

Részvénypiac Szimuláció - Specifikáció Skeleton Programhoz

Rátki Barnabás

2021.04.13

Tartalomjegyzék

1 Feladat	1
1.1 Rövid összefoglaló a problémáról	1
1.2 Bemenetek	2
1.3 Beállítások	2
1.4 Interaktív mód	2
1.4.1 Szimuláció megállítása/elindítása	2
1.4.2 Statisztikák lekérése	2
1.4.3 Cégek listázása	3
1.4.4 Cégek részletes adatai	3
1.4.5 Kereskedők listázása	3
1.4.6 Kereskedők részletes adatai	3
1.5 Kimenetek	3
2 Pontosított feladatspecifikáció	3
3 Terv	3
3.1 Objektum Terv	3
3.1.1 Könyvtár - Terv	3
3.1.2 Rc<T> - Implementáció	6
3.2 Implementációs - Terv	6
4 Megvalósítás	6
4.1 NormalTraderAgent	7
4.2 .config	7
5 Tesztelés	7
6 Programozói dokumentáció - Doxygen	7

1 Feladat

1.1 Rövid összefoglaló a problémáról

A program célja egy képzeletbeli részvénypiac szimulációja.

Megjegyzés: Nem cél a valós rendszer közeli modellezése, mert ez valószínűleg lehetetlen, a feltételezések, amikre a szimuláció épül nem a valóságon alapulnak és lényeges egyszerűsítéseket tartalmaznak.

Az elképzelt piacunkban két különböző szereplő van, cégek illetve kereskedők. A részvények tulajdonos-cseréjét pedig egy központosított piac biztosítja. Az összes többi piaci körülményekben változást elérő hatást véletlenszerű események fogják generálni. A szimulációban arra vagyunk kíváncsiak, hogy az adott beállításokkal és random seed-el milyenek lesznek az árfolyamok grafikonjai a különböző cégeknél és mennyi tőkéjük lesz a kereskedőknek. A kereskedők kezdetben random generált beállításokkal működnek, amik egyedivé teszik az akcióikat és viselkedésüket de összeségében az a céljuk, hogy a maximalizálják profitjaikat. A cégek hasonlóan random generáltak, és a szimuláció elején fix mennyiségű részvényt bocsájtanak ki és igyekeznek maximalizálni az így beszedett pénz mennyiségét. A rendszer ciklusokban működik, minden ciklusban létrejöhetnek események, illetve random aktivált

kereskedők kezelhetik befektetéseik. A piac egy úgynevezett "Aukciós Piac" elvén működik, ahol a kereskedők ajánlatainak egyezése esetén történnek eladások és vételek.

1.2 Bemenetek

A program egy parancsoros interfacen keresztül használható aminek a standard bemeneten lehet megadni különböző beállításokat Ezeket egy egyszerű kulcs-érték pár listaként várja a következő formában: *KULCS=ÉRTÉK* a párok pedig valamilyen "whitespace" karakterrel kell, hogy elválasztva legyenek. A felhasználó dupla "whitespace" karakter bevitelével tudja jelezni, hogy több beállítást nem szeretne megadni.

1.3 Beállítások

Kulcs / Tipus	Leírás
INTERACTIVE_MODE (Bool)	A program interaktív módban vagy limit* módban fusson.
CYCLE_LIMITS (Int)	Maximum mennyi ciklusig tarthat a szimuláció.
SEED (Int)	Mi legyen a random <i>seed</i> **
LOG_LEVEL (Int)	Mennyire legyen a standard kimenetre való naplózás részletes.
TRADER_COUNT (Int)	Hány random generált kereskedők legyen.
COMPANY_COUNT (Int)	Hány random generált cég legyen.
EARNINGS_CYCLES (Int)	Mennyi ciklusonként legyen "earnings" jelentése a cégeknek átlagosan.
DIVIDEND_CYCLES (Int)	Mennyi ciklusonként legyen dividendns fizetés átlagosan.
TRADER_MONEY (Int)	Mennyi legyen a kereskedők átlag kezdő vagyona.
TRADER_INCOME (Int)	Mennyi legyen a kereskedők átlag keresete.
MEDIAN_IPO (Int)	Mennyi legyen a medián kezdetleges részvénykibocsájtás részvényenkénti ára.
PRICE_SAMPLING_RATE (Int)	A szimuláció hűny ciklusonként mintavételzze a cégek árfolyamát. (Kimenet felbontása)

* - Limit módban a program, menü megjelenítése nélkül *CYCLE_LIMITS*-nyi cikluson keresztül fut aztán kilép.

** - A programban ez az egyetlen véletlenszerű forrás, egyébként azonos seedekkel determinisztikusan működik.

Az összes beállítás opcionális és implementáció függő alapértelmezett értékekkel rendelkezik. Az implementáció támogathat még ezeken kívül más beállításokat is. Illetve a program "-help" parancssori argumentummal történő futtatására kiírja a támogatott beállításait és egy rövid használati útmutatót.

1.4 Interaktív mód

Interaktív módban való futtatáskor a szimuláció aktuális állapota egy menü segítségével megfigyelhető. A továbbiakban az ebből a menüből elérhető funkciókat ismertetem:

1.4.1 Szimuláció megállítása/elindítása

El lehet indítani a szimulációt, a ciklusok addig fognak futni ameddig a felhasználó be nem ír valamit a standard bemenetre, illetve előre is meg lehet adni, hogy hány ciklust menjen mielőtt újra megállna. A programot véglegesen is le lehet állítani.

1.4.2 Statisztikák lekérése

Meg lehet tekinteni aktuális statisztikákat a szimulációról, mennyi pénz kering a rendszerben, melyik a legnagyobb cég illetve, ki a leggazdagabb kereskedő, hányadik ciklusban van.

1.4.3 Cégek listázása

Ki lehet listázni a cégeket, szimbólumaikkal, teljes nevükkel illetve jelenlegi árfolyamaikkal.

1.4.4 Cégek részletes adatai

Le lehet kérni egy cég részletes adatait, rejtett tulajdonságaival és visszamenőleges árfolyam adatokkal szimbólum alapján.

1.4.5 Kereskedők listázása

Ki lehet listázni a kereskedőket, nevükkel, egyenlegeikkel, portfólióik méretével illetve egyedi azonosítójaikkal.

1.4.6 Kereskedők részletes adatai

Le lehet kérni egy kereskedő részletes adatait, beállításait, összes nyitott pozícióját a kereskedő egyedi azonosítója alapján.

1.5 Kimenetek

A szimuláció végeztével, a program az elmentett árfolyam, cég és kereskedő információt kiírja JSON formátumban. A konkrét formátum implementáció függő de alapvetően hasonló adatokat tartalmaz, mint ami az interaktív menüben elérhető, csak teljesen részletesen.

2 Pontosított feladatspecifikáció

A laborvezetőnek eddig semmilyen változtatási igénye nem merült fel, ezért az munkát az eredeti tervek alapján folytattam. A következőkben az implementáció részleteivel fogok foglalkozni.

A feladatban elkészített osztályok mezői és függvényei már megtervezésre kerültek, de mivel egy szimulációs feladatról beszélünk, az összes algoritmus konkrét implementációja nem tervezhető meg előre. Mivel ez gyakorlatilag a teljes feladat befejezését igényelné, minden componens belső működése előre nem látható hatásokkal lehet a többi komponens működésére. Ez a tulajdonság teszi magát a szimulációt érdekessé.

A feladat megoldásához nem használok **STL** tárolókat a számelkezeléshez szükséges `std::mutex` osztályon, és más ehhez kapcsolódó és szükséges primitíveken kívül. (Illetve még a `std::function` segéd típust is esedlegetesen alkalmazom de ez nem valódi tároló.)

3 Terv

Az implementáció két részből áll, egy futtatható standard bemenetetn keresztül irányítható programrészből, ami magát a szimulációt is futtatja (ez a program készült végfelhasználásra), illetve egy test programból ami a fordításnál definiálandó `TEST_VAR` szimbólum `ETEST`-értékre való beállításával elérhető. A testprogram a szükséges tárolókat, segéd osztályok működését teszteli, nem célja a teljes kódlefedettség. A program nem könyvtárként való használatra készült, ezért minden hiba amit dob az `std::runtime_error` osztály példánya.

3.1 Objektum Terv

3.1.1 Könyvtár - Terv

A program egy kisebb sablon könyvtárat tartalmaz, a könyvebb fejleszhetőség érdekében. A könyvtár újradefiniáls numerikus típusokat is, hogy azok könyvebben használhatóak legyenek. Az alábbiak ezek a definíciók:

```
typedef std::size_t  usize;
typedef unsigned int  u32;
typedef uint8_t      u8;
typedef uint16_t     u16;
typedef uint32_t     u32;
typedef uint64_t     u64;

typedef int8_t       i8;
typedef int16_t      i16;
typedef int32_t      i32;
typedef int64_t      i64;

typedef float         f32;
typedef double        f64;
typedef long double   f128;
```

Az itt definiált osztályok definícióját és egymáshoz való függőségeit az alábbi UML diagram ábrázolja:

3.1.2 Rc<T> - Implementáció

Egy reference számolt, T típusu dinamikusan tárolt pointert tartalmaz. A reference számolást egy dinamikusan tárolt *usize* pointerrel valósítja meg. Egy *Rc<T>* másolásánál, a reference számláló növekszik egyvel illetve a pointerek másolva vannak. Destrukciónál a reference számláló csökkentve van egyvel, és a belsőleg tárolt adat csak akkor van felszabadítva ha a reference számláló nulla.

3.2 Implementációs - Terv

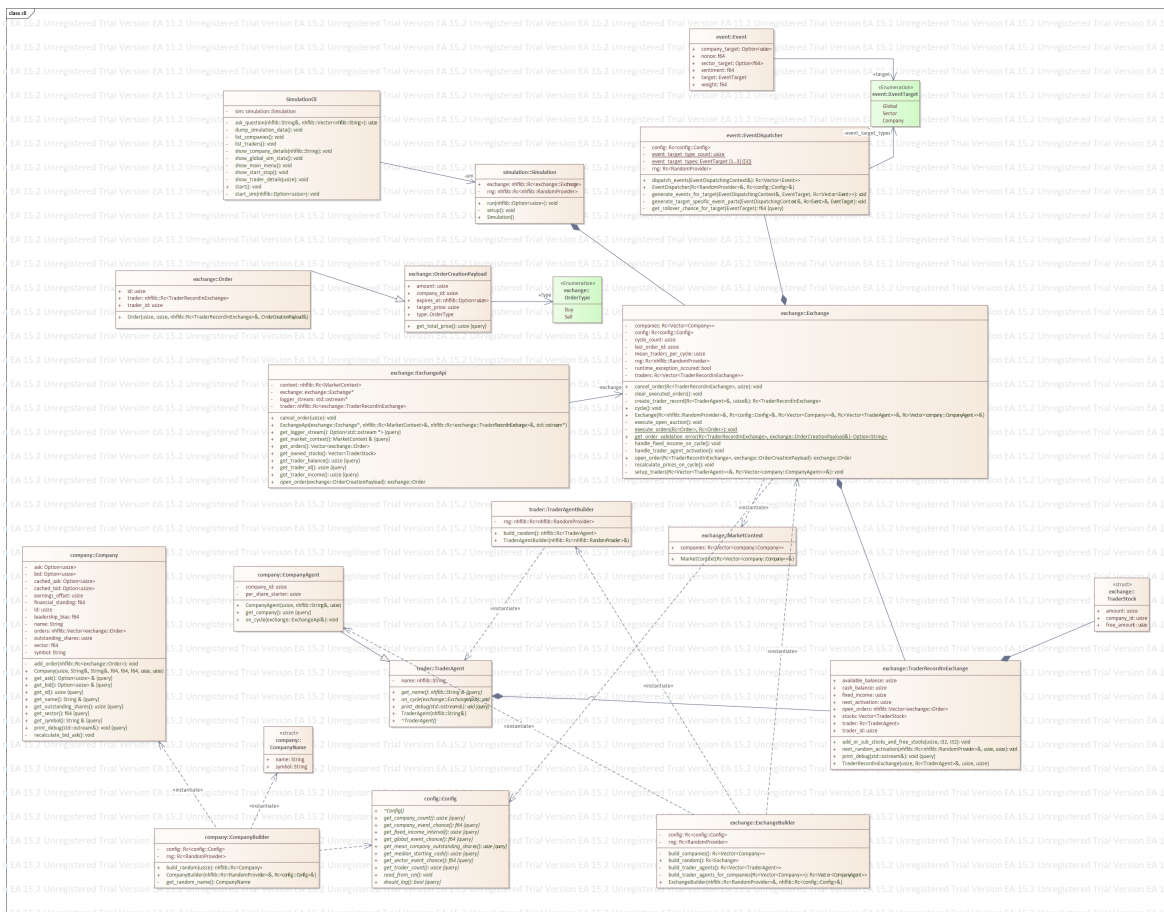
A szimuláció főleg kompozícióval épül fel, a proram a *SimulationCli* osztály példányának létrehozásával és a *start* függvényének meghívásával indul el. A szimuláció osztály *run* függvénye egy második szállon fut, és egy mutex segítségével van a futási állapota szinkronizálva a CLI-vel.

Alapvetően a szimulációban lévő kereskedők egy heterogén kollekcióban vannak tárolva, minden osztály ami kompatibilis a *TraderAgent* osztállyal működhet kereskedő "intelligenciaként". A cégek egy speciális *TraderAgent* implementáción keresztül adják el az első részvényeiket (IPO szimuláció amit brókerek kezelnek).

A kereskedők random időközönként aktiválódnak és az *on_cycle* függvényük meghívódik, a paraméterként kapott *ExchangeApi*-n keresztül tudnak megnyitni új pozíciókat és információt szerezni a piaci körülményekről a döntéshozáshoz.

Alapvető feltételezés, hogy a *TraderAgent* nem megbízható, tehát ha tud akkor akár csalni is fog, ezért úgy került megtervezésre, hogy a benne futó kódban ne kelljen megbízni. Nem dobhat hibákat, és semmilyen "szenzitív" adathoz nem fér hozzá.

Az osztálydiagram sokkal beszédesebben leírja a különböző kompozíciókat:



4 Megvalósítás

A megvalósításhoz pár extra segéd osztály bevezetésére volt szükség: *CliHelper* - standard kimenetre való kiírást kezel, biztonságos beolvasást implementál. *CliQuestioner* - feleletválasztós felhasználói bevétel feldolgozását segítő

osztály. *CliTableBuilder* - standard kimenetre írandó táblázatok építéséért felelős osztály. Ezeken kívül a *TraderAgent* már csak a kereskedő intelligencia őszosztályaként szerepel, az intelligens megvalósítást a *NormalTraderAgent* tartalmazza erről lejjebb kicsit bővebben.

4.1 NormalTraderAgent

A kereskedő intelligencia egy egyszerű algoritmust követ, aminek az az alapja, hogy minden példány random, rejtett adatokkal van létrehozva és ezeket felhasználva reagál az eseményekre. Minden kereskedő fentart egy indikátor értéket minden céghez, amit eredetileg csak az elérhető fundamentális adatokból képez. A szimuláció közben pedig minden aktivációjakor frissíti ezt, események beérkezésére az esemény típusa és súlya alapján. Ár illetve részvény volumen változásokra is reagál, különböző súlyok és diszpozíciók alapján. Megbízásait folyamatosan frissíti az indikátora változása és a megbízás teljesíthetősége miatt.

4.2 .config

A másik újítás, az hogy a program már a munka könyvtárban elhelyezett *.config* fileokat is feldolgozza, konfigurációs kulcs érték párokat olvas be. Ez a fejlesztés megkönnyítést szolgált tenni, a *SKIP_CIN_CONFIG* config változó beállításával a standard bemenetről való konfiguráció teljesen át is léphető.

5 Tesztelés

A program által használt egyedi könyvtár és a legfőbb működés is automata tesztekkel van lefedve a *g_test_lite.h* könyvtár segítségével. A *ETEST* definícióval lefordított program a teszteseteket futtatja le a rendes működés helyett. A beadáshoz a standard input megadását választottam viszont interaktív módban mert így a parancssoros interfész is tesztelve van.

6 Programozói dokumentáció - Doxygen