 MESTERSÉGES INTELLIGENCIA

Mesterséges Intelligencia Snake tanulására

Vass Kornél

CCD1VK

(GKNB\_INTM002)

2023/2024/2

Tartalom

[Használati útmutató 3](#_Toc161055589)

[Probléma bemutatása 4](#_Toc161055590)

[Használt könyvtárak, csomagok, programok 5](#_Toc161055591)

[Anaconda 5](#_Toc161055592)

[Pygame 5](#_Toc161055593)

[Pytorch 5](#_Toc161055594)

[Matplotlib 5](#_Toc161055595)

[Ipython 5](#_Toc161055596)

[Játék bemutatása 6](#_Toc161055597)

[Az alapok 6](#_Toc161055598)

[Inicializálás 6](#_Toc161055599)

[play\_step függvény szerkezete 7](#_Toc161055600)

[Mesterséges intelligencia bemutatása 8](#_Toc161055601)

[Megerősítéses tanulás 8](#_Toc161055602)

[Mély Q tanulás 8](#_Toc161055603)

[Bellman egyenlet 8](#_Toc161055604)

[Az agent ciklusának működése 9](#_Toc161055605)

[Tesztek 10](#_Toc161055606)

[Források 11](#_Toc161055607)

[Szöveges források 11](#_Toc161055608)

[Videós források 11](#_Toc161055609)

# Használati útmutató

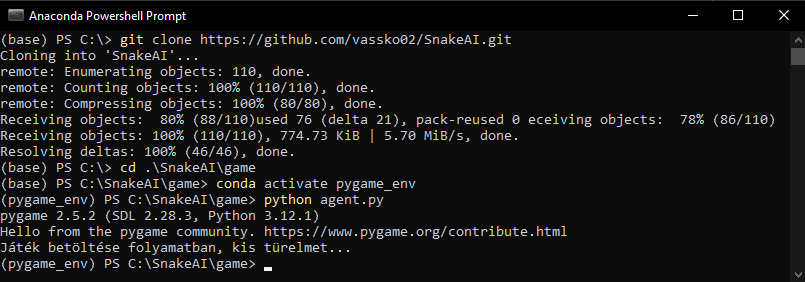
A GitHub repository, amelyben dolgoztam: <https://github.com/vassko02/SnakeAI.git>

1. Telepítsük az Anaconda-t, az alábbi linkről: <https://www.anaconda.com/download>
2. Indítsuk el az Anaconda-t és egy „Anaconda Powershell Prompt” -ot \*
3. Klónozzuk a GitHub repository-t a „git clone <https://github.com/vassko02/SnakeAI.git>” parancs segítségével
4. Lépjünk be a GitHub-ról letöltött fájlba a „cd SnakeAI/game \*
5. Hozzuk létre egy pygame környezetet: „conda create -n pygame\_env python=3.12”
6. Adjuk ki a következő parancsot: „conda activate pygame\_env” \*
7. Telepítsük a szükséges csomagokat:

* pip install pygame
* pip install torch torchvision
* pip install matplotlib ipython

1. Indítsuk el az AI-t a „python agent.py” utasítással \*

(Miután elindult a játék, az ablakok mozgatása, vagy átméretezése összeomlást okozhat, erre próbáltam megoldást keresni, de sajnos nem találtam)

**Sikeres telepítés után a használathoz csak a \*-gal jelölt utasítások kiadása szükséges**

1. ábra Minta a kiadott parancsokról (5. és 7. lépés nélkül)

# Probléma bemutatása

A Snake egy nagyon régi játék, már nyomógombos telefonokon is megjelent (lásd 2.ábra). Az ember gyakran azt gondolná, hogy egy ilyen egyszerű játék, nem okozhat különösebb problémákat, de mégis, ha nem sikerül elérni egy magasabb pontszámot, könnyen felmerülhet frusztráció. De akkor hogyan tudunk kiemelkedően magas pontszámot elérni mindenféle düh és nehézség nélkül? Az egyik lehetséges megoldás egy mesterséges intelligencia fejlesztése, amely képes folyamatosan fejlődni a játékban, és így egyre magasabb pontszámokat ér el.

Amint az AI megfelelően ki lett fejlesztve és tanítva, felmerülnek további kérdések is. Például, mennyi időbe telik az AI-nak az első élelem megtalálása, és mennyi idő alatt éri el a száz pontot? Hány próbálkozás után lehet biztos abban, hogy az elért pontszám nem fog ötven alá esni? Ezek a válaszok nagyon sok tényezőtől függenek. Lényeges például az, hogy milyen paraméterekkel és algoritmusokkal rendelkezik az AI, illetve mennyi időt és erőforrást szánunk a fejlesztésre és tanításra. Ezekre a kérdésekre a [tesztek című fejezetben](#_Tesztek) visszatérünk.

A válaszok keresése és a mesterséges intelligencia továbbfejlesztése során rengeteg kihívással és lehetőséggel találhatjuk magunkat szemben. Fontos, hogy nyitottak maradjunk az új megközelítések és technikák felé, és folyamatosan tanuljunk a fejlesztési folyamat során felmerülő kihívásokból. Ez lehetővé teszi, hogy egyre hatékonyabb és kifinomultabb AI-t hozzunk létre, amely képes megoldani az előtte álló problémákat és elérni a kitűzött célokat a Snake játékban. [1]

2. ábra Snake egy régi telefonon (illusztráció)

# Használt könyvtárak, csomagok, programok

## Anaconda

Az Anaconda egy olyan python alapú program, amelyet gépi tanulásra és adat elemzésre specifikáltak. Több előre telepített csomagot is tartalmaz, például: [matplotlib](#_Matplotlib), [ipython](#_Ipython), vagy pandas. Ez a legnagyobb fő előnye a programnak, amellett, hogy van egy saját csomagkezelője a „Conda”, aminek a segítségével további csomagokhoz tudunk hozzáférni. Mindezek mellett az Anaconda egy biztos környezetet biztosít a programok futtatására, így bízhatunk abban, hogy a programunk minden eszközön ugyanúgy fut, ahol telepítve van a program.[2]

## Pygame

A Pygame több platformfüggetlen Python modul összessége, amelyet videójátékok készítésére használnak. Tartalmaz számítógépes grafika és hangkönyvtárakat, amelyeket a Python programozási nyelvvel használhatunk. A Pygame hivatalosan Pete Shinners által íródott, hogy leváltsa a PySDL-t. A Pygame alkalmas kliensoldali alkalmazások létrehozására, amelyek potenciálisan önálló végrehajtható fájlba csomagolhatók.[3]

## Pytorch

A PyTorch egy nyílt forráskódú gépi tanulási (ML) keretrendszer, amely a Python programozási nyelvre és a Torch könyvtárra épül. A Torch egy nyílt forráskódú ML könyvtár, amelyet mély neurális hálózatok létrehozására használnak, és a Lua szkriptelési nyelven íródott. Ez az egyik preferált platform a mély tanulási kutatásokhoz. A keretrendszer azért lett kifejlesztve, hogy felgyorsítsa a kutatási prototípusok és a bevezetés közötti folyamatot.[4]

## Matplotlib

A Matplotlib egy adatvizualizációs könyvtár Pythonban, amelyet 2D plotokhoz használnak. Matplotlib egy többplatformos adatvizualizációs könyvtár, amelyet NumPy tömbökön alapul, és tervezték a szélesebb körű SciPy veremmel való együttműködésre. John Hunter vezette be 2002-ben. Az egyik legnagyobb előnye a vizualizációnak, hogy lehetővé teszi számunkra a hatalmas adatmennyiség vizuális hozzáférését könnyen fogyasztható ábrák formájában. A Matplotlib többféle diagramot is kirajzolhat, mint például vonaldiagram, oszlopdiagram, szórásdiagram, hisztogram.[5]

## Ipython

Az IPython egy olyan fejlett eszköz, amely számos előnyt kínál a hagyományos Python-nal szemben. Kiemelkedő tulajdonságai közé tartozik egy erőteljes interaktív Python shell, amely lehetővé teszi a gyors és hatékony kódírást és végrehajtást. A beépített objektum-introspekció funkció lehetővé teszi az objektumok tulajdonságainak ellenőrzését futásidőben, ami rendkívül hasznos a fejlesztés során. Az IPython továbbá támogatja a szintaxiskiemelést, tárolja az interakciók történetét, és lehetővé teszi a tabulációval történő kulcsszó, változó és függvénynevek kiegészítését. A varázslatparancs rendszer segítségével könnyen irányítható a Python környezet, és végrehajthatók operációs rendszer feladatok. Az IPython továbbá beágyazható más Python programokba, és hozzáférést biztosít a Python hibakeresőhöz, amely segít a hibák azonosításában és javításában a fejlesztés során.[6]

# Játék bemutatása

Maga a játék a GitHubon a [game.py](game/game.py) fájlban található. Ez tartalmazza a játék mechanikákat, ez nem azt jelenti, hogy ha ezt a fájlt futtatnánk akkor manuális irányítással működne a játék, mivel az AI miatt módosításokat kellett végezni a fájlban, például az irányítás részét teljesen át kellett alakítani mert nyilvánvalóan nem billentyűzetről fogja kapni a bemenetet a program.

## Az alapok

Az alábbi képen a game.py fájl első pár sora látható, ahol a csomagok kerültek beimportálásra és a konstansok kerültek definiálásra. Az irányokat egy enum-ban tároltam, amit akkor használnak, ha egy változónak egy meghatározott értékkészlete legyen, mint most esetünkben az irány. Ezután létrehoztam a Point nevű „namedtuple”-t aminek a segítségével majd a későbbiekben a megjelenített ablak adott pontjaira tudok hivatkozni. Majd definiáltam a kígyó sebességét és a blokkok méretét 20 pixelre, röviden ez annyit takar, hogy felbontottam az ablakot 20\*20-as négyzetekre. Végül pedig színeket definiáltam, ezeket a konstansokat használtam a kígyó, a keret és a háttér színezéséhez. Ezen adatok rögzítése után pedig a játékhoz használt függvényeket hoztam létre.

3. ábra Kódrészlet a game.py fájlból

## Inicializálás

Először inicializálni kell a programot. Ez esetünkben annyit takar, hogy létrehozzuk az ablakot, ahol a játék futni fog, majd az ablak közepére tesszük a kígyót, úgy, hogy jobbra induljon el, majd lehelyezünk egy ételt. Magát a kígyót egy Point-okból álló listában tároltam, aminek a nulladik eleme a kígyó feje utána pedig a folyamatosan növekvő test. Az étel elhelyezéséhez egy következő függvényt hívunk meg amely randomizál, egy képet, hogy melyik gyümölcs kerüljön az ablakra és két számot, amelyek x és y koordinátaként működnek. Fontos, hogy ezek mindig olyan számok lesznek, amelyek oszthatóak a 2. ábrán látható BLOCK\_SIZE-zal. Ha a random pont a kígyóban van akkor új pontot csinál a függvény és azt helyezi le.

Inicializálás után kezdődik a játék, ami egy több lépcsős függvényből áll, amely a kígyó haláláig ismétlődik, majd, ha ez bekövetkezik akkor az előző bekezdésben bemutatott függvény segítségével visszaállítjuk a játékot a kezdeti helyzetbe.

## play\_step függvény szerkezete

* **Bemenet kezelése**

Mivel a programot nem manuálisan vezéreljük hanem AI irányítja ezért ebben a lépésben csak az ablak bezárását kell kezelni, azaz, ha az ablakot bezárjuk akkor a teljes programot le kell állítani.

* **Mozgás**

A mozgás levezényléséhez először azt kell megvizsgálnunk, hogy van-e irányváltozás az előző frame óta. Ha van akkor a Directions enum segítségével megváltoztatjuk az irányt. Végül pedig a blokk méretével eltoljuk a kígyót a megfelelő irányba, úgy, hogy a fejét beszúrjuk a lista elejére.

* **Játék végének vizsgálata**

Egy adott körnek akkor van vége, ha a kígyó meghal, ez akkor következik be, ha beleütközik a falba vagy saját magába. Emellett a játéknak akkor is vége van, ha a kígyó huzamosabb ideig nem ér el semmit. Ha ezek valamelyike bekövetkezik akkor a függvény negatív 10-et ad vissza jutalomként, és igaznak állítja be a game\_over változót.

* **Új étel elhelyezése**

Ennél a résznél azt kell megvizsgálni, hogy a kígyó összeszedte-e a lehelyezett ételt, ha igen akkor a jutalmat 10-re állítjuk és növeljük a pontszámot eggyel, ezzel a kígyó hossza is nő. Ha nem vesz fel ételt az adott frame-ben a kígyó akkor az utolsó blokkját eltávolítjuk. Erre azért van szükség mert a „Mozgás” résznél beszúrtuk a fejet a lista elejére, de azért, hogy ne nőjön a kígyó hossza mikor nem szed össze gyümölcsöt el kell távolítanunk az utolsó blokkot a listából.

* **UI frissítése és a sebesség növelése**

A UI frissítése minden frame-ban lefut. Letisztítja az ablakot, majd újra rajzol rá mindent, beleértve a változásokat. Először feltölt zölddel az egészet, majd a keretet helyezi el. Ezután a kígyó és az étel kerül kirajzolásra. Végül pedig a pontszám felirata. Ha ez sikeresen frissült utána pedig a sebességet növeltem, az alap sebesség 35, ez mindig az adott pontszámmal van növelve.

* **Érték visszaadás**

A függvény végén több értéket is visszaad a függvény, ezek a következők: *jutalom, vége van-e a játéknak, elért pontszám*. Ezt az [agent](game/agent.py) kapja meg ami ezektől az információktól függően folytatja a játékot.

# Mesterséges intelligencia bemutatása

## Megerősítéses tanulás

A megerősítéses tanulás (Reinforcement Learning - RL) a gépi tanulás egy részterülete, amely lehetővé teszi egy AI-generált rendszernek, hogy próbálkozás és hiba útján tanuljon a cselekedeteinek visszajelzése alapján. Ez a visszajelzés, büntetésként vagy jutalomként jelezve, természetesen a jutalomfunkció maximalizálása céljából.[7]

Azért volt jó döntés a SnakeAI megvalósítása ebben a keretrendszerben, mert a Snake egy olyan környezetet biztosít, amely ideális a RL gyakorlására. A játék egyszerű, de elég változatos, hogy különböző stratégiákat tesztelhessünk, és a jutalmazási rendszer segít az ügynöknek megtanulni, hogy milyen lépéseket kell tennie a legjobb eredmények elérése érdekében. A Snake játék tehát remek lehetőséget kínál az RL alapelveinek megértésére és alkalmazására.

## Mély Q tanulás

A mély Q-tanulás és a "Vanilla" Q-tanulás közötti alapvető különbség a Q-tábla megvalósításában rejlik. A mély Q-tanulás esetén a hagyományos Q-táblát egy neurális hálózattal helyettesítik. A (state action) párok Q értékhez történő mappolása helyett (action Q-érték) párokra mappolja.

Az egyik érdekes dolog a mély Q-tanulásban, hogy a tanulási folyamat két neurális hálózatot használ. Ezeknek a hálózatoknak ugyanaz az architektúrájuk, de más súlyokkal rendelkeznek. Minden N lépésben a súlyokat a fő hálózatból átmásolják a célnak nevezett hálózatba. Mindkét hálózat használata stabilitást biztosít a tanulási folyamatban, és segít az algoritmus hatékonyabb tanulásában.[8]

## Bellman egyenlet

A Bellman-egyenlet a megerősítő tanulás alapvető egyenlete, amely kifejezi az állapot vagy állapot-akció pár értéket és az ebből az állapotból várható jutalmak közötti kapcsolatot. Nevét Richard E. Bellmanről kapta, aki jelentős mértékben hozzájárult a dinamikus programozáshoz és vezérléselmélethez.[9]

**Ahol:**

4. ábra Egyszerűsített Bellman egyenlet a [model.py](game/model.py) fájlban

**Q\_new:** az új Q érték a jelenlegi állapothoz

**reward[idx]:** az a jutalom, amit azért kap mert a megfelelő lépést a megfelelő időben végezte el

**self.gamma:** megmutatja mennyire értékes az a jutalom amit egy jövőbeli lépéssel szedhetünk össze

**torch.max(self.model(next\_state[idx])):** a maximum értéke az action value függvénynek a következő állapotban lévő összes művelethez képes

## Az agent ciklusának működése

* **Környezet vizsgálata**

Ebben a fájlban egy végtelen ciklus fut, ez azt jelenti, hogy addig tanul a kígyó amíg mi azt le nem állítjuk. Elsőkörben az ablak állapotát méri fel, ez azt takarja, hogy megvizsgálja, hogy a kígyó fejétől melyik irányban van az étel és azt is, hogy a fej melyik irányában van veszély (fal, vagy a kígyó teste).

* **Következő lépés meghatározása**

Ezen a pontot megjelenik a felfedezés és kihasználás dilemmája (exploration or exploitation) A felfedezés bármely olyan cselekvés, amely lehetővé teszi az ügynök számára, hogy új jellemzőket fedezzen fel a környezetben, míg az kihasználás arról szól, hogy kihasználja az eddig megszerzett tudást. Ha az ügynök csak a múltbeli tapasztalatokat használja ki, valószínűleg egy nem optimális stratégiába ragad. Másrészről, ha folyamatosan felfedez, de nem használja ki az eddig szerzett ismereteket, soha nem találhatja meg a jó stratégiát. Az ügynöknek megtalálnia a megfelelő egyensúlyt a kettő között, hogy felfedezhesse az optimális stratégiát, amely a maximális jutalmat hozza.[10]

* **Lépés megvalósítása**

Ha megvan a következő lépés akkor azt a már megismert [play\_step](#_play_step_függvény_szerkezete) funkció meghívásával végre is hajtja az AI. Ezután ismételten felméri a játék állapotát az agent, ugyanúgy, mint a lépés meghatározás előtt. A megtett lépés után a rövidtávú memória tanítása következik, ezzel a következő lépésre tanítjuk az AI-t. Ebben a lépésben jelenik meg a [Bellman egyenlet](#_Bellman_egyenlet), az új Q érték kiszámolásához. Ezt követően a megtett lépés tulajdonságait eltároljuk, ezek a következők: *mozgás előtti állapot, mozgás utáni állapot, mozgás iránya, a jutalom, és hogy vége lett-e a játéknak*.

* **Játék végének vizsgálata**

Amennyiben vége lett a játéknak (azaz a [play\_step](#_play_step_függvény_szerkezete) igaz értéket adott vissza), a kezdő helyzet kerül visszaállításra. Minden halál után a hosszú távú memória is tanítva van, ez ugyanazon a módon történik, mint a rövid távú memória tanítása, annyi különbséggel, hogy nem csak egy lépés adatait kapja, hanem az összes eltárolt lépéshez tartozó adatot. A következő lépés a rekord frissítése, amennyiben az adott körben a kígyó magasabb pontszámot ért el a jelenlegi rekordnál. Ezután pedig a konzolra kerül kiírásra az adott kör statisztikája. Végül pedig a grafikon kirajzolásához szükséges adatok frissítésére kerül sor, maga grafikon rajzolása a [helper.py](game/helper.py) fájlban történik.

# Tesztek

A [probléma bemutatása fejezetben](#_Probléma_bemutatása) felsoroltam pár „mérföldkövet” az AI életében. Háromszor harminc percen keresztül futtattam a mesterséges intelligenciát, ezen és további statisztikák felderítése érdekében. A használt ablakméret 640\*480, amiből minden oldalon 20 pixel a keret. Nem akartam nagyobb ablakot készíteni mert akkor az elején jóval tovább tartana míg megtalálja a kígyó az ételt, és ez nagyban lelassítaná a tanulási folyamatot.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vizsgált adat | 1. mérés | 2. mérés | 3. mérés | Átlag |
| Hány kör kellett az első étel összeszedéséhez? | 2 | 13 | 6 | **7** |
| Hány kör kellett 20 étel összeszedéséhez? | 98 | 87 | 78 | **87.66** |
| Hányadik kör volt az utolsó 0 végeredménnyel? | 95 | 86 | 89 | **90** |
| Hány kör futott le 30 perc alatt? | 201 | 195 | 196 | **197.33** |
| Mi lett a rekord 30 perc után? | 52 | 70 | 65 | **62.33** |
| Mennyi lett az átlagos eredmény 30 után? | 15.43 | 18 | 18.04 | **17.16** |

A három teszt figyelmen követése után azt vettem észre, hogy az AI viszonylag gyorsan alkalmazkodik a falakhoz, és egy adott ponttól kezdve csak az vet véget a játéknak mikor magába ütközik. Úgy gondolom, hogy további futtatás után ezt is megtanulná kiküszöbölni az AI. A diagrammot vizuálisan szakaszokra tudtam osztani, az első része az volt amikor még falba ütközött, ez egy ponton abba maradt, és onnantól kezdve kilőtt a teljesítmény. Véleményem szerint további futtatásnál ismét kilőne, ha eljutna a következő probléma megoldásához.

# Források

## Szöveges források

[1] „"Nostalgic Nukia Snake Game " Sticker for Sale by PostFresh”, Redbubble. Elérés: 2024. március 7. [Online]. Elérhető: https://www.redbubble.com/i/sticker/Nostalgic-Nukia-Snake-Game-by-PostFresh/141558938.EJUG5

[2] „Anaconda vs Python in Software: Know the Key Differences | Simplilearn”, Simplilearn.com. Elérés: 2024. március 4. [Online]. Elérhető: https://www.simplilearn.com/anaconda-vs-python-article

[3] „Pygame Tutorial - javatpoint”, www.javatpoint.com. Elérés: 2024. március 4. [Online]. Elérhető: https://www.javatpoint.com/pygame

[4] „What is PyTorch?”, Enterprise AI. Elérés: 2024. március 4. [Online]. Elérhető: https://www.techtarget.com/searchenterpriseai/definition/PyTorch

[5] „Introduction to Matplotlib”, GeeksforGeeks. Elérés: 2024. március 4. [Online]. Elérhető: https://www.geeksforgeeks.org/python-introduction-matplotlib/

[6] „IPython - Introduction”. Elérés: 2024. március 4. [Online]. Elérhető: https://www.tutorialspoint.com/jupyter/ipython\_introduction.htm

[7] R. Brooks, „What is reinforcement learning?”, University of York. Elérés: 2024. március 7. [Online]. Elérhető: https://online.york.ac.uk/what-is-reinforcement-learning/

[8] M. Wang, „Deep Q-Learning Tutorial: minDQN”, Medium. Elérés: 2024. március 7. [Online]. Elérhető: https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc

[9] „Bellman Equation | LinkedIn”. Elérés: 2024. március 8. [Online]. Elérhető: https://www.linkedin.com/pulse/bellman-equation-yeshwanth-n/

[10] „What is the exploration vs exploitation trade off in reinforcement learning?”, Scribbr. Elérés: 2024. március 10. [Online]. Elérhető: https://www.scribbr.com/frequently-asked-questions/what-is-the-exploration-vs-exploitation-trade-off-in-reinforcement-learning/

## Videós források

<https://www.youtube.com/watch?v=ORMx45xqWkA&ab_channel=Fireship>

<https://www.youtube.com/watch?v=cO5g5qLrLSo&ab_channel=NicholasRenotte>

<https://www.youtube.com/watch?v=4GhH3d9NsIc&list=PL-9x0_FO_lgn8hXJk5uRv1bO36xyTx00f&ab_channel=MachineLearningwithPhil>

<https://www.youtube.com/watch?v=vhiO4WsHA6c&ab_channel=Chrispresso>

<https://www.youtube.com/watch?v=--nsd2ZeYvs&t=1840s&ab_channel=PatrickLoeber>

<https://www.youtube.com/watch?v=L8ypSXwyBds&t=1041s&ab_channel=freeCodeCamp.org>