagfüggvényei a következők:

Változók:

- `int gameLength`: A játék hossza.
- `Game *actualGame`: Az aktuális játék.
- `Player *players[MAX_NUM_PLAYERS]`: A játékosok tömbje.
- `Rules *rules[MAX_NUM_RULES]`: A szabályok tömbje.
- `Game *games[MAX_NUM_GAMES]`: A játékok tömbje.
- `int numberOfPlayers`: A játékosok száma.
- `int numberOfRules`: A szabályok száma.

- `int numberOfGames`: A játékok száma.

Tagfüggvények:

- `App()`: Az osztály konstruktora.
- `int GetGameLength()`: A játék hosszát adja vissza.
- `void AddPlayer(Player* p)`: Hozzáad egy játékost a játékosok tömbjéhez.
- `void AddRule(Rules* r)`: Hozzáad egy szabályt a szabályok tömbjéhez.
- `void AddGame(Game* g)`: Hozzáad egy játékot a játékok tömbjéhez.
- `void CreateGame()`: Létrehozza a játékot, inicializálja a játékosokat, szabályokat és játékokat.
- `void MainMenu()`: A főmenüt megjelenítő függvény, kezeli a menüpontokat és a felhasználói interakciót.
- `void PrintOptions(Player&, Player&)`: Kiírja a játék beállításait és lehetőségeit.
- `void PrintRules(Rules&)`: Kiírja a játékszabályokat.
- `int GetNumberOfPlayers()`: Visszaadja a játékosok számát.
- `Player* GetPlayer(int i)`: Visszaadja a megadott indexű játékost.

Game, SBS, Simulation és Tournament osztályok (game.cpp és game.h):

Games osztály:

Változók:

- `app`: Az App osztályra mutató pointer, amely az alkalmazást reprezentálja.
- `player1`: Az első játékosra mutató pointer.
- `player2`: A második játékosra mutató pointer.
- `actualRule`: Az aktuális szabályokat reprezentáló objektumra mutató pointer.
- `winp1`: Az első játékos győzelmeinek száma.
- `winp2`: A második játékos győzelmeinek száma.
- `draw`: Döntetlen eredmények száma.

Tagfüggvények:

- `Play()`: Elindítja a játékot.
- `PrintResults()`: Kiírja a játék eredményeit.
- `Simulate()`: Szimulálja a játékmenetet.
- `Interactive()`: Visszaadja, hogy interaktív módban van-e a játék. Alapértelmezetten `false` értéket ad vissza.
- `TwoPlayermode()`: Visszaadja, hogy kétjátékos mód van-e beállítva. Alapértelmezetten `true` értéket ad vissza.
- `GetActualRule()`: Visszaadja az aktuális szabályokat reprezentáló objektumot.
- `SetActualRule(Rules *r)`: Beállítja az aktuális szabályokat.
- `SetPlayer1(Player *p1)`: Beállítja az első játékost.
- `GetPlayer1()`: Visszaadja az első játékost.
- `SetPlayer2(Player *p2)`: Beállítja a második játékost.

- `GetPlayer2()`: Visszaadja a második játékost.
- `IsGameWon(int round, int win1, int win2)`: Ellenőrzi, hogy a játék véget ért-e. Visszaadja a megfelelő logikai értéket.

SBS osztály:

- A `Game` osztály leszármazottja.
- Konstruktorban beállítja az `app`, `player1` és `player2` változókat.
- Az `Interactive()` tagfüggvény `true` értéket ad vissza.
- A `GetModeName()` tagfüggvény visszaadja a játék módjának nevét.

Simulation osztály:

- A `Game` osztály leszármazottja.
- Konstruktorban beállítja az `app`, `player1` és `player2` változókat.
- A `GetModeName()` tagfüggvény visszaadja a játék módjának nevét.

Tournament osztály:

- A `Game` osztály leszármazottja.
- Konstruktorban beállítja az `app` változót.
- Az `Play()` tagfüggvény lejátsza a tornát.
- Az `TwoPlayermode()` tagfüggvény `false` értéket ad vissza.
- A `GetModeName()` tagfüggvény visszaadja a játék módjának nevét.
- Az `PrintResults()` tagfüggvény kiírja a torna eredményeit.

Player, Human, CleverStrat, CopyStrat, RandomStrat és PredictiveStrat osztályok (player.cpp és player.h):

Osztályok:

- `Player`: Absztrakt alaposztály, amely reprezentál egy játékost. Tartalmazza a játékos nevét és pontszámát. Az osztály rendelkezik néhány virtuális tagfüggvénnyel, például a `GetNextShape()`, amely az aláosztályokban implementálva van, és visszaadja a játékos következő választását a szabályok alapján.
- `Human`: Az `Player` osztályból származó leszármazott osztály, amely a felhasználót képviseli. Interaktív játékot tesz lehetővé, és a `GetNextShape()` függvény implementációjával kéri be a felhasználótól a következő alakzatot.
- `CleverStrat`: Az `Player` osztályból származó leszármazott osztály, amely egy okos stratégiát valósít meg. Egyedi következtetéseket von le az ellenfél választásából, és ezek alapján határozza meg a saját választását a `GetNextShape()` függvénnyel.
- `CopyStrat`: Az `Player` osztályból származó leszármazott osztály, amely az ellenfél legutóbbi választását másolja. Az osztály tárolja az ellenfél választását, és a `GetNextShape()` függvénnyel mindig ugyanazt a választást adja vissza.
- `RandomStrat`: Az `Player` osztályból származó leszármazott osztály, amely véletlenszerűen választ az elérhető alakzatok közül a `GetNextShape()` függvénnyel.
- `PredictiveStrat`: Az `Player` osztályból származó leszármazott osztály, amely a játék állásának alapján próbálja megjósolni az ellenfél választását. Tárolja az ellenfél legutóbbi választását és a saját választását, majd a `GetNextShape()` függvénnyel dönti el, hogy milyen alakzattal reagáljon az ellenfélre.

Változók:

- `name`: Karaktertömb, amely tárolja a játékos nevét.
- `points`: Egész szám, amely tárolja a játékos pontszámát.
- `myShapes`: Egész tömb, amely tárolja a CleverStrat osztály által választott alakzatokat.
- `mscount`: Egész szám, amely nyomon követi a CleverStrat osztály által választott alakzatok számát.
- `opponentShapes`: Egész tömb, amely tárolja az ellenfél választott alakzatait a CleverStrat osztály számára.
- `oscount`: Egész szám, amely nyomon követi az ellenfél választott alakzatainak számát.
- `opponentShape`: Egész szám, amely tárolja a CopyStrat osztály által másolt ellenfél választást.
- `actualGame`: Game típusú pointer, amely az aktuális játékot tárolja a PredictiveStrat osztály számára.

Tagfüggvények:

- `SetPlayerName(char* n)`: Beállítja a játékos nevét a megadott karakterláncra.
- `GetNextShape(Rules&)`: Virtuális tagfüggvény, amely visszaadja a játékos következő választását a szabályok alapján.
- `OpponentsShape(int s)`: Rögzíti az ellenfél választását a megadott alakzattal.
- `StartGame(Game* g)`: Elindítja a játékot, inicializálja a szükséges változókat.
- `GetPlayerName()`: Visszaadja a játékos nevét.
- `GetStrategyName()`: Virtuális tagfüggvény, amely visszaadja a játékos stratégiájának nevét.
- `AddPoints(int p)`: Hozzáadja a megadott pontszámot a játékos aktuális pontszámához.
- `GetPoints()`: Visszaadja a játékos aktuális pontszámát.