Kő-papír-olló

Kerek Dominik

2021. 05. 09.

Tartalom

1. Feladat

A feladat a kő-papír-olló játék modellezése lesz. Cél az, hogy a különböző stratégiákat alkalmazó játékosok összesorolva megállapítsuk, melyik a legjobb stratégia.
A modellben szerepel egy „Játékos” és egy „Napló” objektum. Utóbbi a statisztikáért felel.

2. Feladatspecifikáció

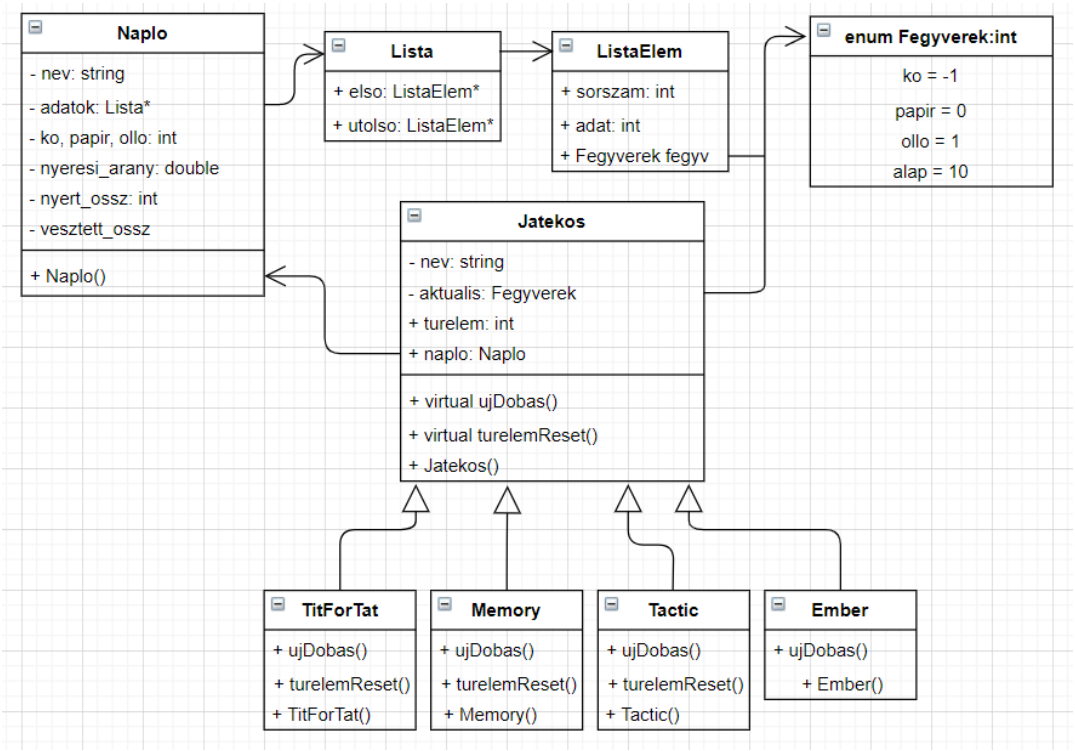
A feladat a kő-papír-olló játék szimulálása. Egyszerű konzolos megjelenítéssel találkozik a felhasználó. Menüvezérléssel lesz irányítható a játék. Főbb menüpontok:
- adatok kezelése (törlése)
- bot játék (folytatása)
- játék a gép ellen (folytatása)
- statisztika

A program összetételét tekintve két fő részre oszlik, az egyik rész a játékosok, amik előre definiált, virtuális játékosok játékstílusait határozza meg. Különböző módon viselkednek és reagálnak a játszmák előrehaladtával. Ezért lesz felelős majd a másik rész, a naplózás, adatgyűjtés, a folyamatosan gyűjtött adatok alapján működnek majd a játékoksok. (pl. egy-egy játékos statisztikái alapján, vagy éppen az alapján mit lépett az ellenfél előzőleg, esetleg az ellenfél utolsó x db mutatása alapján lép) Ezek mellett lehetőség lesz a virtuális játékosok ellen játszani, miközben a program statisztikát gyűjt a felhasználóról, amit később a statisztikák menüpont alatt lehet majd megtekinteni.
A két fő egység egy-egy osztályt fog képviselni. A program a játékosok statisztikáit fájlokba menti ki, hogy folytatni lehessen ezek alapján a játékot.

3. Terv

A kő, papír, olló játék szimulálásához szükséges lesz két nagyobb objektumra. Egyik a Jatekos másik a Naplo lesz. A Jatekos-ból fog öröklődni minden játékos típus, pl. Ember és botok.

3.1. Objektum terv



A Bot osztályban a különböző, előre megírt játékmódszerek külön függvényekben kapnak helyet. Az Ember típus a felhasználótól bekért lépésekkel operál majd.

A Jatekos osztály virtuális függvényeket tartalmaz, így ezeket külön-külön lehet majd definiálni a Bot és Ember típusokra.

4. Algoritmusok és menüpontok

4.1. Játék alapja

A következő függvényen alapul a program, eldönti, hogy melyik nyert a paraméterként kapott két játékos közül. A *get_aktual()* Jatekos classban megtalálható és az aktuális fegyvert adja vissza egy-egy játékosnál. A *CSATA* 0-át ad vissza, ha döntetlen, 1-et ha az első paraméterként kapott játékos nyert és 2-t ha a második.

```
int CSATA(Jatekos *p1, Jatekos *p2) {
    int temp = (int) p1->getAktualis() + (int) p2->getAktualis();
    if (p1->getAktualis() == p2->getAktualis()) { return 0; }
    else if (temp == 20) { throw "hiba"; }
    //ko vs papir --> papir nyer
    if (temp == -1) {
           if (p1->getAktualis() == Fegyverek::papir) { return 1; }
           else { return 2; }
    }
    //ko vs ollo --> ko nyer
    else if (temp == 0) {
           if (p1->getAktualis() == Fegyverek::ko) { return 1; }
           else { return 2; }
    }
    //papir vs ollo --> ollo nyer
    if (temp == 1) {
           if (p1->getAktualis() == Fegyverek::ollo) { return 1; }
           else { return 2; }
    }
    return -1;
}
```

4.2. Adatok kezelése

A program automatikusan menti a játék adatokat a felhasználónál és a botoknál is. Ezeket az adatokat lehetőség van törölni ebben a menüpontban. Itt a program a kiválasztott játékosok adatfájlait törli.

4.3. Bot játék

Ebben a menüpontban sok minden történik. A felhasználó eldöntheti, hogy melyik 2 előre megírt játsszon egymás ellen, hány körön keresztül.

A kiválasztás után az *ujKorok* hívódik meg, amelyben a *JATEK*, ez a függvény levezényli az adott kört, úgy, hogy döntetlen nem lehet. A botoknak van egy türelmi szintjük (3 kör), ez azt jelenti, ha 3 körben egymás után döntetlent léptek, utána random fognak lépni.

A botok játékát figyelemmel kísérhetjük, láthatjuk melyik bot mit lépett és mikor játszottak döntetlent.

4.4. Játék egy bot ellen

Itt lehetőségünk adódik felhasználóként a botok ellen játszani. Hasonlóan a *Bot játéknál* itt is ki kell választani az adott botot és megadni a körök számát.

Ezután a program mindig meg fogja kérdezni a felhasználót a következő lépésről. A játék menetéről szintén az *ujKorok* és a *JATEK* függvények gondoskodnak.

4.5. Statisztika

A játékosokról folyamatosan gyűjtött adatokat ebben a menüpontban lehet megtekinteni.

Összehasoníthatjuk egyes játékosok nyerési arányait százalékos formában. Emellett minden játékosról megtudhatjuk mit választ a legtöbbet, mennyi meccset játszott összesen, ebből mennyit nyert és vesztett.

Ezekért az adatok kezelésért a Naplo osztály a felelős, a *statisztika* függvény az osztályban az adatok kiírását segíti.

5. Osztályok bemutatása

5.1. Jatekos

```
class Jatekos {
protected:
	std::string nev;
	Fegyverek aktualis;
public:
	int turelem;
	Naplo naplo;

	//konstruktorok
	Jatekos() : naplo() {
		nev = "none";
		aktualis = Fegyverek::alap;
		turelem = 3;
	}
	Jatekos(const std::string& n) : naplo(n) {
		nev = n;
		aktualis = Fegyverek::alap;
	}

	//get függvények
	std::string getNev() { return nev; }
	Fegyverek getAktualis() { return aktualis; }

	/*Segéd függvények*/
	void aktualis_kiir(); //kiírja a képernyőre az aktuális fegyvert
	void hozzarendel(int kapott); //a random lépésnél segít

	virtual void ujDobas(Jatekos* ellenfel) = 0;
	virtual void turelemReset() = 0;

	void nyert();
	void vesztett();

	void kiir() { naplo.kiir(); }
	void adatKiir() { naplo.adatKiir(); }

	virtual ~Jatekos() {}
};
```

5.1.1. TitForTat

```
class TitForTat : public Jatekos {
public:
	TitForTat() : Jatekos() {}
	TitForTat(const std::string& nev) : Jatekos(nev) {
		turelemReset();
	}

	void hozzarendel(int kapott) { Jatekos::hozzarendel(kapott); }
	void aktualis_kiir() { Jatekos::aktualis_kiir(); }

	//a viselkedést leíró fgv
	void ujDobas(Jatekos* ellenfel);
	void turelemReset();

	~TitForTat() {}
};
```

Az alábbi kódsor jellemzi a TitForTat (Zsoldos) bot lépését. Ha az ellenfél naplója nem üres és türelme van, akkor az ellenfél naplójába feljegyzett utolsó lépést adja vissza.

```
void TitForTat::ujDobas(Jatekos *ellenfel) {
	if (turelem <= 0 || ellenfel->naplo.naploUres()) {
		int rnd = rand() % 3 + -1;
		hozzarendel(rnd);
	}
	else {
		//visszaadja az ellenfel előző lépését
		aktualis = ellenfel->naplo.utolso();
	}
}
```

A TitForTat (Zsoldos) bot türelme 3 kör.

```
void TitForTat::turelemReset() {
	turelem = 3;
}
```

5.1.2. Memory

```
class Memory : public Jatekos {
public:
	Memory() : Jatekos() {}
	Memory(const std::string& nev) : Jatekos(nev) {
		turelemReset();
	}

	void hozzarendel(int kapott) { Jatekos::hozzarendel(kapott); }
	void aktualis_kiir() { Jatekos::aktualis_kiir(); }

	//a viselkedést leíró fgv
	void ujDobas(Jatekos* ellenfel);
	void turelemReset();

	~Memory() {}
};
```

Az alábbi kódsor jellemzi a Memory (Veteran) bot lépését. Ha az ellenfél naplója nem üres és türelme van, akkor az ellenfél naplóját megvizsgálva, az ellenfél által legtöbbször választott lépés ellen lép.

```
void Memory::ujDobas(Jatekos* ellenfel) {
	if (turelem <= 0 || ellenfel->naplo.naploUres()) {
		int rnd = rand() % 3 + -1;
		hozzarendel(rnd);
	}
```

```
	else {
		Fegyverek tmp = ellenfel->naplo.legtobbszorMutatott();
		if ((int)tmp == -1) aktualis = Fegyverek::papir;
		else if ((int)tmp == 0) aktualis = Fegyverek::ollo;
		else if ((int)tmp == 1) aktualis = Fegyverek::ko;
	}
}
```

A Memory (Veteran) bot türelme 5 kör.

```
void Memory::turelemReset() {
	turelem = 5;
}
```

5.1.3. Tactic

```
class Tactic : public Jatekos {
public:
	Tactic() : Jatekos() {}
	Tactic(const std::string& nev) : Jatekos(nev) {
		turelemReset();
	}

	void hozzarendel(int kapott) { Jatekos::hozzarendel(kapott); }
	void aktualis_kiir() { Jatekos::aktualis_kiir(); }

	//a viselkedést leíró fgv
	void ujDobas(Jatekos* ellenfel);
	void turelemReset();

	~Tactic() {}
};
```

Az alábbi kódsor jellemzi a Tactic (Ujfiu) bot lépését. Ha az ellenfél naplója nem üres és türelme van, akkor az ellenfél naplóját megvizsgálva, az ellenfél által az elmúlt 10 vagy kevesebb körben nyerőnek számító lépést adja vissza.

```
void Tactic::ujDobas(Jatekos* ellenfel) {
	if (turelem <= 0 || ellenfel->naplo.naploUres()) {
		int rnd = rand() % 3 + -1;
		hozzarendel(rnd);
	}
	else {
		aktualis = ellenfel->naplo.legtobb_utolso10();
	}
}
```

A Tactic (Ujfiu) bot türelme 4 kör.

```
void Tactic::turelemReset() {
	turelem = 4;
}
```

5.1.4. Ember

```
class Ember : public Jatekos {
public:
	Ember() : Jatekos() {}
	Ember(const std::string& nev) : Jatekos(nev) {
		turelemReset();
	}

	void aktualis_kiir() { Jatekos::aktualis_kiir(); }
	void ujDobas(Jatekos* ellenfel); //be kér egy lépést felhasználótól
	void turelemReset() { turelem = 0; }
	~Ember() {}
};
```

Az alábbi kódsor a felhasználó által megadott lépések kezelésére szolgál.

```
void Ember::ujDobas(Jatekos* ellenfel)
{
	int temp;
	cout << "Add meg a lepesed: ";
	cout << "\n1. KO"
		"\n2. PAPIR"
		"\n3. OLLO" << endl;
	cin >> temp;
	if (temp == 1) { aktualis = Fegyverek::ko; }
	else if (temp == 2) { aktualis = Fegyverek::papir; }
	else if (temp == 3) { aktualis = Fegyverek::ollo; }
	else { aktualis = Fegyverek::alap; }
}
```

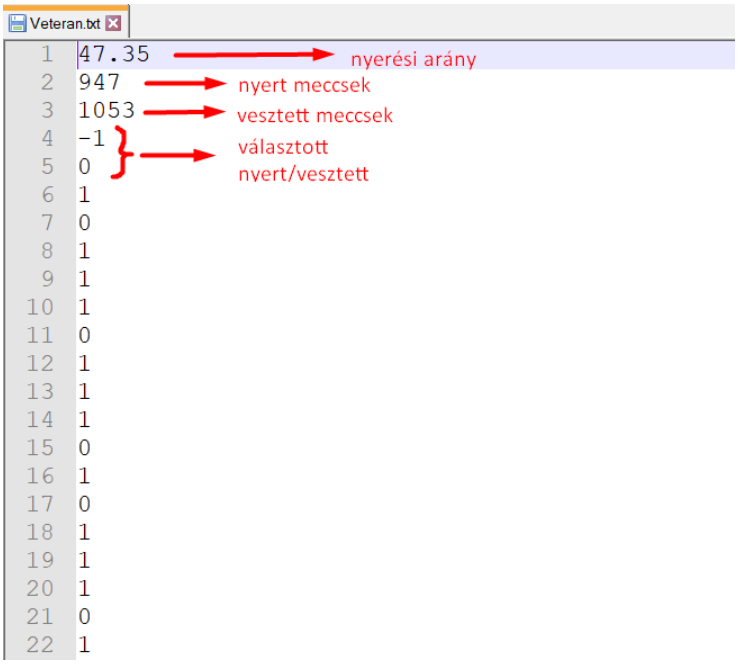
5.2. Naplo

```
class Naplo {
protected:
	std::string nev;
	Lista* adatok;
	int ko, papir, ollo;

	//kalkulált adatok
	double nyeresi_arany;
	int nyert_ossz;
	int vesztett_ossz;

public:
	Naplo() {
		nyeresi_arany = 100;
		nyert_ossz = 0;
		vesztett_ossz = 0;
		ko = 0;
		papir = 0;
		ollo = 0;
		adatok=letrehoz(adatok);
	}
	Naplo(const std::string& n) {
		nev = n;
		nyeresi_arany = 100;
		nyert_ossz = 0;
		vesztett_ossz = 0;
		ko = 0;
		papir = 0;
		ollo = 0;
		adatok = letrehoz(adatok);
		adatok=letrehoz(adatok);
		fajlBeolvas(); }
```

```
        /*Adatkezelés*/
        void kiir(); //kiírja a képernyőre az adott játékos statisztikáit
        void adatKiir(); //elementi egy txt-be az adott játékos statisztikáit
        void fajlBeolvas();
        void ujAdat(int adat, Fegyverek fegyv);
        void ujAdatVissza(int adat, Fegyverek fegyv);


        /*Segéd függvények*/
        bool naploUres(); //eldönti üres e a játkos naplója
        Fegyverek utolso(); //megadja egy játékos utolsó lépését
        std::string fegyverToString(Fegyverek fegyv);
        void osszehasonlit();
        void megszamol();
        double aranySzamol();
        void statisztika();
        Fegyverek max();

        Fegyverek legtobb_utolso10();
        Fegyverek legtobbszorMutatott();

        /*A láncolt listát kezelő függvények*/
        Lista* letrehoz(Lista* list);
        void hozzafuz(Lista* list, int adat, Fegyverek fegyv);
        void felszabadit(Lista* list);

        virtual ~Naplo() { felszabadit(adatok); }
};
```

A Naplo osztály felelős a játékosok adatainak kezeléséért. Az adatokat a játék végeztével a program fájlba menti, a fájlok neve megegyezik a játékosok nevével.

5.2.1. Fájlkezelés



A fenti képen látható milyen formátumban menti az adatokat a program. Az első szám jelöli az adott játékos nyerési arányát, utána a nyert meccsek és vesztett meccsek száma.

Ezután a számokat párosával kell értelmezni, egy-egy számpár egy kört reprezentál. Az első a körben választott fegyvert jelöli (Fegyverek osztály), a második 0 ha veszett és 1 ha nyert a játékos. Az adatok visszaolvasásáért a *fajlBeolvas()* függvény felel.

5.2.2. Lista

A Naplo osztályban szereplő Lista* adattag egy duplán láncolt, strázsás láncolt listára hivatkozik.

6. Tesztelés

A feladat során elkészített kő-papír-olló játék manuálisan tesztelhető, kipróbálható.

7. Memóriakezelés tesztje

A memóriakezelés ellenőrzését a laborgyakorlatokon használt MEMTRACE modullal végeztem. Ehhez minden önálló fordítási egységben include-oltam a memtrace.h állományt a standard fejlécállományok után. Memóriakezelési hibát nem tapasztaltam a futtatások során.