**Féléves feladat**

*Tűzeset Szimuláció*

Szoftverfejlesztés Párhuzamos Architektúrákra

Mérnökinformatikus, Msc

*Készítette:*

Liszkai Keve Áron

T0BQQL

Tartalom

[Kezdeti bevezető 4](#_Toc499457413)

[Célkitűzés 4](#_Toc499457414)

[Projekt témája 4](#_Toc499457415)

[Feladat 4](#_Toc499457416)

[Játéktér 5](#_Toc499457417)

[Tűz 5](#_Toc499457418)

[Ember 5](#_Toc499457419)

[Füst 5](#_Toc499457420)

[Egyszálas vázlat 5](#_Toc499457421)

[Több szálas vázlat 6](#_Toc499457422)

[Időmérés, összehasonlítás 7](#_Toc499457423)

[Grafikus Interfész (GUI) 7](#_Toc499457424)

[Technológiák 7](#_Toc499457425)

[Verziók 7](#_Toc499457426)

[Utólagos visszatekintés 7](#_Toc499457427)

[Feladat megoldása 8](#_Toc499457428)

[Bevezetés 8](#_Toc499457429)

[Általános 8](#_Toc499457430)

[Koordináta geometria 8](#_Toc499457431)

[Hőterjedés 9](#_Toc499457432)

[Hőterjedés matematikai leírása 9](#_Toc499457433)

[Hőterjedés példaprogram 10](#_Toc499457434)

[Ember 11](#_Toc499457435)

[Célkövetés 11](#_Toc499457436)

[Játéktér 11](#_Toc499457437)

[Program működése 12](#_Toc499457438)

[Párhuzamosítás 14](#_Toc499457439)

[Eredmények 14](#_Toc499457440)

[GUI és kezelése 17](#_Toc499457441)

[Full Benchmark 18](#_Toc499457442)

[Hivatkozások 19](#_Toc499457443)

# Kezdeti bevezető

## Célkitűzés

Nem titkolt célom az, hogy a mesterképzés színvonalának megfelelő minőségű munkát adjak be a fél éves otthoni munkám végeredményeképpen. Ezért igyekezni fogok minőségi munkát kiadni a kezemből, hogy az minél magasabb érdemjegyben részesüljön.

## Projekt témája

A párhuzamos programozás témakörében való kisebb internetes kutatásom során két témát találtam, amelyek engem komolyabban megmozgattak.

Az egyik ilyen témakör a különböző szimulációk témaköre. Sok internetes forrás ajánlotta a jól párhuzamosítható példaprogramok között a különböző gázok/folyadékok, és más fizikai jelenségek szimulációját (pl.: Gravitációs N-test probléma). Az általam érdekeseknek vélt projektek főleg a fizikai valóság lemodellezésén alapuló, és jól párhuzamosítható feladatokra épülő programok.

A másik téma, amit érdekesnek találtam, az a különböző (akár számítógépes játékokból is ismerhető) „mesterséges intelligenciák” (AI) párhuzamos kezelése. Természetesen itt egyszerűbb mesterséges intelligenciákról van szó (bár találtam forrást, ami egészen a tanuló algoritmusokig bonyolítja a dolgot), itt a hangsúly a párhuzamosíthatóságon van, hiszen minden egyes egyed (bizonyos korlátokon belül) rendelkezhet akár saját szállal, így a teljesítmény egyértelműen növelhető.

A fent említett két téma összefűzéséből született meg a projekt, amivel a félév során szeretnék foglalkozni, és úgy gondolom, hogy megfelel az elvárt szintnek.

## Feladat

Adott egy fizikai, kettő dimenziós, zárt tér (pl. iroda). A térben jelen vannak bizonyos emberek. Az adott térben kialakul egy tűzeset. A feladat szimulálni a tűz terjedését, valamint a térben lévő emberek menekülését, természetesen mindezt grafikusan is ábrázolva. Végezetül elkészíteni ennek a több szálas, valamint az egy szálas megoldását, és azokat összevetni teljesítmény szempontjából.

A feladatnak három fő alkotó része van: játéktér, ember, tűz. A későbbiekben ez a három alkotóelem lesz bemutatva. A feladat bonyolultsága nagyban függ a megvalósítás minőségétől, amely a ráfordított idővel arányos. Hiszen, ha csak arra gondolunk, hogy a fent említett három szereplő bármelyike akár lehet random generált is, de lehet beégetett is. Az egyes emberek akár rendelkezhetnek bonyolultabb pathfinding algoritmussal is, de akár még egyéni jellemzőkkel, viselkedési formákkal is. Természetesen ezek inkább a szimuláció életszerűségét növelik.

## Játéktér

A szimuláció statikus tagja. Rendelkezik legalább egy kijárattal. Rendelkezik falakkal, amik az emberek számára átjárhatatlanok, a tűz számára lassító tényező. A játéktér szobákra osztható, amelyek rendelkeznek egy, vagy több ajtóval (egyszerű pathfinding esetén ezek az ajtók lesznek a menekülési útvonal pontjai).

## Tűz

A tűznek egy kiindulási pontja van, a tűzfészek. A szimuláció paraméterei szerint lehet egy, vagy több darab, lehet random, vagy egérkattintás helyén. A tűz az epicentrumtól kifele mutató irányba, egy állandó sebességgel táguló kör. Ha a tűz útjába fal kerül, akkor az egy bizonyos időre lelassítja a tüzet, és természetesen innentől kezdve megváltozik a geometriai alakja. A tűz egyik fő tulajdonsága, hogyha elér egy embert, akkor az adott ember megég, és kiesik a szimulációból.

## Ember

A legérdekesebb, és leginkább számolásigényes tagja a szimulációnak. Lehet egy, vagy több, elhelyezkedésük lehet random, vagy egérrel elhelyezett. Az ember rendelkezik egy látótérrel, amin belül képes érzékelni a tüzet. Az emberek alapesetben random mozgást végeznek, vagy egy helyben állnak (ezeket az állapotokat akár váltogathatják is), egészen addig, amíg meg nem látják a tüzet. Miután észlelték a tüzet, a játéktér kijárata felé veszik az irányt (ez persze lehet egy bonyolultabb, vagy akár egy egyszerűbb algoritmus alapján). Amelyik ember elérte a kijáratot, az megmenekült.

Az emberek kapcsán megemlíthető esetleges színesítésként még elképzelhető akár az is, hogy az emberek képesek legyenek figyelmeztetni egymást, ha már látták a tüzet. Vagy akár még az is, hogy bizonyos emberek, meg direkt ne csinálják ezt.

## Füst

Esetleg plusz tényezőként a füst még felsorolható, mint nehezítő elem. Ezt nem tartozik szorosan szimulációhoz, azonban egy jó példa arra, hogy a szimuláció összetettsége még sok módon növelhető. A füst is egy érdekes résztvevő lehet, amely hasonlóan viselkedik a két dimenziós térben, mint a tűz, azzal a két különbséggel, hogy gyorsabb, valamint a falak számára áthatolhatatlanok.

## Egyszálas vázlat

Az egyszálas változat folyamán az egyes komponensekkel kapcsolatos műveletek (pl.: mozgatás) egymás után hajtódnak végre egyetlen közös szálban. Logikai vázlatát a következő ábra mutatja:

Ember 1 mozgatás

Ember N mozgatás

Tűz mozgatás

Füst mozgatás

Játéktér generálás

SZÁL 1

## Több szálas vázlat

A többszálas megoldás során a bevárásos szinkronizáció módszerét fogom alkalmazni. Azaz, egy főszál első lépésként legenerálja a játékteret, majd létrehoz további szálakat, amelyek mindegyik egy komponensért felelős (Ember, Tűz). Az egyes komponensek Kiszámolják a hozzájuk tartozó objektumok jövőbeli koordinátáit, valamint egyéb értékeit, és utána egy randevú ponton találkoznak. Itt a főszál a kapott koordináták segítségével frissíti a képernyőt, így a mozgás létrejön.

**Randevú pont**

Ember 1 mozgatás

Ember N mozgatás

Játéktér generálás

Tűz mozgatás

Füst mozgatás

Képernyő frissítése

Szál 1

Szál N

Szál N+1

Szál N+2

## Időmérés, összehasonlítás

Mind a két megoldás során a teljesítmény, és az idő folyamatos mérés alatt lesz, így a két szimuláció futtatása után az eredmények egyértelműen összehasonlíthatóak lesznek. Természetesen több szimuláció adatait fogom átlagolni a pontosabb végeredmény érdekében.

## Grafikus Interfész (GUI)

A grafikus megjelenéshez a Visual Studio-ban megtalálható Windows Forms Application-t fogom alkalmazni. Ez képes ablakos alkalmazások létrehozására, valamint azokon objektumok mozgatására. Úgy gondolom, hogy a projekt igényeinek eleget tud tenni.

## Technológiák

A projekthez a Visual Studio 2017 nevű fejlesztőkörnyezetet fogom használni a .NET keretrendszerrel. A projekt fejlődése nyomon követhető a következő nyilvános GitHub linken: [tuzeset-szimulacio](https://github.com/keveliszkai/tuzeset-szimulacio).

## Verziók

Természetesen, mivel a projekt nehézségének megbecslése rizikós, ezért több lehetséges verzió képzelhető el a végeredmény tekintetében, így elképzelhető, hogy a jelen dokumentum nem lesz teljes mértékben összhangban a végeredménnyel.

## Utólagos visszatekintés

Az eddig olvasott fejezet az év elején került leadásra. Úgy gondoltam, hogy utólag már nem szeretném megváltoztatni a leírást, akkor sem, ha idő közben egy-két dolgon változtatnom is kellett. Talán így ennek a bevezetőnek a fényében jobban kiértékelhető a projekt elejétől végéig tartó út.

# Feladat megoldása

## Bevezetés

A feladat kidolgozása során egyértelművé vált számomra, hogy a korábban elképzelt feladatokhoz társított nehézségi szintek nem tükrözik a valóságot, ugyanis úgy gondoltam, hogy az egyes mozgó elemek (emberek) mozgatása lesz a leg számításigényesebb, és leg bonyolultabb feladat a projekt során. Azonban rá kellett eszmélnem, hogy a tűz lemodellezése sokkal nagyobb feladat, és sokkal jobban párhuzamosítható, így ezt az elemet tekinteném mostantól a projekt főszereplőjének.

## Általános

A szoftver, amit készítettem egy felülnézetes, kettő dimenziós szimulátor. Ezért minden egyes eleme a szoftvernek közvetlen kapcsolatba van a koordináta geometriával, és elengedhetetlen volt annak szoftveres implementálása. Saját Vector osztályt hoztam létre a sokszor használt metódusok jobb áttekinthetősége érdekében.

A szoftver szerkezetileg három fő részre bontható: Modellek (*Models* mappa), Kontrollerek (*Controllers* mappa) és maga a *MainForm.cs* fájl, amit egy main függvényként használok.

### Koordináta geometria

A koordináta geometriából öt sokszor használt elemet írtam bele a Vector osztályba:

* Két pont közötti távolság
* Helyvektorok összege
* Helyvektorok különbsége
* Helyvektor szögének kiszámítása
* Helyvektor skalárral való szorzása

A fent említett alapműveletek nem bonyolultak, azonban, ha a két dimenziós térben gondolkozunk, akkor ezek a leggyakrabban használt műveletek, főleg a mozgó egyedeknél.

Az objektumokról általában elmondható, hogy a rajzolás miatt rendelkeznek egy Position tulajdonsággal, ami a bel felső sarka az adott modellnek, de mindegyiknek van külön egy Center tulajdonsága, amivel könnyebben kezelhető, hiszen a matematikában is többnyire egy testet a középpontjával pozícionálunk.

Ezek a műveletek a „Models/Vector.cs” fájlban találhatóak meg.

## Hőterjedés

A tűz természetének behatóbb utánanézését követően ráébredtem arra, hogy a keresett tématerület a hőterjedés, hőátadás területe (1). Több dokumentumot is találtam az interneten, amik a hőterjedés matematikai leírásával foglalkoznak. Professor Jung-Yang San a (2)-es hivatkozásban jól levezeti a matematikai hátterét a két dimenziós hőterjedésnek, amely nagyban elősegítette a téma megértését. Azonban mégis a szoftver megírása közben főleg a CMSoft által készített esettanulmányt vettem alapul, bár ők grafikus kártyát használtak a megoldáshoz.

### Hőterjedés matematikai leírása

A hőterjedés alaptétele (1) tetszőleges koordinátarendszerre:

, ahol

A fenti egyenletet megfelelő matematikai átalakítások, és rendezések után (felhasználva a parciális differenciálegyenleteket (3)) a következő egyenlethez jutunk:

,ahol

Talán a fenti egyenlet bonyolultnak tűnik, de ezek már jól szoftverbe implementálható egyenletek. Lényegében egy pont jövőbeli hőmérsékletének a megállapításához elegendő a vízszintes, és függőleges szomszédok hőmérsékletét venni, hiszen ilyenkor a ∆x = 1 és ∆y = 1. Ezzel a könnyítéssel a négyzetes tag is könnyebben kezelhető.

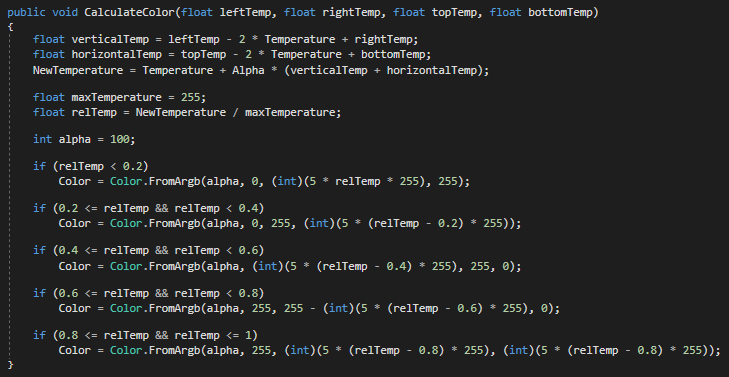
Ezeket a megállapításokat alkalmazva a következő egyenletet kapjuk:

Ahol

## Hőterjedés példaprogram

A Hőterjedés szimulációjához létrehoztam egy HeatBlock nevű osztályt. A név is utal arra, hogy a játékteret blokkokra osztottam fel, amiket egy két dimenziós tömbben tárolok. Minden tömbnek van egy hőmérséklete, amit kívülről mindenki lát, és nem változik az új hőmérséklet kiszámítása közben. Illetve természetesen van egy függvénye, ami a belső kiszámított hőmérsékletet szinkronba hozza a publikussal. Mindezzel elkerülhető, hogy valaki például akkor olvasson ki a publikus változóból, amikor éppen változik az értéke.

A hőmérséklet megjelenítéséhez, illetve a szín megválasztásához – hogy a megszokott kék/zöld/vörös hő kamerás megjelenést tudjam biztosítani – kölcsönvettem a CMSoft (4) színezési algoritmusát. Ezt azért is fontos megemlíteni, mert a fenti matematikai levezetés végül annyira leegyszerűsödött, hogy egybe vontam a szín megválasztásával. Így a kapott függvény a következőképpen néz ki:



‑. ábra - Hőmérséklet és szín meghatározása szoftveresen

## Ember

Az ember az egyetlen mozgó elem a szimulációban.

### Célkövetés

Az ember objektum célkövetése úgy történik, hogy, ha megadunk neki egy célpont pontot (PointF), akkor addig megy a pont felé egyenesen, amíg a távolságuk megfelelően kicsire változik. Így a megfelelő pontok kijelölésével akár tetszőleges pályát is meg lehet adni az egyednek.

Az ütközés detektálás is hasonlóképpen történik. Minden kalkuláció során először kiszámolja az objektum, hogy melyik lenne a következő koordináta, ahova lépnie kéne. Majd az új pontra megvizsgálja, hogy ütközési távolságba van-e bármelyik fallal. Ha igen, akkor az új célpont a jelenlegi koordinátája lesz (megáll), ha nem, akkor tovább lép.

A vészhelyzetet úgy detektálja, ha a hozzá legközelebb eső hő blokk egy bizonyos hőmérséklet fölé emelkedik, akkor megkezdi a menekülést. Minden egyed tudja, hogy melyik terembe (Room) van. A terem ajtókból és falakból áll. Vészhelyzet esetén megkeresi a hozzá tartozó szoba legközelebbi ajtaját, és azt vészi célpontba.

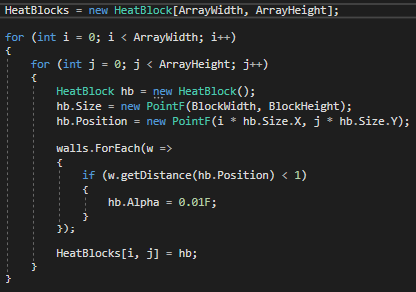
A játéktéren van egy előre meghatározott menekülési útvonal, ami ellenőrzőpontokból áll. Ezzel modelleztem le, hogy a valóságban is mindenki tudja, hogy merre kell menni, mert általában ezt fényút is jelöli. Az ellenőrző pontok tárulják magukban a következő pontot is (merre kell mennie tovább). Így az ajtó elérése után az objektum a menekülési útvonalon halad tovább a kijáratig.

Nyugalmi állapotban az emberek random időközönként random helyekre mozognak. Ezt azért találtam fontosnak betenni, mert így a szimuláció elindítása után egyértelműen látszik, hogy fut a szimuláció.

## Játéktér

A játéktér az első tervezés során még úgy szerepelt, mint random generálható elem. Azonban, mivel a játéktér csak egyszer generálódna le, és onnantól kezdve nem venne sokat igénybe a processzor erőforrásából, így a párhuzamosítás szempontjából nem lényeges elem. Mindezek miatt egy előre beégetett/kiszámolt pálya kerül bemutatásra.

A játéktér falakból és ajtókból áll. Az ajtó sajátossága, hogy az Ember objektum át tud rajta menni. A fal sajátossága pedig az, hogy a hőterjedési együttható itt sokkal kisebb, tehát a tűz/hő sokkal lassabban halad falon keresztül. Ezt mutatja az alábbi kódrészlet:



‑. ábra - Falhoz tartozó hőterjedési együttható kiszámolása szoftveresen

A fenti kódban a *getDistance* függvény pont és egyenes távolságát adja vissza, ahol az egyenes értelemszerűen a fal, a pont pedig a bemenő paraméter. A kódból látszik, ha 1-nél közelebb van a falhoz (rajta van) a blokk, akkor az adott blokknak a hőterjedési együtthatója 0.01 lesz, aminek az alapértéke 0.25.

## Program működése

A program természetesen több algoritmus alapján is elkészült. Először is meg kellet találnom azokat a pontokat, amik a számítási teljesítmény/idő alapján kritikusak, hiszen ott van értelme párhuzamosítani. Nagy tanulság, hogy a korábbi elképzelésem, miszerint az egyes Emberek mozgatása, ütközésdetektálása veszi majd a leg több időt igénybe, nem állta meg a helyét.

A program először is felbontottam funkciók szerinti csoportokra, és elkezdtem a különböző funkciók idejét mérni. A funkciók két csoportra bonthatóak:

1. Kalkulációk
   1. Emberek kalkulációi
   2. Hő blokkok kalkulációi
2. Rajzolások
   1. Játéktér kirajzolása
   2. Emberek kirajzolása
   3. Hő blokkok kirajzolása

A folyamatot összegzem egy blokkdiagramon. Az alábbi ábra a szekvenciális megoldást mutatja:

0‑3. ábra - Program szekvenciális megoldása

**Szimuláció előkalkulációi**

**Emberek kalkulációi**

**Hő blokkok kalkulációi**

**Emberek és játéktér kirajzolása**

**Hő blokkok kirajzolása**

**Statisztika frissítése**

Az ábrán narancsszínnel jelöltem a kritikus pontokat. Ugyanis az emberek kalkulációja (5 embernél, 10-szeres szorzóval, párhuzamosítás nélkül) ms-ban nem mérhető. Itt, és a Hő blokkok kalkulációjánál is át kellet térnem a Tick (stopper) mértékegységre. Az emberek kalkulációja 30-150 Tick idő alatt megtörténik, míg a hő blokkok kalkulációja 3000-8000 Tick. Itt már látszik, hogy a kalkulációban is komoly különbség van. Valószínűleg ez még sokáig így lenne a szoftver fejlesztése során, hiszen pixelből mindig több lesz, mint emberből.

Azonban, amit a mérések kihoztak az az, hogy a legtöbb idő a kirajzolással megy el. A kirajzolás ideje ennél a szimulációnál 70-90 ms! Itt már komoly mértékben eltér, ezért egyértelmű, hogy a párhuzamosításnak erre a területre kell koncentrálnia.

A szimulációkról általánosan elmondható, hogy a legjobb mérőszám hozzá az FPS (Frame Per Second), azaz az egy másodperc alatt kirajzolt képek száma. A későbbiekben is ezt fogom használni a párhuzamosítás sikerességének a méréséhez.

# Párhuzamosítás

A párhuzamosítás fő helye a korábban említett Hő blokkok kirajzolása. Viszont a másik kritikus hely is a hő blokkokhoz tartozik, ezért a két kritikus pont párhuzamosítása azonos metodika alapján ment végbe. Minden szimuláció elején van lehetőség megadni egy függőleges, és egy vízszintes felbontási számot. Ez azt takarja, hogy vízszintesen, illetve függőlegesen hány részre osztom fel a játékteret. Ez azért fontos, mert a két szám szorzata meghatározza a maximális szálak számát.

Mindkét ponton felbontom a játékteret kis részekre, és az adott részeket külön szálak számolják. Ez a kalkuláció esetén nem okoz problémát, hiszen van egy publikus, és egy belső változójuk. Viszont a rajzolásnál ez már egy picit komolyabb probléma, ugyanis ugyanarra a képre kéne rajzolni, ami nem a legszerencsésebb, mert elakad a program.

A rajzolás során minden szál lemásolja magának az eredeti képet, és csak azon a szekción dolgozik, ami az ő feladata. Miután végzett a képalkotással, az adott szekcióját visszamásolja az eredeti képre. Ez az egyetlen hely, ahol összeakadhatnak a szálak. Hozzáteszem ez ritka, hogy ugyanakkor végezzenek, de természetesen beleraktam egy lock object-es megoldást, így ez a probléma is át lett hidalva.

Mindkét kritikus pontnál elmondható, hogy a szálak megvárják egymást, és csak azután halad tovább a program, hogy a szálak mindegyike végzett a feladatával.

A Thread, és Task alapú megoldás csak annyiban tér el egymástól, hogy melyik eszközt használom, de maga az algoritmus, és a szálak száma ugyanaz. Azonban meglepő, hogy mégis van különbség a kettő között, de ezt majd az Eredmények részben fogom részletezni.

A Thread és Task típusú párhuzamosításokhoz az adott módszerrel elnevezett függvények nyújtanak segítséget. Ezeket a függvényeket a hő blokkokhoz tartozó kontrollerben kell keresni (HeatTransferController).

A ThreadPool-os megoldás sem sokban különbözik, csak annyiban, hogy az eszközhöz tartozó sajátossághoz kellett igazodnom.

## Eredmények

Az elkészült szoftver sok féle variációt kínál eredményvizsgálás szempontjából. Az emberek száma, szorzó, és a szálak számának változtatása rengeteg lehetséges esetet vázol fel. Én a sok lehetőség közül én egy olyan szimulációt választottam, ahol az emberek száma **öt**, a szorzót (multiplier) **tízre** állítottam, és **harminc** másodperces szimulációkat néztem a szálak számának függvényében.

A lenti eredményt kaptam. Több érdekesség is feltűnt.

Először is ami egyértelmű, hogy bár nem tűnik nagynak a gyorsulásnövekedés, mégis- jelentős. Például, ha megnézzük a Task alapú párhuzamosítást, akkor jól látszik, hogy 2 szál esetén 14 Fps-re nőtt a képfrissítés 10 Fps-ről. Ez 40%-os teljesítménynövekedés, ami viszont jelentős. Azonban egyértelműen látszik, hogy, hacsak nem Task alapú párhuzamosítást használunk, akkor felesleges 3 szálnál többet alkalmaznunk.

‑. ábra - FPS értékek a szálak függvényében

Második érdekesség, hogy a Thread és a Task alapú párhuzamosítás semmiben nem különbözik, csak a kulcsszóban, de az algoritmus ugyanaz, mégis jelentős eltérés van a Task javára.

Azért választottam pont ezt a konfigurációt, mert a 10 Fps egy szaggatott látványt reprezentál, míg a 14 Fps hozzá képest egy sokkal folytonosabb kép, így a teljesítmény növekedés szemmel látható.

Próbáltam természetesen nagyobb szorzóérték mellet is, de ott a 2 Fps-ről 5 Fps-re való növekedés nem nagyon látványos. A 40-ről 50-re úgyszintén nem látványos (kisebb szorzóérték esetén).

A következő ábrákon most ábrázolni fogok pár eredményt. A következőknél ugyanúgy az emberek száma 5, szálak száma változik és a szorzó is.

‑. ábra - Benchmark, 1-es szorzóval

‑. ábra- Benchmark, 5-ös szorzóval

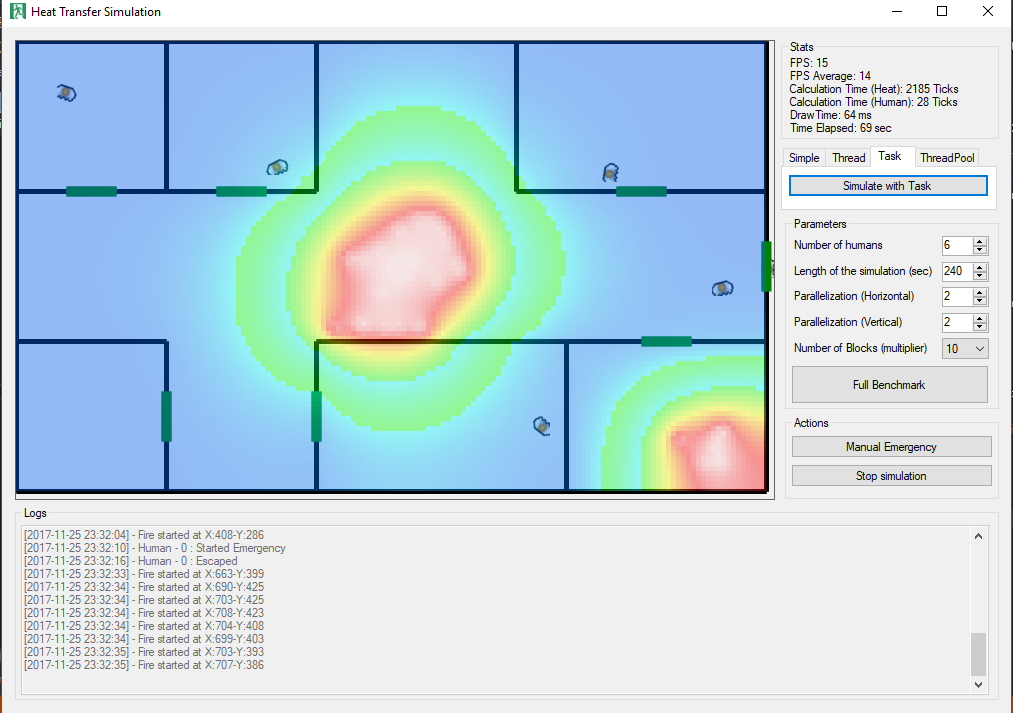
‑. ábra- Benchmark, 10-es szorzóval

# GUI és kezelése

A szimulációk indítása, és kezelése a grafikus interfészen keresztül történik. A lenti ábra mutatja a szoftvert működés közben. A jobb felső sarokban az aktuális adatokat láthatjuk a jelenlegi szimulációról.

Alatta egy tabos választón lehet kiválasztani, hogy milyen típusú szimulációt szeretnénk futtatni. Alatta a paramétereket tudjuk állítani, ami az összes szimulációhoz közös. A paraméterek a következőek:

* Number of humans – A generált emberek száma
* Length of the simulation (sec) – A szimuláció hossza másodpercben
* Parallelization (Horizontal) – Azt adja meg, hogy hány részre osztja fel vízszintesen a képet
* Parallelization (Vertical) - Azt adja meg, hogy hány részre osztja fel függőlegesen a képet
* Number of Blocks (multiplier) – Szorzó. Azt adja meg, hogy hányszorosa legyen a blokkok száma az alapértéknek. Az alapérték: 15 x 9 blokk. A maximális: 150 x 90 blokk



‑. ábra - Grafikus interfész működés közben

A paramétereket választó fül alatt található egy *Full Benchmark* gomb.

## Full Benchmark

Az adott funkció talán a leghasznosabb. A gomb megnyomásakor a megadott paraméterekkel lefut mind a négy fajta szimuláció, valamit utána az eredményt a log résznél ismerteti a megfelelő értesítés után.

A Benchmark gomb alatt találhatóak az akció gombok. Két akcióra van lehetőség:

* Manual Emergency – A gomb lenyomásakor, egy tűzjelzőt szimulál a program, azaz minden ember elkezd azonnal menekülni a kijárat felé
* Stop simulation – Szimuláció azonnali megállítása. Benchmark közben ez nem lehetséges

Az ablak legalján pedig egy log rész található, ahol minden rendszerüzenet olvasható. Itt logolásra kerülnek a tűzzel, emberekkel, illetve a szimulációkkal kapcsolatos információk.

# Hivatkozások

1. **Heat equation (Wiki). [Online] https://en.wikipedia.org/wiki/Heat\_equation.**

**2. SIMPLE Algorithm for Two-Dimensional Flow. [Online] http://www.me.nchu.edu.tw/lab/lab516/2014/%E7%86%B1%E5%B0%8D%E6%B5%81(%E6%97%A5)-pdf-2014/5.pdf.**

**3. Partial Differential Equations. [Online] http://www.math.odu.edu/~jhh/chs9.pdf.**

**4. CMSoft Study Case. [Online] http://www.cmsoft.com.br/videoLecs/CMSoft%20Case%20Study%20Heat%20Transfer%20Simulation.pdf.**

**5. Case study: heat transfer simulation using CLGL interop. [Online] http://www.cmsoft.com.br/opencl-tutorial/case-study-heat-transfer-simulation-using-clgl-interop/.**

**6. Heat Equation. [Online] http://www4.ncsu.edu/~rsmith/MA573\_F17/Heat\_Equation.pdf.**

**7. Convective heat transfer. [Online] https://en.wikipedia.org/wiki/Convective\_heat\_transfer.**