



Eötvös Loránd Tudományegyetem

Informatika Kar

Programozáselméleti és Szoftvertechnológiai Tanszék

MI kliens 2 dimenziós ügyességi játékhoz

Témavezetők:

Bordás Henrik
Szoftverfejlesztő
CAE-Engineering kft.

Szerző:

Vecsernyés Márk
nappali tagozat
programtervező informatikus szak

Budapest, 2019

TARTALOMJEGYZÉK

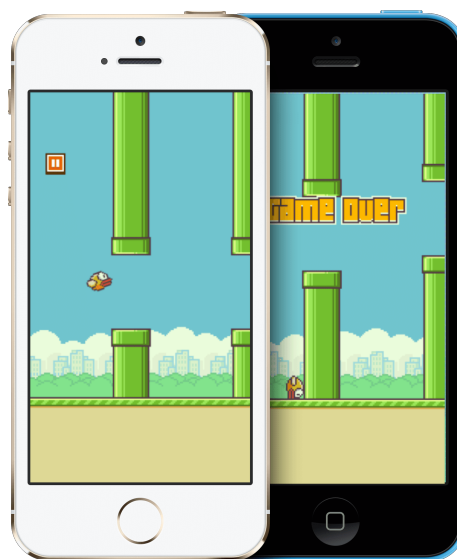
1. Bevezetés.....	3
2. Bevezetés az evolúciós algoritmusba	4
3. Felhasználói dokumentáció.....	9
3.1. A program által megoldott feladat	9
3.2. Célközönség.....	10
3.3. Minimális rendszerkövetelmények	10
3.3.1. Számításokért felelős programcsomagok.....	10
3.3.2. Megjelenítésért felelős programcsomagok.....	11
3.3.3. Hardveres minimális rendszerkövetelmények.....	11
3.4. Első üzembehelyezés	11
3.5. A program futtatása	12
3.5.1. Kezdőoldal	12
3.5.2. Egyszemélyes modul	14
3.5.3. Mesterséges intelligenciát használó modul	15
3.6. Hálózati kommunikációt használó műveletek.....	20
3.7. Adatbázist használó műveletek	20
3.8. Nagyobb erőforrásokat igénylő műveletek.....	21
3.9. Hibaüzenetek, azok elhárításai.....	21
3.10. Kapcsolat a fejlesztővel	22
4. Fejlesztői dokumentáció.....	24
4.1. Általános információk	24
4.2. Megoldási terv	24
4.2.1. A program futásának terve	25

4.2.2. Rendszer architektúrája	28
4.2.3. Adatbázis felépítése.....	30
4.2.4. Komponensek leírásai.....	32
4.2.5. Fájszerkezet (14. ábra).....	35
4.2.6. Futási algoritmus, architektúra, komponensek és fájlok kapcsolata.....	36
4.2.7. Felhasználói felület terve	36
4.3. Megvalósítás.....	37
4.3.1. Egyszemélyes játék forkolása.....	37
4.3.2. Refaktorizáció	38
4.3.3. Gombok hozzáadása a felülethez	38
4.3.4. Flask hozzáadása.....	38
4.3.5. Adatbázis hozzáadása.....	38
4.3.6. POST kommunikáció bevezetése.....	39
4.3.7. Refaktorizált játék hozzáadása a frontendhez	39
4.3.8. Kérések bevezetése a játék logikájához	39
4.3.9. Evolúciós algoritmus prototípusának létrehozása.....	39
4.3.10. Neurális hálók bevezetése	40
4.3.11. Neurális hálók tárolása	40
4.3.12. Evolúciós algoritmus implementálása	40
4.3.13. Hibakezelés	41
4.3.14. Kommentezés	41
4.4. Tesztelés.....	42
4.5. Továbbfejlesztési lehetőségek	42
5. Irodalomjegyzék	43

1. BEVEZETÉS

A világ legnagyobb videómegosztóján számos példa van arra, amikor a gép megtanul játszani egy játékkal anélkül, hogy ismerné annak szabályait. Komplexebb többszemélyes kompetitív játékokban a mesterséges intelligencia képes legyőzni a világ legjobb játékosait¹ is. Mindig is érdekelt, hogy ez hogyan lehetséges, ezért szakdolgozatom témájaként a gépi tanulást választottam egy ismert játékon evolúciós algoritmussal.

A tanítást a Flappy Bird² (1. ábra³) átiratán végeztem. Az eredeti 2013. Május 24-én jelent meg okostelefonos platformra. *Dong Nguyen* fejlesztette a *dotGears* cégnél, 2014 elején napi 50.000 dollárt hozott reklámbevételekből. Két dimenziós, retro játékokra hasonlít, az irányítandó karakter egy sárga madár, aki folyamatosan halad előre balról jobbra, közben a magassága a gravitáció hatására csökken. Egyetlen parancsot



1. ábra

lehet neki adni, ez az ugrás, így kell minél messzebb manővereznie az akadályok (oszlopok) között. 2014. Február 10-én *Nguyen* személyes okokra hivatkozva törölte a játékot. Ezután az egyik legtöbbször másolt játék lett az *Apple Storeban*, megközelítőleg napi 60 másolatot töltöttek fel, de PC-re is számos átirat megjelent.

Dolgozatom célja egy létező algoritmus implementálása egy olyan grafikus környezetben, amiben a felhasználó láthatja a működését a gyakorlatban.

Ezúton szeretnék köszönetet mondani a *családomnak* tanulmányaim támogatásáért, illetve *Bordás Henriknek*, aki konzulensként tudásával és tanácsaival segített felépíteni dolgozatomat.¶

2. BEVEZETÉS AZ EVOLÚCIÓS ALGORITMUSBA

A feladat *célja* egy működő program létrehozása. Jelen esetben az elméleti háttér minimális ismerete nélkül nem értelmezhető kellő mélységben a program által megoldott feladat, ezért szeretném röviden, a teljesség igénye nélkül ismertetni az evolúciós algoritmust működését a programomban. Az evolúciós algoritmus egy *heurisztikákat* nem használó egy speciális *keresőrendszer*. Heurisztikus, vagy informált keresőrendszernek nevezzük azokat, amely a probléma definícióján túlmenően problémaszpecifikus tudást is felhasznál – hogyan képes hatékonyabban megtalálni a megoldást⁴. A *keresőrendszer*⁵ egy algoritmus (2. ábra), ami a problématerben keresi egy helyes választ. Általános alakja a következő:

Procedure KR

1. **ADAT** := *kezdeti érték*
2. while \neg *terminálási feltétel*(**ADAT**) loop
3. **SELECT SZ FROM** *alkalmazható szabályok*
4. **ADAT** := **SZ**(**ADAT**)
5. endloop
- end

2. ábra

A mesterséges intelligenciában használt keresőrendszereknek ennek az algoritmusnak speciális esetei, ilyen például az evolúciós algoritmus is.

A evolúció alapalgorithmusa⁶ (3. ábra) a következő:

***Procedure* EA**

populáció := kezdeti populáció

while terminálási feltétel nem igaz ***loop***

szülők := szelekció(*populáció*)

utódok := rekombináció(*szülők*)

utódok := mutáció(*utódok*)

populáció := visszahelyezés(*populáció*, *utódok*)

endloop

3. ábra

Egy adott pillanatban a problémátérnek sosem egy egyedét (lehetséges választ), hanem egy halmazát tároljuk. Ezt nevezzük *populációnak*. A populációt egyedeit valamivel kódolnunk kell, erre *neurális hálót* fogok használni (ami mátrixokkal van reprezentálva, de jelen dolgozatnak nem célja a neurális háló bemutatása).

Arra törekszünk, hogy az egyedek minél jobbak legyenek, ennek megítélésére rátermettségi függvényt (későbbiekben fitnessz) fogunk használni. Egy madár fitnessz értéke annál magasabb, minél messzebb jut a pályán (minden időpillanatban egyet nő). Minden pálya véletlenszerűen van generálva, sosem egy adott pályára tanulnak rá az egyedek. A tanulás mindig a generáció végén történik.

Legfontosabb fogalmak rövid áttekintése:

- *Egyed*: problémátér egy lehetséges válasza.

- *Kódolás*: egyed reprezentációja.
- *Populáció*: problémátér egy halmaza, az egyedeket tartalmazza.
- *Fitnessz*: az egyed rátermettségi függvénye.
- *Generáció*: az algoritmus egy ciklusa.
- *Evolúciós operátorok*: Szelekció, rekombináció, mutáció, visszahelyezés.
- *Terminálási feltétel*: az a feltétel, amikor az algoritmus véget ér (a fitnessz eléri a megadott küszöbértéket).

Az algoritmus futása vázlatosan a programomban:

1. Az algoritmus elején a populáció méretével megegyező számú egyedet (madarat) hozunk létre, tetszőleges neurális hálóval. Ez az első generáció inicializálása.
2. Elkezdünk egy játékot, amiben a madarak a neurális hálójuk alapján ugrálnak az oszlopok között (itt még természetesen nem tanulnak). A neurális hálónak 3 input paramétere van: madár magassága (y koordinátája), az következő oszloppár x koordinátája, illetve a következő oszloppár közül a felső tetejének y koordinátája. Minden időpillanatban minden madár lefuttatja a neurális hálóját a jelenlegi pozíciójához tartozó paraméterekre, majd ez alapján kétféle döntés hozhat: ugrik vagy nem ugrik. Ha az egyik madár fitnessz értéke elér egy küszöbindexet, akkor az algoritmus terminál. Amennyiben nem éri el egyik sem, és minden madár meghal, a 3. pontra lépünk.
3. Az első evolúciós operátorunk a szelekció. Itt kiválasztunk néhány (feltehetőleg rátermett) egyedet szülőnek. A cél, hogy a kevésbé rátermett egyedek is esélyt kapjanak (persze kisebb valószínűséggel). Erre több módszer létezik, én a *versengéssel* módszert választottam, ahol tetszőleges

csoportok (itt párok) legjobbját választom ki, pontosan $\left\lfloor \frac{s}{100} * p \right\rfloor$ darabot, ahol $s \in \{50, 51, \dots, 100\}$ szelekciós együttható, illetve $p \in \{20, 25, \dots, 100\}$ a populáció mérete.

4. A második evolúciós operátor a rekombináció. Feladata, hogy a szülőkből olyan utódokat készítsünk, amik a szülők tulajdonságait öröklik. Erre a következő módszert használok: egyszerű jelcsoportokat cserélek két véletlenszerűen választott szülőben. A szülők neurális hálóval vannak kódolva, ami 2 mátrixból, és 2 vektorból áll (példa 4.3.12/16. ábra). A mátrixokat ugyanabban a tetszőleges pontban kettévágom és összeolvasztom, a vektoroknál a kettő közül az egyiket választom ki, és azt adom az utódnak. Pontosán $\left\lfloor \frac{c}{100} * p \right\rfloor$ (alsó egészrész) darab utód jön létre, ahol ahol $c \in \{40, 45, \dots, 90\}$ rekombinációs együttható, illetve $p \in \{20, 25, \dots, 100\}$ a populáció mérete.
5. A harmadik evolúciós operátor a mutáció. Feladata egy utód kisméretű változtatása. Meghatározásához 2 együtthatót kell definiálnunk, $M \in \{5, 10, \dots, 30\}$ és $m \in \{5, 10, \dots, 30\}$. $\left\lfloor \frac{M}{100} * p \right\rfloor$ számú utódot választunk ki, mindegyiknek $\left\lfloor \frac{m}{100} * p \right\rfloor$ darab komponense megváltozik (a neurális háló minden eleme $(-1, 1)$ intervallumba esik, így ezen az intervallumon belül változhat a kiválasztott elem).
6. A negyedik evolúciós operátor a visszahelyezés. Feladata a populáció utódokkal való frissítése. Kiválsztja a lecserélendő egyedet, majd azok helyére az újakat teszi. A populáció mérete legyen $p \in \{20, 25, \dots, 100\}$ és a törlési együttható $d \in \{1, 2, \dots, 10\}$. Az utolsó (legrosszabb fitnessz értékkel

rendelkező) $\left\lfloor \frac{d}{100} * p \right\rfloor$ darab egyed helyett újat generálunk, majd a sorban következő (növekvő sorrendben) egyedeket lecseréljük az összes utódra. A maradék (legjobb fitnessz értékű) egyed marad. A generáció számát növeljük eggyel, majd 2. pontra lép az algoritmus.

Megjegyzés: az együtthatók intervallumait tapasztalati úton hoztam létre

Az evolúciós algoritmus egy nem-módosítható stratégia, a populáción végzett változtatások visszavonhatatlanok. Nem determinisztikus, a populáció módosítása mindig ugyanazzal a stratégiával, de véletlenszerű választásokkal történik. Ugyanarra a kezdőpopulációra más eredményeket ad általában. A program kipróbálásakor ezért tapasztalhatunk eltérő tanulási időket, például valamikor egy 3. generációs egyed eléri a megadott küszöbértéket, valamikor pedig a 200. generációra sem képes egyetlen egyed sem eljutni a 3. oszlopig.

3. FELHASZNÁLÓI DOKUMENTÁCIÓ

A következőkben ismertetjük *a program által megoldott feladatot, célközönségét*, minimális rendszerkövetelményeit, majd a felhasználó szempontjait szem előtt tartva áttekintjük az *első üzembehelyezést*, majd részletesen bemutatjuk a *program használatát*. Felsoroljuk a hálózati kommunikációt, adatbázist, nagyobb erőforrásokat használó műveleteket. Bemutatjuk a rendszer hibaüzenetét, és azok megoldásait. Jelen fejezet az olvasási szokások miatt (az egyes részeknek önmagukban is teljeseknek kell lennie) redundáns információkat tartalmazhat.

3.1. A program által megoldott feladat

A program a gépi tanulás egy módszerének, az *evolúciós algoritmusnak* a futását, nem pedig magát az algoritmust akarja részletesen megmutatni a felhasználónak. A szoftver korlátozottan, de paraméterezhető.

Bármilyen módszer működése egy adott probléma megoldására akkor szemléltethető a legjobban, ha a modell kellően bonyolult, viszont közérthető. A bonyolultságra azért van szükség, mert az esetleges egyszerűsítések következtében ha túl sok tulajdonságot elhagyunk, akkor a módszer hátrányait nem minden esetben láthatjuk. Közérthetőség nélkül pedig a probléma sokak számára értelmezhetetlenné válik. A bevezetőben ismertetett modellt fogom úgy átalakítani, hogy a gép képes legyen hozzáférni a megfelelő interfészekhez ("lássa, amit a felhasználó láthat", illetve "parancsot tudjon neki adni, hogy a felhasználó", de semmi többet).

A *kiindulási modell* a játék egy változata melynek forráskódja⁷ ingyenesen letölthető az internetről. Itt a felhasználó manuálisan, egy gomb lenyomásával tudja irányítani a karaktert. Ezt az input interfészt le kell cserélnünk, hogy a számítógép irányítsa madarat, illetve a populációnak méretének megfelelő számú madarat kell elhelyezni a játékban. A minden madarat eltérő mesterséges

neurális hálóval kell kódolni, futása végén (egy futás játék indításától a madár haláláig tart) pedig fitnessz értékét feljegyezni. Az algoritmus a ezen értékek ismeretében tudja elvégezni a kiértékelést. *A program által megoldott feladat a feljebb ismertetett modellen történő optimális egyed keresés evolúciós algoritmussal.*

3.2. Célközönség

A program célközönsége azok az emberek, akik már megismerkedtek az evolúciós algoritmussal, és szeretnék látni működés közben. A módszer előnyei, illetve hátrányai közül számos előjön futtatások során. Például egy alkalmazási terület amikor a tanár elmagyarázza az algoritmust, majd elindítja ezt a programot, és a diákok láthatják működés közben.

3.3. Minimális rendszerkövetelmények

A program jelenleg csak asztali számítógépen futtatható, operációs rendszertől függetlenül, amennyiben a szükséges programcsomagok telepítve vannak.

3.3.1. Számításokért felelős programcsomagok

Név	Verzió	Feladat	Telepítés
Python	3.7.0	Interpretálás	Binárisból
Pip3	18.0	Python3 csomagok telepítése	Binárisból
PostgreSQL	10.5	Adatbázis-szolgáltatás	Binárisból
Flask	1.0.2	HTML szerver	Pip3-mal
Psycopg2	2.7.5	Adatbázis utasítások kezelése	Pip3-mal
Numpy	1.15.2	Mátrixok kezelése	Pip3-mal
Torch	0.4.1	Neurális háló kezelése	Pip3-mal

3.3.2. Megjelenítésért felelős programcsomagok

Elég egy böngésző következők közül tetszőlegesen a teljesség igénye nélkül (html, javascript, illetve ajax támogatás szükséges):

Név	Verzió
Google Chrome	70.0
Mozilla Firefox	63.0
Opera	53
Safari	12.0

3.3.3. Hardveres minimális rendszerkövetelmények

Amennyiben a 3.3.1-es, és 3.3.2-es pontban leírt komponensek futnak az adott személyi számítógépen, akkor a program is futni fog. Pontos minimális hardveres követelmények a csomagok hivatalos honlapjain elérhetőek.

3.4. Első üzembehelyezés

Ha a minimális szoftveres és hardveres követelmények a rendelkezésünkre állnak, akkor a forráskód birtokában el fogjuk tudni indítani az alkalmazást, külön telepítést nem igényel, mert interpretert használ. Indítás előtt győződjünk meg a következőkről:

- Python3 interpretert el tudjuk indítani konzolos felületről
- PostgreSQL szolgáltatás fut, illetve rendelkezik 'postgres' role-lal

Első lépésben a számításokért felelős szolgáltatást fogjuk elindítani: nyissunk meg egy konzolt (terminált), navigáljunk a forráskód gyökérkönyvtárába, majd

a "python3 app.py" parancsot kiadva a program elindul. Ekkor az alkalmazás a standard outputra kiír pár üzenetet (4. ábra).

```
markvecsernyes@szonor fbai (master) $ python3 app.py
Connecting to the PostgreSQL database...
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
Connecting to the PostgreSQL database...
```

4. ábra

Ha sikerült csatlakoznia az adatbázishoz, és az üzenetek között nincs hiba, akkor a program fut, több teendőnk ezzel nincs (nem szükséges a továbbiakban leállítani, illetve újraindítani ezt).

3.5. A program futtatása

A böngészőbe <http://127.0.0.1:5000/> címet beírva az URL sávba, megkapjuk a program grafikus felhasználói felületét, amit GUI-nak (graphical user interface) nevezünk. Összesen 3 különböző oldal található meg a programban (Kezdőoldal (3.5.1), egyjátékos modul (3.5.2), mesterséges intelligenciát használó modul (3.5.3), ezeket a következőkben ismertetem.

3.5.1. Kezdőoldal

Leírás: egy üdvözlőképernyő, ahonnan a felhasználó a többi oldalra tud navigálni.

Elérés: A 3.5-ös pontban ismertetett URL a program kezdőlapjára navigálja a felhasználót (5. ábra).



5. Ábra

Felület: A GUI két jól elkülöníthető részre van osztva:

- Gombokra, ahol a "Single player mode" az egyjátékos modulhoz (3.5.2), illetve "AI player mode" ami a mesterséges intelligenciát használó modulhoz (3.5.3) vezet.
- Rajzvászonra 576 pixel * 512 pixel-es mérettel, melynek világoskék a színe. Nyelve angol, a felirat jelentése pedig: "Üdvözljük! Kérem válassza ki a játék módját felül, jó szórakozást!"

Használat: A gombok valamelyikére kell kattintani, hogy el tudjuk kezdeni a játékot valamelyik verzióban.

3.5.2. Egyszemélyes modul

Leírás: a modul célja, hogy a felhasználó kipróbálhassa a probléma nehézségét, nevezetesen, hogy a megfelelő időpillanatokba parancsot adva (egy gombot lenyomva) milyen messze tud jutni a madárral.

Elérés: Erre az oldalra (6. ábra) a *kezdőoldaltól* (3.5.1), illetve a *mesterséges*



intelligenciát használó 6. ábra *modulból* navigálhatunk (3.5.3).

Felület: A GUI két jól elkülíthető részre van osztva:

- Gombokra, ahol a "<Home" a kezdőoldalra (3.5.1), illetve "AI player mode" ami a mesterséges intelligenciát használó (3.5.3) verzióhoz vezet.
- Rajzvászonra 576 pixel * 512 pixel-es mérettel, melynek világoskék a színe. Középen egy vonallal van elválasztva bal oldalán a játék fut (automatikusan indul), jobb oldalon pedig az aktuális pontszám látható, minden időpillanatban frissítve. A madár meghal, ha oszloppal ütközik, ekkor a pontszám 0-ra csökken, a játék pedig újraindul. A szöveg fordítása: "Egyjátékos mód! Nyomja le bármelyik gombot, hogy a madarat vezérelje. A cél elkerülni az oszlopokat."

Használat: A modul használatához bármelyik gombot lenyomhatjuk a billentyűzeten, ekkor ugrik a madár. Az oldalt elhagyni valamelyik fent említett gombra kattintva lehet.

3.5.3. Mesterséges intelligenciát használó modul

Leírás: a modul célja, hogy mesterséges intelligenciát (evolúciós algoritmust) használva a számítógép megpróbálja megtanulni a 3.5.2-ben futtatható játékot minél jobban játszani.

Elérés: Erre az oldalra a *kezdőoldaltól* (3.5.1), illetve az *egyszemélyes modulból* navigálhatunk (3.5.3).

Felület: A GUI három jól elkülöníthető részre van osztva:

- Gombokra, ahol a "<Home" a kezdőoldalra (3.5.1), illetve "Single player mode" ami az egyszemélyes verzióhoz (3.5.2) vezet.
- Rajzvászonra 576 pixel * 512 pixel-es mérettel, melynek világoskék a színe. Két állapota van:

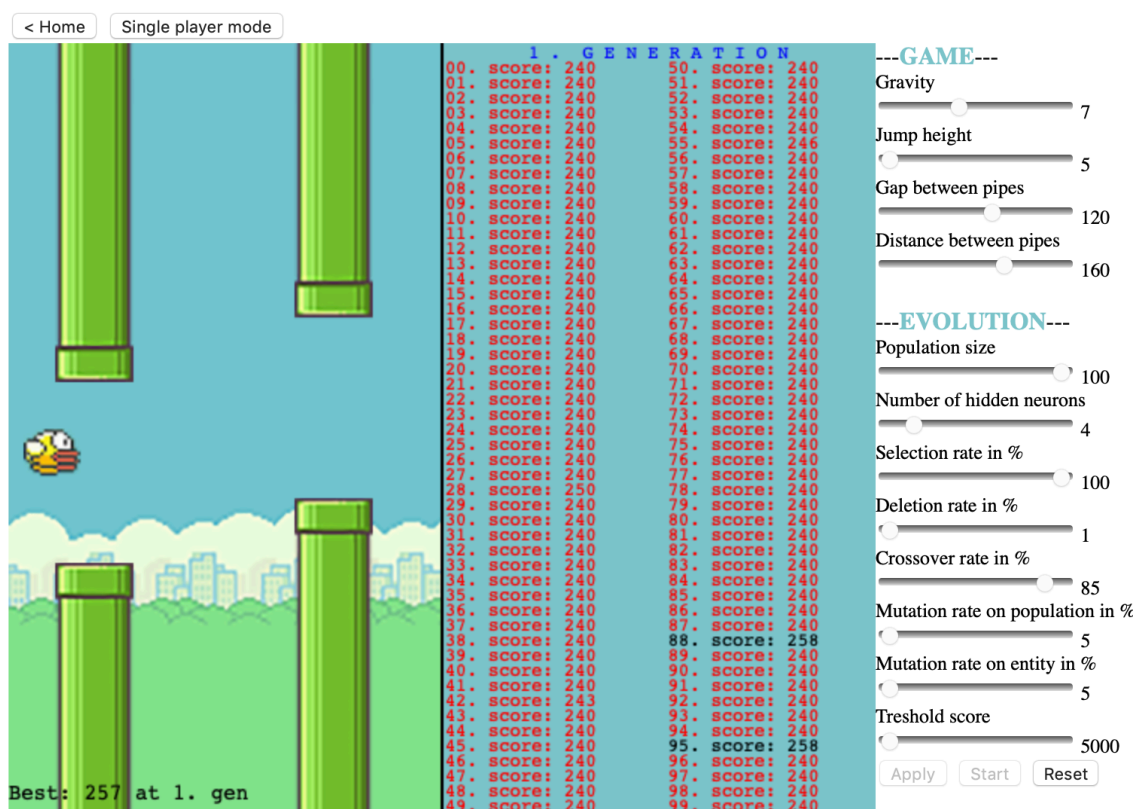
- Indítás előtt (7. ábra): Utasítások vannak kiírva középre. A szöveg fordítása: "Mesterséges intelligenciát használó mód! 1. Állítsa be a futási paramétereket, 2. kattintson az Apply-ra, 3. kattintson a Start-ra."



7. ábra

- Indítás után (8. ábra): egy vonallal van a rajzvásznon kettéosztva. Bal oldalt a játék fut, minden időpillanatban frissítve. A populáció méretével megegyező számú madár van kirajzolva, folyamatosan haladnak balról jobbra, a magasságuk a gravitáció hatására csökken. Az ugrási parancsok mesterséges intelligenciával vannak vezérelve. Ha egy madár oszloppal ütközik, akkor meghal, és eltűnik a képernyőről. Jobb oldalt felül az aktuális generáció száma, alatta az egyes madarak fitnessz értékei láthatóak sorszámozva feketével. A madár halálával a fitnessz értékét tartalmazó szöveg színe piros lesz. Ha minden madár

meghal, akkor lefut a tanító algoritmus, majd a generáció száma eggyel nő, és a játék újraindul, de a madarak már máshogy fognak ugrálni.



8. ábra

- Input paraméterek, és a játék indítása. A bemeneti paramétereket csúszkával lehet állítani, majd a gombok segítségével lehet véglegesíteni, majd indítani a játékot. Alaphelyzetben a *csúszkák*, és az *Apply*, illetve a *Reset* gomb aktív, a *Start* pedig inaktív. A felület ezen része részletesen a következő elemeket tartalmazza, az első 4 csúszkával a játék, az utolsó nyolccal pedig az evolúciós algoritmus paramétereit lehet állítani:
 - Gravity (csúszka): a gravitáció értéke, megadható vele, hogy milyen gyorsan csökkenjen a madár magassága. Állítható: 5-10 között, 1-es lépéseközzel.

- Jump height (csúszka): ugrás magassága, megadható vele, hogy egy ugrással mennyit nőjön a madár magassága. Állítható: 5-10 között, 1-es lépésekkel.
- Gap between pipes (csúszka): egy oszloppár közti függőleges távolság. Állítható: 90-140 között, 5-ös lépésekkel.
- Distance between pipes (csúszka): két egymást követő oszloppár közti vízszintes távolság. Állítható: 120-180 között, 10-es lépésekkel.
- Population size (csúszka): populáció mérete, azt állítja, hogy hány madár (egyed) legyen a populációban. Ezek az egyedeket vezérli a neurális háló, illetve rajtuk fut a tanítási algoritmus. llítható: 20-100 között, 5-ös lépésekkel.
- Number of hidden neurons (csúszka): a neurális háló rejtett neuronjainak száma. Állítható: 3-10 között, 1-es lépésekkel.
- Selection rate in % (csúszka): szelekciós ráta, az evolúciós algoritmus pontosan $\left\lfloor \frac{s}{100} * p \right\rfloor$ darab pár közül választja ki a legjobbat szülőnek (s a szelekciós ráta, p a populáció mérete). Állítható 50-100 között 5-ös lépésekkel.
- Deletion rate in % (csúszka): törlési ráta, az evolúciós algoritmus pontosan $\left\lfloor \frac{d}{100} * p \right\rfloor$ darab egyed helyett teljesen újat hoz létre (d a törlési ráta, p a populáció mérete). Állítható 1-10 között 1-es lépésekkel.

- Crossover rate in % (csúszka): rekombinációs ráta, az evolúciós algoritmus a kiválasztott szülők közül pontosan $\left\lfloor \frac{c}{100} * p \right\rfloor$ darab egyedpár kódjait keresztezi. Állítható 40-90 között 5-ös lépésekkel.
- Mutation rate on population % (csúszka): mutációs ráta a populáción, az evolúciós algoritmus pontosan $\left\lfloor \frac{M}{100} * p \right\rfloor$ darab utódot választ ki, ezek fognak mutálódni. Állítható 5-30 között 5-ös lépésekkel.
- Mutation rate on entity % (csúszka): mutációs ráta az egyeden, az evolúciós algoritmus a mutációra kiválasztott egyedek pontosan $\left\lfloor \frac{m}{100} * p \right\rfloor$ darab kódját lecseréli egy véletlenszerű értékre a (0,1) intervallumban. Állítható 5-30 között 5-ös lépésekkel.
- Threshold score (csúszka): a terminálási feltétel pontszáma, azt állítja, hogy az evolúciós algoritmus mekkora fitnessz értéknél álljon le. Állítható 5000-50000 között 5000-es lépésekkel.
- Apply (gomb): nyugtázás, az input paraméterek véglegesítése. Start gomb előfeltétele.
- Start (gomb): játék indítása, a gomb az Apply gomb megnyomása után válik elérhetővé.
- Reset (gomb): az oldal újratöltése, minden beállítás elveszik, és alapértelmezettre áll vissza.

Használat: A modul használatához először tetszőlegesen módosítsuk az input paramétereket a csúszkával (hagyhatjuk az alapértelmezett beállításokon is). Majd az Apply gombbal véglegesítsük (ekkor az Apply, illetve az input paraméterek inaktívvá válnak, a start gomb pedig aktívvá), majd a Start gomb

megnyomásával elindíthatjuk a játék futását, Innentől semmilyen felhasználói input nem szükséges. A Reset gombbal bármikor újratölthetjük a lapot, ekkor minden eddigi beállítás elveszik. Az oldalt elhagyni valamelyik felső gombra kattintva lehet.

3.6. Hálózati kommunikációt használó műveletek

Hálózati kommunikációt csak az oldalak betöltése és a mesterséges intelligenciát használó modul (3.5.3) végez. A program jelenleg csak *localhoston* (a számítógép saját magára mutató neve⁸) az *5000*-es portot használva fut. A kliens a következő műveletekkel létesíthet kapcsolatot a szerverrel:

- *Oldalak betöltése*: a GIU-n felül található gombokra kattintva
- Mesterséges intelligencia modulban:
 - *Apply* gomb megnyomása
 - *Start* gomb megnyomása
 - *Reset* gomb megnyomása
 - *Automatikusan* minden generáció előtt és után
 - *Automatikus* a játék minden időpillanatban

3.7. Adatbázist használó műveletek

Adatbázist csak a mesterséges intelligenciát használó modul (3.5.3) használ a következő műveletekkel.

- *Apply* gomb megnyomása
- *Start* gomb megnyomásával
- *Automatikusan* minden generáció előtt

- *Automatikusan* minden generáció után

3.8. Nagyobb erőforrásokat igénylő műveletek

Nagyobb erőforrásokra csak a mesterséges intelligenciát használó modulnak (3.5.3) lehet szüksége. Minél magasabb értékre állítjuk a *Population size* (több neurális hálót kell lefuttatni, illetve nagyobb halmazon kell elvégezni a tanítást) vagy a *Number of hidden neurons* (nagyobb neurális hálókkal kell vezérelni a madarakat) csúszka értékét, annal több számítást kell végeznie a gépnek, ezáltal lassulhat.

Megjegyzés: a tanítás nem feltétlenül lesz hatékonyabb, ha ezek minél magasabb értékek, de dolgozatom célja nem az optimális kiindulási paraméterek keresése.

3.9. Hibaüzenetek, azok elhárításai

A program használata közben kétféle hibaüzenet jelenhet meg csak a kliens oldalon. Az egyik a hálózati kommunikációra, a másik pedig az adatbázisra vonatkozik.

- Kommunikációs hiba a szerverrel (3.6)
 - Sikertelen oldalbetöltés esetén beépített, a használt böngészőre jellemző hibaüzenetet kapunk, hogy a szerver nem érhető el (Safari-ban *9. ábra*)
 - További műveletek esetén pedig egy *Server side error* (szerver oldal hiba) feliratú HTML alert (figyelmeztetés) típusú felugró ablak jelzi (*10. ábra*), hogy az elküldött üzenetre nem érkezett válasz.
- Adatbázis kapcsolat (3.7) hibája csak az *Apply* gomb megnyomásakor jöhet elő HTML alert formában (*11. ábra*)

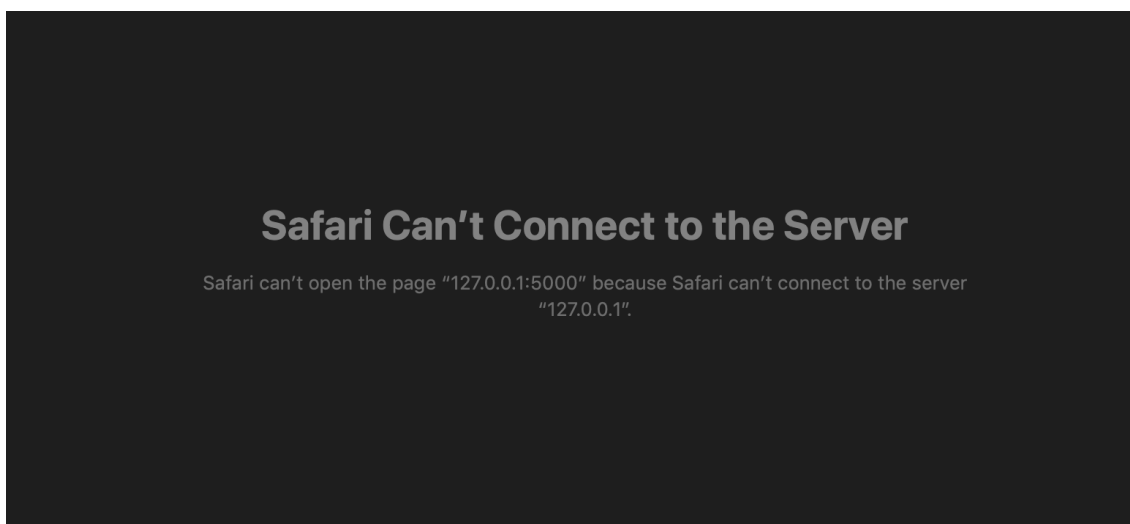
Hibaelhárítás *hálózati kommunikációs* hiba esetén: leggyakoribb hiba, hogy leáll a szerver, ezért ellenőrizzük, hogy fut-e a szerver. Amennyiben leállt, indítsuk el újra a 3.4-be ismertetettek alapján.

Hibaelhárítás *adatbázis kapcsolat* hibája esetén: indítsuk el újra az adatbázis szolgáltatást, amit különböző platformokon a következő parancsokkal tehetők meg konzolos felületről:

- *Windows*: `net start postgresql`
- *Linux*: `systemctl start postgresql`
- *MacOS*: `brew services start postgresql`

3.10. Kapcsolat a fejlesztővel

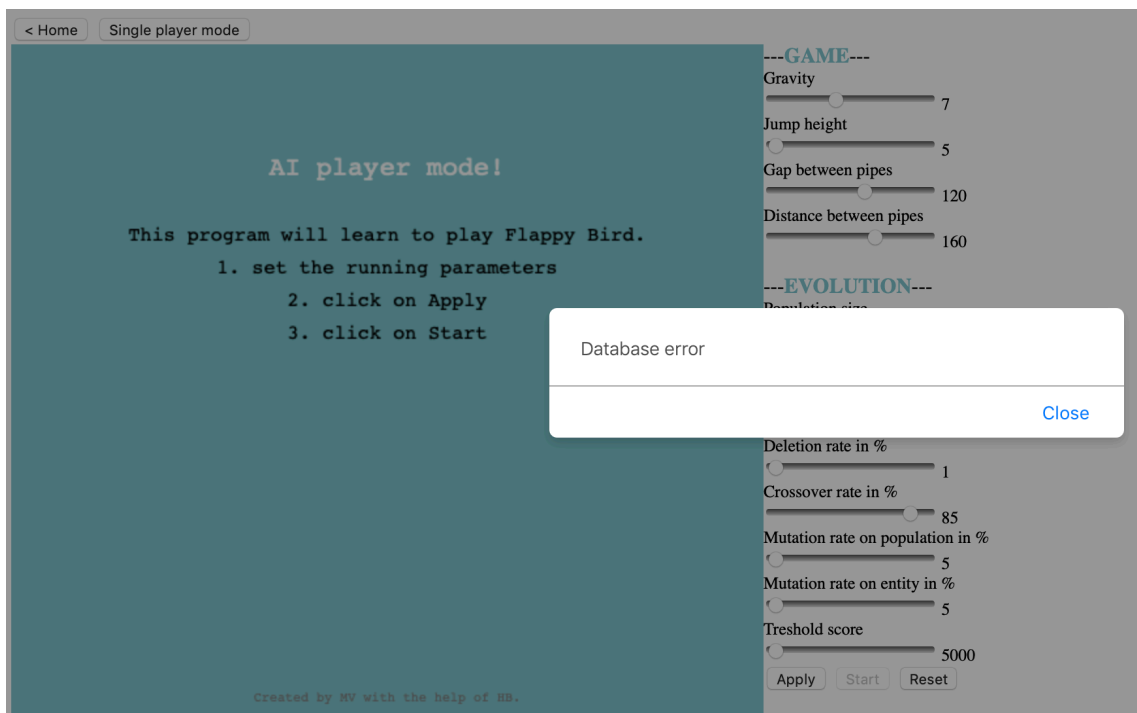
A projekt a <https://github.com/fovecsernyes/fbai> oldalon elérhető. Felmerülő hiba esetén új Issue létrehozásával lehet hibákat jelezni, illetve a fovecsernyes@gmail.com e-mail címen.



9. ábra



10. ábra



11. ábra

4. FEJLESZTŐI DOKUMENTÁCIÓ

A következőkben ismertetjük a *program megoldási tervét, megvalósítását, tesztelését*, illetve a továbbfejlesztési lehetőségeket fejlesztői szemléletet követve. Jelen fejezet az olvasási szokások miatt (az egyes részeknek önmagukban is teljeseeknek kell lennie) redundáns információkat tartalmazhat.

Csak a mesterséges intelligenciát használó (3.5.3) modulra vonatkozóan tartalmaz információt ez a fejezet.

4.1. Általános információk

A kiindulási projekt a <https://github.com/CodeExplainedRepo/FlappyBird-JavaScript> (2018.11.18.) címen található nyílt forráskódú kódbázisból lett forkolva (a szoftverfejlesztési projekt elágaztatása során a fejlesztők veszik a szoftvercsomag forráskódját és megkezdik annak az eredeti fejlesztéstől független továbbfejlesztését, egy új szoftverterméket hozva létre⁹) a fejlesztő engedélyével. Az egyszemélyes modul (3.5.2) lényegében megegyezik ezzel a kóddal. A mesterséges intelligenciát használó modul csak a képeket, illetve játék általános logikáját tartotta meg (például milyen feltételek bekövetkeztekor haljon meg a haljon meg a madár).

A projekt elérhető a <https://github.com/fovecsernyes/fbai> (2018.11.18.) címen, ahol a fejlesztés egész folyamata nyomonkövethető.

4.2. Megoldási terv

A következőben a megoldás részletes tervét (a program futásának tervét, rendszer architektúrájának leírását, adatbázis felépítését, komponensek leírásait, fájlstruktúrát, a komponensek elhelyezését a futási tervben, illetve a felhasználói felület) ismertetem.

4.2.1. A program futásának terve

A modul működési tervét a rendszer architekturájától eltekintve általánosan, a teljesség igénye nélkül ismertetem stuktogrammokkal:

Program működési terve

adatbázis := AdatbázisInicializálás()	
futási_paraméterek := FelhasználóBemenet()	
neurális_hálók := NeurálisHálókatGenerál(futási_paraméterek)	
AdatbázisbaÍr(adatbázis, ciklusok_tábla, játék_id)	
AdatbázisbaÍr(adatbázis, neurális_hálók_tábla, neurális_hálók)	
	Ciklus amíg IGAZ
	neurális_hálók := AdatbázisbólBetölt(adatbázis, neurális_hálók_tábla, neurális_hálók)
	fitnessz := JátékotFuttat(neurási_hálók, futási paraméterek)
	AdatbázisbaÍr(adatbázis, fitnessz_tábla fitnessz)
	neurális_hálók := EvolúciósAlgoritmus(adatbázis, futási_paraméterek)
	AdatbázisbaÍr(neurális_hálók)

Függvény AdatbázisInicializálás()

adatbázis := CsatlakozásAzAdatbázishoz()	
\	Léteznek-e a táblák az adatbázisban /
-	Táblák létrehozása
Return adatbázis	

Függvény FelhasználóiBemenet()

Be: gravitáció, ugrás, populáció, nyílás, távolság, rejtett_neuron, szelekció, törlés, keresztezés, mutáció1, mutáció2, küszöb
futási_paraméterek := Tömb[gravitáció, ... küszöb]
futási_paraméterek[gravitáció] := gravitáció
futási_paraméterek[ugrás] := ugrás
futási_paraméterek[populáció] := populáció
futási_paraméterek[nyílás] := nyílás
futási_paraméterek[távolság] := távolság
futási_paraméterek[rejtett_neuron] := rejtett_neuron
futási_paraméterek[szelekció] := szelekció
futási_paraméterek[törlés] := törlés
futási_paraméterek[keresztezés] := keresztezés
futási_paraméterek[mutáció1] := mutáció1
futási_paraméterek[mutáció2] := mutáció2
futási_paraméterek[küszöb] := küszöb
Return futási_paraméterek

Függvény NeurálisHálókatGenerál(futási_paraméterek)

neurális_hálók := Tömb[1..futási_paraméterek[populáció]]
Ciklus i=1..futási_paraméterek[populáció]
neurális_hálók[i] := random_neurális_háló(3, futási_paraméterek[rejtett_neuron])
Return neurális_hálók

Procedúra Adatbázisbaír(adatbázis, táblanév, adat)

Az adatbázisban található táblanév nevű táblába beszúrja az adatot
--

Függvény AdatbázisbólBetölt(adatbázis, táblanév, adat)

Return az adatbázisban található táblanév nevű táblából kiolvasott adat

Függvény JátékotFutatt(neurális_hálók, futási_paraméterek)

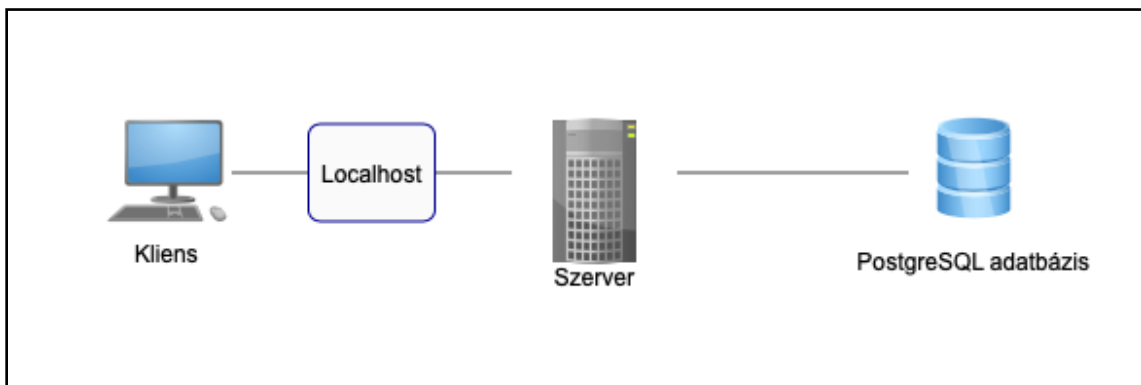
Madár := Rekord{ pozíció=(0,0), fitness=0, él=Igaz }				
Madarak = Madár[1..futási_paraméterek[populáció]]				
Ciklus amíg van élő madár				
	Ciklus 1..futási_paraméterek[populáció]			
		\ Madarak[i].él == Hamis /		
		-	Madarak[i] futtatása neurális háló alapján	
			Madarak[i].pozíció== \ oszlop pozíció /	
			Madarak[i].él := Hamis	Madarak[i]. fitnessz++
			Madarak[i].fitness >= futási_paraméterek[küszöb]	
			Program terminál	-
Return Madarak.fitnessz				

Függvény EvolúciósAlgoritmus(adatbázis, futási_paraméterek)

fitnessz := AdatbázisbólBetölt(adatbázis, fitnessz_tábla, fitnessz)
neurális_hálók := AdatbázisbólBetölt(adatbázis, neurális_hálók_tábla, neurális_hálók)
szülők := szelekció(fitnessz, neurális_hálók, futási_paraméterek)
utódok := keresztezés(szülők, futási_paraméterek)
utódok := mutáció(utódok, futási_paraméterek)
új_neurális_hálók := visszahelyezés(utódok, fitnessz, neurális_hálók)
Return új_neurális_hálók

4.2.2. Rendszer architektúrája

A rendszer architektúrája egy egyszerű kliens-szerver modell (12. ábra), ahol a kliens az alkalmazás frontendjéért (megjelenítés, felhasználói interakció) felel, míg a szerver a backendért (felhasználói interakciók kezelése, kérések kiszolgálása, számítások végzése, adatbáziskezelés). A kommunikáció egy számítógépen belül (localhoston) folyik.



12. ábra

ADATBÁZIS

Alkalmazott technológiák: PostgreSQL-t használok, ami nyílt forráskódú, és ingyenesen használható.

Feladata: a hosszabb távon és több, egymástól független helyen használt adatok tárolása.

BACKEND

Alkalmazott technológiák: Python3 nyelven írt *Flask* nevű framework-öt használ, ami lényegében egy HTTP szerveret valósít meg.

Feladatai:

- A kientől érkező kérések fogadása, feldolgozása, és válasz küldése
 - Oldalak átirányítása

- Adatbázis inicializálása, futási paraméterek tárolása
- Játék indítása
- Neurális hálók betöltése minden generáció elején
- Fitnessz értékek rögzítése minden generáció végén
- Madár neurális hálójának lefuttatása adott pozícióra
- Adatbázisműveletek
 - Adatbázishoz csatlakozás
 - Táblák létrehozása
 - Táblákba való írás
 - Táblákból olvasás
- Neurális hálók kezelése
 - Neurális hálók létrehozása
 - Lefuttatása adott input paraméterekre
- Evolúciós algoritmus

FRONTEND

Alkalmazott technológiák: HTML, illetve JavaScript.

Feladatai:

- HTML a statikus megjelenítésért
 - A gombok, input mezők, rajzvászon pozíciójáért felel
- JavaScript a dinamikus elemekért

- Gombok, csúszkák státuszának változtatása
- Rajzvászonra rajzolás, például a madarak megjelenítése

FRONTEND ÉS BACKEND KÖZTI KOMMUNIKÁCIÓ

A frontend és a backend localhoston kommunikál egymással GET illetve POST kérésekkel. Mindig a frontend kezdeményezi a kommunikációt, ezt a backend fogadja, feldolgozza, majd válaszol rá.

- GET kérések:
 - Oldalak betöltése, hogy milyen címre irányítsanak át a navigáló gombok (felhasználói interakció)
- POST kérések
 - Input paraméterek küldése (felhasználói interakció)
 - Játék indítása (felhasználói interakció)
 - Generáció indítása (automatikusan)
 - Generáció befejezése (automatikusan)
 - Neurális háló kiértékelése adott input paraméterekkel (automatikusan)

4.2.3. Adatbázis felépítése

A program PostgreSQL nevű adatbáziskezelő szolgáltatást használ, amihez csak a backend fér hozzá. Először a táblákat és feladatait ismertetem, majd egy ábrával megadom a táblák mezőit, és a táblák közti kapcsolatot.

CYCLE TÁBLA

A program jelenlegi állapotában nem játszik szerepet, a továbbfejlesztésnél lesz hasznos, ha bevezetjük a ciklusokat. Miután elindul egy generáció, és meghal

minden madár, akkor ne legyen rögtön tanulás és generációlépés, hanem ugyazok az egyedek fussanak más pályán többször, fitnessz értékük adódjon össze. Ezzel elkerülhető az, hogy az olyan egyedeket "kidobjuk", akik csak egy pályán teljesítenek rosszul, de a többin magas fitnessz értéket képesek elérni.

A teszteseteknél lesz fontos a *game_id* (játék azonosítója) mező, illetve a *running_parameters* (futási paraméterek), mert ezeket felhasználva különböző lekérdezések készíthetők az egyes játékokra vonatkozóan.

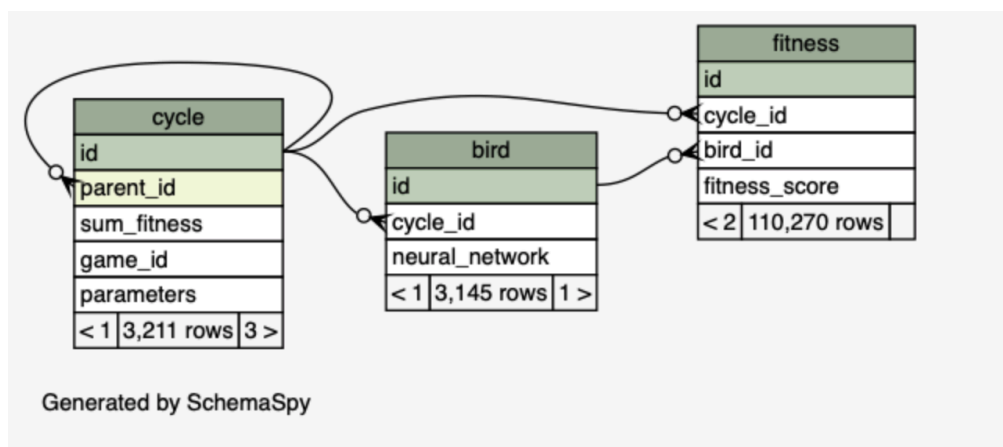
FITNESS TÁBLA

A madarakhoz tartozó fitnessz értékét tárolom benne minden generáció után. Az evolúciós algoritmus használja annak eldöntésére, hogy melyik egyed volt jobb.

BIRD TÁBLA

A madarak adatait tartalmazza: minden madárnak egyedi azonosítója van (sosem törölünk madarat, minél többet játszunk, annál nagyobb lesz a tábla mérete), itt tárolom a neurális hálókát (az evolúciós algoritmus innen olvassa ki), illetve a továbbfejlesztési lehetőségeket szem előtt tartva a ciklus azonosítója is megjelenik.

TÁBLÁK MEZŐI, AZOK KÖZTI KAPCSOLAT (13. ÁBRA)



13. ábra

4.2.4. Komponensek leírásai

BACKEND

- **APPLIKÁCIÓ:** nem tartalmaz osztályt, ezt "fő" fájl, ami kezeli a frontendről jövő kéréseket (a kérés lekezeléseinek metódusai másik fájlokban találhatóak). Ezek a kérések a következők lehetnek:
 - Oldalak betöltése
 - Apply gomb megnyomásának kezelése
 - Start gomb megnyomásának kezelése
 - Generáció előtti kérés kezelése
 - Generáció utáni kérés kezelése
 - Neurális hálóra vonatkozó kérés kezelése
- **KONFIGURÁCIÓ:** adatbázis konfiguráció olvasása fájlból
- **ADATBÁZIS:** az adatbáziskezelés műveleteiért felel, két adattagja van: conn (adatbázis kapcsolat) és az params (adatbázis konfigurációja):
 - Táblák létrehozása
 - Madár beszúrása
 - Ciklus beszúrása
 - Fitnessz beszúrása
 - Madarak lekérdezése
 - Fitnessz lekérdezése
 - Játék azonosítójának lekérdezése

- Neurális háló frissítése
- JÁTÉKSZOLGÁLTATÁS: az APPLIKÁCIÓBAN lévő kérések ennek a fájlnek a metódusait hívják meg. Nincs osztályba szervezve, a következő metódusokat tartalmazza:
 - Apply gomb megnyomásának kezelése: inicializálja az adatbázist, majd beállítja a futási paramétereket.
 - Start gomb megnyomásának kezelése: neurális hálókat generál, majd ezeket beszúrja az adatbázisba a madarak táblába.
 - Generáció előtti kérés kezelése: generáció számát növeli eggyel, majd kiolvassa a neurális hálókat, és betölti, hogy a madarak használni tudják.
 - Generáció utáni kérés kezelése: beszúrja az adatbázisba a fitnessz értékeket, majd lefuttatja az evolúciós algoritmust
 - Neurális hálóra vonatkozó kérés kezelése: minden élő madárra lefuttatja az aktuális pozíciójára a neurális hálóját.
- EVOLÚCIÓS ALGORITMUS: az evolúciós algoritmust tartalmazza:
 - Kiolvassa az adatbázisból a madarak fitnessz értékét és a hozzá tartozó neurális hálókat.
 - Szelektál.
 - Keresztez.
 - Mutációt alkalmaz.
 - Visszahelyez.

- Az adatbázisban frissíti a neurális hálókat a visszahelyezett populációra.
- NEURÁLIS HALÓ: egyrétegű neurális hálót tartalmaz, mely alapján a madarak ugrálnak.

FRONTEND

SABLONOK: a HTML fájlokat tartalmazza, a felhasználó ezeket az oldalakt tudja böngészőből betölteni:

- INDEX oldal: a kezdőképernyő, ezt az oldalt tölti be először a felhasználó (nem a mesterséges intelligencia modul része)
- EGYSZEMÉLYES oldal: itt próbálható ki a játék egyszemélyes módban (nem a mesterséges intelligencia modul része)
- MESTERSÉGES INTELLIGENCIÁS oldal: itt található a mesterséges intelligencia modul

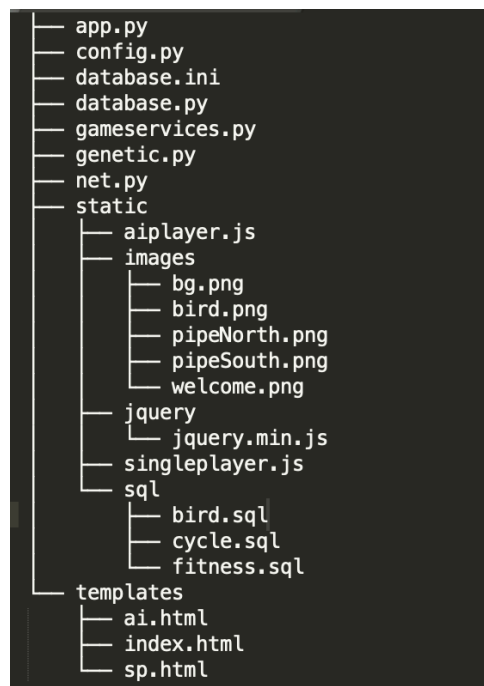
STATIKUS ELEMEEK: a képeket, jQuery fájlt, SQL lekérdezéseket, illetve a frontend scriptjeit tartalmazza. A scriptek a következők:

- EGYSZEMÉLYES script: ez a script tartalmazza az egyszemélyes játék modult (nem a mesterséges intelligencia modul része)
- MESTERSÉGES INTELLIGENCIA script: a mesterséges intelligencia modult tartalmazza:
 - Betölti a képeket.
 - Madarak adatait tartalmazza: pozíció, fitnessz érték, életben van-e még.
 - Oszlopok koordinátái tartalmazza.

- A játékot tartalmazza: betölti a futási paramétereket, illetve futtatja a játékot. Kirajzolja a háttérrel, oszlopokat, élő madarakat, fitnessz értékeket, ellenőrzi, hogy ütközött-e oszloppal valamelyik madár, ha igen akkor megöli. Minden időpillanatban elküldi a backendnek a madarak pozícióit, majd a válasz utasítása szerint kezeli a madarakat (ugorjon vagy sem) majd növeli a fitnessz értékeket, ha él a madár. Ha mindenki meghalt, akkor a fitnessz értékeket elküldi a backendnek.

4.2.5. Fájszerkezet (14. ábra)

A fájlok elrendezésében a Flask framework alapértelmezett mappaszerkezetét használom. A projekt *gyökérkönyvtárában* a Python3 fájlok, a *static* mappában a JavaScript fájlok, a képek, illetve az táblákat létrehozó parancsok vannak. A *templates* mappában pedig a HTML fájlok.



14. ábra

4.2.6. Futási algoritmus, architektúra, komponensek és fájlok kapcsolata

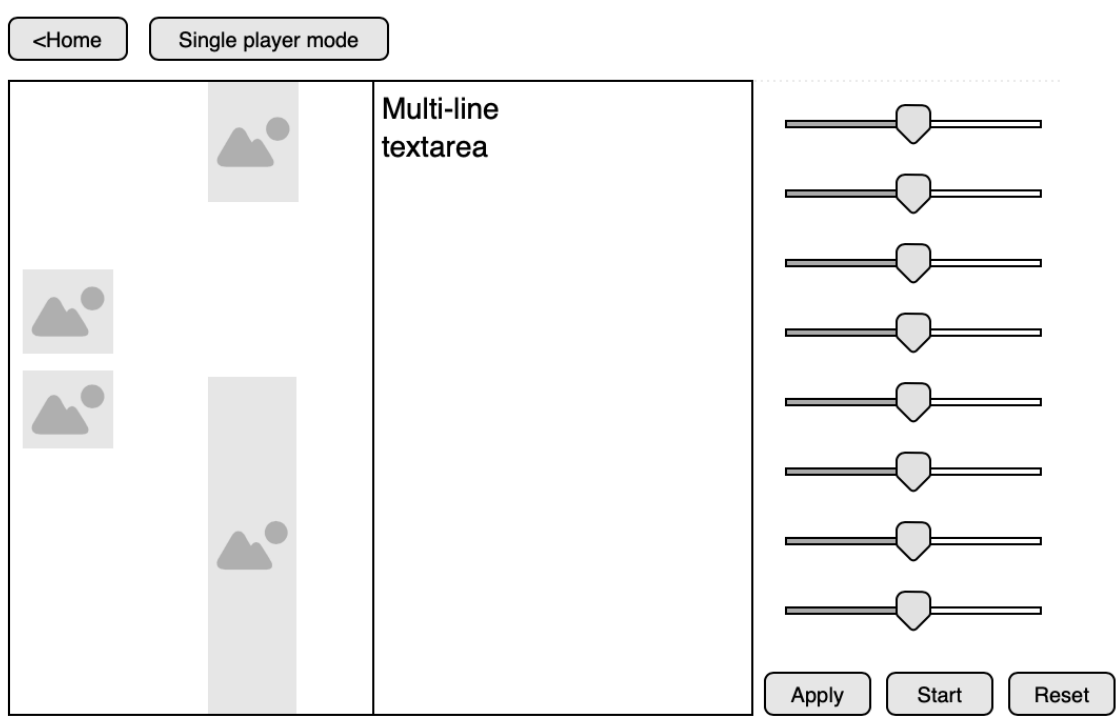
A program futási algoritmusát (4.2.1) architektúráról (4.2.2), komponensekről (4.2.4), fájlszerkezetről (4.2.5) függetlenül adtam meg. A következőkben szeretném megadni ezek kapcsolatát egy táblázattal.

Áttekintés

Komponens neve	Architektúra	Tartalmazó fájl	Futási algoritmus
APPLIKÁCIÓ	Backend	app.py	-
KONFIGURÁCIÓ	Backend	config.py	-
ADATBÁZIS	Backend	database.py	AdatbázisInicializálás() AdatbázisbaÍr() AdatbázisbólBetölt()
JÁTEKSZOLGÁLTATÁS	Backend	gameservices.py	-
EVOLÚCIÓS ALGORITMUS	Backend	genetic.py	EvolúciósAlgoritmus()
NEURÁLIS HÁLÓ	Backend	net.py	NeurálisHálótGenerál()
SABLON	Frontend	ai.html	FelhasználóiBemenet()
STATIKUS ELEMÉK	Frontend	aiplayer.js	JáteketFuttat()

4.2.7. Felhasználói felület terve

Az alkalmazás felületéért a SABLON komponenst felel. Három egyszerű komponensből fog állni: felül a gombok, alatt a rajzvászon, a rajzvászontól jobbra pedig a felhasználói paramétereket állító csúszkák, és az ezeket véglegesítő gombok (14. ábra).



15. ábra

4.3. Megvalósítás

Ebben a fejezetben a szoftver megvalósításának fő lépéseit ismertetem és indoklom azokat a döntéseket, amelyeket a szoftver megalkotása során hoztam.

A megvalósítás összes lépése megtekinthető a <https://github.com/fovecsernyes/fbai/commits/master> (2018.11.18) oldalon.

A program generált doxyGen dokumentációja elérhető a források között a [/documents/html/index.html](#) oldalon

4.3.1. Egyszemélyes játék forkolása

A kiindulási projektet *lemásoltam* a következő címről: <https://github.com/CodeExplainedRepo/FlappyBird-JavaScript> (2018.11.18.). Ez egy egyszemélyes játék, majdnem teljes egészében megegyezik a 3.5.2-ben lévő játékkal.

4.3.2. Refaktorizáció

Kiindulási játék kódja a programozási szakzsargonban úgynevezett spagetti kód (strukturálatlan és nehezen karbantartható forráskód pejoratív megnevezése¹⁰). *Refaktorizálni* kellett ahhoz, hogy a későbbiekben könnyen olvasható és módosítható legyen. Külön osztályt hoztam létre a *madaraknak*, *oszlopoknak*, *képeknek*, és a *játéknak*. Már képes voltam több madarat kirajzoltatni a rajzvásznzon. Minden madárnak implementáltam a *fitnessz* értékét, és egy random generált szám alapján ugrott vagy sem (a későbbiekben ezt kell majd lecserélni neurális hálóra). Beállítottam a madarakra a játék *logikáját*, azaz ha oszloppal ütközik, akkor meghal. Átméreteztem a rajzvásznat és kirajzoltam a fitnessz értékeket. Ezt a fájlt ideiglenesen félreraktam.

4.3.3. Gombok hozzáadása a felülethez

A felületi terv (4.2.7) alapján elkészítettem az input mezőket egy üres HTML fájlban, illetve a gombokat, de ezeket még nem kötöttem be.

4.3.4. Flask¹¹ hozzáadása

Hozzáadtam a *backend* szolgáltatást, majd létrehoztam a frontend és backend közötti *kapcsolatot*: a backend felelt az oldalak betöltéséért, illetve bekötöttem a frontenden található gombokat. Az Apply gombot bekötöttem, így el tudtam küldeni a *futási paramétereket* GET kéréssel a backend-nek.

4.3.5. Adatbázis hozzáadása

Feltelepítettem a PostgreSQL¹² nevű relációsadatbázis-kezelőt, majd a psycopg2¹³ nevű Python3 modullal csatlakoztam az adatbázishoz. Létrehoztam a táblákat a korábban a konzulensemme elkészített tervek alapján, majd az adatbázisműveletek kiszerveztem egy `database.py` nevű fájlba.

4.3.6. POST kommunikáció bevezetése

Az eddig használt GET kéréseket az oldalbetöltéseket leszámítva POST kérésre cseréltem, ekkora a kérés paramétere nem látható az URL mezőben, így biztonságosabb lett. A különböző kérésekhez különböző végpontokat hoztam létre, megoldottam a JSON formátumú adat feldolgozását, majd a Start gombot is bekötöttem, ami csak egy "{respond: start}" JSON üzenetet küldött vissza. Megadtam a gombok, és input paraméterek logikáját (kezdetben az Apply és az input mezők aktívak és a Start nem. Az Apply megnyomásával ez ellentettjére változik).

4.3.7. Refaktorizált játék hozzáadása a frontendhez

A 4.3.2-ben refaktorizált játékot hozzáadtam a frontendhez. Megírtam, hogy ciklusba fusson, azaz ha minden madár meghal, akkor újraindul (ezt itt még a frontend kezeli).

4.3.8. Kérések bevezetése a játék logikájához

Implementáltam, hogy a játék a Start gombra induljon a játék, minden generáció előtt, után, és a játék minden időpillanatában egy aszinkron (megvárja a választ) kérést küldjön a frontend. Minden generáció végén elküldtem az összes fitnessz értéket a backend-nek, majd az adatbázis fitnessz táblájába kiírtam.

4.3.9. Evolúciós algoritmus prototípusának létrehozása

Létrehoztam az evolúciós algoritmus prototípusát, majd egy kiszerveztem a genetic.py nevű fájlba. Ebben az állapotában az egyes függvényhívásokat megírtam, de csak a standard outputra írta ki, hogy meg volt hívva, majd visszatért, tehát még semmit nem csinált.

4.3.10. Neurális hálók bevezetése

Létrehoztam egy egyszerű neurális hálót tartalmazó osztályt a PyTorch¹⁴ nevű Python3 csomag segítségével. A cél az volt, hogy minél egyszerűbb legyen, és a lehető leggyorsabban tudja elvégezni a kiértékeléseket. 3 tetszőleges inputra lebegőpontos számot ad végeredményben, én 0-nál húztam meg a határt. Negatív szám esetén nem ugrik a madár, nem negatív esetén igen. Populáció méretének megfelelő számú neurális hálót generáltam. A frontenden a játék minden időpillanatában összegyűjtöttem a madarak pozícióit (madár y koordinátája, következő oszloppár x , illetve y koordinátája) és egy tömbben elküldtem a *backend*-nek, aki le tudta futtatni a *neurális hálókat* ezekre az inputokra, majd a választ visszakülde. A válasz egy 0-kat vagy 1-eket tartalmazó populáció számával megegyező tömb lett. 0 válasz esetén nem csinált semmit, 1 esetén ugrott.

4.3.11. Neurális hálók tárolása

A neurális hálók egy osztállyal vannak reprezentálva, valahogy adatbázisban szerettem volna tárolni, hogy utána vissza tudjam tölteni, illetve az evolúciós algoritmussal módosítani tudjam. Hosszas keresés után találtam rá a Pickle¹⁵ nevű Python3 csomagra. Ezzel bináris formában ki tudom írni az adatbázisba a neurális hálókat, majd be is tudom tölteni. Megírtam, hogy a generálás után kiírja adatbázisba a hálókat, majd minden generáció elején adatbázisból olvassa ki, és töltse be.

4.3.12. Evolúciós algoritmus implementálása

A 4.3.9-ben létrehozott prototípus funkcióit implementáltam. Kiolvastam a z aktuális neurális hálókat, és a hozzá tartozó fitnessz értékeket az adatbázisból, majd fitnessz értékek alapján sorbarendeztem. A szelekciót, keresztezést, mutációt és visszahelyezést a 2. fejezetben leírtak szerint implementáltam. A

neurális hálókat konvertálni kellett, mivel OrderedDict formátumban nehéz volt módosítani, így NumPy¹⁶ mátrixos reprezentációvá alakítottam. A neurális háló két mátrixból, és 2 vektorból áll (16. ábra).

```
r0 matrix:
[[-0.41734403  0.4247005 -0.38231313]
 [-0.31708774 -0.4491085  0.47553277]
 [ 0.11922002  0.0170604  0.30270827]
 [ 0.41681165 -0.30448186  0.41370636]]

r1 matrix:
[ 0.08892071  0.4028464 -0.35963738 -0.5507271 ]

r2 matrix:
[[-0.23181033  0.07741141 -0.3402595  0.10194528]]

r3 matrix:
[0.07358599]
```

16. ábra

Miután ezeket a mátrixokat az evolúciós algoritmus módosítja, frissítettem az adatbázisban az értéküket az újjal.

4.3.13. Hibakezelés

Implementáltam az adatbáziskapcsolat, illetve a backend elérhetetlenségének hibakezelését (lásd bővebben 3.9)

4.3.14. Kommentezés

DoxyGen kompatibilisan kommenteztem az összes fájlt, hogy lehessen belőle *referenciát* generáltatni.

4.4. Tesztelés

TO BE DONE

4.5. Továbbfejlesztési lehetőségek

CIKLUSOK ALMALMAZÁSA

Ha rögtön minden generáció után lefuttatjuk az evolúciós algoritmust, akkor könnyen megtörténhet, hogy olyan egyedeket is kidobunk, akik más pályán nagyon jól szerepeltek volna, így érdemes lehet bevezetni a ciklusokat. Egy példa a ciklus futására: ha a 3 ciklust állítunk be, akkor tanítás nélkül három alkalommal fut a játék, a fitnessz értékeket összeadjuk, így futtatjuk az evolúciós algoritmust. Ezzel lehetőséget adunk az egyedeknek arra, hogy megmutassák fitnesszüket másik pályán is.

TÖBB KLIENS KISZOLGÁLÁSA

A szoftver jelenleg csak egy klienst tud kiszolgálni, ha rácsatlakozunk egy másik klienssel, akkor kifagy. Meg lehetne csinálni, hogy képes legyen egy időben, nyilván erőforrástól függően, de egynél több klienst kiszolgálni.

PUBLIKÁLÁS A WEBEN

Ha több klienst ki tud már szolgálni, akkor érdemes lehet nyilvánosan elérhetővé tenni a az interneten, hogy bárki ki tudja egyszerűen próbálni.

KÜLÖNBÖZŐ MÓDSZER ALKALMAZÁSA PARAMÉTERKÉNT EVOLÚCIÓS OPERÁTORA

Jelenleg általam választott módszerek vannak beégetve az evolúciós algoritmusba. Ha implementálva lenne több módszer is, például versengéses szelekció helyett rulettkerék-alapú¹⁷, akkor ezt is meg lehetne adni felhasználói inputként, majd összehasonlítani a különböző tanítási formákat.[¶]

5. IRODALOMJEGYZÉK