ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Hajdú András**
2023	Hallgató törzskönyvi száma:	**T/006210/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Hajdú András**
Törzskönyvi száma: T/006210/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Mesterséges intelligencia fejlesztése Heroes of Might and Magic 3 számára
Developing an artificial intelligence for Heroes of Might and Magic 3

Intézményi konzulens: Balázs Elemér
Külső konzulens:

Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés specializáció

A feladat

A hallgató feladata a Heroes of Might and Magic III játék open-source változatának (VCMI) megismerése, feltérképezése és egy mesterséges intelligencia készítése a már meglévő két megközelítéstől eltérő módon. A számítógép döntései korábbi játékokban elvégzett műveleteket alapján kerüljenek meghatározásra. Ezek kiértékeléséhez készítsen egy olyan alrendszert, amely képes a játékmenetet strukturált formában eltárolni, továbbá legyen lehetőség az egyes lépések visszajátszására. Az elmentett adatok alapján készítsen egy olyan döntéshozó egységet, mely valós időben képes egy adott játékbéli szituációban a győzelem elérése érdekében utasítást adni.

A dolgozatnak tartalmaznia kell:

- a feladat leírását,
- játékok esetében alkalmazott mesterséges intelligenciák fajtáinak bemutatását,
- az open-source projekt strukturális felépítésének ismertetését,
- a feladat megoldásához szükséges releváns irodalom feldolgozását,
- a fejlesztendő rendszer részletes tervét,
- a megvalósítás menetének részletes leírását,
- az elért eredmények bemutatását, értékelését és az alkalmazás tesztelési jegyzőkönyvét,
- az elkészült egységek továbbfejlesztési lehetőségeit,
- a dokumentációt, a programot (kivéve a Heroes of Might and Magic III hang, kép, videó és térképfájljait, mert az a 3DO Company tulajdonát képezi), a szükséges inputadatokat, valamint a rendszert bemutató honlapot és prezentációt CD mellékleten.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar 1 Budapest

Ph.

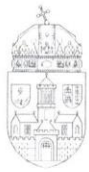
.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 20 22. 12. 15.

Kajdi András

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: HAJDÚ ANDRÁS

Neptun kód: [redacted]

Tagozat: NAPPALI

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

MESTERSÉGES INTELLIGENCIA FEJLESZTÉSE HEROES OF MIGHT AND MAGIC 3 SZÁMÁRA

Szakdolgozat / Diplomamunka[2] címe angolul:

DEVELOPING AN ARTIFICIAL INTELLIGENCE FOR HEROES OF MIGHT AND MAGIC 3

Intézményi konzulens: BALÁZS ELEMÉR

Külső konzulens: —

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.09.	PROJEKT INICIALIZÁLÓ MEGBESZÉLÉS	[signature]
2.	2021.10.01.	HASONLÓ RENDSZEREK ÁTTEKINTÉSE	[signature]
3.	2021.11.04.	RENDSZERTERV PONTOSÍTÁSA	[signature]
4.	2021.12.06.	ELÉRT EREDMÉNYEK VÉLEMÉNYEZÉSE	[signature]

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021.12.09.

[signature]
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: HAJDÚ ANDRÁS

Neptun Kód: [redacted]

Tagozat: NAPPALI

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat / Diplomamunka[5] címe magyarul:

MESTERSÉGES INTELLIGENCIA FEJLESZTÉSE HEROES OF MIGHT AND MAGIC 3 SZÁMÁRA

Szakdolgozat / Diplomamunka[6] címe angolul:

DEVELOPING AN ARTIFICIAL INTELLIGENCE FOR HEROES OF MIGHT AND MAGIC 3

Intézményi konzulens: BALÁZS ELEMÉR

Külső konzulens: —

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.09.07.	PROJEKT FÜGGŐSÉGI GRÁFJÁNAK ISMERTETÉSE	
2.	2022.09.30.	REPLAY SYSTEM BEMUTATÁSA	
3.	2022.11.04.	MESTERSÉGES INTELLIGENCIA KORLÁTAIRÓL EGYEZTETÉS	
4.	2022.11.27.	ELKÉSZÜLT DOLGOZAT VÉLEMÉNYEZÉSE, PROJEKT ZÁRÁSSAL KAPCSOLATOS TEENDŐK ÁTBESZÉLÉSE	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat / Szakdolgozat I. / Szakdolgozat II. / Diplomamunka II. / Diplomamunka III. / Diplomamunka IV. / Projeklabor 2. / Projektlabor 3. / Záródolgozati projekt[7] tantárgy követelményét teljesítette, beszámolóra / védésre[8] bocsátható.

Javasolt érdemjegy: 5

Intézményi konzulens

Budapest, 2022.11.28.

---

[5] Megfelelő aláhúzandó!
[6] Megfelelő aláhúzandó!
[7] Megfelelő aláhúzandó!
[8] Megfelelő aláhúzandó!

Tartalom

Bevezető

Szakdolgozatunk alapötletét a gyerekkorom egyik kedvenc időtöltése, a Heroes of Might and Magic 3 (HOMM3) című videójáték adta. A szoftver 1999-ben jelent meg, műfaja a körökre osztott stratégiai játék. A célunk röviden összefoglalva: az ellenség hőseinek legyőzése és kastélyainak elfoglalása saját hőseinkkel, akik ezt saját szintük növelésével és különféle mitológiai és kitalált élőlények toborzásával érik el. A munkánk megvalósításához a VCMI [1] nevű nyílt forráskódú projektet használjuk fel. A játékmotor C++-ban íródott, a készítők célja az eredeti játék újraírása volt könnyen bővíthető formában.

1. A játékmenet bemutatása és elemzése

A következő fejezetben a játékmenet bemutatása és elemzése valósul meg. A játékmenet kastély, térkép, valamint csata nézetre oszlik fel.

1.1. Kastély nézet

A kastély nézetben (*1.1. ábra*) fejleszthetjük kastélyunkat, mely segítségével nagyobb számú, fejlettebb egységeket tudunk toborozni, akiknek száma hetente feltöltődik. Piactereket is építhetünk, melyek lehetővé teszik nyersanyagjaink kicserélését más nyersanyagokra. A piacterek hatékonysága kastélyaink számával arányos. Itt bérelhetünk fel új hősöket is. Mivel a nézet mesterséges intelligenciájának megvalósítása nem túl nehéz (az egységekhez erő/ár arányban értékeket rendelünk, ez alapján választunk), ezzel a területtel nem sokat fogunk foglalkozni.

1.1. ábra - Kastély nézet

A kastély nézet az alábbi főbb területekből tevődik össze:

Városközpont (1.) – itt választjuk ki, mit fejlesszünk a kastélyunkban, egység lakóhely (2.) – egy tetszőleges egység lakóhelye, innen toborozhatjuk őket, kastély összefoglaló (3.) – a kastély nevét, heti bevételét és toborozható egységek számát mutatja, hős egységek (4.) – a kastélyban tartózkodó hőst és egységeit jelzi, kastély egységek (5.) – a kastélyban tartózkodó egységeket jelzi.

1.2. Térkép nézet

A térkép nézet (*1.2. ábra*) a játékmenet legfontosabb része. Hőseinkkel ezen keresztül fedezzük fel a pályát, nyersanyagokat és varázstárgyakat gyűjtünk, semleges lényekkel harcolunk. Hőseink bizonyos mennyiségű mozgási ponttal rendelkeznek, melyek elhasználása után tovább adjuk a kört a következő játékosnak. A pályán lévő fontos elemek közé tartoznak a semleges kastélyok, illetve az ellenfél játékosok hősei és kastélyai, akikkel harcba bonyolódhatunk a csata nézetben.

1.2. ábra - Térkép nézet

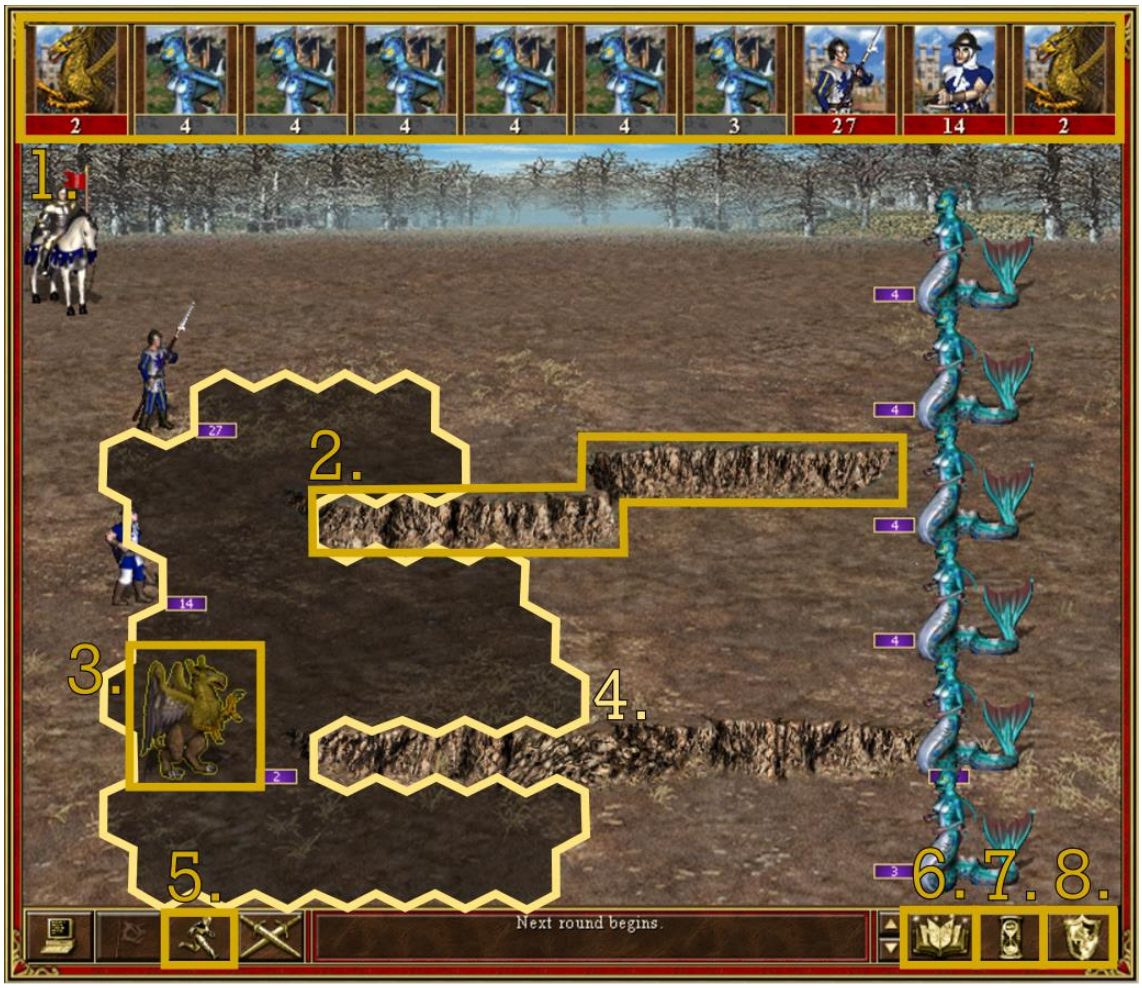
A térkép nézet az alábbi főbb területekből tevődik össze:

Semleges egység (1.), saját kastély (2.), saját hős (3.), semleges nyersanyagbánya (4.), semleges épület (5.), felvehető nyersanyag (6.), saját hősök listája (7.), saját kastélyok listája (8.), saját nyersanyagok listája és játékon belüli idő (9.)

1.3. Csata nézet

A térképen lévő csatakezdeményezés után ebbe a nézetbe (*1.3. ábra*) kerülünk. A nézet egy sakktáblaszerű, 11x15 nagyságú, hatszögekből álló csatatér, ahol a játékosok körökre osztva mozgathatják egységeiket és támadhatnak. A pályán különféle akadályok is el vannak szórva, melyek gátolják a mozgást. Kastélyharc esetén a védő jelentős előnnyel rendelkezik, egységei a kastély falain belül kezdenek, melyek mesterséges akadályok a csatatéren. Továbbá a kastély lövésztornyai is részt vesznek a harcban, amennyiben a kastélyban ki lettek fejlesztve. A harcban részt vevő egységeket 3 kategóriába csoportosíthatjuk, sima közelharci egységekre, lövészekre akik messziről képesek sebezni és repülő egységekre akik ignorálják a földön lévő akadályokat. A támadási sorrend az egységek sebességétől függ.

A várakozás gomb megnyomásával a kör végére halaszthatjuk a kiválasztott egység mozgatását, a védekezés gombbal pedig elhalasztjuk körét, védekezése megnövelése fejében. Hősünk nagyban befolyásolja a csatát, saját erősségei hozzáadódnak egységeihez és varázslatok használatára is képes. A csataterünk talaja is különféle előnyöket illetve hátrányokat biztosít, típusától függően.



1.3. ábra - Csata nézet

A csata nézet az alábbi főbb területekből tevődik össze:

Támadási sorrend (1.), akadályok (2.), tetszőleges egység (3.), egység lehetséges mozgásai (4.), menekülés a csatából (5.), varázslat használata (6.), várakozás (7.), védekezés (8.)

1.4. ábra - Egység és tulajdonságai

Egy tetszőleges egység tulajdonságainak bemutatása:

Támadási és védekezési erő, zárójelben a hős erejének hozzáadásával keletkezett valódi értékek (1.), lövések száma - ha lövész (2.), maximális és minimális sebzés (3.), teljes és jelenlegi élet (4.), sebesség (5.), morál - nagy morál esetén kétszer támad, ellenkező esetben kimarad a köre (6.), szerencse - nagy szerencse esetén maximális sebzés, ellenkező esetben minimális (7.), egységen lévő pozitív és negatív hatások (8.), egység speciális tulajdonságai (9.)

2. A játék lehetséges optimalizálási pontjai

Saját tapasztalataink alapján listázott észrevételek, megfigyelések a mesterséges intelligenciával (Artificial Intelligence - továbbiakban AI) kapcsolatban, mint az eredeti játékban, mint a VCMI-ben. A problémákra lehetséges megoldásokat is felsorolunk.

2.1. Térkép nézet

Fontos, hogy hőseink mozgáspontjai teljesen és hatékonyan ki legyenek használva. A talaj fajtájától (például út, mocsár, rét, víz), egységek sebességétől és saját képességeiktől függően különböző távolságokat tudnak megtenni. A játékos cserélgetheti egységeit és varázstárgyait hősei között is, így nagy távolságú mozgósításra képes, ha optimális távolságban helyezkednek el egymástól.

Mivel az egységeink száma korlátozott, főleg a játék kezdetén, célszerű hőseinknek szerepeket osztani. Egy-két hős rendelkezzen seregeink nagy részével, összpontosítson a tapasztalatszerzésre, a semleges egységek és az ellenséges hősök legyőzésére. A többi hősünk célja a felfedezés, nyersanyaggyűjtés és a fő hőseink támogatása új egységekkel.

Bizonyos nyersanyagokat erősebb lények védik, ilyenkor az AI feladata eldönteni megéri-e harcolni vagy inkább elkerüli a csatát. Bizonyos nyersanyagok fontossága is változhat a játékos jelenlegi mennyisége és kastélyainak igénye alapján. Célunk megtalálni ezeknek a döntéseknek az optimális kockázat/haszon arányát.

A játékban hetente nemcsak a toborozható egységek száma frissül, hanem a pályán lévő előnyöket/egységeket adó létesítmények nagy része is. Ezek heti meglátogatása is része az optimális játékmenetnek.

A Térkép AI hiányosságai

A jelenlegi AI nehézségi fokozatait több esetben az ellenség vagy a játékos kezdőnyersanyagának csökkentésével szabályozza, miközben maga az AI döntései nem változnak. Ezen kívül a túl nagy prioritás van a hősök felbérlésére, így sokszor feleslegesen vándorolnak ide-oda. A hősök közötti interakciók (egységcserék, egységek utánpótlása) sem az erőssége.

Saját céljaink

A térkép bejárására genetikus algoritmusok, különböző optimalizációs stratégiák, mint például a visszalépéses keresés, nyers erő algoritmus, mohó algoritmus, szétválasztás és korlátozás algoritmus, és egyéb útvonalkereső algoritmusok implementálása és összehasonlítása, mint például Dijsktra legrövidebb út algoritmusa. Valamint célunk az előzőleg leírt optimalizási pontok kidolgozása.

2.2. Csata nézet

A legtöbb egység egy támadás után csak egyszer indít ellentámadást, így célszerű összpontosítani támadásainkat. A legerősebb egységhalmazokat célszerű fókuszálni, hogy minél kevesebb kárt tegyenek seregeinkben, ez kifejezetten vonatkozik a lövészekre.

Nem célszerű gyorsabb egységeinkkel azonnal befutni, mivel az ellenfél egységei könnyen rácsoportosulhatnak. Az egység körének átadása helyett viszont a várakozás sokszor optimálisabb, mivel a kör végére az ellenfél egységei támadótávolságba mozoghatnak. Fontos feladat a várakozási funkció optimális használatának meghatározása. Szintén megfontolandó egységeink előreküldése, ha lövészeink jóval erősebbek ellenfeink lövészeitől. Ilyenkor érdemes csoportosulni és körbevenni értékes egységeinket.

Hőseink és egységeink varázslatainak és képességeinek kihasználása a harc fontos része.

Például egy egység támadása lefedhet több harcmezőt is, ilyenkor több figyelmet kell fordítanunk az egységünk helyezkedésére a pályán.

A Csata AI hiányosságai

A jelenlegi AI nem használja ki elég jól a várakozás funkciót, és sokszor választja a rohamozást a lövészek védelmével szemben. A varázslathasználata elég kiszámítható, az egyszerű erősítéseket preferálja.

Saját céljaink

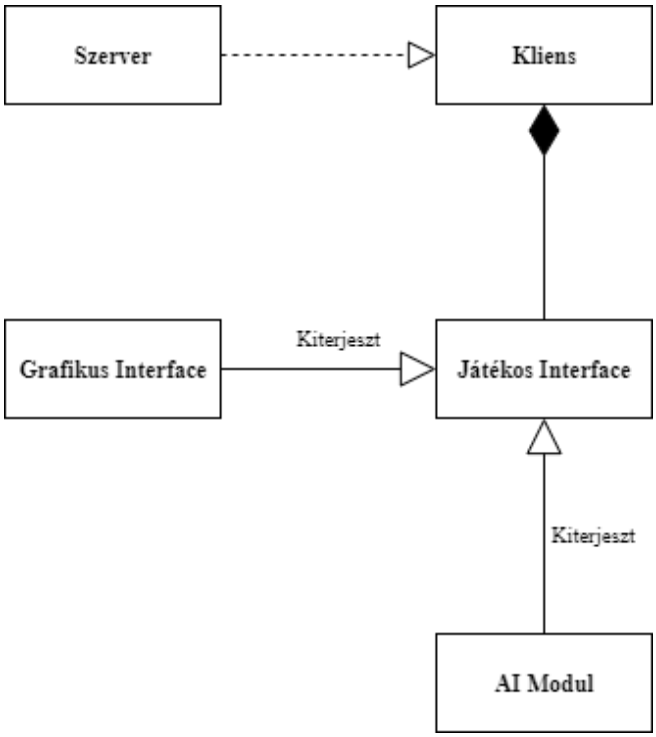
Célunk a hagyományos sakktábla megoldó algoritmusok átírása erre a feladatra, öntanuló algoritmusok bevezetése, melyekhez szükséges implementálnunk egy visszajátszás (replay) funkciót.

3. VCMI felépítése

Az alábbi fejezetben a VCMI rendszer nagyságrendi áttekintése látható. A rendszer felépítésének ismertetése mellett a saját komponensek kapcsolata, rendszerbe való beilleszkedése is nyomon követhető.

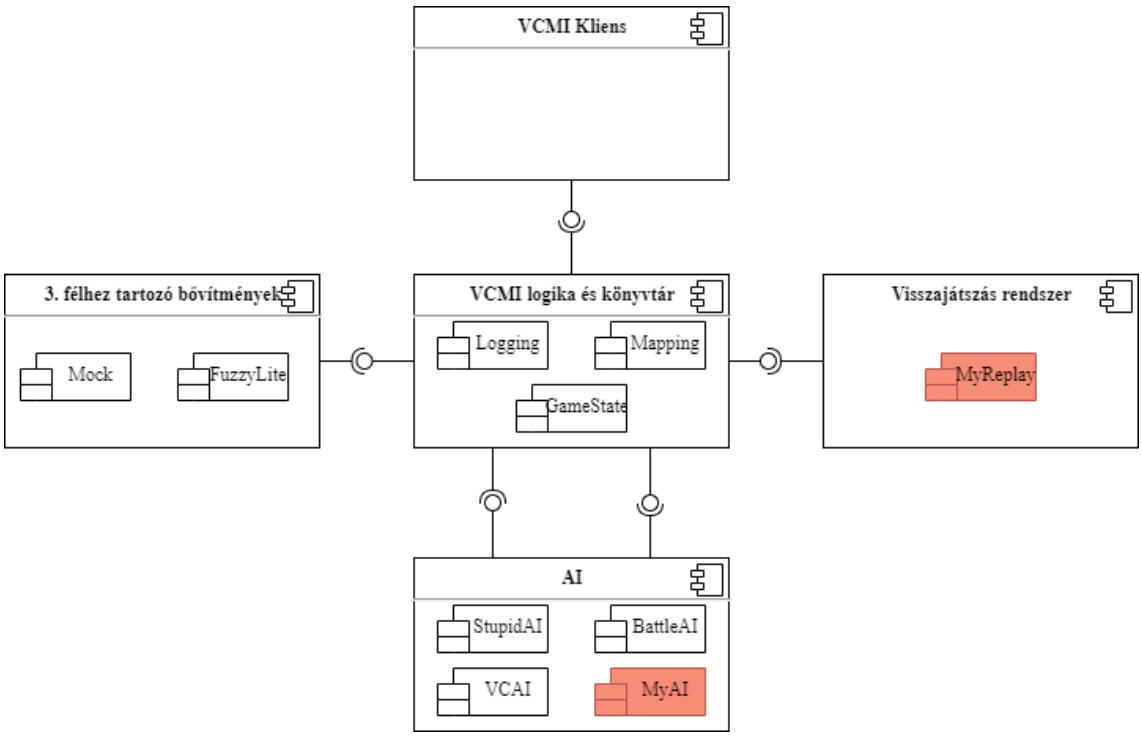
3.1. A VCMI általános felépítése

Az általános felépítés (*3.1. ábra*) a VCMI oldalán feltüntetett ábra alapján lett készítve. Látható, hogy a mi rendszerünk a Játékos Interface modulra alapul.



3.1. ábra - A VCMI felépítése (Forrás: [2])

A részletesebb kidolgozást a következő ábrán (*3.2. ábra*) ismertetjük.



3.2. ábra - Saját kódtérképünk

Az ábrán piros kijelöléssel jelezzük saját rendszereinket. Látható, hogy mindkét modulunk szoros kapcsolatban áll a közös VCMI logika-könyvtár modullal, mivel játék állapotáról rögzített információ nagy része itt van tárolva. A Visszajátszás rendszerünket (*3.2. ábrán* MyReplay) viszont külön modulként illesztjük, mivel logikailag nem kötődik a létező modulokhoz. Az AI többi változatával később foglalkozunk (*5. fejezet Hasonló rendszerek bemutatása*).

4. AI elemzése a játékos szemszögéből

Egy komplex AI tervezése és tökéletesítése közben könnyű elfeledkezni a végső célunkról. Sok munka gyümölcseként elérhetjük azt, hogy az általunk programozott mesterséges intelligencia teljes mértékben kihasználja a játék korlátjait anélkül, hogy bármilyen előnnyel rendelkezne a játékossal szemben. A végső termék viszont nagy valószínűséggel egy kudarc lenne. Miért? Bár a legtöbb területen a végcél egy optimális AI készítése, a mi célunk ennél jóval egyszerűbb. Nekünk az a lényeges, hogy a játék élvezetes legyen, az AI intelligensnek és becsületesnek tűnjön. Ez a fejezet a játékos szemszögéről, illetve játék AI történelméről, fajtáiról és egyéb apró észrevételeiről szól, melynek segítségével egy jobb AI-t tudunk készíteni.

4.1. AI történelmi áttörései

Az első játék AI-k előre beprogramozott mozgási mintákat használtak. Később ezek hasító függvényekkel lettek bővítve, melyek a játékos bemenete alapján működtek. Az új játékműfajok megjelenése miatt különböző rendszereket vezettek be, például a formális AI-ra alapuló determinisztikus véges állapotú gépeket (Finite State Machine - FSM) [3]. A későbbi játékokban elkezdtek megjelenni az öntanuló intelligenciák is, ami nem meglepő, hiszen 1997-ben az IBM által kifejlesztett DeepBlue legyőzte Garri Kaszparovot, az akkori sakkvilágbajnokot [4]. Nagy úttörő volt a Halo 2 (Bungie, 2004), mely bevezette a viselkedésfák használatát [5] [6], illetve a F.E.A.R. (Monolith Productions, 2005), mely dinamikus problémamegoldása és komplex csapatviselkedése miatt [7] máig az egyik legjobb mesterséges intelligenciával rendelkező játék. A modern játékok nagy része FSM-t, illetve különböző útkereső AI-kat használ [7], melyekről később bővebben is beszélünk.

4.2. Intelligencia illúziója

A Halo készítésekor már tisztában voltak a fejlesztők azzal, hogy az agresszív ellenfelek intelligensebbnek tűnnek [6], azonban ez kifejezetten hátrány volt a Doom 2016 (Id Software, 2016) készítésekor, mivel az ellenfelek viselkedése miatt a játékosnak folyton hátrálnia kellett [8]. A mi esetünkben a pálya felfedezése a játék nagy részét képezi, így a túl agresszív játékosok kifejezetten rontják a játékélményt.

Szintén effektív illúzió lehet különböző személyiségek rendelése az ellenfelekhez - ebből adódóan az AI kiszámíthatóbb lesz. Ez lehetővé teszi, hogy a játékos szándékossággal játsszon: „A játékos képessége céljai kitűzésére és megvalósítására, a játékról kialakított tapasztalata és a játék által nyújtott erőforrások kihasználásával” – Clint Hockings [9].

4.3. AI és a csalás

Játék AI szempontjából akkor beszélhetünk csalásról, amikor a programozó olyan információt vagy utasítást ad meg az AI ágensének, mellyel ugyanabban a helyzetben a játékos nem rendelkezne [10]. A régebbi játékokban a csalás elengedhetetlennek tűnt, mivel a mesterséges intelligencia területének a kezdetein jártunk és a fejlesztők csak így tudták kompetitív szintre emelni az ellenfeleket. A játékosok persze nem szeretik, ha az AI csal, de ennek felismerése nem könnyű feladat. Jó példa erre a Sid Meier's Civilization (MicroProse, 1991) nevű játék, ahol a készítőnek szándékosan ki kellett vennie az AI-ok szövetségkötését, mivel túl jól használták funkciót, így a játékosok csalásnak vélték [11]. A modern játékok világában is sok rejtett csalás van, legtöbbször viszont a játékosok javára. Az Uncharted 2 (Naughty Dog, 2009) játékban például az akadályok mögül való kitekintéskor az ellenfelek pontossága nullára csökken, így lehetőséget ad a játékosnak egy „ingyen” lövésre [12].

5. Hasonló rendszerek bemutatása

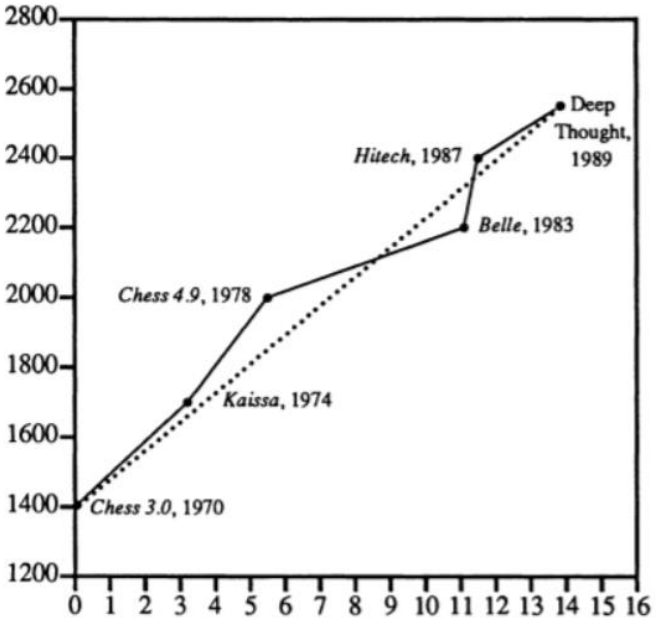
Ebben a fejezetben az játékunkhoz hasonló rendszereket mutatunk be, nem feltétlenül műfaj alapján (VCMI – körökre osztott stratégia), hanem mesterséges intelligenciájuk hasonlósága és ebből eredő tanulságai miatt. A további fejezetekben bemutatjuk a VCMI AI további változatait.

5.1. Külső rendszerek bemutatása

Ez az alfejezet a VCMI-n kívüli rendszereket részletezi.

5.1.1. Sakk

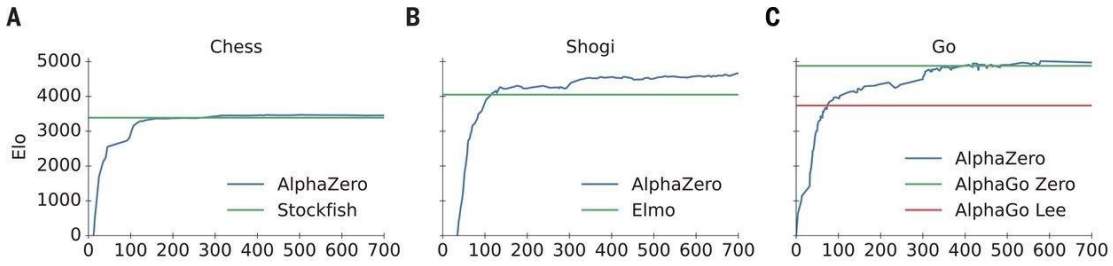
A sakk nagy múlttal rendelkezik a mesterséges intelligencia történelmében. Már 1968-ben létezett egy átlagos emberi szinten játszó számítógép [13], a Mac Hack Six, mely emberileg beállított szabályok és keresési fák segítségével működött. A következő évtizedeket a brute force elv dominálta, kifejezetten elterjedt volt az alfa-béta keresés nevű algoritmus [14]. Ekkor merültek fel kétségek a sakk AI területének hasznosságáról is. A gépek nem emberszerűen játszottak, fejlődésük szimplán a hardver fejlődésének mellékterméke volt, mivel nagyobb mélységű keresőfákat tudtak feldolgozni (*5.1. ábra*). Ennek élő példái az évszázad végén épült szuperszámítógépek, például a DeepBlue [4].



5.1. ábra - AI USCF Képesség / átnézett csúcsok száma mozgásonként: $10000 * 2^n$ (Forrás: [13])

Bár a sakkhoz épített specifikus rendszerek verhetetlenek voltak, területükön kívül csak nagy emberi beavatkozással működtek. A 2017-ben publikált AlphaZero program viszont más szögből közelíti meg a mesterséges intelligenciát. A program visszacsatolt mély neurális hálózatok segítségével működik [15] [16], és saját magával való játszás során tanítja magát.

A rendszer Monte Carlo fa keresést (MCTS) használ [17] a tradicionális alfa-béta kereséssel szemben. Elegendő információ gyűjtése után, több játékterületen is sikeres eredményeket ért el (*5.2. ábra*).



5.2. ábra - AlphaZero Élő-pont / lépések száma ezrenként (Forrás: [17])

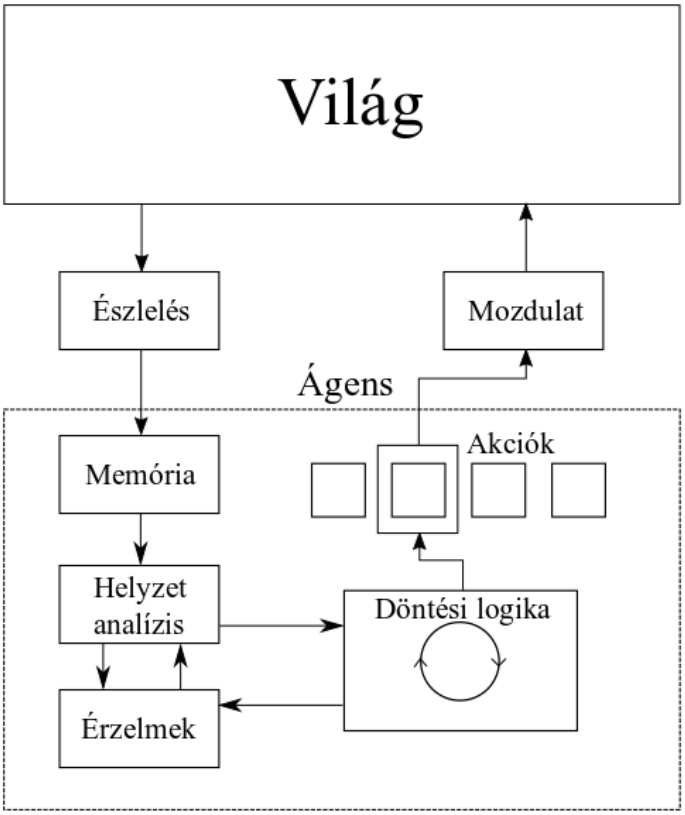
5.1.2. Sid Meier's Civilization

A Sid Meier's Civilization videójáték-sorozat játékmenete (*5.3. ábra*) sokban hasonlít a VCMI-hoz. A játékot I.e. 4000-ben kezdjük kiválasztott civilizációnkkal. Az évszázadok során terjeszkedhetünk, kiépíthetjük városainkat, háborút indíthatunk, illetve szövetséget köthetünk a többi civilizációval. A játéknak több nyerési feltétele is van: diplomatikus, kulturális, tudományos, hadi úton is győzhetünk. A játékmenetet a saját rendszerünkkel összehasonlítva, egy bonyolultabb térkép és kastély nézetű, de csata nézet nélküli rendszerként tudnánk leírni. A játékok alapján több nyílt forráskódú projekt keletkezett, mint például a FreeCiv és a C-evo. A civilizációk fontos tulajdonsága a hozzájuk rendelt személyiségük. Ez határozza meg diplomáciai döntéseiket, preferált győzelmi módjukat, terjeszkedési taktikáikat. Az AI-ok a játékon belüli pozíciójuk alapján különféle stratégiákat is alkalmaznak, például a játék elején a felfedezésre és a terjeszkedésre összpontosítanak, gyenge szomszédok esetén gyorsháborúkat indítanak.

5.3. ábra - Sid Meier's Civilization VI (Forrás: [18])

5.1.3. Halo

A Halo sorozat műfaját tekintve nem hasonlít az eddigi példákhoz, AI szempontból sok hasznos információt és érdekességet találhatunk működésében. A készítők az intelligencia illúzióját több módszer kombinálásával oldották meg. Az egyik módszer a 5.1.2. fejezetben már említett személyiségrendelés [6]. Ez könnyebbé teszi a játékosnak az ellenfelek kategorizálását, így tapasztalatai alapján kiismerheti a játék működését. Továbbá stratégiai célokat rendeltek az ellenfelekhez, hogy az AI ne egy semleges nézőnek tűnjön, hanem aktívan akadályozza a játékost. Monotonitás elkerülése végett célszerű volt a játékmenetbe bizonyos mennyiségű kiszámíthatatlanságot vezetni. A véletlenszerű akciók viszont az eddig említett pontokkal elvben ütköznek. A megoldás a játékos döntéseire épülő reakciók bevezetése volt. Erre először egy Fuzzy logikára alapuló rendszert használtak, később véletlenszerűsége miatt egy érzelemtáblázatot alakítottak ki az ágenseknek. Az AI rendszer végleges felépítése az alábbi ábrán (*5.4. ábra*) található.



5.4. ábra - Általános felépítés (Forrás: [6])

5.1.4. The Battle for Wesnoth

A Battle for Wesnoth (2003, David White) című, nyílt forráskódú projekt hasonló szabályrendszert alkalmaz az eddig említett példákhoz. Az AI szintén akciólistákat használ, majd a játékmeneten belüli körülményektől függően, érték szerinti rendezés alapján végrehajtja azokat.

5.2. A VCMI létező mesterséges intelligenciái

Ez az alfejezet a VCMI jelenlegi AI moduljait fejti ki.

5.2.1. StupidAI bemutatása

A StupidAI lényegében a készítők által nyújtott üres AI sablon. A modul csak a legalapvetőbb metódusokat tartalmazza. Referenciákat is tartalmaz a VMCI logika és könyvtár modulhoz, melyben a játék és ágenseinek állapotáról való információk vannak eltárolva. Az ágensek viselkedése alapvető szinten van, egyszerű útkeresési algoritmusokat használ.

5.2.2. VCAI bemutatása

A VCAI a VCMI alapértelmezett és egyben fő mesterséges intelligenciája. Az AI a fuzzy logikára épül, melyet a Fuzzylite nevű projekt segítségével valósít meg. A StupidAI-hoz hasonlóan sokban függ a VCMI_lib (logika és könyvtár) modultól. A modul egy Goals mappát tartalmaz, mely az AI fő céljait részletezi. A Pathfinding mappa a térkép nézeten való útvonalkereséssel foglalkozik. Tartalmazza a lehetséges mozgási akciók listáját, a játék mozgáshoz kötött szabályait. Maga az útvonalkereső algoritmus csomópont rendszert alkalmaz, vagyis a térképen lévő fontosabb objektumokhoz (például nyersanyagbányák) csomópontokat rendel. A csomópontok értékeinek meghatározásáról egy külön osztály felel, végül fuzzy logika segítségével végső döntést hoz. Az egységek toborzásával és a nyersanyagokkal való gazdálkodással is külön-külön osztályok foglalkoznak.

5.2.3. BattleAI bemutatása

Bár külön modulnak vesszük, a BattleAI lényegében a VCAI bővítménye a csata nézetre. Működése ugyanúgy a fuzzy logikára alapul. Az AI lépésénként összegyűjti egységeinek listáját és potenciális sebződését, melyek az ellenfél egységek hatókörein belül állnak. Ugyanezt a lépést fordítva is megteszi, vagyis tárolja azon ellenfél egységek listáját, melyeket saját egységei elérnek, valamint az ekkor okozott potenciális sebzéseket. Az értékek összevetése után az AI fuzzy logika segítségével dönt.

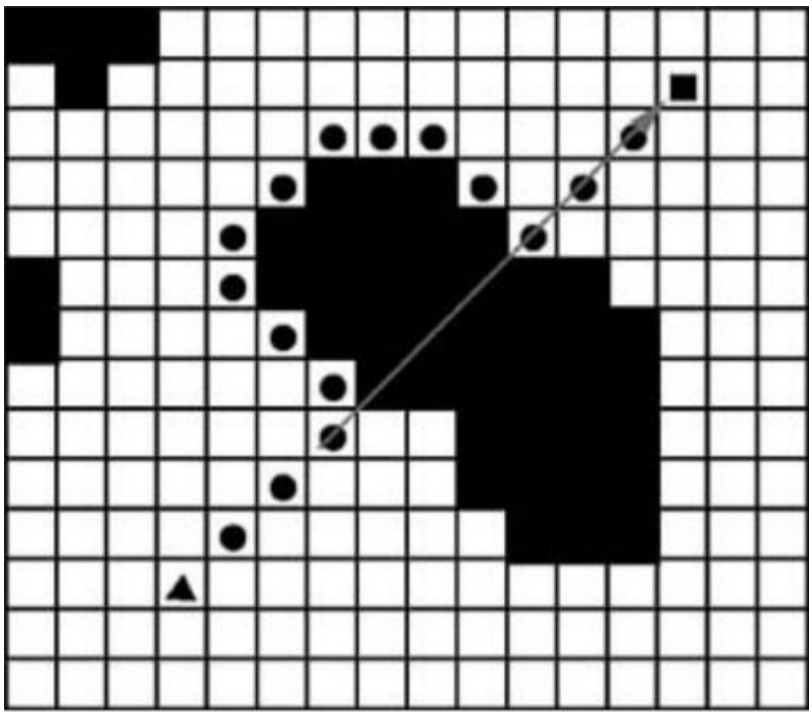
6. Irodalomkutatás

Ebben a részletben jobban megismerkedünk a szakdolgozattal kapcsolatos irodalommal.

6.1. Útvonalkeresés bemutatása

Az útvonalkeresés egy nagyon sokrétű ág a játékok mesterséges intelligenciájában. Minden játékban más-más módszereket alkalmaznak a készítők, melyek sokszor a rendelkezésre álló időtől, és a pálya területének nagyságától függenek. Az AI for Game Developers [19] című könyv részletez pár lehetséges változatot.

Az egyik ilyen megoldás az ún. Akadályok mentén mozgó algoritmus. Első lépésben az algoritmus egy egyenes vonalat rajzol jelenlegi állapota és célja között. Az algoritmust 2 állapotra tudjuk szétosztani, az elsőben az ágens elindul egyenesen a célja felé, majd egy akadályba ütközve átvált másik állapotába. A második állapotban az ágens addig járja körbe az akadályt, amíg el nem éri az első lépésben kirajzolt egyenes vonalat. Ilyenkor az algoritmus ismét visszavált az első állapotra (*6.1. ábra*).



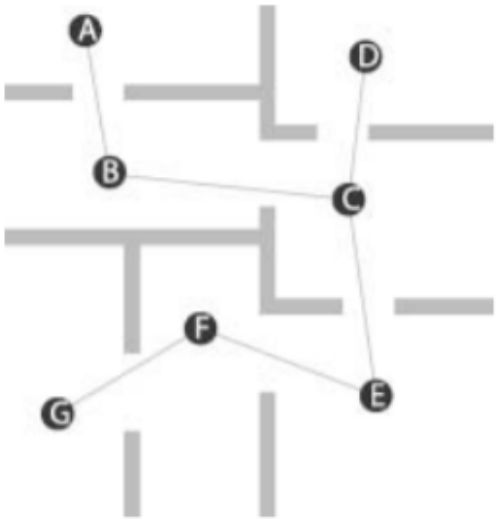
6.1. ábra - Akadályok mentén mozgás (Forrás: [19])

Egy következő megoldás a Kenyérmorzsa útkereső algoritmus. Egy kiegészítő algoritmusnak tekinthetjük, viszont bizonyos helyzetekben kifejezetten hasznos lehet, például több ágenst tartalmazó pályán. Az algoritmus a többi ágens által kivájt utakat követi, így például üldözésnél is hasznos lehet.

Továbbá említésre méltó a Monte Carlo fa keresés (MCTS) is. Az algoritmussal már egy előző fejezetben *(5.1.1. fejezet*) találkoztunk, a sakk AI területén nagy sikereket ért el. Az algoritmus keresőfa segítségével működik, egy nagy mélységű fát épít ki. A játék során

a fa sikeres ágait választja ki, kibővíti őket, szimulációval kiszámítja értéküket, majd a kiszámolt értékkel arányosan visszafele menőleg módosítja szüleinek értéket.

A bevezetőben említett időkorlát nagy hatással volt az útkereső algoritmusok fejlődésére, így új módszerek jelentek meg az idő csökkentése végett. Dijkstra legrövidebb út algoritmusa ezeknek a módszereknek az atyja. Az alapötlet, hogy a fontos pontokban súlypontokat helyezzünk el, így elég csak a súlypontok közötti útvonalkeresést megoldani (*6.2. ábra*), mellyel sok időt megspórolhatunk.



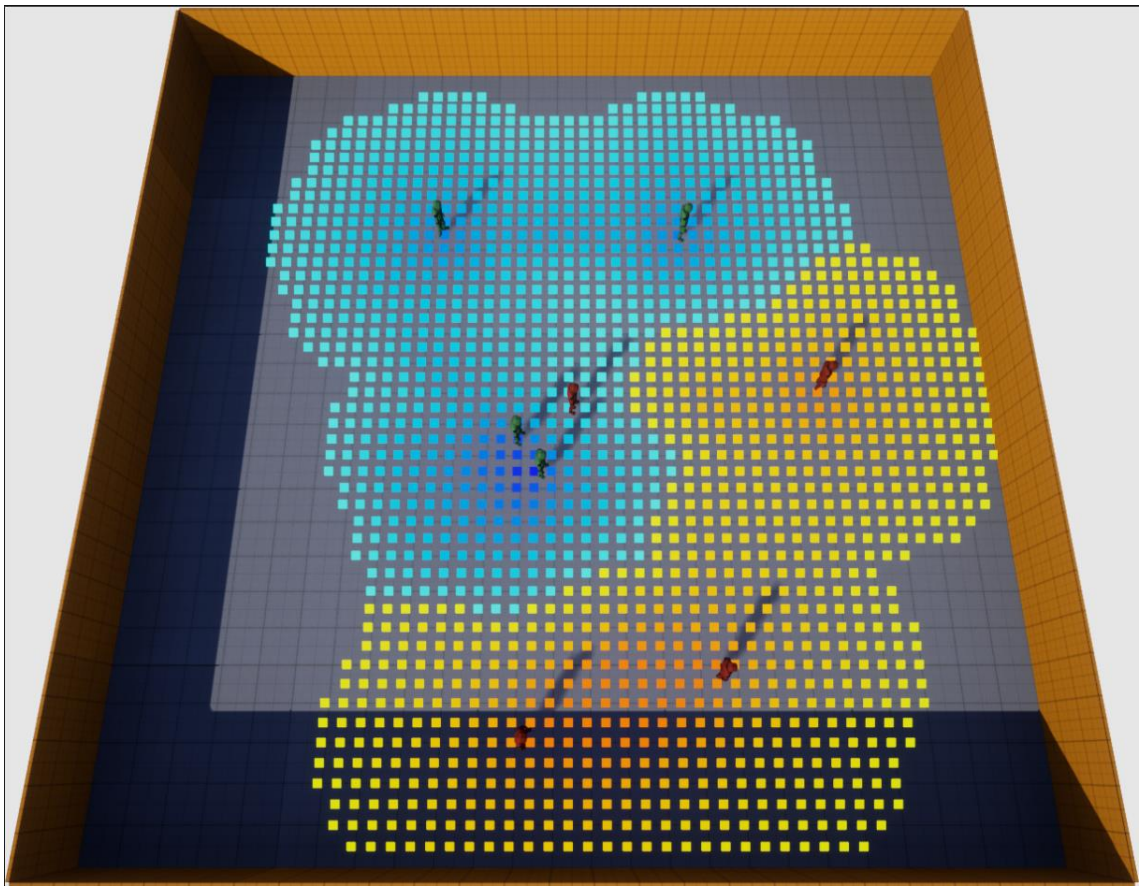
6.2. ábra - Csúcspontok kialakítása és összekötése (Forrás: [19])

Végül, de nem utolsósorban ide tartozik az A* keresés is, ami Dijsktra algoritmus egyik változata, és mára a leggyakrabban használt kereső algoritmus. Az algoritmust működését egy hullámhoz hasonlíthatjuk, mely a csomópontokból kiindulva terjed, amíg el nem éri a keresett csomópontot. Az algoritmus több helyes utat is találhat, viszont az optimális útvonalat a hullámok alatti talajokhoz rendelt értékek segítségével számítja ki.

6.2. Hatástérképek

A mesterséges intelligenciát használó játékok világában egy jelentős probléma a játékban lévő komplex adatok konvertálása egyszerű, AI által feldolgozható részekre [20]. Ennek megoldására számtalan fajta megoldás létezik, melyek közül mi a hatástérképekkel fogunk foglalkozni. A hatástérképek fő célja az információgyűjtés, melyet a játékkörnyezeten belüli AI ágensek fel tudnak dolgozni. Itt fontos megjegyezni, hogy a hatástérképek sosem adnak direkt utasításokat, csak segítenek azok meghozásában [21]. Működésük alapja arra a koncepcióra épül, hogy a játékon belül minden objektum egyfajta hatást sugároz ki magából. Ezeknek a hatásoknak az összevetésével, változásaik követésével pedig értékes információkat adhatunk át az AI számára. A hatástérképek 3 különböző fajta információt nyújtanak [22]. Először is egy könnyen értelmezhető helyzetösszefoglalót adnak a játék környezetéről. Ebből megtudhatjuk, mely objektumok irányítják a pálya bizonyos részeit, hol vannak ezen területek közti határok. Erre egy

példát az alábbi ábrán (*6.3. ábra*) láthatunk. Az ábrán a zöld és a piros csapat kombinált hatástérképe látható. A zöld csapat hatása kék színnel, a piros csapaté pedig sárga színnel van jelölve. Látható, hogy a hatások erősségét a színek sötétsége határozza meg.



6.3. ábra - Két ellentétes oldalon álló csapat közös hatástérképe (Forrás: [20])

Egy másik fajta információhoz juthatunk a hatástérképek változásainak feldolgozásával. Ennek segítségével ellenőrizhetjük bizonyos akciók sikerességét. Végül a jövőbeli döntések meghozásában is támogathatja AI-unkat. Különböző tendenciák felfedezésével megjósolhatjuk bizonyos objektumok következő lépéseit.

6.3. Mesterséges neurális hálózat bemutatása

A Csata AI megvalósításához egy neurális hálót is fel szeretnénk használni, így ez az alfejezet annak általános bemutatására szolgál. Az ötlete biológiai ihletésű. Egy gráf alapú modellből áll, melyben rétegekbe rendezett mesterséges neuronok kommunikálnak egymással aktivációs függvények segítségével. A hasonló neuronokat egy összesített neuronrétegként kezeljük. A neurális hálót 3 részre tudjuk felosztani:

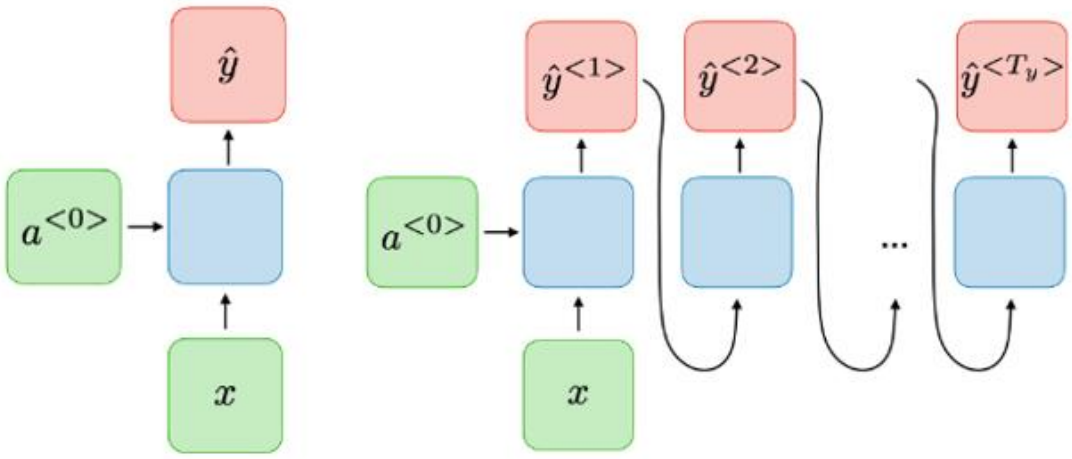
- Bementi réteg – a bemenetként kapott adatok szerint meghatározza a háló neuronjainak számát, majd továbbítja aza adatokat a rejtett rétegnek.
- Rejtett rétegek – a bemeneti információkat transzformálják aktivációs függvények segítségével, illetve köztes reprezentációkat hoznak létre a neuronok között

- Kimeneti réteg – a kimeneti réteg funkciója a feladattól, és a kimeneti függvény típusától is függ.

Több alkalmazásprogramozási felület segítségével is lehet neurális hálózatot implementálni C++-ban, például a Microsoft által fejlesztett CNTK könyvtárral, és Google Brain csapat által kiadott TensorFlow könyvtárral.

6.4. Rekurrens neurális hálók

Az általános neurális hálók működésekor egy tanítási ciklusban a háló feldolgozza a bemeneti rétegen érkezett adatot, a rejtett rétegekben lévő neuronok értékei módosítva vannak az adat által, majd a feldolgozott adat távozik a kimeneti rétegen keresztül. Ezek után egy új ciklus indul a következő kötegnyi tanító adattal. Azonban mi van akkor, ha szeretnénk, hogy a ciklusok eredményei hatással legyenek egymásra? Ez a rekurrens neurális hálók alapja. A *6.4 ábrán* a két háló közti különbségeket láthatjuk.



6.4. ábra - Tradicionális háló (NN) és rekurrens neurális háló (RNN) (Forrás: [23])

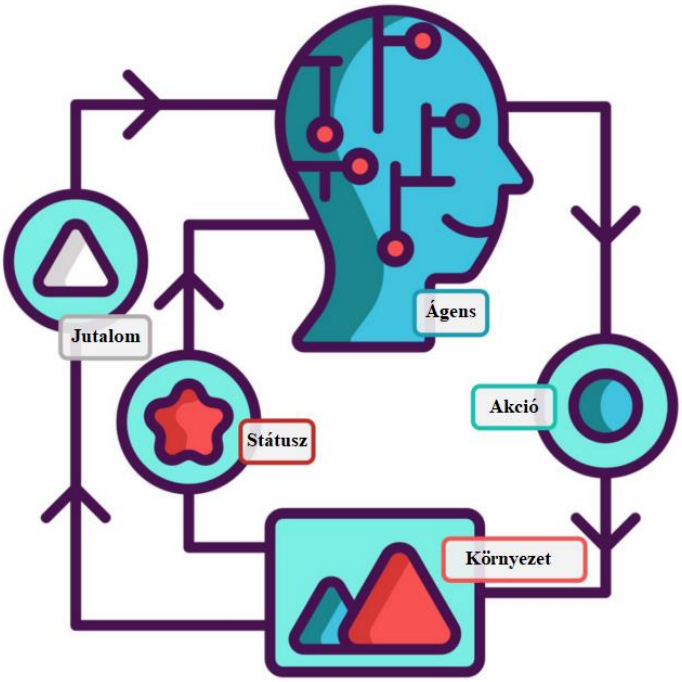
A rekurrens háló(jobb) kimeneti értékeit betáplálja az utána következő ciklusba, így az adatok között egyfajta időbeli, dinamikus kapcsolatot hoz létre. A neurális háló tulajdonképpen így egy rövid távú memóriára tesz szert. Több fajta módon is történhet ez az adatátadás, különböző módszerek különböző felhasználási tereken effektívek. A képen például egy Egy-a-Többhöz típusú rekurrens hálót láthatunk, mivel az első ciklus kimeneti adatai befolyásolják a többiét. Ilyen fajta hálókat használnak például zenék generálására.

Mikor előnyös egy rekurrens hálót használni? Célszerű, ha a bemenetünk egymást követő, sorrendiségükben összefüggő paraméterekből áll. Ezenkívül, ha fontos, hogy ez az összefüggés hatást gyakoroljon a tanítás folyamatára is. Ezek a hálók főleg a természetes nyelvfeldolgozás és a beszédfelismerés területén elterjedtek.

6.5. Megerősítéses tanítás

A megerősítés tanulás (Reinforcement Learning - RL) a gépi tanulás harmadik területe, a felügyeletes tanítás és a felügyelet nélküli tanítás mellett. Célja egy intelligens ágens lehető legnagyobb kumulált értékének elérése egy környezetben, melyet optimális akciókkal tehet meg. A másik két területtel ellentétben – amik alá az előző fejezetekben részletezett neurális hálók is tartoznak (*6.2-6.3 fejezetek*) – működésükhöz nem szükséges nagy mennyiségű megcímzett adatmennyiség betáplálása. Ehelyett a hangsúly az egyensúly megtalálására helyeződik az új információk felfedezése és a már megismert információ feldolgozása között [24].

Egy RL rendszer működtetéséhez először célszerű pontosan meghatározni a megoldani kívánt problémát, mivel ez alapján van a jutalomtartomány kialakítva [25]. A célunk ennek a jutalomnak a maximalizálása, amit különféle akciók elvégzésével tehetünk meg. Maga a folyamat szekvenciális, és a benne lévő akciók kihatnak egymásra. Ez miatt néha jobb feláldozni egy azonnali jutalmat egy hosszú távú jutalom fejében. A megfelelő akciók kiválasztása és továbbítása az ágens feladata. Ezt a döntést a környezetétől kapott státusz, és az ehhez tartozó jutalom segítségével hozza meg. A döntésben nagy szerepe van az ágens irányelvének és értékfüggvényének is. Az ágens kiválasztott akcióját ezután továbbítja a környezetének. Majd a ciklus megismétlődik, amíg el nem érünk valamilyen megállási feltételt. A folyamatot vizuálisan reprezentálva a *6.5. ábrán* láthatjuk.

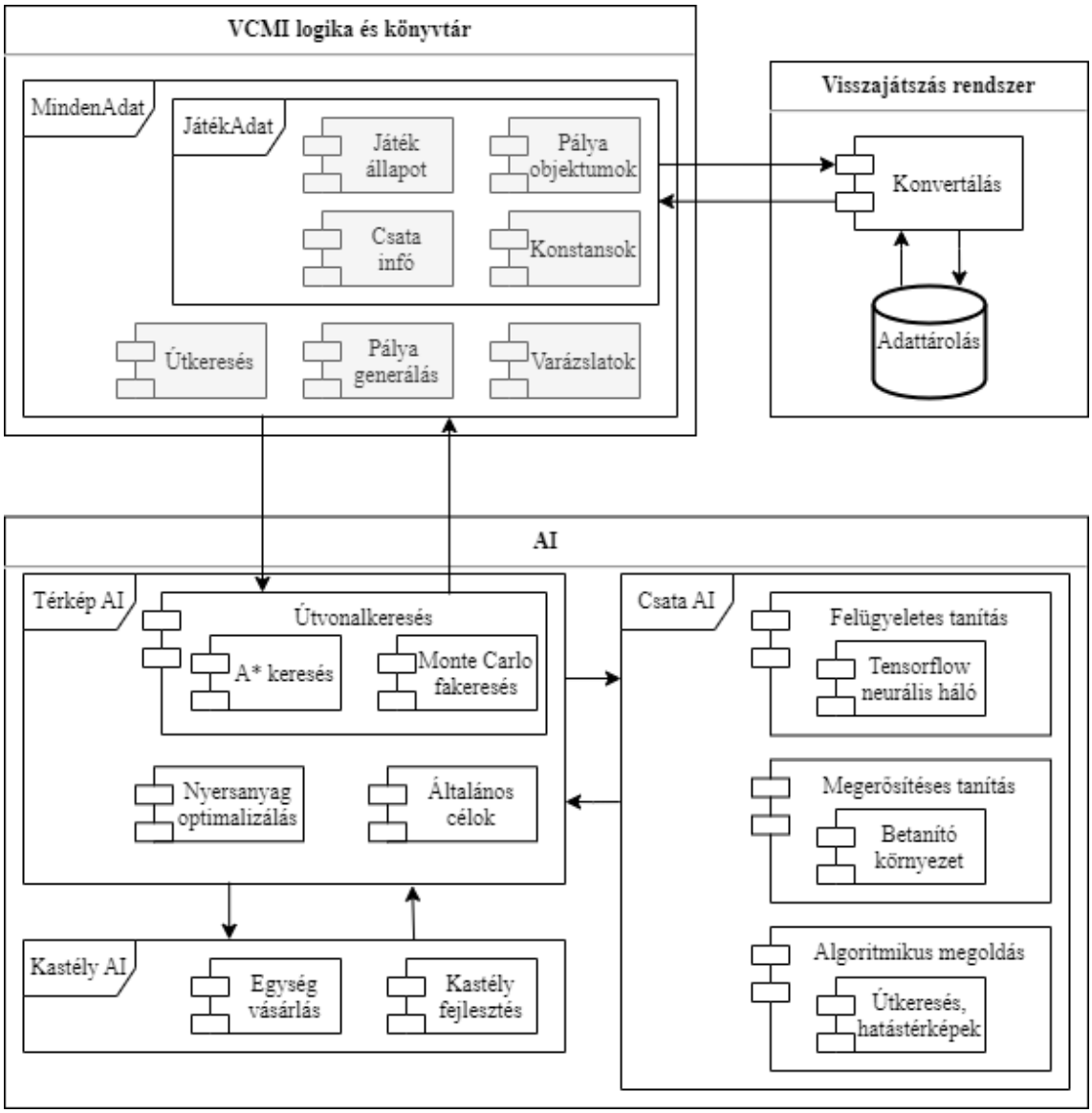


6.5. ábra - RL rendszer működése (Forrás: [26])

A megerősítéses tanításnak létezik egy alterülete mély megerősítéses tanítás névvel, mely kombinálja az RL felépítését, egy nagy mennyiségű adat feldolgozására képes neurális hálóval.

7. Rendszerterv

Ebben a fejezetben felvázoljuk a projektünk felépítését (*7.1. ábra*), majd a soron következő alfejezetekben modulokra szedve részletezzük őket.



7.1. ábra - A rendszerterv felépítése

7.1. VCMI logika és könyvtár

A VCMI logika és könyvtár modul (*7.1. ábra*) már a meglévő rendszer része, mégis fontos szerepet tölt be projektünkben. Az egyes elemei szürkével vannak kijelölve, ezzel is jelezve, hogy már kész elemekről beszélünk. A JátékAdat keret a következő egységekből épül fel: Játék állapot, Pálya objektumok, Csata infó, Konstansok.

A Játék állapot az általános információkat tartalmazza a játék jelenlegi állapotát tekintve, például a hősök, kastélyok, nyersanyagok számát. A Pálya objektumok egység a térkép nézeten lévő jelentős objektumok tulajdonságait és metódusait öleli fel. A Csata infó-ban

a csata nézetről tárolt összes információ van eltárolva, beleértve a csatatér típusát, a csatában részt vevő hősöket és egységeiket, a pályán lévő akadályokat. A Konstansokban pedig a játékhoz tartozó konstans változók, és egyéb statikus információk vannak eltárolva.

A bővebb MindenAdat keret tartalmaz egyéb segédosztályokat is, melyek segítséget nyújtanak az AI modul készítésénél. Említésre méltóak az Útkeresés, Pálya generálás és Varázslatok egységek. Az Útkeresés a térkép nézeten való mozgással kapcsolatos információkat tartalmazza, ideértve a mezők elérhetőségét, a hősök mozgáspontjainak számát, a mozgások típusát (például vízi, szárazföldi). A pálya generálás a véletlenszerűsített pályák elkészítését végzi el. A Varázslatok pedig játékban lévő összes varázslatot, és azok hatásait tartalmazza.

7.2. Visszajátszás rendszer

A Visszajátszás rendszerünk (*7.1. ábra*) felépítése nem túl bonyolult, szerepe viszont jelentős a munkánkban. A Konvertálás, valamint az Adattárolás egységből tevődik össze.

A Konvertálás egység a JátékAdat keretből kapott információkat konvertálja az Adattárolás egység struktúrája alapján, majd eltárolja azokat a kiválasztott adattárolónkban. A Csata AI-hoz tartozó neurális hálózat betanítása esetén, a JátékAdat keretben lévő adatokat a Visszajátszás rendszer biztosítja, az adattárolónkból lekért adatok visszaalakításával. Az adatokat játékmeccsenként szeretnénk tárolni, melyek körökre vannak tovább osztva. A körök pedig tovább vannak bontva különböző játékon belüli akciókra.

7.3. Térkép AI

A Térkép AI keret (*7.1. ábra*) a térkép nézet mesterséges intelligenciájával, és az általános játék AI-al is foglalkozik. A modul szerepei közé tartozik a VCMI logika és könyvtár modul többi AI-al való összeköttetése, illetve azok működéséhez szükséges releváns információk továbbítása is. Erre a keretre ruháztuk fel ezeket a felelősségeket, mivel mindkét másik nézet a térkép nézeten való interakcióval érhető el.

A keret több egységből áll össze. Az Útvonalkereső modul a térkép nézeten való útvonalkereséssel foglalkozik. Különböző útvonalkereső algoritmusokkal – például: A* keresés, Monte Carlo fa keresés– kiszámítja az optimális mozgást, majd ezeket az információkat betanítja egy neurális hálónak. A végső döntést a háló kimeneti rétege alapján hozza meg. A Nyersanyag optimalizálás az optimális nyersanyaggyűjtést biztosítsa, nagyobb prioritást ad az egységeink megvásárlásához és épületeink megépítéséhez szükséges nyersanyagoknak. Az Általános célok pedig a játékmenettel kapcsolatos általános célokat határoz meg az AI számára.

7.4. Csata AI

A Csata AI (*7.1. ábra*) 3 különböző fajta megoldást kínál egy sikeres Csata AI kiépítéséhez. A megvalósításhoz szükséges adatokat a Visszajátszás rendszer biztosítja, melyet a Térkép AI-n keresztül kapunk meg. A Felügyeletes tanítás modulban a tárolt adathalmazunk segítségével tanítjuk be a neurális hálónkat. A Megerősítéses tanítás modul egy segédkörnyezet kialakításáról, majd az abban lévő ágensek betanításáról szól. Az Algoritmikus megoldás pedig útkereső algoritmusok és hatástérképek felhasználásával építi fel AI modulunkat.

7.5. Kastély AI

A Kastély AI (*7.1. ábra*) felépítése viszonylag egyszerűbb a többi modulhoz képest, viszont szerepe ugyanúgy fontos az AI szempontjából. Az Egység vásárlás az optimális egységvásárlással foglalkozik, ügyel a hadseregek egyenletes kompozíciójára (közelharci-lövész-repülő egység) és a rendelkezésre álló nyersanyagok számára. A Kastély fejlesztés pedig az épületek megépítésének optimális sorrendjét határozza meg.

8. Specifikáció

Ez a fejezet a projekt megvalósításához szükséges konkrét technikákat és módszereket részletezi.

8.1. Általános specifikáció

A kódot jelentős részét C++ nyelven fogjuk írni, a Microsoft Visual Studio 2019 keretrendszert használva. A neurális hálónkat Python nyelv segítségével valósítjuk meg.

8.2. Visszajátszás rendszer specifikáció

A Visszajátszás rendszer adattárolását json formátumú fájlok segítségével valósítjuk meg. Minden fájl egy meccsnyi adatot fog tartalmazni. A fájlokat egy közös Maps mappában tároljuk majd.

8.3. Térkép AI specifikáció

Az útvonalkeresésre több opciót biztosítunk. A fő algoritmusaink - A* keresés algoritmus és a Monte Carlo fa keresés - mellett kísérletezünk további útvonalkereső algoritmusokkal is, az összehasonlítás célja végett.

8.4. Csata AI specifikáció

Neurális hálónk megvalósításához egy virtuális környezetet hozunk létre, melyben a Jupyter Notebook fejlesztői környezet segítségével dolgozunk. A TensorFlow nevű programkönyvtár mellett, több kisebb segédkönyvtárt is használunk.

A megerősítéses tanítást RLlib vagy Gym könyvtárak segítségével szeretnénk megvalósítani. Ezek szintén Python könyvtárak, melyek a tanításhoz szükséges környezet felépítésében segítenek.

Az algoritmikus megoldást a VCMI környezetén belül szeretnénk kialakítani.

8.5. Kastély AI specifikáció

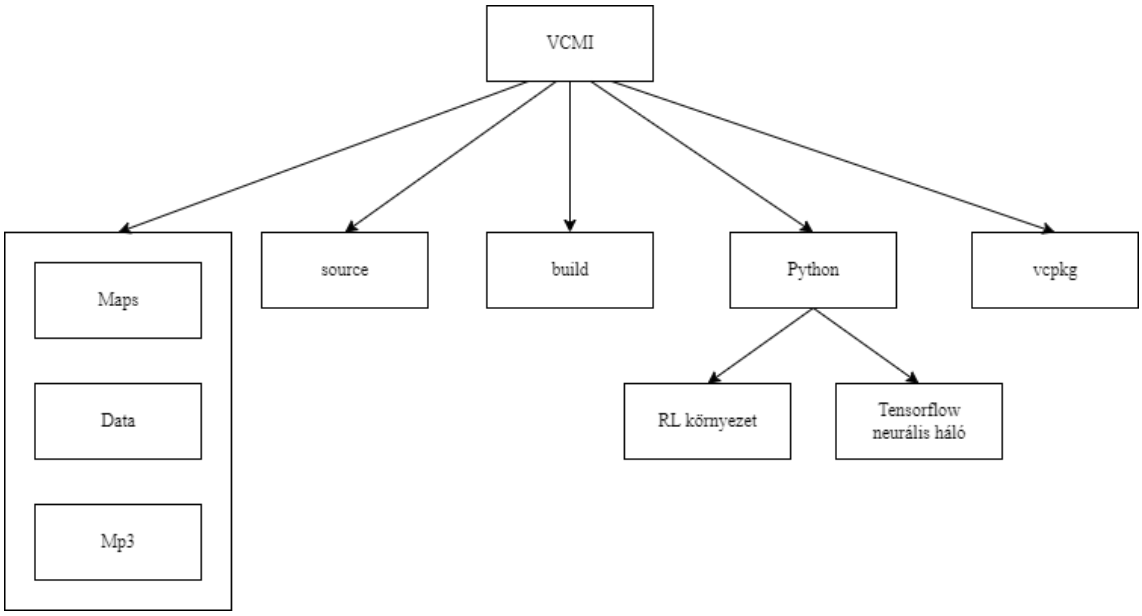
A kastély nézetben lévő algoritmusok implementálásához különféle listákat alkalmazunk, melyek tartalmazzák az egységek típusát, árát és erejét. A kastély fejlesztésére használt dinamikus lista az aktuálisan fejleszthető épületeket, és azok hatásait tartalmazza.

9. Megvalósítás

A fejezetben munkánk megvalósításának menetét fejtjük ki bővebben. A részfejezetek a megvalósítandó modulok, és azok megvalósításához szükséges lépésekből összegyűjtött modulokból épülnek fel. Törekedünk az implementáció időbeli sorrendjének megtartására, kivéve, ha az veszélyezteti a feljebb említett logikai felépítést.

9.1. VCMI inicializálása

A fejezet összefoglalja VCMI környezetet inicializálásához, telepítéséhez és működtetéséhez szükségünk lépéseket. A környezetünk felépítését az alábbi ábrán láthatjuk (*9.1. ábra*).



9.1. ábra - A projektünk mappa struktúrája

9.1.1. Előfeltételek

A VCMI egy Microsoft Visual Studio-ban fejlesztett projekt, pontosabban a 2015-ös verziójában. A fejlesztők az évek során folyamatosan kompatibilisen tartják az újabb verziókkal, ezért a működtetéséhez használhatjuk a 2015, 2017 és a 2019-es Visual Studio-t is.

A projekt git segítségével van publikálva egy interneten lévő privát repository-ba. A git egy nyílt forráskódú, elosztott verziókezelő rendszer. A rendszer több előnnyel is rendelkezik, a projektünkben például növeli az átláthatóságot, a különböző változtatások, és azokhoz fűzött megjegyzések tárolásával. A VCMI nyílt forráskódú, ezért bárki letöltheti a kódot a megfelelő forrásról. A folyamat megkönnyebbítése érdekében egy gitkezelő alkalmazás segítségét vesszük igénybe, a Sourcetree-t.

Mérete miatt nem fér el a VCMI teljes terjedelme a GitHubon – a weboldalon, ahol a projekt publikálva van -, ezért egy CMake nevű szoftverfelépítést segítő, és egyben automatizáló alkalmazás is szükséges a sikeres telepítéshez.

A Microsoft által fejlesztett vcpkg csomagkezelő rendszert is igénybe vehetjük, de ez csak opcionális, mivel a VCMI csapata biztosít nekünk egy általuk összerakott rendszert is.

9.1.2. Futtatható kód előállítása

A projektnek egy ASCII-karakter nélküli, nem védett mappát válasszunk. A mappában készítsünk egy almappát a vcpkg csomagoknak. Mi az előre megépített csomagot használjuk, ezért a vcpkg manuális összerakásával nem foglalkozunk.

A következő lépés a VCMI leklónozása GitHubról, a gitkezelő programunk (Sourcetree) segítségével. A letöltött fájlokat a source (forrás) mappában tároljuk. Az így kapott kódot a projektünk csontvázaként kezelhetjük. Ezek az osztályok határozzák meg az egész projektünk működését, de jelenleg még nem rendelkezünk az általuk működtetett adatokkal, és működő hozzáférésünk sincs az osztályokhoz.

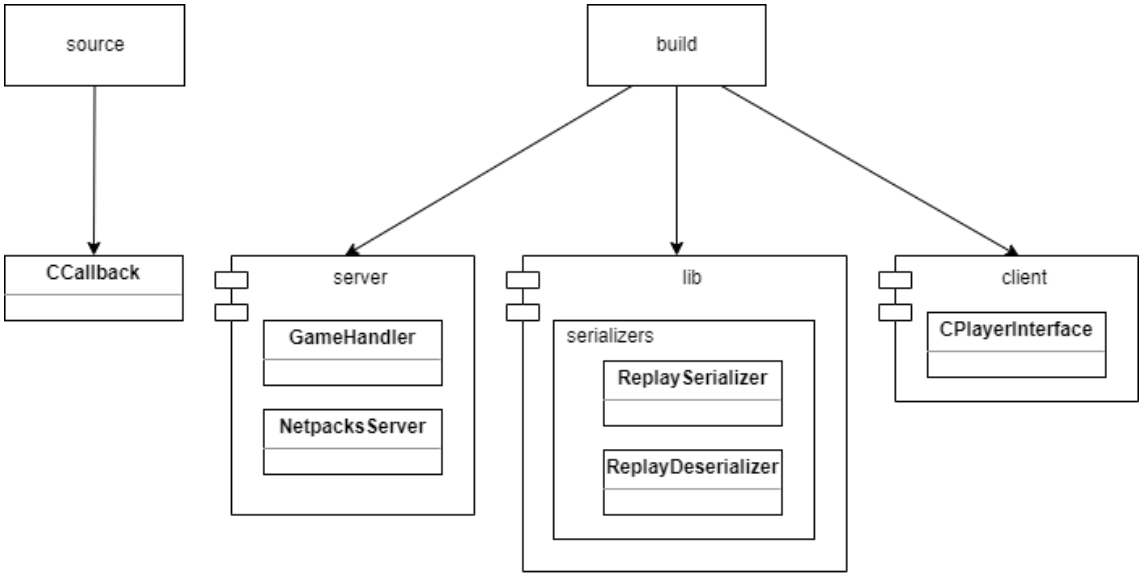
Létrehozhatjuk a következő almappánkat is build néven. A parancssoron keresztül generálhatjuk le az ide tartozó fájlokat a CMake segítségével, mely felhasználja a vcpkg-ben tárolt könyvtárakat is. Az így keletkezett Visual Studio solution a hozzáférési pontunk a projekthez. A source mappában lévő osztályokra hivatkozik, és tartalmazza működésükhöz szükséges könyvtárakat is, így ezzel a környezettel dolgozunk.

A szükséges változtatások elvégzése után, a projektben lévő BUILD_ALL paranccsal lefordíthatjuk programunkat, amit a build mappa bin almappájában találhatunk meg és indíthatunk el.

A megépített alkalmazás sikeres működéséhez először még át kell másolnunk az alap mappában tárolt Maps, Data és Mp3 mappákat. A mappák a játék működtetéséhez szükséges vizuális, hang- és egyéb fontos fájlokat tartalmaznak. Ezek a fájlok az eredeti játék forrásfájljai között vannak, így szerzői jogok megsértése nélkül nem biztosíthatják nekünk a VCMI fejlesztői. Ez a lépés tehát egyben biztosítja azt is, hogy az eredeti játék megvétele nélkül senki sem tudja működtetni a projektet.

9.2. Visszajátszási rendszer

A Visszajátszás rendszerünknek 2 fontos szerepe van. Az egyik a játékon belüli akciók elmentése, és későbbi visszajátszása a könnyebb tesztelés érdekében. A másik az adatok kinyerése a Csata AI felügyeletes tanítás változatú moduljához, mivel a neurális háló betanításához sok adatra van szükségünk. A rendszerünk felépítését a *9.2. ábra* szemlélteti.



9.2. ábra - A Visszajátszás rendszer kiemelt elemei

A projektünk óriási mérete, és a minimális dokumentáció miatt, az adatkinyerési pont meghatározása nehéz feladatnak bizonyult.

Az első próbálkozásaink a server mappában kezdődnek, ahol a GameHandler osztály végrehajtja a kliensek általi utasítások nagy részét. Bár az adatkinyerés sikeres, a haladás menete nagyon lassú. A munkánkat akadályozza a sok egymásra épülő metódus, és a számunkra hasznos adatok kiválogatása a többi adat közül.

A második lehetséges kinyerési pontunk a CCallback osztály. Az osztály a szerver és a kliensek közötti adatcsomagok szállítására szolgál. Bár az osztály felépítése jó nekünk - csak az akció lefuttatásához fontos adatokat szállítja-, az osztályhoz nincs hozzáférésünk futásidőben, mivel maga az osztály nem kerül bele a lefordított fájlok közé.

A jó megoldást végül a szerver oldali NetPacksServer osztályban találjuk meg. Ez az osztály kapja meg a CCallback osztály utasításait a klienstől, így a két osztály felépítése sokban hasonlít, ezenkívül futásidőben is képesek vagyunk az itt áthaladó adatokat lekérdezni. Mivel a számítógép által irányított játékosok is ugyanezt a csatornát használják, ezért sikeresen el tudjuk érni a játékban történt összes fontos eseményt.

A visszajátszási pont alatt azt a helyet értjük, ahol a kinyert adatokat visszatápláljuk a rendszernek, és az így létrejött utasítások automatikusan lefutnak körünk kezdetén. Ezt a helyet sikerült második próbálkozásunkra megtalálnunk, a kliens modul CPlayerInterface osztályában.

A visszajátszás inicializálása mellett egy fontos hitelesítő pont is. Itt ellenőrizzük le, a betölteni való visszajátszás fájl jelenlétét, illetve a fájl nevében található egyedi azonosító értékét. Ez az azonosító egy szám, ami minden pálya generálásakor létrejön. A játékmenet véletlenszerűsített értékei ennek segítségével vannak beállítva. Ebből az okból kifolyóan a Visszajátszás rendszer csak egy meglévő játék betöltésénél működik, új játék kreációjánál nem. Mit tegyünk akkor, ha egy egész játékot akarunk visszajátszani? Ez a probléma könnyen megoldható, ha az új játék készítése után azonnal mentünk. Ezen a mentésen keresztül pedig már a játék első akcióit is a program irányítja.

Az utasításaink nagy része sikeresen visszajátszódik, pár kivétellel, melyek a program párhuzamos felépítése miatt szálkezelési problémákba ütköznek.

A fejezetben megemlített modulok és osztályok, a fejezet elején lévő (*9.2. ábra)* ábrán vannak vizualizálva.

9.2.1. A szerializálás megvalósítása

Mind az adatkinyerési, mind a visszajátszás pont metódusai csak segédmetódusok, maga a szerializálás és a deszerializálás folyamata más helyen megy végbe. Ezt a projektünk library (könyvtár) mappájának serializer almappájában, 2 új osztály létrehozásával valósítjuk meg. Ügyelnünk kell, hogy a projekt bővítésekor módosítsuk a felépítést megvalósító CMake utasításainkat is. A lépés kihagyása esetén sikeresen tudunk dolgozni az új osztályokkal és bővítményekkel, azonban a kód lefordítása után ezek a fájlok nem lesznek becsomagolva, vagyis programunk nem tud majd rendesen lefutni.

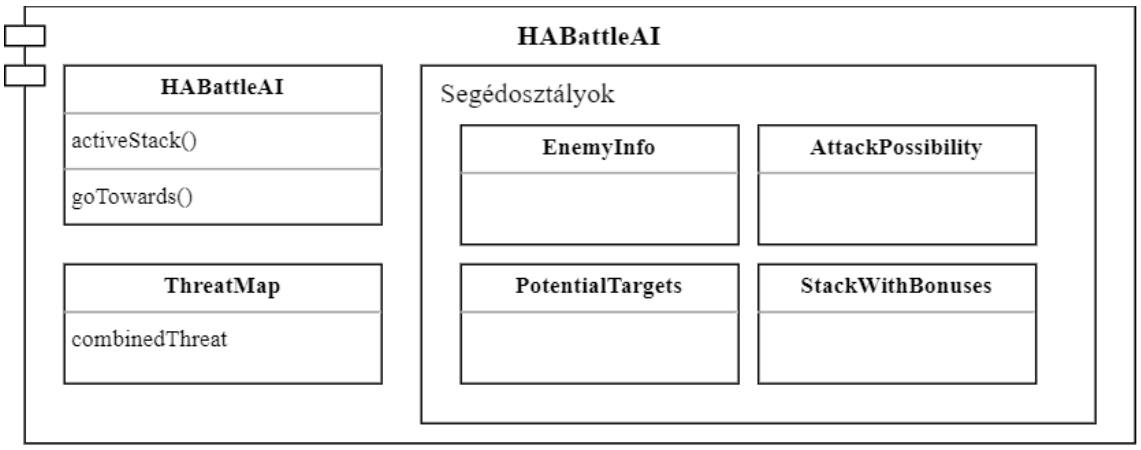
A két új osztályunk közül az első a ReplaySerializer, mely az adatok json formátumú elmentésére szolgál. C++ programozási nyelvben nincs lehetőségünk beépített json kezelésre, ezért az nlohmann_json nevű bővítmény segítségével tesszük ezt meg. Az osztály sok kis metódusból áll össze, melyeknek megnevezései megegyeznek a játékon belüli lehetséges akciók nevével. Egy hős lépését például a serializeMoveHero metódus valósítja meg. A metódusok kis-kis json struktúrákat építenek fel, melyek tartalmazzák a specifikus akció adatait. Egy közös akciótípus tulajdonsággal is rendelkezik minden struktúra, így beolvasásnál ez szerint tudjuk megkülönböztetni őket egymástól. Egy játékon belüli kör lefolyása alatt az akciókat az ReplaySerializer osztályon belüli közös json listához adogatjuk, majd a kör végén ezt a listát egy egyedi néven elmentjük egy közös gyűjtőhelyre. Végül a lista értékeit kitöröljük, így a belső memóriánkban mindig csak maximum 1 környi adatot kell tárolnunk. Az osztályban egy másik json listát is kezelünk, melyben a Csata AI-hoz gyűjtött részletes információkat halmozzuk. Az információgyűjtés megkönnyebbítéséhez pár segédmetódust készítünk, melyek a szerver oldalon lévő, beérkező adatcsomagokat feldolgozó GameHandler osztályban (*9.2. ábra)*

vannak megvalósítva. A Csata AI-nank szánt listánk is a feljebb említett lépések szerint gyűjti és tárolja adatait, így azt tovább nem részletezzük.

A második új osztályunkat ReplayDeserializer néven hozzuk létre. A visszajátszási pontnál van szükségünk metódusaira. Az osztály az elmentett json fájlok megnyitásával, azok körökre, majd akciókra bontásával foglalkozik. A sikeresen szétválasztott akció json-öket pedig eljuttatja a CPlayerInterface osztálynak (*9.2. ábra)*, ahol az utasítások sikeresen visszajátszódnak.

9.3. Csata AI létrehozása

Az alábbiakban megépítjük saját Csata AI modulunkat a VCMI rendszerben, valamint szót ejtünk a további Csata AI megvalósításainkról is. A végső Csata AI-unk felépítését a *9.3. ábrán* láthatjuk.



9.3. ábra - HABattleAI felépítése

9.3.1. Csata modul felépítése

A VCMI bemutatásánál (*3. fejezet VCMI felépítése*) már láthattuk, hogy a saját modulunkat egy létező AI rendszerbe építjük be negyedik elemként. A további három modul közül az egyik – StupidAI – csupán egy üres AI modul keret, ezért nem része a rendszer működésének. A VCAI viszont a játék központi AI-ját öleli magába, ami a térkép és a kastély nézetben való döntéshozásról gondoskodik. Csaták esetén átadja az irányítást a BattleAI-nak, ami egyben az utolsó modul is a mappában. Ez a két AI modulárisan lett kiépítve, így a felhasználó egy json fájl mezőjének módosításával könnyen tud a működő AI-ok között váltani. Ez azonban azt is jelenti, hogy a saját csata modulunknál figyelnünk kell, hogy betartsuk a már létrehozott rendszer szabványait.

Az ábrán (*9.3. ábra*) a modul nagyságrendi felépítését láthatjuk, pár fontos metódussal és változóval kiemelve.

A modul nevével osztozó HABattleAI osztály a legfontosabb elemünk, melynek elnevezése a monogramom és a BattleAI szavak kombinációja által történt. Itt inicializálódik a modulunk, és egyben ez a csatlakozási pontunk a térkép AI-al. Az ábrán két metódust is kiemeltünk, az activeStack() a játékkörét tartó egység optimális lépését dönti el, a goTowards() pedig a egységek mozgásának logikáját tartalmazza.

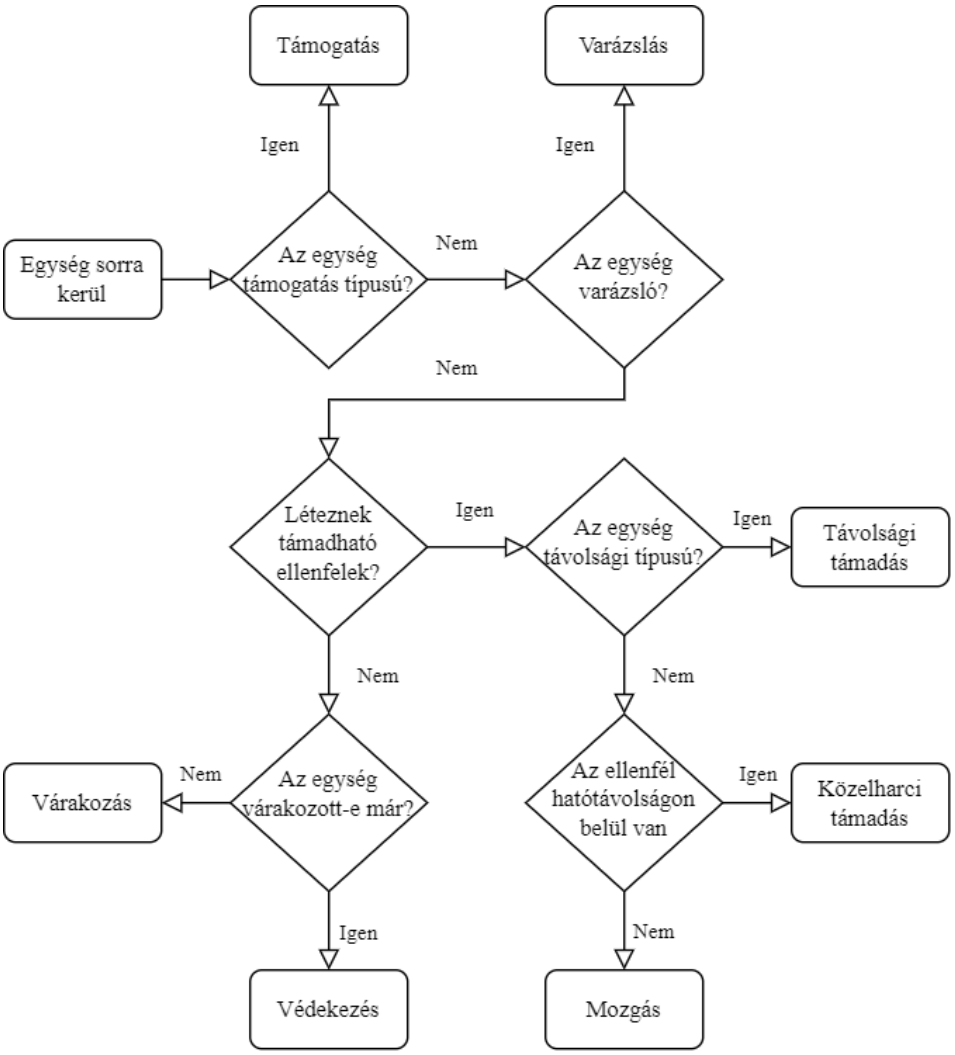
A ThreatMap osztály az AI logikánk legfontosabb részét tartalmazza, a csata jelenlegi helyzetét írja le hatástérképekben. A metódusok a BattleAI standardjai alapján vannak elnevezve, a működésüket a következő fejezetekben részletezem.

A különböző segédosztályok a BattleAI modulból vannak áthozva. Az EnemyInfo és a StackWithBonuses osztályok részletesebb adatokat adnak a csatában részt vevő

egységekről, az AttackPossibility és PotentialTargets osztályok pedig az egységek kezeléséhez biztosítanak pár segédmetódust.

9.3.2. Csata modul megvalósítása

A Csata AI logikájának implementációját egy példán keresztül szeretnénk bemutatni, melyben segítséget nyújthat az ábrán (*9.4. ábra*) lévő egyszerűsített döntésfa.



9.4. ábra - Egy egység döntésfája

Egy csata kezdete után inicializálódik a HABattleAI, majd a megkezdi körét az első egység, vagyis meghívódik rajta az előbb említett activeStack() metódus. Az AI először ellenőrzi az egység típusát, ami alapján specializált utasításokat adhat.

Az egyik ilyen típus a támogató egységek csoportja. Ide tartoznak például a katapultok és a gyógyító sátrak. Ezek az egységek a csatatéren első és utolsó oszlopaiban helyezkednek el, és pozíciójuk statikus. Az egységek logikájához egyszerű algoritmusok tartoznak, a katapultok próbálják egy fal azonos részét támadni, a gyógyító sátrak pedig a legjobban sérült egységeinket gyógyítani.

A következő szakasz a varázslatok kezelését dolgozza fel, mind a hősünk, mind az egységünk szempontjából. A varázslatok optimális kezelése egy Csata AI legnehezebb feladata. A varázslatok sokasága és egyedisége miatt nehéz jó döntést hozni. Ezeket a funkciókat az alapértelmezett BattleAI metódusainak implementálásával kezeljük, ami az ellenfélen ejthető lehető legnagyobb sebzést részesíti előnyben.

Optimalizálás érdekében csak ezek a lépések után fut le a logika magja. Először is létrejön egy másolat a jelenlegi csatamezőről, melyet a lehetséges lépések jóságának megállapítására van felhasználva. Ez a feljebb említett StackWithBonuses segédosztály, HypotheticBattle alosztályának segítségével történik meg. Ezen kívül létrejönnek az egység hatástérképei a ThreatMap osztály segítségével. Végül az AI összegyűjti az ellenfél egységeit a PotentialTargets segédosztályt használva.

Ezen a ponton érdemes tisztázni a példa gyanánt meghatározott kentaur típusú egységünket is, melynek csataterét a vizuális segítség érdekében az alábbi képen (*9.5. ábra*) láthatjuk. A kentaur se támogató, se varázsló típusú, ezért idáig nem kapott semmilyen utasítást.

9.5. ábra - Kentaur csatatere

A ThreatMap osztály példányosításához továbbítódik kentaur egységünk referenciája. Az osztály először átmegy az összes pályán lévő egységen és megjelöli mindazon mezőket, melyeket az egység közelharcban tud támadni. A távolsági típusú egységek minden

ellenfelet tudnak támadni, ezért távolsági támadásnál ez a lépés kimarad. Ezután a megjelölt mezők alapján kitöltődik a kentaurunk ellenfél és szövetséges hatástérképe. Ezek a térképek kétdimenziós tömb és vektor keverékek. A csatatér nagysága adja meg a tömb méretét, a benne lévő vektorok mérete pedig a konkrét csatamezőre ható egységek számától függ. Minden egység egy-egy BattleInfo struktúrával bővíti azokat a vektorokat, melyeknek mezői a feljebb említett módszer alapján ki lettek jelölve. A BattleInfo struktúrák különféle információkat nyújtanak egységeikről, beleértve azok erejét és típusát. A kész hatástérképek adatait felhasználva ezután erőpotenciál (EP) térképeket készítünk. Az EP a hadseregben lévő egységek száma, támadó- és védekezőereje alapján generált érték. Mivel a két térkép húzása ellentétes, ezért a szövetséges térkép negatív értékekkel van kitöltve. A kentaurunk ellenfél és szövetséges EP térképei egy közös térképbe vannak transzformálva. Normalizáció nem szükséges, mivel mindkét térkép azonos típusú adatokat használ fel, súlyozásra viszont van lehetőség. A mi esetünkben a szövetséges térkép súlyai felezve vannak, mivel a gyakorlatban így nagyobb sikereket értünk el. A kentaurunk kombinált EP térképét az alábbi képen láthatjuk (*9.6. ábra*).

	-20	-28	-28	-28	-28	-20	-20	-20	-20	21	21	21	21	21	21	
	-20	-28	-34	-34	-28	-28	-20	-20	-20	1	21	21	21	21	21	
	-34	-34	-34	-34	-28	-28	-20	-20	1	1	21	40	40	40	40	
	-34	-34	-34	-34	-34	-28	-28	-20	-20	1	21	40	40	40	40	
	-34	-34	-34	-34	-34	-28	-28	-20	-20	21	40	40	40	40	40	
	-14	-34	-34	-34	-34	-34	-28	-28	-20	0	40	40	40	40	40	
	-34	-34	-34	-34	-34	-28	-28	-28	0	19	40	40	40	40	40	
	-34	-34	-34	-34	-34	-28	-28	-28	0	19	19	40	40	40	40	
	-34	-34	-34	-34	-8	-8	-8	-8	19	19	19	40	40	40	40	
	-14	-34	-34	-34	-28	-8	-8	-8	-8	19	19	19	19	19	19	
	-14	-34	-34	-28	-8	-8	-8	-8	0	19	19	19	19	19	19	

9.6. ábra - Egységünk kombinált erőpotenciál térképe

Kék színnel és negatív értékekkel ábrázolva a szövetséges EP térképünk hatáskörét, piros színnel és pozitív értékekkel pedig az ellenfél EP térkép hatáskörét láthatjuk. Fekete színnel a pálya elérhetetlen mezőit jelezzük.

A térképek létrehozása után tovább fut a fő metódus, ahol össze van szedve az egységünk által támadható ellenfelek listája. Egy egység nem elérhető, ha körül van véve más egységekkel, vagy a csatatéren lévő akadályokkal. Amennyiben a lista üres, az AI

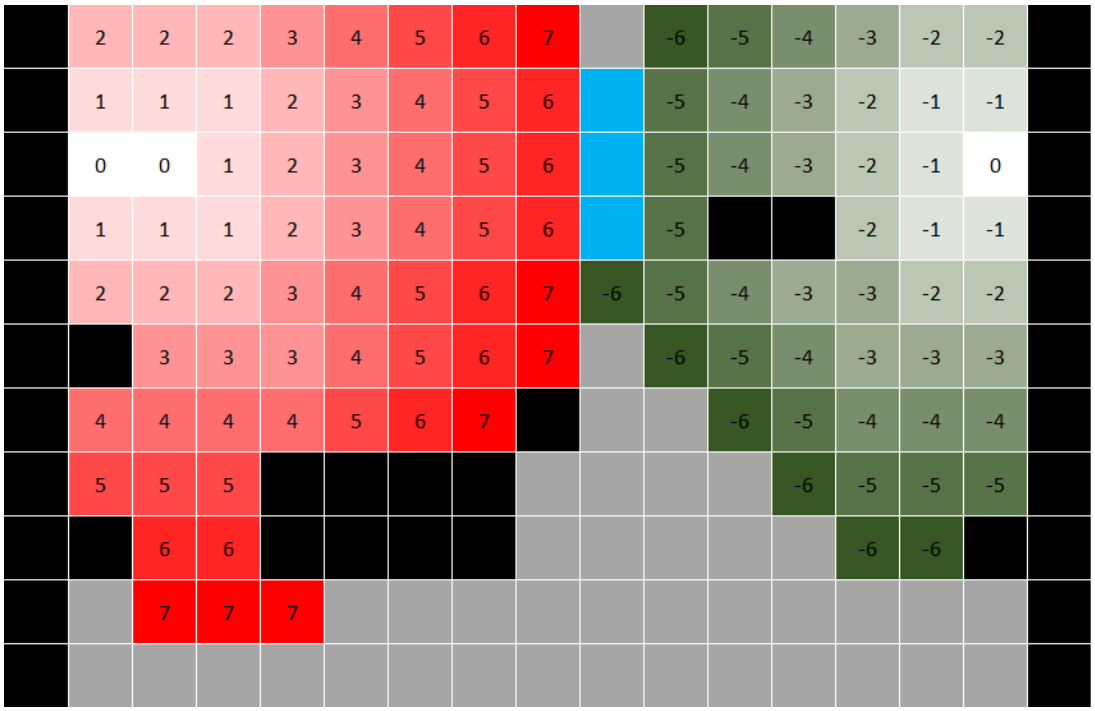
ellenőrzi, hogy egységünk várakozott-e már a körben. A várakozás késlelteti egy egység lépését a kör végéig. Ha egy egység már várakozott, akkor újra ellenőrizve van az elérhető ellenfelek listája. Sikeres keresés esetén egy mozgás akció van indítva az ellenfél mezője felé. Ellenben pedig először a várakozás akció van kiadva, második alkalommal pedig egy védekezés utasítás, ami az egység körének kihagyását jelenti.

A mi esetünkben viszont az ábra alapján (*9.8. ábra)* látható, hogy a kentaur mindkét ellenfele elérhető, így az AI támadást próbál indítani. Először a távolsági egységek lépése van feldolgozva, mely könnyeb feladat. A cél a legtöbbet sebezhető ellenfél támadása. Viszont, ha egy egység közelharci típusú – ami a mi példánknál igaz -, bonyolultabb számítások szükségesek. Kezdésként, ha nincs az ellenfél a hatótávolságunkban, akkor egy mozgás akció indul, amit a goTowards() metódus kezel. Ellenkező esetben pedig egy közelharci támadás akció indul, melyet sok különböző tényező által kell megvalósítani az AI-nak:

- melyik hatótávolságon belüli ellenfél legyen megtámadva?
- melyik mezőről legyen az ellenfél támadva?
- mennyit sebzés lesz okozva sikeres támadás esetén?
- mennyit sebzés lesz elszenvedve visszaütés esetén?
- van-e lehetőség több ellenfél sebzésére?
- van-e kockázat baráti tűzre?

Ezen tényezők mérlegelése után végül kiválasztódik a támadni kívánt ellenfél, és a hatástérképek segítségével, a támadásra használt csatamező.

A kentaur hatótávolságában azonban nincs ellenfél, így egy mozgás akció van meghívva. A célmező megtalálására egy A* nevű útkereső algoritmus van implementálva, melyet az irodalomkutatás során bővebben kifejtettünk (*6.1 fejezet Útvonalkeresés bemutatása*). A keresés kezdőpontja a kiválasztott ellenfél mezője. A kereső algoritmus addig fut, míg átfedés nem keletkezik az algoritmus által bejárt terület és az egységünk hatótávolsága között (*9.7. ábra*). Ilyenkor a metszetben lévő mezők közül, a hatástérképe által legjobbnak ítélt mezőre lép egységünk.

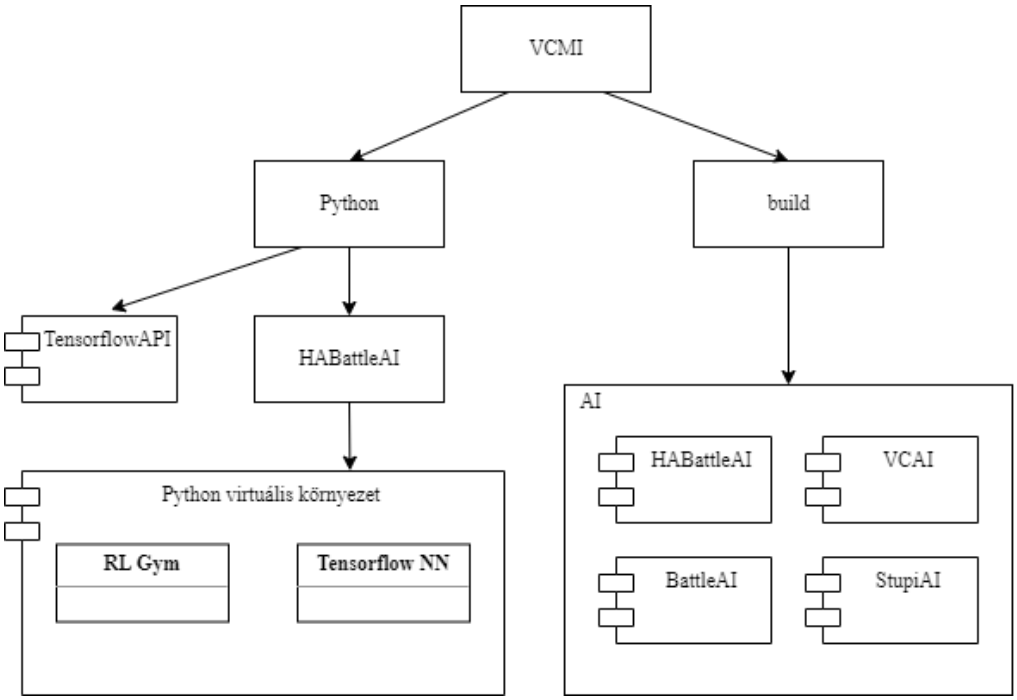


9.7. ábra - Optimális mozgás meghatározása a kentaur egység példáján

Fekete színnel a pálya elérhetetlen mezőit jelöljük. Piros színnel és pozitív értékekkel ábrázolva az egység hatótávolságát jelezzük, tehát azokat a mezőket, melyekre egységünk léphet. Zöld színnel és negatív értékekkel pedig az ellenfelünk mezőjéről kiinduló A* útvonalkereső algoritmus által lefedett területet láthatjuk. A kék mezők pedig az egységünk hatótávolságának és az útvonalkereső algoritmus utolsó hullámának metszetét jelzi, vagyis egységünk lépésének lehetséges mezőit.

9.3.3. További Csata AI változatok

A Csata AI-unkat végül az Algoritmikus megoldással oldottuk meg, hatástérképek és útvonalkeresés felhasználásával. A megerősítéses és felügyeletes tanítású próbálkozásaink sajnos sikertelenül végződtek. Az ábrán (*9.8. ábra*) látható módon nézett volna ki tervezett rendszerünk.



9.8. ábra - A tervezett rendszer felépítése

A felügyeletes tanítást az ábrán (*9.8. ábra*) látható Tensorflow NN (Tensorflow neurális háló) modullal végeztük volna el. A folyamatban sokáig el is jutottunk, letöltöttük a Tensorflow futásához szükséges API-t, inicializáltuk a Python virtuális környezet, majd megépítettük neurális hálónkat is. A hálónk két bemeneti és egy kimeneti ágból álló rekurrens neurális háló (*6.4. fejezet Rekurrens neurális hálók*) volt. Az egyik bemeneti ágon a csatáról fontos általános adatokat töltöttük be, mint például a csatában lévő felek összegzett EP értékét. A háló másik oldalára pedig a csatatér aktuális állapotát tápláltuk be számszerűsítve. Mivel a rekurrens hálók alapból csak rövid távú memóriával rendelkeznek, ezért Hosszú rövid távú memória (Long short-term memory - LTSM) egységeket is hozzáadtunk hálónkhoz. Az implementációt végül két okból szakítottuk meg. Egyrészt rájöttünk, hogy a háló betanításához való nagy mennyiségű adatot nem tudtuk volna összegyűjteni. Másrészt, a folyamat során megtapasztaltuk a neurális hálók korlátait. Éreztük, hogy még elegendő adat tulajdonában se generált volna jó megoldásokat hálónk.

A megerősítéses tanítást használó rendszerünknél is problémákba ütköztünk. A modult szintén a Python virtuális környezeten belül hoztuk létre. A megerősítés tanulás (Reinforcement Learning - RL) területén sok olyan könyvtár létezik, mely segít az RL környezet kialakításában. Ezeket a környezeteket edző- vagy tornatermeknek, angolul gym-eknek nevezzük. Innen ered modulunk RL Gym elnevezése. Mi az RLlib nevű könyvtárat akartuk volna felhasználni, azonban több problémába is ütköztünk. A környezet kiépítéséhez újra kellett volna írni a csata környezetünket a Python nyelvre, mely komplikált feladat lett volna. Ezen kívül kétségeink voltak a VCMI és RL környezetünk közötti adatcsatorna megvalósításáról, illetve annak negatív hatásáról futásidőnkre.

9.4. További AI modulok

A Térkép és Kastély AI moduljainkat végül nem készítettük el. A szakdolgozatunk implementációja során nyilvánvalóvá vált, hogy muszáj összpontosítanunk munkánkat a rendszertervünk bizonyos részeire. Végül a Csata AI-t és a Visszajátszás rendszert választottuk. A Visszajátszás rendszer egy teljesen új funkció volt a VCMI környezetében, illetve a szükségünk volt annak begyűjtött adataira is a Csata AI modul részére. A Csata AI modulban pedig nagyobb lehetőséget láttunk, mint a Térkép AI-ban, mivel a megvalósítását több szögből is meg tudtuk közelíteni. Ezen kívül a Térkép AI-hoz kutatott irodalmunkat az útkeresésről, végül fel tudtuk használni Csata AI modulunknál.

A Kastély AI modulunk pedig szorosan kötődött volna a Térkép AI modulunkhoz, melynek hiányában nem volt ésszerű elkészítenünk.

10. Tesztelés

Az alábbiakban az implementáció során a kiépített rendszerünk minőségét és versenyképességét teszteljük.

10.1. Csata AI

A Csata AI-unk tesztelése érdekében egy megfelelő környezetet kell kialakítanunk. Egy átlagos HOMM 3 meccsben majdnem lehetetlen olyan szituációba kerülni, ahol a csatában szereplő egyik fél ugyanazokkal az eszközökkel rendelkezik, mint ellenfele. Ezeknek a tényezőknek a minimalizálása miatt, egy rövid meccsekre kialakított, aréna típusú pályát választottunk. A pályán rendelkezésünkre állnak előre összerakott és kiegyensúlyozott hadseregek, melyek a játékban lévő minden frakció részére ki lettek alakítva. Hőseink nagy hatással vannak egy csata kimenetére, ezért ügyeltünk, hogy mindkét fél azonos varázslatokkal, készségekkel és szinttel rendelkezzen. Minden hősnek van egy egyedi specializációja, mely lehet egy bizonyos egység vagy varázslat megerősítése. Figyeltünk, hogy ezek ne legyenek hatással a tesztelni kívánt csatánkra. Persze minden frakció más-más fajta egységekből áll össze, melyek különböző erősségekkel és gyengeségekkel rendelkeznek. Az autentikus tesztelés érdekében ezt figyelembe kell vennünk.

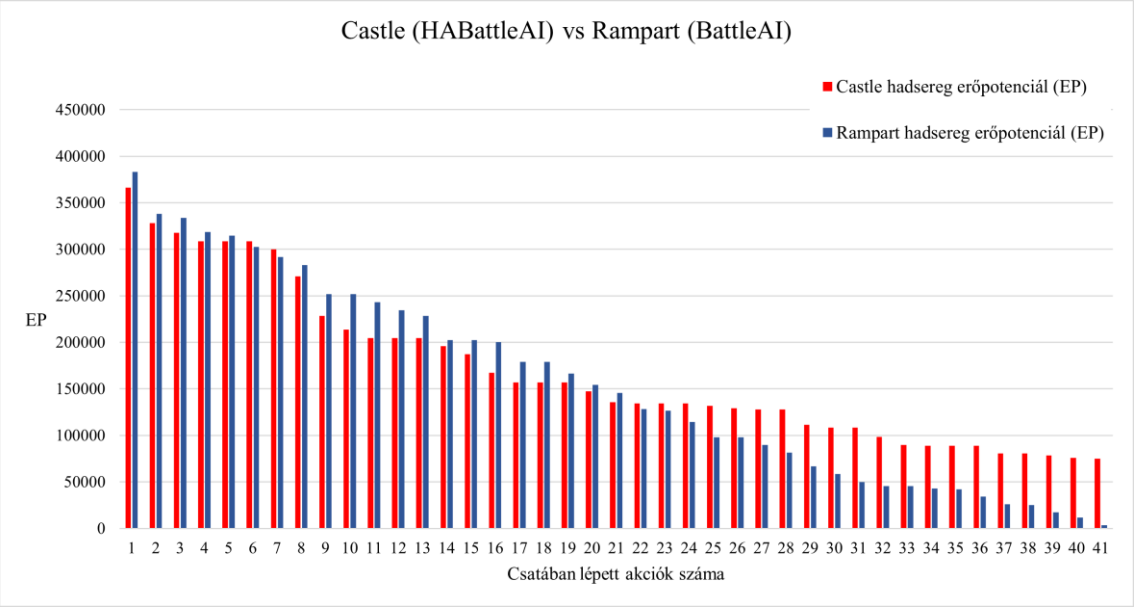
Tesztjeinket két különböző fajta frakció között készítjük. A csatát 3 fajta módon is lejátsszuk. Először a saját HABattleAI-unk játszik az ellenfél BattleAI-ával szemben, a csatatér mindkét feléről nézve. Majd lejátszatjuk két BattleAI-al is a meccset, melynek eredménye alapján értékelhetjük saját AI-unk teljesítményét. A csatákat nem szükséges többször lefuttatnunk, mivel a játék véletlenszám generátora a környezethez van kötve. Erre egy egyszerű példa az egységek sebzése a harcban. Elméletileg mindig változó értékeket kellene sebezniük, mivel támadóerejük egy számskála véletlenszerű értéke, nem egy fix szám. A gyakorlatban viszont minden csatában azonos sebzések generálódnak.

Első tesztünkben a Castle és Rampart frakciókat használjuk. A csatában lévő két hadsereg felépítését az alábbi ábrákon láthatjuk (*10.1 – 10.2 ábra*). A hadseregek erejét az erőpontenciál (EP) érték segítségével jelezzük. Az EP a hadseregben lévő egységek száma, támadó- és védekezőereje alapján generált érték. Castle hadsereg erőssége 366137 EP nagyságú, míg a Rampart frakcióé 382985 EP értékű.

10.1. ábra - Castle hadsereg

10.2. ábra - Rampart hadsereg

Az első meccset a Castle frakció (piros oldal) nyeri, a HABattleAI irányításával. Az ábrán (*10.3. ábra*) a csata menetét láthatjuk. Az X tengely a csatában megtett akciók számát jelzi, az Y tengely pedig a csatában lévő hadseregek EP értékét. Látható, hogy a csata első felében a kék oldal fenntartja a tempót, sőt nyerő helyzetben is áll, azonban a csata végére szétesik a hadserege.



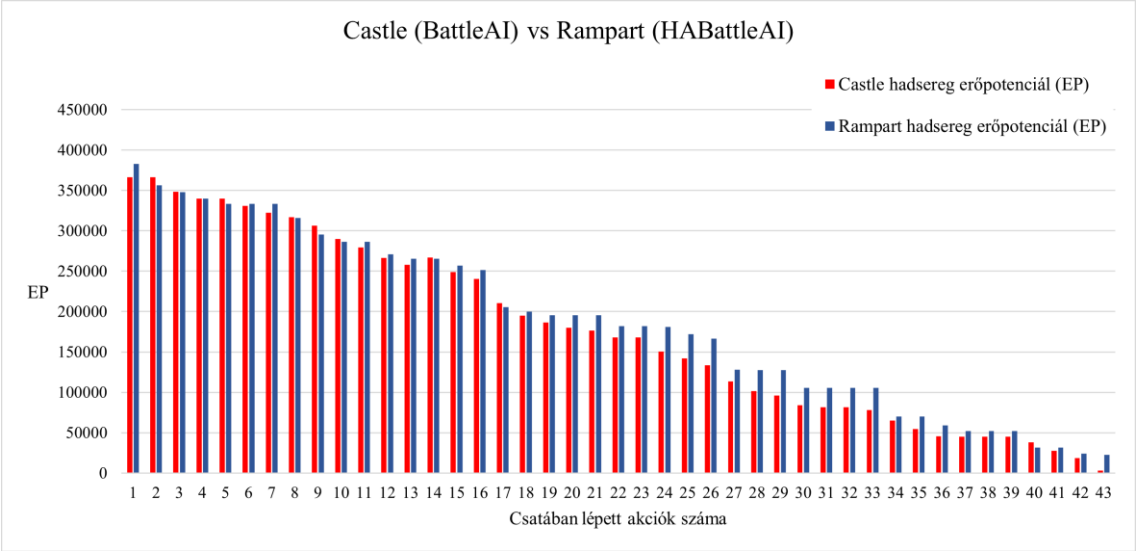
10.3. ábra - Első csata menete

A piros oldal végül megnyeri a csatát, melyet egy 75000 EP erősségű sereg él túl (*10.4 ábra*).

10.4. ábra - Első csata maradék hadsereg

Vizsgáljuk meg azt az esetet, amikor mindkét oldalt a BattleAI irányítja. Ezt a meccset is a Castle frakció nyeri, bár az összecsapást egy csupán 39750 EP értékű hadsereg éli túl. Ez az eredmény azt jelzi, hogy a BattleAI Castle oldalt tartja erősebbnek.

Végül lefuttatjuk a 3. fajta esetet is, ahol HABattleAI irányítja a Rampart frakciót (kék) és a BattleAI a Castle oldalát, melynek menetét az ábrán (*10.5. ábra*) láthatjuk. A gráfról leolvashatjuk, hogy egy szoros összecsapás történik. Végül a saját AI modulunk kerül ki győztesként, 22484 EP különbséggel. Az eredmények alapján elmondhatjuk, hogy a HABattleAI ebben a szettben felülmúlja ellenfelét, mivel a csata mindkét felét nyeri.



10.5. ábra - Harmadik csata menete

Nézzük meg, hogyan teljesítenek az AI modulok egy más környezetben. Felállítunk egy új környezetet, melyekre ugyanúgy érvényesek a fejezet elején felvázolt egyenlő esélyeket biztosító szabályok. Az egyetlen különbség az, hogy a csatában két új frakció vesz részt, Necropolis és Fortress elnevezésekkel. Célunk annak meghatározása, hogy saját AI modulunk általánosan jobban teljesít ellenfeléhez képest, vagy csak az előző tesztben résztvevő frakciókkal játszik jobban. A csaták sorrendje azonos az előző tesztünkkel. Először a HABattleAI-unk játszik a piros oldalon (Necropolis) egy BattleAI-la szemben, majd a két BattleAI méreti össze erejét, végül a HABattleAI a kék oldalról (Fortress) próbálja legyőzni BattleAI ellenfelét.

A csaták kimenetelét a táblázatban (*10.1 táblázat*) találhatjuk meg. Az eredményekből megállapíthatjuk, hogy a BattleAI ellenfeléhez képest okosabban irányítja ennek a szettnek a frakcióit.

Frakció	1. játszma végső hadsereg EP	2. játszma végső hadsereg EP	3. játszma végső hadsereg EP
Necropolis (piros)	0	0	46197
Fortress (kék)	11862	97426	0

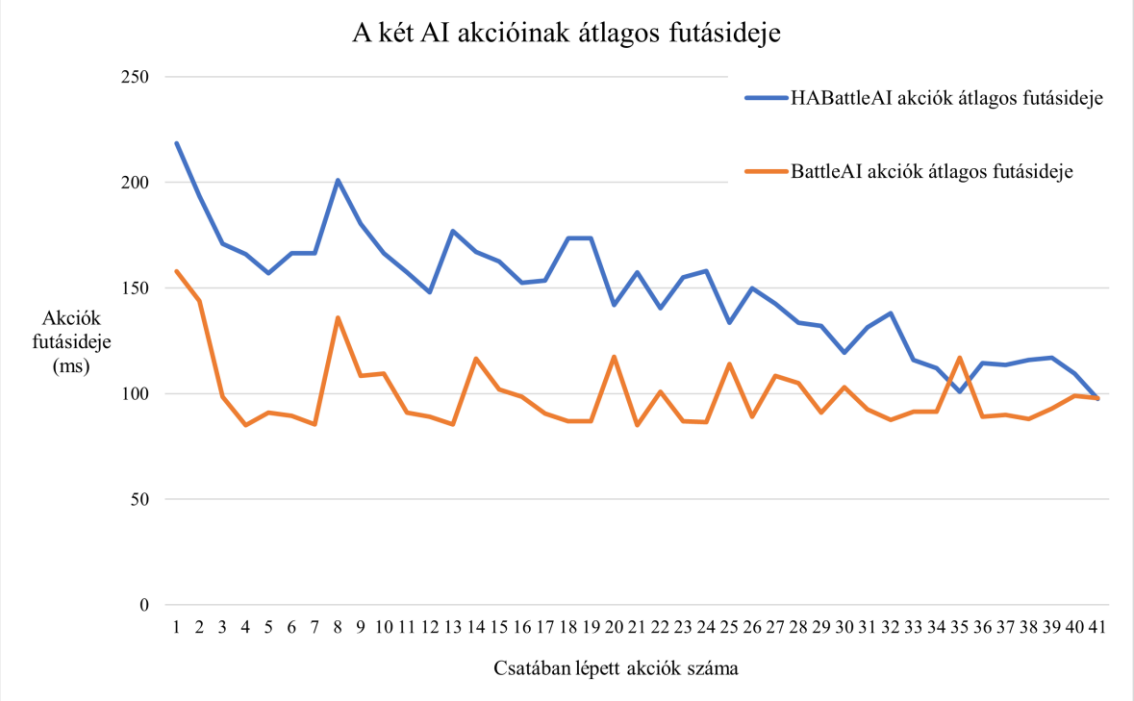
10.1. táblázat - A második szett eredményei

A tesztek eredményeiből egy egyszerű következtetést vonhatunk le. Egyik AI modul se múlja felül a másikat, csupán különböző frakciókkal játszanak effektívebben. Pontosítva az AI stratégiája jobban illik bizonyos frakciók játékstílusához, azok egységeit jobban irányítja.

Felmerülhet a kérdés, hogy miért teszteltük két BattleAI között is a csatákat? A célunk egy különböző AI modul készítése volt, így érdekeltek mennyit változik egy csata kimenetele a HABattleAI jelenlététől függően. Az első szettnél összhangot figyelhetünk meg, mindkét AI modul a Castle frakciót találja erősebbnek, ami a csaták eredményeiben is kimutatható. A második szettnél viszont ennek pont az ellentéte jelenik meg. A BattleAI saját maga ellen jobban teljesít a Fortress frakcióval, viszont a HABattleAI ellen a Necropolis frakciókkal jobb eredményeket ér el. Ezek az eredmények is alátámasztják előző következtetésünk hitelességét.

A csaták eredményei mellett az egyes AI modulok futásidejét is megfigyeljük. Az ábrán (*10.6. ábra*) az AI modulok átlagos futásidejét láthatjuk akciónként. Látható, hogy a kezdeti lépéseknél nagyobb különbségek vannak a futásidők között, a csata végére viszont azonos szintre konvergálnak. A HABattleAI modulnál az extra időt a hatástérképek generálása, illetve a mozgásnál használ útkereső algoritmusok generálása teszik ki. A futásidők soha nem haladják meg a 220-230 ms időtartamot. Mivel a tesztek során nagy hadseregeket használtunk, ezt a futásidőt az átlagos csaták során nem érjük el.

Maga az időkülönbség emberi szemmel sem látható, de ez egy lehetséges optimalizálási pont a programunkban.



10.6. ábra - Az AI modulok futásidejének összehasonlítása

10.2. Visszajátszás rendszer

A Visszajátszási rendszerünk funkcionalitását már a Csata AI implementációja során teszteltük. Ennek segítségével automatizáltuk a csaták teszteléséhez való lépések futtatását. A visszajátszási adatok szerializációja tökéletesen működik, mind az adatok tárolása és beolvasása terén. A visszajátszás folyamata viszont tartalmaz hibákat. Az adatok beolvasása és akciókká alakítása hiba nélkül megtörténik. Viszont bizonyos akciók vizuális elemei a játékképernyőn maradnak az akció lefutása után is. Ezek hibát nem okoznak, viszont eltávolításukat manuálisan kell elvégeznünk. Ezen kívül egy minimális számú akció lefuttatásánál hibákba ütközünk.

11. Eredmények értékelése

Szakdolgozatunkban végül két jelentős modulra fókuszáltunk, a Visszajátszás rendszerünkre és HABattleAI nevű mesterséges intelligenciánkra.

A Visszajátszás rendszerünk egy újdonság a VCMI környezetében, így hasonló belső rendszerrel nem tudjuk összevetni. A játékok világában viszont egy elterjedt koncepció. Célja a játékmenet visszajátszása, melyet beadott adatok és utasítások alapján tesz meg. Modulunk nem csak a visszajátszást oldja meg, de az ahhoz szükséges adatok elmentését és feldolgozását is. Az adatmentés és adatbetöltés funkciók tökéletesen működnek. Az adatokat json formátumban tároljuk, melyek egyedi azonosítókkal vannak elmentve. Az adatbetöltés funkció pedig először hitelesíti a betöltött adatokat, majd továbbítja azokat visszajátszásra. A játék akcióinak visszajátszása azonban nem zökkenésmentes. Bizonyos akciók felesleges vizuális elemeket hagynak a képernyőn, illetve pár akció meghívásánál hibákba ütközünk. Ezek okára sajnos nem tudtunk rájönni a VCMI projekt hatalmas terjedelme és komplexitása miatt. Ennek ellenére a Visszajátszás rendszer funkcionális, a Csata AI rendszerünk implementálása során nagy segítséget nyújtott különböző játékon belüli szituáció tesztelésében.

A Csata AI modulunkat a már meglévő VCMI mesterséges intelligenciák mellé illesztettük be. Az AI környezet modulárisan van felépítve, így figyelnünk kellett modulunk szabályos felépítésére. Ugyanakkor ezt azt is jelentette, hogy saját Térkép AI implementálása híján, felhasználhattuk a beépített modult is. Feladatunkat több szögből is megközelítettük. Próbálkoztunk többfajta neurális háló implementálásával, illetve a megerősítéses tanítást megvalósító, gym környezet összerakásával is. Ezek a kísérletek sajnos eredménytelenül végződtek. Az AI-t végül hatástérképek és útvonalkereső algoritmusok segítségével implementáltuk. Célunk egy egyedi AI felépítése volt, saját észrevételeink és tapasztalataink alapján. Véleményem szerint sikeresek voltunk. A tesztelés alapján látható, hogy saját HABattleAI modulunk versenyképes a beépített BattleAI modullal. A tesztek alatt tapasztalhattuk, hogy a csatában részt vevő hadseregektől függően felül- és alul is múlta ellenfelét. A két AI pontos eltéréseit nehéz megállapítani, mivel számtalan tényezőtől függ egy csata kimenetele. A modulok kódjának ismerete, illetve saját megfigyelésünk alapján viszont jó pár következtetést le tudunk vonni. Saját AI-unk a hatástérképei alapján mozog, és jobban odafigyel milyen mezőkről támadjon. Továbbá az egységei jobban csoportosulnak. Ezek a tényezők azt eredményezik, hogy defenzívebben játszik. Ezen kívül összpontosítsa támadásait azokra az ellenfél egységekre, melyek túl messze helyezkednek csapattársaiktól. Így mondhatjuk, hogy AI-unk megbünteti túlzott agressziót. Ugyanakkor ezek a tulajdonságok hátrányosak is lehetnek. A beépített BattleAI modul nyereségeinél azt láthattuk, hogy bizonyos egységeinek feláldozásával nagy előnyöket szerzett. Ezen kívül a lövészei valamennyivel nagyobb biztonságot élvezhettek. A két AI közötti viselkedési különbségek végül érdekes meccseket eredményeztek, mely szintén célunk volt.

12. Továbbfejlesztési lehetőségek

A szakdolgozatunk készítése során jó pár lehetséges optimalizálási pontot, és továbbfejlesztéséi lehetőséget találtunk. Sajnos időkorlátok miatt sok ötletünket nem tudtunk megvalósítani, ezeket részletezzük a továbbiakban.

Maga a rendszer továbbfejlesztésére számtalan lehetőségünk van. A VCMI egy nyílt forráskódú projekt, így a program bármely része módosítható, bővíthető. Itt nem csak az AI területére gondolunk. Készíthetünk új frakciókat, más-más fajta egységekkel. Felfordíthatjuk a játék szabályzatát, belenyúlhatunk a játék pályáinak generálásába, és még számtalan más játékon belüli területbe.

Saját munkánk kibővítésére is sok opciónk van. Energiánkat a Csata AI és a Visszajátszás rendszerünk implementációjára fordítottuk, ezért megvalósíthatjuk a végül nem elkészített moduljainkat. Itt a Térkép és Kastély mesterséges intelligenciákat értjük, melyek segítségével egy teljesen független AI modult alakíthatunk ki.

A VCMI működésének teljes átlátása fejében javíthatjuk a Visszajátszás rendszerünk hiányosságait. Illetve, ha adataink mennyisége túl nagyra nőne, kialakíthatunk egy SQL adatbázist is.

A Csata AI megvalósítása alatt is akadtak ötleteink. Implementálhatjuk azt más eszközök segítségévével is, például genetikus algoritmusokkal. A tesztelésnél láthattuk, hogy nagyobb seregek esetében AI-unk futásideje magasabb volt ellenfelénél. Ezt orvosolhatjuk például a hatástérképeink okosabb frissítésével. Egy másik érdekes kibővítési pont a varázslatok használata. Létrehozhatók különböző varázslási stratégiák is, melyeket egy magas szinten játszó játékos segítségével lehetne összerakni.

13. Összefoglalás

Szakdolgozatunk célja egy egyedi mesterséges intelligencia fejlesztése volt a HOMM3 nevű játékhoz, melyet annak nyílt forráskódú újraírásában, a VCMI nevű projektben valósítottunk meg. Első lépésként felvázoltuk a játék működését, elemeit és a saját fejlesztésünk céljait. Majd elvégeztük a projektünk megvalósításához szükséges irodalomkutatást. Szót ejtettünk a mesterséges intelligencia fejlődéséről, a szakdolgozatunkhoz hasonló rendszerekről és különböző megvalósítási módszerekről.

Ezután feltérképeztük a VCMI rendszer környezetét, és megterveztük rendszerünket. A kezdetekben egy teljes játékmenetet átfedő AI-t szerettünk volna implementálni, később a csaták automatizálására összpontosítottuk figyelmünket. A Csata AI mellett egy visszajátszás rendszert is megvalósítottunk. Ennek célja a rendszer egyedi funkcióval való bővítése, valamint a Csata AI megvalósításához szükséges adatok összegyűjtése volt. A rendszer először összegyűjtötte a játékmenet adatait, melyeket egyedi azonosítókkal tárolt el. Egy játék betöltésénél pedig lehetőséget adott a játékmenet betöltésére és visszajátszására.

A Csata AI megvalósítását több irányból is megközelítettük. Próbálkoztunk felügyeletes tanítással, melyet egy Tensorflow neurális háló, és a Visszajátszás rendszerünk adatainak segítségével tettük. Majd megkíséreltük egy megerősítés tanítást lehetővé tévő környezet kiépítését is. Sikeres megoldáshoz azonban algoritmikus úton jutottunk, útvonalkereső algoritmusok és hatástérképek segítségével. Tesztjeink során tapasztalhattuk, hogy saját rendszerünk versenyképes a konkurenciájával. Egyedi stratégiája révén bizonyos helyzetekben túl-, másokban alulteljesítette ellenfelét.

14. Abstract

The aim of our thesis was to develop a custom artificial intelligence for the game HOMM3, which we implemented using an open-source rewrite of the game, a project called VCMI. Firstly, we outlined the game's functionality, its elements, and our own development goals. We then carried out a literature search for the implementation of our project. We discussed the development of artificial intelligence, systems like our thesis, and different implementation methods. After that, we mapped the environment of the VCMI system and designed our own system. In the beginning, we wanted to implement an AI that covered the whole gameplay. Later we focused our attention on the automation of battles. Besides a Battle AI, we also implemented a replay system. The goal was to add a unique feature to the existing system and to collect the data needed to implement our Battle AI. The system first collected gameplay data, which was stored with unique identifiers. When a game was loaded, it provided us with the possibility to load and replay a game using our stored data.

The implementation of our Battle AI was approached from several directions. We tried supervised learning, using a Tensorflow neural network and data from our replay system. We also attempted to build a reinforcement learning environment. However, the successful solution was obtained algorithmically, using pathfinding algorithms and influence maps. Our tests proved that our own system is competitive with its counterpart. Its unique strategy allowed it to outperform its opponent in some situations, but underperform them in others.

Irodalomjegyzék

[1] VCMI, „About VCMI,” [Online]. Elérhető: https://vcmi.eu/about/. [Hozzáférés dátuma: 2020. 3. 31.].

[2] VCMI, „VCMI Wikipedia - Big Picture,” [Online]. Elérhető: https://wiki.vcmi.eu/images/0/0b/Architektura.png. [Hozzáférés dátuma: 2020. 04. 16.].

[3] B. Schwab, „AI Game Engine Programming,” Charles River Media, 2004, pp. 97-112..

[4] IBM, „DeepBlue,” IBM, [Online]. Elérhető: https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/. [Hozzáférés dátuma: 2020. 04. 15.].

[5] C. Butcher, Interviewee, *The Artificial Intelligence of Halo 2.* [Interjú]. 2004. 11. 17..

[6] C. Butcher és J. Griesemer, „The Illusion of Intelligence,” Bungie Inc., 1999-2012, Microsoft 2012-2020,[Online]. Elérhető: http://halo.bungie.org/misc/gdc.2002.haloai/talk.html. [Hozzáférés dátuma: 2020. 04. 15.].

[7] J. Orkin, „Three States and a Plan: The A.I. of F.E.A.R.,” in *Game Developer's Conference (GDC)*, 2006, 2006.

[8] H. Martin, Interviewee, *Make me think, make me move': New Doom's deceptively simple design.* [Interjú]. 2020. 04. 15..

[9] C. Hocking, „Designing to Promote Intentional Play,” in *Game Developers Conference*, San Jose, California, USA, 2006.

[10] B. Scott, „The Illusion of Intelligence,” in *AI Game Programming Wisdom*, R. Steve, Szerk., Charles River Media, 2002, pp. 16-20..

[11] S. Meier, „The 7th International Computer Game Developers Conference,” *Computer Gaming World*, %1. kötet 108, p. 34., 1993.

[12] B. Russel, „The Secrets Of Enemy AI In Uncharted 2,” 2010. 11. 03.. [Online]. Elérhető: https://www.gamasutra.com/view/feature/134566/the_secrets_of_enemy_ai_in_.php. [Hozzáférés dátuma: 2020. 04. 16.].

[13] K. D., „Advances in Man-Machine Play,” in *Computers, Chess, and Cognition*, (1990, pp. 9-32.

[14] S. J. Donskoy M.V., „Perspectives on Falling from Grace,” in *Computers, Chess, and Cognition*, 1990, pp. 259-268.

[15] A. H. I. S. D. S. C. J. Maddison, „International Conference on Learning Representations,” 2015.

[16] e. a. D. Silver, „Mastering the game of Go with deep neural networks and tree search,” *Nature*, %1. kötet 529., p. 484–489, 2016.

[17] e. a. David Silver, „A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,” *Science*, pp. 1140-1144, 2018.

[18] „https://civilization.com/,” Firaxis Games, 21. 10. 2016.. [Online]. [Hozzáférés dátuma: 2020.05. 07.].

[19] G. S. David M. Bourg, AI for Game Developers, O'Reilly, 2004.

[20] H. Kent, „Influence mapping system,” [Online]. Elérhető: https://harrykent.games/game-ai/influence-mapping/. [Hozzáférés dátuma: 2022. 12. 2.].

[21] D. Mark, „Modular Tactical Influence Maps,” in *Game AI Pro 2*, CRC Press, 2015., pp. 343-364..

[22] A. J. Champandard, „The Core Mechanics of Influence Mapping,” 8. 6. 2011. [Online]. Elérhető: https://www.gamedev.net/tutorials/programming/artificial-intelligence/the-core-mechanics-of-influence-mapping-r2799/. [Hozzáférés dátuma: 2022. 12. 2.].

[23] S. A. Afshine Amidi, „https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks,” Stanford University. [Online]. [Hozzáférés dátuma: 2022. 11. 23.].

[24] L. L. M. M. A. Kaelbling, „Reinforcement Learning: A Survey,” *Journal of Artificial Intelligence Research (JAIR)*, %1. kötet4., pp. 237-285., 1996.

[25] R. Wong, „Reinforcement Learning: An Introduction to the Concepts, Applications and Code,” Towards Data Science, 23. 9. 2018.. [Online]. Elérhető: https://towardsdatascience.com/reinforcement-learning-an-introduction-to-the-concepts-applications-and-code-ced6fbfd882d. [Hozzáférés dátuma: 2022. 12. 2.].

[26] C. Mahoney, „Reinforcement Learning - A Review of the Historic, Modern, and Future Applications of this Special Form of Machine Learning,”

Towards Data Science, 2021. 6. 12.. [Online]. Elérhető: https://towardsdatascience.com/reinforcement-learning-fda8ff535bb6. [Hozzáférés dátuma: 2022. 12. 2.].

Ábrajegyzék