**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Numerikus Analízis** **Tanszék**

**Reaktív mesterséges intelligencia megvalósítása videojátékban**

**Témavezető:**

Fábián Gábor

tanársegéd

**Szerző:**

Barton Patrik

Programtervező informatikus BSc.

Budapest, 2020

Tartalom

[1. Bevezetés 3](#_Toc39146195)

[2. Felhasználói Dokumentáció 4](#_Toc39146196)

[2.1 A program által megoldott feladat 4](#_Toc39146197)

[2.2 A program alapvető működése 4](#_Toc39146198)

[2.3 Felhasznált módszerek áttekintése 5](#_Toc39146199)

[2.3.1 Wavefront algoritmus 5](#_Toc39146200)

[2.3.2 Monte Carlo Tree Search algoritmus 5](#_Toc39146201)

[2.4 A program használata 5](#_Toc39146202)

[3. Fejlesztői Dokumentáció 6](#_Toc39146203)

[Elméleti áttekintés 6](#_Toc39146204)

[A program szerkezete 6](#_Toc39146205)

[Tesztelés 6](#_Toc39146206)

[4. Összefoglalás 7](#_Toc39146207)

[5. További fejlesztési lehetőségek 8](#_Toc39146208)

# Bevezetés

A legtöbb mai videojátékban a számítógép által irányított karakterek Mesterséges Intelligencia algoritmusa nem veszi figyelembe, hogy a játékos karakterének milyen lehetőségek állnak rendelkezésére az adott pillanatban ahhoz képest, hogy összesen milyen képességei lehetnek.

Vegyünk például egy általános, online szerepjátékot. A játékos a játék elején tipikusan kiválasztja a karaktere alapvető tulajdonságait, többek közt a kasztját. Ennek függvényében a karaktere lehet egy mágus, íjász, esetleg lovag. A választott kaszttól pedig nagymértékben függ a játék során szerezhető képességek és felszerelések típusa. Mindezek a játékstílust alapvetően változtathatják meg a játékos számára, az ellenfelek viszont tipikusan ezt figyelmen kívül hagyják. Ugyanúgy reagálnak egy közelharcos karakterre egy pajzzsal, aki ellen egy emberi játékos esetleg oldalba támadást alkalmazna a pajzs megkerülése érdekében, és egy távolharcos íjász karakterre, akit egy emberi játékos valószínűleg a lehető leggyorsabban meg akarna közelíteni, hogy ne használhassa hatékonyan a fegyverét.

Ennek a programnak a célja, hogy egy olyan Mesterséges Intelligenciával lássa el a játékos ellenfeleit, amely reagál a játékos által elérhető lehetőségekre. Ha a játékos használja a képességei egyikét, az ellenfelek tudomást szereznek róla, hogy a játékos hozzáfér az adott lehetőséghez. Ennek tudatában az MI által irányított ellenfelek más döntéseket hozhatnak, annak függvényében, hogy a játékos milyen képességeket ér el.

Ezen megoldás előnye, hogy a játékélményt sokkal színesebbé és izgalmasabbá tehetné. A játékosoknak minden eltérő tulajdonságú karakter esetében fel kell készülniük, hogy az ellenfeleik is máshogy fognak viselkedni. Emellett azon is érdemes lehet elgondolkodniuk, hogy egy adott képességet ne használjanak túl korán, így taktikai előnyt szerezve a meglepetés erejével.

# Felhasználói Dokumentáció

A következőkben ismertetjük a program által megoldandó feladatot, majd a felhasználó szempontjait szem előtt tartva áttekintjük a megoldás elméleti hátterét. Végül részletesen bemutatjuk a program használatát, a játékos lehetőségeit és a Mesterséges Intelligencia által irányított ellenfelek várt működését.

## 2.1 A program által megoldott feladat

A mai videojátékok nagyrészében a játékos által irányított karakteren kívül szerepelnek „Nem-Játékos Karakterek” is. Ezekre a dolgozat hátralevő részében NPC néven hivatkozunk.

Ezek a karakterek tipikusan Mesterséges Intelligencia algoritmusok alapján cselekednek. Általában beállításoktól, vagy az adott karakter alapvető tulajdonságaitól függ, hogy az MI algoritmus milyen fajta viselkedést imitál. Egy magas védekezési értékekkel rendelkező NPC például lehet, hogy igyekszik elsőnek megütközni az ellenfelével, ezzel megpróbálva magára vonni az ellenség figyelmét a kevésbé strapabíró társai helyett. Jellemzően viszont ennek a karaktertípusnak minden helyzetben ugyanez lesz a stratégiája, attól függetlenül, hogy hány, milyen lehetőséggel bíró ellenfele van.

Ebben a programban egy ismert MI algoritmust, a *Monte Carlo Tree Search* -innentől MCTS - módszert alkalmazzuk. Ezt felhasználva mutatjuk be, hogy ugyanaz az ellenfél másképpen reagálhat annak tudatában, hogy a játékos karaktere milyen képességekkel rendelkezik.

## A program alapvető működése

A játék célja, hogy a játékos elpusztítsa az összes ellenfelet. A program során a játékos a képernyő közepén látható karaktert irányítja. Őt egy segédkarakter támogatja, akit közvetett módon irányíthat.

A Q és E gombokkal változtatható meg a kiválasztott akció. A játékos számára elérhető akciók listája: Mozgás, Alap támadás, Képesség1, Képesség2, Segédkarakter irányítása. A Q gombbal a listában balra, az E gombbal jobbra léphetünk.

A kiválasztott akció végrehajtása a WASD iránygombokkal történik. Ezekkel a kiválasztott akció végrehajtásának irányát adhatja meg a játékos. Felhasználásuk: W–fel, A–balra, S–le, D–jobbra.

Amennyiben a Mozgás akció van kiválasztva, a játékos a WASD iránygombok használatával a karakterét mozgatni képes. Az adott gomb megnyomásakor a karakter a megfelelő irányba mozog pontosan egy mezőnyit.

Amennyiben az Alap támadás, Képesség1 vagy Képesség2 akció van kiválasztva, a játékos a WASD iránygombok használatával támadni képes. A játékos karaktere az iránygomboknak megfelelő irányba a kiválasztott akciónak megfelelő támadást végez.

Amennyiben a Segédkarakter irányítása akció van kiválasztva, a játékos az A gombbal a saját karakterének közelébe hívhatja a Segédkaraktert, míg a D gombbal az ellenfelek közelébe küldheti őt, hogy megtámadja őket. A W gomb megnyomásának következtében a Segédkarakter véletlenszerű akciót hajt végre, az S gomb megnyomásának következtében pedig egyhelyben marad.

A játék körökből áll. Minden kör elején a Játékos karaktere cselekszik először. Két mozgás akciót, egy támadás jellegű akciót, és egy segédkaraktert irányító akciót hajt végre. Ezután a Segédkarakter következik, aki a Játékos utasításainak megfelelően cselekszik. Ezt követően az ellenséges karakterek következnek. Az ellenséges karakterek végrehajtandó akcióját a MCTS algoritmus határozza meg, erről a 2.3.2 pontban írunk bővebben.

## 2.3 Felhasznált módszerek áttekintése

Most, hogy bemutattuk a vizsgált feladatot és a program alapvető működését, áttérünk a feladat megoldására. Ehhez mindenekelőtt a programban felhasznált algoritmusok, módszerek bemutatására kerítünk sort, felhasználói szempontok szerint, informálisan. Ezalatt a fő (MCTS) algoritmust, valamint a játék működtetéséhez felhasznált útkereső algoritmust értjük. Az algoritmusok részletes, formális leírására a [Fejlesztői Dokumentációban](#_Fejlesztői_Dokumentáció) kerül sor.

### 2.3.1 Wavefront algoritmus

A karakterek a játéktéren mozogni képesek. Nem minden mezőre léphetnek viszont. Ha egy mezőn terepakadály, pl. egy fa van, vagy egy másik karakter áll az adott mezőn, a karakter nem léphet oda. Ahhoz, hogy ezeket az akadályokat figyelembe véve mozogjanak a karakterek, a Wavefront algoritmust használjuk. Ez az algoritmus egy ***kiinduló pozícióból*** egy ***cél pozícióba*** keresi meg a legrövidebb elérhető utat.

Ennek megvalósításához rekurzívan bejárja a ***kiinduló pozícióból*** elérhető játékteret, és a rekurzió minden egyes lépésével egyel nagyobb értéket ad az adott mezőknek. Ez a rekurzió a ***kiinduló pozícióból*** indul, az annak megfelelő mezőnek pedig az *1* értéket adja. Ezután a közvetlen azzal szomszédos mezőnek egyel nagyobb, *2* értéket ad, és így tovább, amíg több szomszédos mezőt már nem lehet felfedezni a játéktéren.

Miután a bejárható mezőt feltöltötte, a ***cél pozíciónak*** megfelelő mezőből kiindulva visszaköveti és elmenti az utat a ***kiinduló pozícióig***. Ezt úgy éri el, hogy a szomszédos mezők közül mindig a legkisebb értékű mezőt választja az út következő elemének. Ez a módszer végül megtalálja az összes közül a legkisebb értékű mezőt, ami pont a ***kezdő pozíciónak*** felel meg, mivel ennek értékét az algoritmus elején egyre állítottuk, és az összes többi mező értékét ehhez képest növeltük.

### 2.3.2 Monte Carlo Tree Search algoritmus

Az MCTS algoritmus határozza meg, hogy egy adott körben az ellenfelek milyen lépést tegyenek meg. Ez az algoritmus négy lépésből áll, melyek folyamata ciklikusan ismétlődik, egészen addig, amíg a megadott időkeret le nem jár.

Ez a négy lépés a *Kiválasztás* (Selection), *Kibővítés* (Expansion), *Szimuláció* (Simulation/Rollout) és *Visszaterjesztés* (Backpropogation). Az algoritmus egy csúcsokból álló fa gráfot használ, mely kezdetben egy gyökér csúcsból áll, és működése közben az algoritmus egyre több csúcsot ad hozzá a fához.

##### A csúcsok tulajdonságai.

Minden egyes csúcs egy körön belül vagy a játékos- és a segédkarakter, vagy az ellenfél karakterek lépéseit határozza meg. Ennek megfelelően minden csúcs tartalmaz egy *akciót*, ami megfeleltethető az egyik fél karaktereinek lépéseinek; és egy logikai változót, amely megadja, hogy az adott csúcs az ellenfél egységeinek akcióját adja-e meg, vagy a játékos- és segédkarakterét. Ez a két érték minden csúcs létrejöttekor inicializálódik, és később nem változik.

Minden csúcs tartalmaz továbbá két nem negatív egész számot, a *Győzelmek* (Wins) és *Látogatások* (Visits) értékeket. Ez a két érték minden újonnan létrejött csúcsban alapvetően nullával egyenlő, és a negyedik lépésben, a *Visszaterjesztésben* változik.

A csúcsok tartalmaznak még mutatókat, mind a szülőjükre, mind a gyermekeikre. A szülő csúcs nem változik az algoritmus futása közben, a csúcs létrejöttekor inicializálódik. A gyerek csúcsok egy listában tárolódnak, amely kezdetben üres, az algoritmus futása során pedig bővül.

Mindezeken kívül a csúcsok még egy logikai változóval rendelkeznek, ez azt adja meg, hogy a szimulációban létrejövő csúcsok-e, vagy sem. Ennek jelentősége az algoritmus harmadik lépésében, a Szimulációban lesz. Ennek a változónak az értékét is a csúccsal együtt inicializáljuk.

Minden esetben, ha egy csúcs szülője egy másiknak, az egyik csúcs az ellenfelek, a másik pedig a játékos- és segédkarakter akcióit adja meg. Két szülő-gyermek kapcsolatban álló csúcs tehát sosem adhatja meg ugyanannak a csapatnak az akcióit.

##### A lépések összefoglalása

A *Kiválasztás* lépésben a fának egy csúcsa választódik ki. Ez a kiválasztás lépésenként egy képlet segítségével történik, melynek jelentősége, hogy a legígéretesebb csúcsot válassza ki a lehetségesek közül, de a túl régen kiválasztott csúcsoknak is adjon esélyt, ezzel csökkentve az esélyt, hogy a legjobb megoldást csupán azért elkerülje az algoritmus, mert eleinte rossz értékeket mutatott, és ezért nem vizsgálta tovább.

A *Kibővítés* lépésben a *Kiválasztás* lépés során kiválasztott csúcs gyermekeként generálunk egy új csúcsot. Ennek a csúcsnak az *akció* változója véletlenszerűen jön létre. Ez tehát egy véletlenszerű lépésnek felel meg.

A *Szimuláció* a *Kiválasztás* lépésben véletlenszerűen létrehozott *akció*jú csúcsból indul ki. További, véletlenszerűen létrehozott *akció*jú csúcsokat hoz létre, melyek lépésről lépésre az előző csúcs gyermekeiként keletkeznek. Ezzel gyakorlatilag egy véletlenszerű kimenetelt szimulál, amelyben a két fél tetszőleges lépéseket tesz.

A *Szimuláció* a MCTS algoritmusokban tipikusan addig fut, amíg az egyik fél egyértelműen meg nem nyeri a játszmát, így egyértelműen eldöntve a győztes felet. Mivel a mi környezetünkben a terminális végkimenetel nagyon sok véletlenszerű lépést jelenthetne, ezzel jelentősen lecsökkentve az algoritmus hatékonyságát, ezért a *Szimuláció* ebben a programban csak bizonyos számú lépést tehet meg. Emiatt a *Szimuláció* végén nem feltétlen lesz terminális végkimenetel bármelyik fél számára; ebben az esetben azt a felet tekintjük győztesnek, amelyiknek a karakterei együttesen több életponttal rendelkeznek.

A *Visszaterjesztés* a *Szimuláció* eredményére alapoz. A legutolsó, *Szimuláció* során létrehozott csúcsban elindítunk egy rekurzív folyamatot, amely mindig a csúcs szülőjében folytatódik, egészen a gyökér csúcsig. Ez a folyamat minden érintett csúcsnak egyel növeli a *Látogatások* értékét, és amennyiben a *Szimuláció* végén az ellenfelet tekintettük győztes félnek, szintén egyel növeli az érintett csúcsok *Győzelmek* értékét.

Kiválasztás (Selection).

*Ebben a lépésben a fának egy csúcsa választódik ki.* A *Kiválasztás* első lépése mindig a gyökér csúcsból indul. A *Kiválasztás* minden lépésében kiválaszt egy csúcsot az adott csúcs és a gyermekei közül.

Ez a választás a következő képleten alapszik: , ahol: *w* az adott csúcs *Győzelmek* értéke; az *n* az adott csúcs *Látogatások* értéke; az *N* az adott csúcs szülőjének *Látogatások* értéke; a *C* pedig egy adott konstans, ami a mi esetünkben .

Ezt a képletet minden megvizsgált csúcsra kiszámoljuk: az adott csúcsra és a gyermekeire is. Amelyik csúcsnak a legnagyobb a képlet által kiszámolt értéke, azt a csúcsot fogja a *Kiválasztás* lépés választani.

Miután kiválasztott egy gyermek csúcsot, a választott csúcsra ismét végrehajtja a kiválasztást, amennyiben a csúcs nem levél csúcs. Ha azonban nem gyermek csúcsot választ ki hanem önmagát, vagy levél csúcsot, akkor a kiválasztott csúccsal tér vissza, és a *Kiválasztás* lépés befejeződik.

Kibővítés (Expansion).

*Ebben a lépésben a Kiválasztás lépés során kiválasztott csúcs gyermekeként generálunk egy új csúcsot.* Ennek a csúcsnak az *akció* változója véletlenszerűen generálódik. Ebből a csúcsból fogjuk indítani a *Szimuláció* lépést.

##### Szimuláció (Simulation/Rollout).

*Ez a lépés a Kiválasztás lépésben véletlenszerűen létrehozott akciójú csúcsból indul ki.* Minden szimulációs lépésben létrehoz egy csúcsot, véletlenszerű akcióval, és az előzőleg szimulált csúcs gyermekévé teszi. (Nyilván a legelső csúcs, amit létrehoz, a *Kibővítés* során létrehozott csúcs gyermeke lesz.) Ha a szimulációs lépések száma meghaladja a megadott határértéket, vagy a játék terminális végkimenetelre jut az egyik fél számára, a *Szimuláció* véget ér.

##### Visszaterjesztés (Backpropogation).

*A Visszaterjesztés a Szimuláció eredményére alapoz.* A legutolsó csúcsból indul ki, ami a Szimuláció során jött létre, és szülőről szülőre járva a gyökérig halad. Minden érintett csúcsnak egyel növeli a Látogatások értékét, valamint a Győzelmek értékét akkor, ha a Szimuláció végén az ellenfél volt a győztes fél.

## 2.4 A program használata

# Fejlesztői Dokumentáció

## Elméleti áttekintés

## A program szerkezete

## Tesztelés

# Összefoglalás

# További fejlesztési lehetőségek