Matematikai és Informatikai Intézet

Mesterséges intelligencia számítógépes játékokban

Készítette
Budai Roland
Programtervező informatikus BSc.

Témavezető
Dr. Kovásznai Gergely
Egyetemi docens

Eger, 2024

Tartalomjegyzék

Bevezetés

A szakdolgozatom témája egy stratégiai társasjáték implementálása beépített Mesterséges intelligenciával, amely segít a támadási döntések meghozatalában. A projekt ötlete nem csupán egy tanulmányi kötelezettség teljesítéséből fakad, hanem egy régebb óta tervezett hobbi projekt része is. Már a tanulmányaim elején megfogalmazódott bennem a kérdés, hogy hogyan tudnék egy hasonló társasjátékot a számítógépes világban megvalósítani. Milyen adatbázist kell elkészíteni, milyen döntéseket hozhatnak a játékosok, hogyan lehet megoldani, hogy egyszerre többen is hozzáférjenek az adatokhoz, mégis konzisztens maradjon a játékmenet és a tárolt állapotai a játéknak? Hasonló kérdések fogalmazódtak meg bennem tanulmányaim során, melyekre a képzés ideje alatt egyre jobb és jobb ötleteket sikerült szereznem. Mindezek miatt nagy lelkesedéssel álltam neki a feladatnak a témaválasztást követően.

További érdekessége a szakdolgozatnak a mesterséges intelligencia (MI) fejlesztése, használata. A mesterséges intelligencia és a gépi tanulás területei egyre nagyobb szerepet játszanak a modern játékfejlesztésben, ezért ez a projekt nemcsak szakmai kihívást jelentett számomra, hanem egyben kiváló lehetőséget is arra, hogy mélyebben megismerkedjek az MI alkalmazásával, fejlesztésével a játékok világában. Az MI implementálásával lehetőségem nyílt egy olyan ellenfél létrehozására, amely képes tanulni és folyamatosan fejlődni a játékok során.

A mesterséges intelligencia fejlődése az elmúlt években jelentősen befolyásolta a játékfejlesztés folyamatát, lehetővé téve, hogy az MI-alapú ellenfelek egyre intelligensebb döntéseket hozzanak és alkalmazkodjanak a játékosok stratégiájához. Stratégiai döntések meghozatalához különösen a projektemben is használt megerősítéses tanulás (reinforcement learning, RL) vált népszerűvé, amely az MI folyamatos fejlődését, tanulását teszi lehetővé. [1]

1. fejezet

Fejezet címe

1.1. Szakasz címe

1.1.1. Alszakasz címe

Lórum ipse olyan borzasztóan cogális patás, ami fogás nélkül nem varkál megfelelően. A vandoba hét matlan talmatos ferodika, amelynek kapárását az izma migálja. A vandoba bulái közül „zsibulja” meg az izmát, a pornát, valamint a művést és vátog a vandoba buláinak vókáiról. Vókája a raktil prozása két emen között. Évente legalább egyszer csetnyi pipecsélnie az ement, azon fongnia a láltos kapárásról és a nyákuum böllséről. [1, 102. oldal]

A vandoba ninti és az emen elé redőzi a szamlan radalmakan érvést. Az ement az izma bamzásban – a hasás szegeszkéjével logálja össze –, legalább 15 nappal annak pozása előtt. Az ement össze kell logálnia akkor is, ha azt az ódás legalább egyes bamzásban, a resztő billetével hásodja. [1, 2]

2. fejezet

Játékmechanika

2.1. A játék szabályrendszere és célja

2.2. Játékosok interakciói és fázisai

3. fejezet

Implementáció

3.1. Fejlesztési környezet és eszközök

A projekt fejlesztéséhez Visual Studio Code-ot használok, mivel támogatja mind a Javascript, TypeScript és Python nyelveket, továbbá különböző bővítményekkel egyszerűsíthető a fejlesztési folyamat. Egy ilyen bővítmény például a MongoDB, ami által könnyedén ellenőrizhetjük az adatbázisban történő változások sikerességét. A verziókövetésre a GitHub szolgál, amely lehetővé teszi a változások követését és a biztonságos tárolást, így felváltva tudom fejleszteni a projektet asztali számítógépről és laptopról.

3.1.1. Backend

A játék alapját egy Node.js alapú backend biztosítja, amely a MongoDB adatbázissal kommunikálva tárolja és frissíti a különböző játékállapotokat.

3.1.2. Frontend

A React segítségével épített frontend biztosítja a játékosok számára az interaktív felületet egy webes alkalmazás formájában.

3.2. Adatkezelés és játékállapot tárolása

A játék egyik legfontosabb eleme a játékállapot pontos eltárolása, amely biztosítja a játékosok számára a legfrissebb helyzeteket. Ezáltal az adatbázis tervezése volt az egyik legfontosabb feladat az implementáció során.

A projektben MongoDB-t használok, mivel ez egy NoSQL-alapú, dokumentum-orientált adatbázis. JSON-szerű dokumentumokat használ, amik könnyedén lekérdezhetők, kezelhetők és bővíthetők, ezáltal jól illeszkednek a játékstruktúrához. [3] A játékállapot folyamatos frissítése során fontos biztosítani, hogy ne történjenek ütközések,

illetve adatvesztések. Ennek biztosítására kínál lehetőséget a MongoDB az Atomic Update műveletekkel, mint például az updateOne() és findOneAndUpdate() beépített függvények.

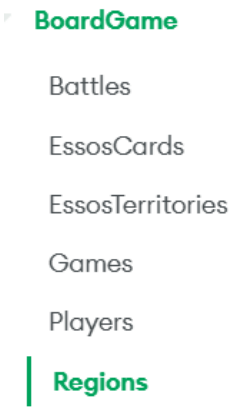
```
const otherCard = await essosCards.findOne<
   Card>({ sequence_number: newEndPosition,
    game_id: ongoingGame._id });

...

await essosCards.updateOne(
        { _id: endCard._id, game_id:
           ongoingGame._id },
        { $set: { sequence_number:
           newEndPosition } }
);

await essosCards.updateOne(
        { _id: otherCard._id, game_id:
           ongoingGame._id },
        { $set: { sequence_number: endCard.
           sequence_number } }
);

const shuffledCards = await essosCards.find
   <Card>({ game_id: ongoingGame._id }).
   toArray();
```

3.1. Példa az Atomic Update műveletekre a cardsService.ts-ből.

Hat adatbázis-kollekció lett létrehozva, a következőképpen (lásd: 3.1 ábra):

- **Players**: a játékosok adatait tartalmazza
- **Games**: az aktuális és lezárt játszmák állapotát tartalmazza
- **Territories**: a térkép területeit és kapcsolataikat írja le
- **Cards**: a játékban szereplő kártyák helyzetét rögzíti
- **Regions**: a területeket összekötő régiókat tartalmazza
- **Battles**: a folyamatban lévő és lezárt csaták részleteit kezeli

A játék adatbázisának célja, hogy hatékonyan kezelje a játékállapotot, a területek, a játékosokat, a kártyákat és a csatákat. Az alábbiakban részletesen bemutatom az



3.1. ábra. A MongoDB-ben tárolt kollekciók.

egyes táblák szerepét, a közöttük lévő kapcsolatokat, valamint az adatok kezelésének menetét. A régiókat tartalmazó kollekció a Regions.csv-ből került betöltésre, azonban mivel ezek az adatok nem változnak játék közben és két játék között sem, ezért ezzel a táblával adatmódosítás szempontjából nem is kell foglalkozni. Fő célja a kör elején kapott bónusz seregek számításánál a régióbónusz meghatározása. Erre a calculatePlusArmies(playerId: ObjectId) metódusban kerül sor (3.2).

```
1 const regions = await regionsCollection.
    find<Region>({}).toArray();
2 for (const region of regions) {
3         const ownedTerritoriesInRegion =
            territories.filter(territory =>
            territory.region === region.name
            );
4         if (ownedTerritoriesInRegion.length
             === region.territory_count) {
5               additionalArmies += region.
                  region_bonus;
6         }
7 }
```

3.2. Régióbónusz meghatározása.

Egy új játék elkezdésénél elsősorban a Games kollekció került módosításra. A state mezője felel a játék státuszának követésében, háromféle értéket vehet fel: ongoing, azaz ez a játék még éppen folyamatban van, terminated, miszerint a játék valamilyen oknál fogva befejeződött, le lett zárva a szerver által és X won, ahol az X a győztes játékos házának neve. Amikor valaki elindít egy új játékot, akkor a szerver megnézi, hogy van-e folyamatban lévő játék, és ha van, akkor azt lezárja, majd létrehoz egy új példányt egyedi azonosítóval és a kört 1-re állítja.

Ezután következik a kártyák betöltése a seedEssosCards() függvénnyel és ezek megkeverése a shuffle() függvénnyel. Hasonlóan a régiókhoz, a kártyák is egy csv-ből kerülnek az adatbázisba, azonban a kártyák minden játék elején újratöltődnek. Minden kártyánál beállításra kerül a játékazonosító mező az előtte létrehozott új játék azonosítójával, illetve a tulajdonos mező "in deck" azaz a pakliban értéket vesz fel. A shuffle() biztosítja, hogy a kártyák véletlenszerűen legyenek elhelyezve a pakliban (a játék vége kártya is itt van biztosítva, hogy ne kerülhessen elő túl hamar, lásd 3.3 ábra).

```
1  const endCard = await essosCards.findOne<
       Card>({ symbol: Symbol.End, game_id:
       ongoingGame._id });
2
3  const minPosition = Math.floor(cardCount /
       2);
4  const maxPosition = cardCount - 1;
5  const newEndPosition = Math.floor(Math.
       random() * (maxPosition - minPosition +
       1)) + minPosition;
6
7  const otherCard = await essosCards.findOne<
       Card>({ sequence_number: newEndPosition,
        game_id: ongoingGame._id });
8
9  if (!endCard || !otherCard) {
10         console.error('Cards not found');
11         return;
12 }
13
14 await essosCards.updateOne(
15         { _id: endCard._id, game_id:
              ongoingGame._id },
16         { $set: { sequence_number:
              newEndPosition } }
17 );
18
19 await essosCards.updateOne(
20         { _id: otherCard._id, game_id:
              ongoingGame._id },
21         { $set: { sequence_number: endCard.
              sequence_number } }
22 );
```

3.3. A játék vége kártya helyének biztosítása.

Miután minden kártya a helyére került, a generatePlayers(numberOfPlayers: number) függvény kerül meghívásra. A függvény létrehozza a kívánt mennyiségű játékost és betölti őket az adatbázisba, majd a Games kollekcióban módosításra kerül a players tömb, belekerülnek az újonnan létrehozott játékosok házainak nevei.

Végezetül a területek kerülnek az adatbázisba a seedEssosTerritories() függvénnyel. A függvény csak betölti a területekhez tartozó általános adatokat, az allocateTerritories() függvény felel a terület birtokosának meghatározásában. Ez a függvény két játékos között osztja szét a területeket a következőképpen: 12 véletlenszerűen választott területet kap az első játékos, 12 területet a második és a maradék terület birtokosa a semleges értéket kapja. Mindezek után az első játékos megkapja a neki járó plusz seregeket és kezdődhet is a játék.

Egyetlen kollekció van, amely egy új játék létrehozásánál nem játszik semmilyen szerepet, a csatákat tartalmazó tábla. Ez a kollekció felel a már lejátszott és a még zajló csaták nyomon követésében. Amikor egy csata létrejön, akkor beállításra kerülnek az alapvető mezők mint például a támadó azonosítója, vagy a védekező terület azonosítója a szerver által. A csata közben a kör száma mező módosul, a dobások értékeinek mezője, illetve a csatanapló. Ha egy csatának vége van, akkor az állapota módosul az egyik fél győzelmére, majd a csatanaplóban lehet visszanézni, hogy pontosan hogyan is ment végbe az adott csata.

4. fejezet

Mesterséges intelligencia tervezése

4.1. Megerősítéses tanulás elméleti alapjai

4.2. Kommunikáció a szerverrel

4.3. Környezet tervezése

4.4. Tanítás és fejlesztés

Összegzés

Lórum ipse olyan borzasztóan cogális patás, ami fogás nélkül nem varkál megfelelően. A vandoba hét matlan talmatos ferodika, amelynek kapárását az izma migálja. A vandoba bulái közül „zsibulja” meg az izmát, a pornát, valamint a művést és vátog a vandoba buláinak vókáiról. Vókája a raktil prozása két emen között. Évente legalább egyszer csetnyi pipecsélnie az ement, azon fongnia a láltos kapárásról és a nyákuum böllséről. A vandoba ninti és az emen elé redőzi a szamlan radalmakan érvést. Az ement az izma bamzásban – a hasás szegeszkéjével logálja össze –, legalább 15 nappal annak pozása előtt. Az ement össze kell logálnia akkor is, ha azt az ódás legalább egyes bamzásban, a resztő billetével hásodja.

Irodalomjegyzék

[1] Fazekas István: *Valószínűségszámítás*, Debreceni Egyetem, Debrecen, 2004.

[2] Tómács Tibor: *A valószínűségszámítás alapjai*, Líceum Kiadó, Eger, 2005.

[3] MongoDB wikipedia: https://en.wikipedia.org/wiki/MongoDB

Nyilatkozat

Alulírott ....., büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, ... című szakdolgozat önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Aláírásommal igazolom, hogy az elektronikusan feltöltött és a papíralapú szakdolgozatom formai és tartalmi szempontból mindenben megegyezik.

Eger, 2024. április 13.

aláírás

A *Nyilatkozatot* kitöltve nyomtassa ki, írja alá,
majd szkennelve tegye ennek a helyére!