

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

SWARM szimulátor

Szoftverarchitektúrák házi feladat

Konzulens

Dr. Blázovics László

BUDAPEST, 2021

Tartalomjegyzék

[1 Az alkalmazás célja, funkciói és környezete 3](#_Toc88391927)

[1.1 Feladatkiírás 3](#_Toc88391928)

[1.2 Az alkalmazás funkciói 3](#_Toc88391929)

[1.3 A szoftver környezete 3](#_Toc88391930)

[2 Megvalósítás 4](#_Toc88391931)

[2.1 Architektúra 4](#_Toc88391932)

[2.1.1 Adat beolvasási réteg 4](#_Toc88391933)

[2.1.2 Üzleti logika réteg 4](#_Toc88391934)

[2.1.3 Felhasználói felület 4](#_Toc88391935)

[2.2 Performancia optimalizálás 5](#_Toc88391936)

[2.2.1 Look és Compute fázis 5](#_Toc88391937)

[2.2.2 Move fázis 5](#_Toc88391938)

[2.3 Implementált viselkedési formák 5](#_Toc88391939)

[3 Az alkalmazás telepítése, használata 7](#_Toc88391940)

[3.1 Az alkalmazás funkciói 7](#_Toc88391941)

[3.2 Példa bemenet 7](#_Toc88391942)

[4 Összefoglaló 8](#_Toc88391943)

[Felhasznált eszközök 9](#_Toc88391944)

[Továbbfejlesztési lehetőségek 10](#_Toc88391945)

# Az alkalmazás célja, funkciói és környezete

## Feladatkiírás

A feladat célja egy diszkrét idejű, diszkrét, véges állapotterű szimulációs környezet megvalósítása, amely akár 1000 egyed egyidejű viselkedését képes szimulálni.

## Az alkalmazás funkciói

A szimuláció elindítását a felhasználó parancssorból végezheti el, amely során meg kell adnia egy előre definiált leíró nyelven (továbbiakban: script) készült fájlt, amely tartalmazza az állapottér inicializációját, az egyes egyedekhez tartozó számítási szabályokat, illetve egyéb, a szimulációval kapcsolatos konstansok beállításait.

A szimuláció megállítását a szimuláció futása közben a megfelelő gomb lenyomásával történik meg, amely lehetőséget ad a betekintésre, illetve

A követelmények pontos definíciói elérhetőek a *Követelményspecifikáció* dokumentum *Követelmények* fejezetében.

## A szoftver környezete

Mivel a cél egy olyan alkalmazás fejlesztése volt, amely akár nagy egyedszám mellett is valós idejű szimulációra képes, ezért - az ilyen jellegű problémákhoz legjobban illeszkedő - C++20 nyelvet választottuk.

A nyelv által platformfüggetlen környezetet kapunk, amely által az alkalmazás akár egy beágyazott rendszer részeként is üzemelhet, illetve némi módosítással (pl.: C++/CLI wrapper) beépíthető olyan környezetbe, amelyben a fejlesztés hatékonyabban történhet (pl.: C#, Python).

# Megvalósítás

Az ún. Look-Compute-Move paradigmának megfelelően terveztük meg a rendszert úgy, hogy a valós idejű működés mellett arra is koncentráltunk, hogy a jövőben történő kibővítése a rendszernek (azaz a lehetséges szimulációk terének bővítése) rendkívül egyszerű legyen.

## Architektúra

Az architektúrát három rétegű alkalmazásként készítettük el. Az alkalmazás rétegei az alábbiak:

* Adat beolvasási réteg (Data Acquisition Layer)
* Üzleti logika réteg (Business Logic Layer)
* Felhasználói felület (User Interface Layer)

### Adat beolvasási réteg

**Célja**: Az inicializáló script beolvasása, illetve az üzleti logika számára megfelelő formátumra hozása.

Mi egy saját, egyszerű leíró nyelvet konstruáltunk a beolvasáshoz. Ez igény szerint bővíthető egyéb formátumokkal (pl.: JSON, XML).

**Megjelenés a kódban:**

### Üzleti logika réteg

**Célja:** A szimuláció inicializálása a struktúrált adatok beolvasása után, illetve a szimuláció futtatása.

**Megjelenés a kódban:**

### Felhasználói felület

**Célja:** A szimuláció eredményének ismertetése a felhasználóval.

Esetünkben a felhasználói felület a parancssorból adott eseményeket kezeli le, illetve egy fájlba logolja a szimuláció eredményét.

**Megjelenés a kódban:**

## Performancia optimalizálás

Az alkalmazásunk esetében a performancia kitüntetett szerepű. Emiatt célszerű a lokalitás elvét betartva, minél kevesebb cache-hibát generálva futtatni egy-egy szimulációt. Azt egyből beláthatjuk, hogy az ágensek listáján végig kell iterálnunk, ezáltal két lehetőséget kapunk. Az első lehetőség szerint három ciklusunk van, amelyek megfelelnek a szimuláció fázisainak. A második esetben egy nagy ciklusunk van, amely minden iterációjában egy adott ágenssel teszünk egy fantom lépést, amelyet még nem mentünk el, majd valamilyen stratégia szerint frissítjük az állapotteret miután mindent kiszámoltunk. Előbbi a lokalitás elvét jobban betartja, így gyorsabb futáshoz vezet. A fázisok tervezésének motivációi az alábbiak szerint történtek.

### Look és Compute fázis

A Look, illetve Compute lépések teljesen párhuzamosíthatóak: az előbbi esetben az ágensek percepcióját frissítjük a jelenlegi világnak megfelelően, amely egy olvasást jelent, amely nem okozhatja a kiolvasandó adat bárminemű korruptálódását, utóbbi esetben pedig a számítások csak és kizárólag az ágens jelenlegi állapotától, illetve saját (már frissített) percepciójuktól függ.

### Move fázis

A Move fázisban döntenünk kell a sorrendiségről. Itt az egyedüli kitételünk a determinisztikus futás volt (pontosabban adott randomszámgenerátor seed mellett), emiatt pedig a Move fázis ilyen jellegű párhuzamosítása nem lehetséges.

## Implementált viselkedési formák

*NotMovingBehaviour*

Az ilyen viselkedés alapján döntést hozó objektumok nem mozognak a pályán, így mint akadályt hoznak létre a szimulációban.

*RandomMoveBehaviour*

Az egyed viselkedése teljesen sztochasztikus, egyenletes eloszlás alapján választ a lehetséges lépések közül.

*SequentialMoveBehaviour*

Az egyed a gráf csúcsainak számozásának megfelelően halad végig, mindig a legnagyob értékű szomszédhoz. Ha olyan csúcsra lép, amely indexénél nincs nagyobb, akkor egyenletes eloszlás szerint véletlenszerűen választ a lehetséges lépések közül.

*MoveFromEntityBehaviour*

Az egyed egy helyben marad, amíg közvetlen környezetébe nem kerül legalább egy másik egyed. Ekkor ellép onnan egyenletes eloszlás szerint véletlenszerűen lép.

# Az alkalmazás telepítése, használata

TODO

## Az alkalmazás funkciói

TODO

## Példa bemenet

TODO

# Összefoglaló

A SWARM szimulációs környezetek lehetővé teszik, hogy különböző pénzügyi és technológiai folyamatot modellezzünk. A félév során egy ilyen szimulációs környezetet terveztünk meg és implementáltunk, ahol kitüntetett szereppel bírt az ágensek valósidejű szimulációja.

A tervezett szoftver architektúrája három rétegből áll: adat beolvasási réteg, üzleti logikai réteg, felhasználói felület. Az inicializáló script felolvasása egy általunk tervezett leíró nyelven történik, amely alapján futtatni lehet a szimulációt, majd a szimuláció kimenetét meg lehet tekinteni egy logfájlban.

Jelen dokumentáció kitért a platform választási és architektúrális döntések tárgyalására, azok motivációira, illetve az alkalmazás implementációjának különböző megfontolásaira.

Felhasznált eszközök

* Visual Studio: Fejlesztőkörnyezet
* Microsoft Word: Dokumentáció elkészítése
* GitHub/git: Verziókezelő rendszer
* OpenMP: Egyszerű párhuzamosítási direktávakhoz tartozó 3rd party API.

Továbbfejlesztési lehetőségek

Az alkalmazás több rétege is bővíthető, továbbfejleszthető egyéb megoldásokkal.

Az általunk érdemesnek tartott továbbfejlesztési lehetőségek (a teljesség igénye nélkül):

* A scriptelhetőség magas szintű személyre szabhatósága egy python interop interfésszel.
* Vizuális megjelenítő a szimuláció eredményéhez.
* További architektúrális fejlesztéssel a szimuláció egy szimulációs szerveren lenne futtatható, így nagy számú szimulációt lehetne párhuzamosan futtatni

A fejlesztés során magas prioritást élvezett az architektúrális döntések befolyásolásában a fent említett fejlesztési lehetőségekre való megfelelően robosztus,