Gymnázium, Praha 6, Arabská 14

Obor Programování

Ročníková práce

Genetický algoritmus aplikovaný na neuronovu síť

Adam Rubeš

2024

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne Autor

Anotace

Tato ročníková práce se zabývá možností aplikace genetického algoritmu na vytvoření neuronové sítě, která bude schopná hrát jednoduchou počítačovou hru Flappy Bird. Program obsahuje grafické rozhraní, ve kterém je možné sledovat proces učení v grafické reprezentaci hry Flappy bird, a k tomu živě ukazovat jak neuronová síť reaguje na přicházející vstupy. Mimo výše zmíněné nabízí aplikace také možnost upravování parametrů genetického algoritmu, takže si uživatel může vyzkoušet jak které parametry ovlivňují rychlost a stabilitu učení klientů.

Zadání

Cílem ročníkového projektu je pomocí getického algoritmu trénovat neuronovou síť pro hraní počítačové hry flappy bird. Program se bude skládat ze ze dvou částí.

- První část - Program schopný vycvičit Neuronovou síť na hraní hry floppy bird pomocí genetického algoritmu.
- Druhá část - Lightweight grafické rozhraní počítačové hry flappy bird, pro vizuální reprezentaci úspěšnosti neuronových sítí vycvičených první částí.

Obsah

1 Úvod **3**

2 Použité technologie **4**

3 Přehled funkcí a ukázky **5**

3.1 Training 6

3.2 Pretrained Clients 8

3.3 Player game 9

3.4 Hra 10

4 Architektura a mechanizmy **11**

4.1 Main UI 11

4.2 Hra 12

4.2.1 State managment 12

4.2.2 Game backend 12

4.2.3 Herní grafika 13

4.2.4 Napojení neuronové sítě na hru 13

4.3 Genetický algoritmus 14

4.4 Implementace neuronové sítě 16

4.4.1 Struktura 16

5 Teorie **18**

5.1 Umělá neuronová síť 18

5.1.1 Struktura 18

5.1.2 Neuron 19

5.1.3 Aktivační funkce 20

5.2 Normalizace dat a obor hodnot 20

6 Genetický algoritmus **23**

6.1 Pojmy 23

6.2 Průběh 23

7 Závěr **25**

Seznam příloh **27**

1. Úvod

Účelem této práce bylo vyzkoušet aplikaci genetického algoritmu na neuronovou síť. Abych však mohl neuronovou síť naučit řešit problém musel jsem si nejdřív nějaký vymyslet. Z mnoha problémů, které jsou neuronové sítě schopné řešit jsem zvolil hraní hry. Flappy bird jsem tedy zvolil především z následujících důvodů:

1. projekt má mít lehce naučnou povahu, tak jsem od začátku počítal s tím, že s programem se mohou setkat i zástupci neodborné veřejnosti, kteří by se rádi dozvěděli něco málo o neuronových sítích a třeba i o genetických algoritmech. A proto bylo důležité aby koncept problému, který bude neuronová síť řešit, byl snadno uchopitelný a vizualizovatelný pro člověka. Vizualizace je pojatá tak že uživatel může živě pozorovat průběh hry a jak neuronová síť uvnitř reaguje na podněty z prostředí (Například klasifikace obrazu je podstatně hůře pochopitelný a vizualizovatelný problém).

2. důvodem proč jsem zvolil hru je, že genetické algoritmy jsou jako dělané pro vytváření řešení na problémy, kde nemáme přístup ke klasifikovaným datům, na základě kterých bychom síť vycvičili třeba pomocí Back propagation (Zpětné šíření).

Na druhou stranu co se týče genetického algoritmu, zde je naučná část pojata tak, že uživatel má možnost parametrizovat vlastnosti genetického algoritmu a vyzkoušet si jak který parametr ovlivní proces učení.

2. Použité technologie

K tvorbě svého ročníkového projektu jsem využil:

- Programovací jazyk Java
- Vývojové prostředí InteliJ Idea Ultimate
- GUI editor SceneBuilder
- Grafické knihovny JavaFX, JFXtras, ControlsFX, Swing

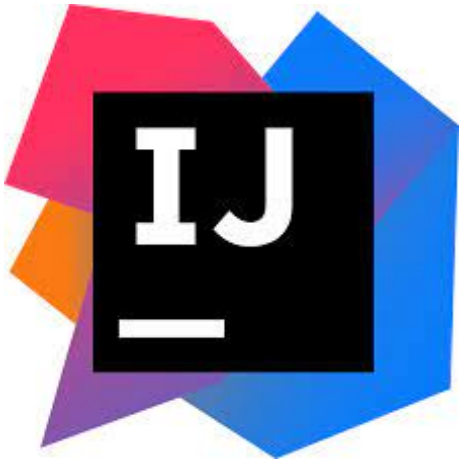

Figure 1: InteliJ Idea

Figure 2: Java

Figure 3: JavaFX

Figure 4: Scene builder

3. Přehled funkcí a ukázky

Program využívá grafické uživatelské rozhraní k interakci s uživatelem. Po spuštění se tedy uživateli zobrazí okno ve kterém je možné (v horní části obrazovky) přepínat mezi několika režimy, a to: "Training" (Učení klientů), "Pretrained Clients" (Hra už uložených klientů) a "Player Game" (FlappyBird pro uživatele).

Ve spodní části okna (Ta je nezávislá na aktuálním režimu) jsou převážně informativní prvky, které uživateli poskytují informace o aktuálním stavu hry (Score - kolik překážek už klient zdolal, Birds - Kolik klientů je aktuálně aktivních, Engine a UI frequency udávají s jakou frekvencí se obnovuje stav herní fyziky a grafické vizualizace).

Směrem doprava od těchto informativních prvků se nachází Game settings (Nastavení hry), zde má uživatel možnost nastavit rozlišení (velikost) okna ve kterém se bude vykreslovat herní prostředí, rychlost běhu hry (Hru je možné zpomalit, aby uživatel mohl lépe pozorovat rozhodování neuronové sítě, a nebo naopak hru zrychlit, aby uživatel nemusel dlouho čekat na výcvik klientů). Hru je také možné spustit v režimu Headless (Hra se spustí na pozadí bez GUI za účelem snížení nároků na výkon)

V pravém dolním rohu je potom umístěn ilustrativní obrázek neuronové sítě, kterou mají klienti v základu přednastavenou (Vzhled sítě se může lišit v závislosti na preferenci uživatele, nicméně je nutné aby měla 4 vstupní a 1 výstupní neuron). U jednotlivých vstupních neuronů jsou umístěny popisky obsahující informaci o tom, jaké hodnoty se do nich vkládají.

3.1. Training

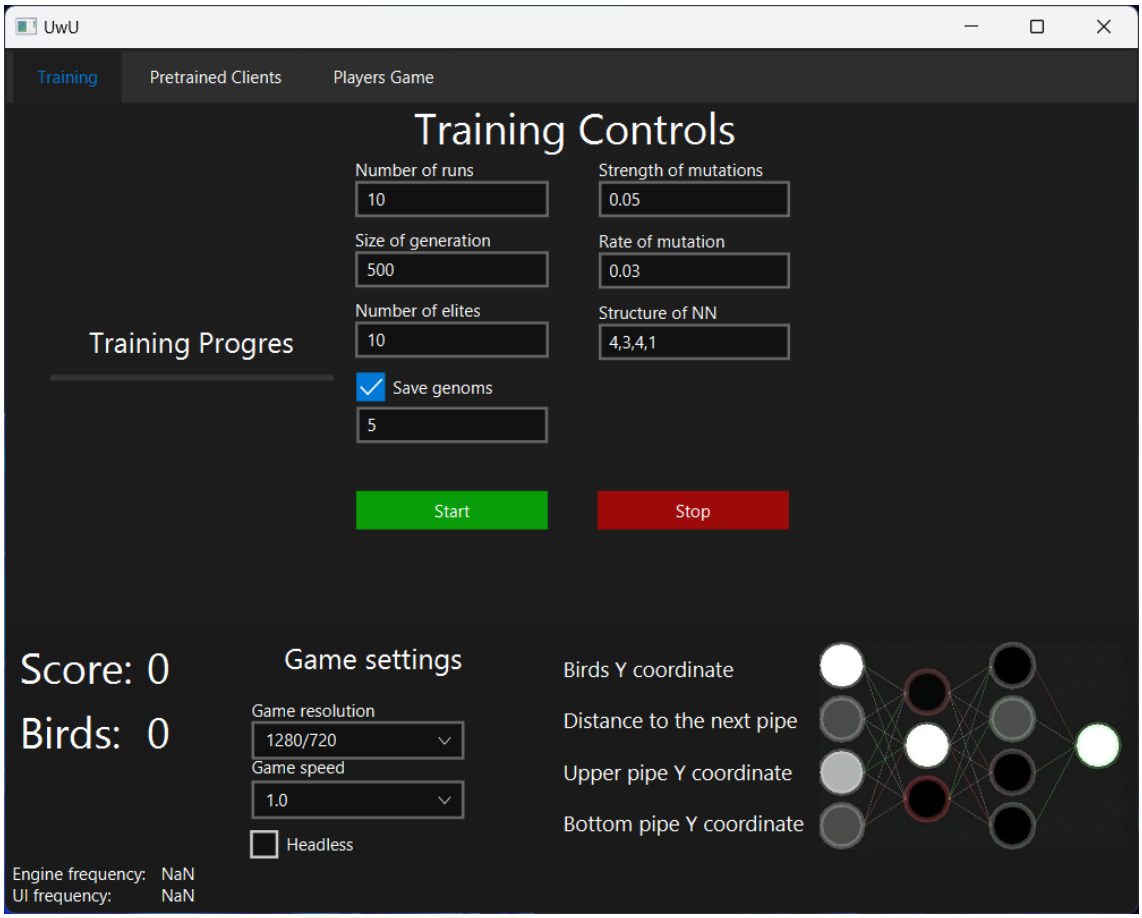

Figure 5: Režim: Training

Režim trénování klientů, kde jsou klienti trénováni, jak bylo zmíněno v úvodu, ve hře flappy bird, je hlavní částí mého programu - splňuje totiž hlavní zadání práce a tedy aplikovat genetický algoritmus na neuronovou síť.

Program se automaticky spustí v režimu Training, kde uživatel může spravovat/sledovat postup tréningu, definovat parametry genetického algoritmu a upravovat vlastnosti všech klientů. V této sekci nechci zacházet příliš do detailu o genetickém algoritmu samotném, a proto vysvětlím jednotlivé parametry jen velmi stručně.

Při pohledu na obrazovku/obrázek zleva, je jako první vidět Training progress (Progres tréningu). Pod ním se nachází indikátor progresu, který se postupně plní modrou barvou podle toho jak se tréning blíží ke konci. Zároveň pokud je spuštěna hra (jakákoliv) tak se pod indikátorem progrese zobrazí ještě animovaný indikátor že zrovna probíhá hra. Tento indikátor je užitečný především v případech že hra

hra běží v režimu headless a bez indikátoru by nemuselo být uživateli zřejmé zda-li program "něco dělá".

Dominantou obrazovky je poté množství pojmenovaných textových polí, která umožňují upravovat parametry genetického algoritmu. Popíšu je tedy z levého horního rohu po sloupcích. Number of runs (Počet her/běhů) Tento parametr určuje kolikrát má proběhnout proces učení. Size of generation (Velikost generace) Tento parametr určuje kolik klientů bude vytvořeno pro každý běh učení. Number of elites (Počet elit) Tento parametr určuje z kolika jedinců se vytvoří nová generace. Další sloupec (přeskakuji "Save genoms" protože nesouvisí s genetickým algoritmem). Strength of mutations (Síla mutací) Tento parametr určuje jak moc budou vybrané geny pozměněny. Rate of mutation (Frekvence mutací) Tento parametr určuje kolik genů bude ovlivněno mutací.

Structure of NN (Struktura neuronové sítě) Tento parametr se netýká genetického algoritmu, nýbrž klienta určuje totiž velikost jednotlivých vrstev neuronové sítě, kterou klienti obsahují - je nutné aby měla vstupní vrstvu velikosti 4 a výstupní vrstvu velikosti 1. viz. Napojení neuronové sítě na hru.

Nyní zpátky k přeskočenému parametru tréningu Save genom (Uložit geny). Tento parametr určuje jestli si uživatel přeje ukládat genomy nejúspěšnějších klientů do počítače, za účelem pozdějšího použití. Zaškrtávací políčko určí jestli se budou geny ukládat (Zaškrtnuto - Ano, Nezaškrtnuto - Ne). V textovém poli pod zaškrtávacím políčkem poté uživatel určí kolik genů si přeje na konci každého běhu uložit. Číslo by nemělo přesáhnout Number of elites (Počet elit) - protože není důvod ukládat neúspěšné klienty. Tento parametr a zaškrtávací políčko je možné měnit i v průběhu tréningu a za chodu hry a program vezme jeho změnu v potaz.

Co se týče parametrizace tréningu klientů je nutné zmínit že, program nijak nehlídá správnost/logičnost vstupních parametrů a tedy je pravděpodobné že pro špatné parametry nebude proces učení fungovat efektivně, jestli vůbec.

Na obrazovce pak už zbývají k popsání pouze dvě tlačítka a to zelené Start (Začít) a červené Stop (Zastavit). Tlačítko start spustí tréning s výše vyplněnými parametry. Tréning pak končí tehdy, když proběhne odpovídající počet běhů, a nebo pokud je explicitně přerušen uživatelem a to stiskem tlačítka Stop.

3.2. Pretrained Clients

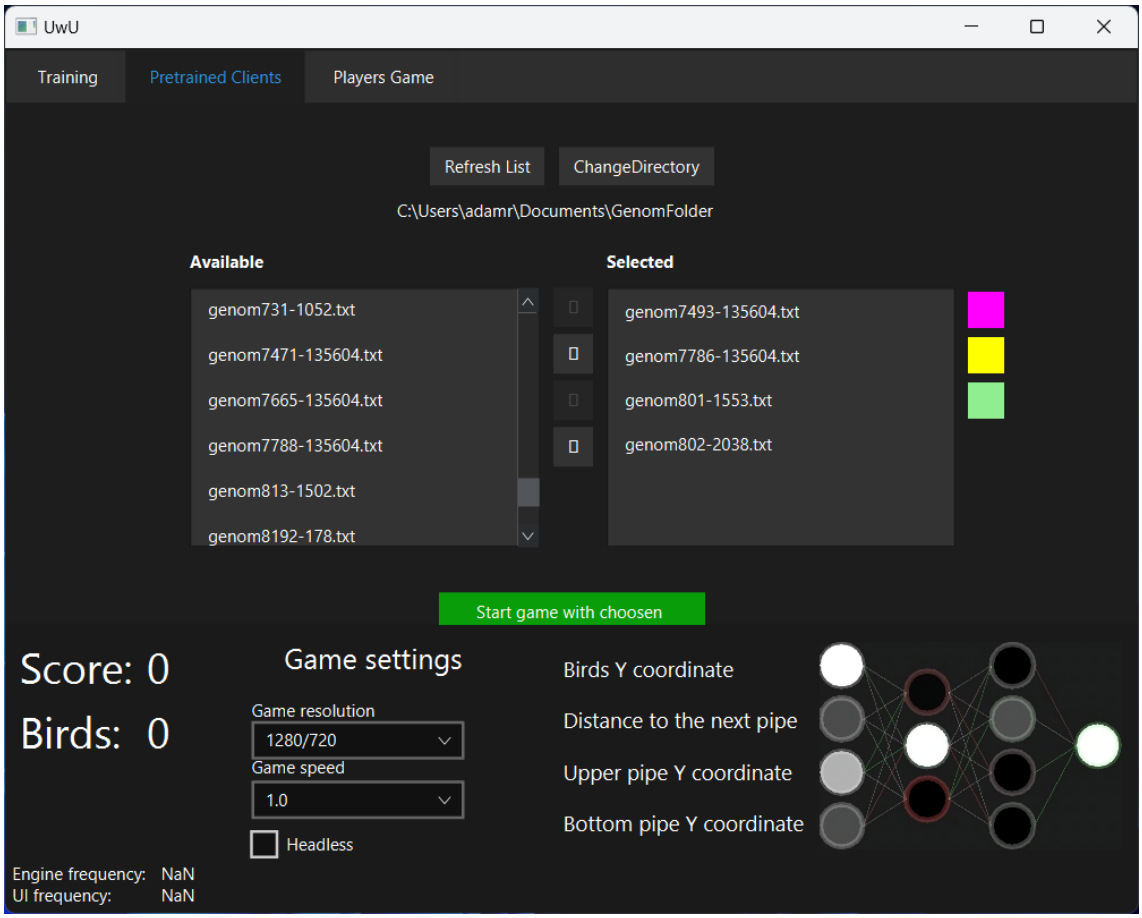

Figure 6: Režim: Pretrained Clients

Režim Pretrained Clients (Předem vycvičení Klienti/Jedinci) je doplňkový režim mé aplikace. Uživatel v tomto režimu může spustit hru s předem vytvořenými klienty s účelem pozorování chování jednotlivců.

Ve středu obrazovky jsou dva seznamy první nadepsaný Available (Dostupné) a druhý Selected (Vybrané). V prvním seznamu jsou vypsané uložené geny v počítači (Program si sám vytvoří a automaticky vybere složku, uživatel nemusí nic konfigurovat). Druhý seznam je v základu prázdný a uživatel do něj může přesunout libovolný počet genomů vybraných klientů, které chce vidět hrát. Uživatel přesune klienta ze seznamu do seznamu dvojitým kliknutím na jeho název.

Prvním třem klientům bude na začátku hry přiřazená barva (Barvy jsou zobrazeny napravo od seznamu Selected a jejich pořadí koresponduje s pořadím vybraných klientů a to tedy - První bude růžovo/fialový, druhý žlutý a třetí světle zelený). Barvy jsou přiřazovány klientům jednak za účelem snadného rozpoznání

který je který a jednak aby bylo poznat, která aktuálně zobrazovaná neuronová síť náleží kterému klientovi.

Pokud je vybráno více klientů než-li barev, poté bude po smrti obarveného klienta barva předána doposud neobarvenému klientu.

3.3. Player game

Jedná se o čistě doplňkový režim hry, který sloužil původně k testování herního enginu. V tomto režimu má uživatel, stejně jako v předchozích dvou režimech, možnost spustit hru Flappy bird, avšak s tím rozdílem že velení nad ptákem převezme uživatel do svých vlastních rukou.

3.4. Hra

Vizualizace hry je mandatorní součást mé aplikace, neboť v zadání práce je zadáno vytvořit "lightweight grafické rozhraní počítačové hry flappy bird...". Zde jsem zadáni dodržel opravdu do detailu, protože vzhled samotné hry je opravdu minimalistický ale i tak plnící svůj účel, a tím jest "vizuální reprezentace úspěšnosti neuronových sítí...".

Rychlé shrnutí pravidel hry: Hra pro hráče končí ve chvíli kdy zemře. (Hráč 1 kostička/ 1klient) Smrt nastane jestliže:

1. Narazí do vrchní hranice obrazovky
2. Narazí do spodní hranice obrazovky
3. Narazí do zeleného obdélníku - překážky (v originální hře často zelená trubka)

(Zmínit se o hitboxech v implementaci hry)

Hra samotná se po spuštění otevře v samostatném okně. Kde pohybující se červeno mordé a jinak zbarvené kostičky reprezentují aktuálně živé klienty hrající hru a zelené obdélníky reprezentují překážky do kterých kostičky nesmí narazit. Každý jeden klient obsahuje neuronovou síť která za něj činí rozhodnutí jestli skočit a nebo neskočit na základě 4 vstupů reprezentujících aktuální polohu a stav prostředí. viz Napojení neuronové sítě na hru. Následně po levé straně okna/obrázku je vidět jedna až tři neuronové sítě, které jsou ohraničeny jednou ze tří již dříve zmíněných barev (růžovo/fialová, žlutá, světle zelená). Tyto barevně ohraničené vizualizace neuronových sítí živě ukazují stav neuronových sítí klientů obsažených ve stejně zabarvených kostičkách.

V případě že hra běží v režimu Training tak hra končí ve chvíli, kdy počet žijících kostiček odpovídá počtu elit, protože nemá smysl aby hra pokračovala dále.

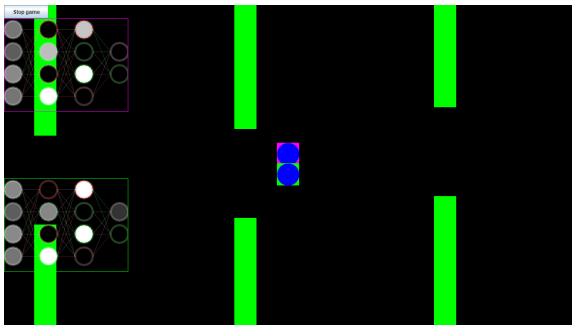

Figure 7: Ukázka hry Pretrained clients (Předem vycvičení Klienti/Jedinci)

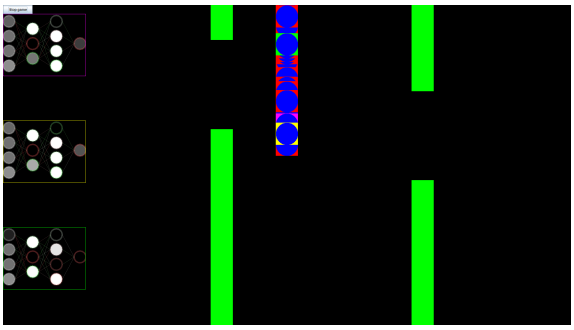

Figure 8: Ukázka hry Training (Tréning klientů/jedinců)

4. Architektura a mechanizmy

V této sekci zajdeme do mírně technických detailů ohledně rozvržení aplikace a implementace jednotlivých funkcí. Bude zde rozepsáno jak funguje hlavní uživatelské rozhraní, herní engine a skoro krok po kroku proces trénování za využití genetického algoritmu.Také zde bude popsaná konkrétní implementace neuronové sítě.

4.1. Main UI

Hlavní uživatelské rozhraní je vyhotoveno za využití grafické knihovny JavaFX a jí rozšiřujících knihoven ControlsFX a JFXtras. (Přidat link na knihovny). ControlsFX přidává mnoho nových/vylepšených kontrolních prvků a validátory vstupních dat. Ze všech nových funkci jsem nakonec využil pouze dvojitý seznam na výběr již vycvičených klientů. JFXtras přidává mimo jiné nové skiny pro nody z JavyFX. Což je jediná funkce kterou jsem z této knihovny využil.

K tvorbě hlavního okna jsem využil funkci JavyFX, kde rozvržení nodů na scéně se načítá z .FXML souboru, který je spřažen s tzv. controllerem (class MainWindowController) ve kterém se odehrává veškerá komunikace mezi frontendem a backendem. Výhoda použití tohoto způsobu je skryta především v tom že je možné si nody ve scéně rozvrhnout pomocí nástroje s grafickým rozhraním jako je například mnou použitý Scene Builder (LINK), který zajistí vytvoření korektního FXML souboru s uživatelem definovaným rozložením nodů. Dle mého názoru se jedná o podstatně snazší způsob designovaní GUI než-li vlastnoručně definovat každý element v programovacím jazyku Java.

Kvůli striktní politice JavyFX ohledně manipulace s proměnnými ve vláknech

JavyFX z jiných vláken jsem musel často využívat knihovní třídy typu IntegerProperty, StringProperty apod. které se dle režije JavyFX aktualizují aktualizují samy. Také bylo nutné používat Platform.runLater() při manipulaci s proměnnými ve vláknu JavyFX.

4.2. Hra

S ohledem na to, že jsem ještě nikdy neprogramoval žádnou hru, tak jsem se rozhodl že k tvorbě hry neužiji žádné knihovny, která by mi zajistila funkční backend hry. Z knihoven jsem využil pouze Java SWING a AWT pro tvorbu grafického rozhraní.

Hra je v mém projektu reprezentovaná třídou Game, která zastřešuje všechny podtřídy zodpovědné za jednotlivé části hry. Třída game poskytuje základní metody pro manipulaci s hrou jako např. spustit hru, zastavit hru, restartovat apod.

4.2.1. State managment

Klíčovou třídou využívanou v herní logice je třída GameState. Tato třída v sobě drží skoro všechny podstatné informace o hře, za všechny uvedmě např. aktuální score, jestli běží hra, počet živých klientů a různé veličiny pro výpočty herní fyziky. Kvůli důležitosti informací obsažených v této třídě jsem se rozhodl že tato třída bude implementova návrhový vzor singleton, aby byla dostupná kdekoliv v programu a nemusel jsem ji všude ručně předávat, a zároveň abych předešel de synchronizaci údajů.

4.2.2. Game backend

Hlavní třídou zajištující backend hry je abstraktní třída GameBackendSuper, která je potomkem třídy Thread. Je důležité zmínit že tato třída obsahuje hlavní herní cyklus, abstraktní metody volané v herním cyklu (popsány později) a link na GraphicalInterface který vykresluje obraz.

Tato třída má dva potomky GameBackendAI a GameBackendPlayer, kde každý implementuje zděděné abstraktní metody tak, aby jednou fungoval herní backend pro počítačem řízené hráče a podruhé pro živého hráče. Toto rozdělení nebylo asi nezbytně nutné ale obecně mi to přijde jako čistější přístup - rozdělit kód do menších lépe spravovatelných bloků.

Mezi abstraktní metody patří metody zodpovědné za zpracování vstupu, aktualizace herní fyziky, kontrola kolizí a počítání bodů. Ze všech zde zmíněných považuji za důležité rozepsat podrobněji pouze jednu a to GameBackendAI.handleInput() která zprostředkovává napojení neuronové sítě na herní logiku v následující sekci.

Co se týče vyhodnocování kolizí, je nutné zmínit že kolize nastane pouze pokud dojde k průniku modrého kruhu uvnitř kostičky se zelenou překážkou, nebo s hranicí obrazovky. Kostička je kolem kroužku vykreslena čistě za účelem lepší vizualizace.

4.2.3. Herní grafika

Grafika hry je zpracovaná v co možná nejjednodušším stylu. Jedná se totiž defacto o plátno na které se na korespondující souřadnice (poskytnuté hrením backendem) vykreslí jeden ze dvou tvarů obdélník a nebo kruh. K tomuto překreslení by mělo docházet minimálně tak třicetkrát za vteřinu.

4.2.4. Napojení neuronové sítě na hru

Před napojením neuronové sítě na hru je třeba provést úvahu, co potřebuje člověk (neuronová síť) aby se mohl správně rozhodnout, jestli skočit a nebo neskočit a pokusit se neuronové síti takové informace předat. Po krátké chvíli přemýšlení jsem došel, stejně jako mnoho jiných lidí na internetu, k závěru že pro správné rozhodnutí je bez pochyby potřeba vědět jak je kostička vysoko (Aby nenarazila do vrchní hranice a do spodní hranice obrazovky), dále že je potřebné vědět jak je kostička daleko od nejbližší následující trubky (Aby hráč věděl kdy je potřeba se začít rovnat do správné polohy) a nakonec je potřeba dalších dvou informací horní hranice otvoru v překážce a dolní hranice otvoru v překážce (Aby hráč věděl kam se má trefit). Všechny vstupní hodnoty jsou normalizovány do intervalu $\langle 0; 1 \rangle$. viz teorie (LINK ADDD).

Při vymýšlení struktury jsem se zaobíral i různými alternativami nyní zvolených vstupů. Zvažoval jsem např. že bych ještě dal neuronové síti další, pátý vstup, a to její rychlost na ose Y. Nakonec jsem ale došel k závěru že se současným modelem fyziky není třeba tento parametr přidávat protože klienti jsou i bez tohoto parametru dostatečně úspěšní. Nebo, a to jsem si uvědomil až s odstupem času, že když je velikost průletu mezi překážkami statická, což v aktuální verzi hry je (i přesto

že engine umožňuje dynamické změny velikosti), není potřeba mít dva parametry určující horní a spodní mez a stačil by jen jeden.

A podobnou úvahu je třeba provést i u vybírání parametrů výstupů neuronové sítě. I přesto že je to v tomto případě poměrně triviální úvaha, tak existuje docela dost možností jak to provést, a většina z nich vede nejspíš ke správnému výsledku. Hodně možností se nám otevře speciálně ve chvíli kdy si uvědomíme že existuje vlastně docela velké možností způsobů jak klasifikovat to že se neuronová síť rozhodla skočit, ona se totiž přizpůsobí skoro všemu.

A tak jsem dospěl k tomu že bude pouze jeden výstupní neuron s aktivační funkcí sigmoid. Aktivační funkci sigmoid jsem však ve výstupní vrstvě brzy nahradil za ReLU z čistě praktického důvodu, a to že při vizualizaci vypadalo ReLU už ze své povahy lépe. Pokud je tedy výstup větší než 0.5 tak je klasifikováno jako skok.

1. input: Y pozice kostičky - normalizováno do $\langle 0; 1 \rangle$
2. input: Vzdálenost kostičky od nejbližší překážky - normalizováno do $\langle 0; 1 \rangle$
3. input: Y pozice vrchní hranice překážky - normalizováno do $\langle 0; 1 \rangle$
4. input: Y pozice spodní hranice překážky - normalizováno do $\langle 0; 1 \rangle$

1. output: Pokud větší než 0.5 potom skok

4.3. Genetický algoritmus

V této sekci se budu věnovat jak krok po kroku probíhá proces tvorby optimálních neuronových sítí pro hraní hry Flappy bird.

Proces tréninku spustí uživatel v grafickém rozhraní, jak bylo zmíněno dříve v textu, uživatel má také možnost upravit celou škálu parametrů genetického algoritmu. Parametry jsou podrobně rozzepsané v teorii (DOPLNIT ODKAZ)

Celý genetický algoritmus je vlastně cyklus, který má tolik opakování jako je dáno parametrem Number of runs. Při prvním kole se vytvoří sada náhodně vygenerovaných neuronových sítí, které jsou součástí třídy Client, která obsahuje i jiné potřebné informace pro genetický algoritmus. Po vytvoření klientů je vytvořena nová instance hry (prostředí ve kterém se budou klienti pohbovat). V průběhu hry jsou žijící klienti oceňováni přímo úměrně uražené vzdálenosti od počátku hry. Hra

končí pokud počet žijících klientů klesne na hodnotu zadanou v parametru Number of elites.

Poté co skončí hodnocení klientů tak začne, tvorba další generace. Tvorba nové generace klientů je kompletně orchestrována ve třídě GeneticAlgorithm. Tato třída má i množství podpůrných tříd reprezentujících některé význačné kroky ve tvorbě nové generace, za účelem rozdělení kodu do menších kusů. Jmenovitě potom: GenomProcessor, ScoreEvaluator (využíván pouze k testování), GenomHybridizer a GenomMutator.

Proces tvorby nové generace začíná tedy zavoláním metody GeneticAlgorithm.evolve(Client[] inputClients), kde jejím vstupem je již ohodnocená generace klientů. Z těchto klientů je nejdříve vybráno 10 nejlepších - elita. (množství je možné změnit parametrem). Z neuronové sítě elity jsou následně přeloženy do genomu a přesunuty do pole genomů připravených pro novou generaci. Následně se vždy náhodně vyberou dva "rodičovské" geny z řad elit a použitím metody uniformCrossover implementované ve třídě GenomHybridizer se vytvoří nový genom, který je následně přesunut také do pole ze kterého bude sestavena nová generace klientů. Tento proces tvorby nových genomů se opakuje "Size of generation" - "Number of elites" krát - doplnit počet genomů na počet původních klientů.

Poté co je vyrobeno dostatečné množství nových genů, podrobíme všechny geny bez vyjímky náhodné mutaci. Mutace genů je implementována třídou GenomMutator. Mutace je náhodná a funguje tak, že se při iteraci přes každou hodnotu obsaženou v genomu položí podmínka: If(mutationRate ¿ náhodné číslo). MutationRate je dán uživatelem a náhodné číslo náleží do intervalu $\langle 0; 1\rangle$ toto číslo by mělo mít v dané intervalu rovnoměrnou distribuci. Pokud je podmínka nepravdivá pokračuje se na další hodnotu genomu. Pokud je podmínka pravdivá dojde k mutaci. Mutace jako taková je potom přičteni náhodného čísla s gausiánskou distribucí vynásobeného parametrem mutation strength. Gausiánskou distribucí v praxi zajišťuje vysoký výskyt hodnot blízko 0 a vzácněji silnější hodnoty - silnější mutace jsou tedy vzácnější.

Poté co jsou všechny mutace provedeny dojde k překladu finálních genetických informací na neuronové sítě a může začít celý proces od začátku. Znovu budou klienti hrát hru, budou hodnoceni atd.

4.4. Implementace neuronové sítě

Tato sekce je věnována popisu implementace neuronové sítě. Neuronovou síť jsem se snažil implementovat s důrazem na OOP (Object Oriented Programing) a modularitu, abych ji mohl do budoucna ještě rozvíjet. Podobně tedy jako u výše popsaného genetického algoritmu je zde hlavní třída Network, která orchestruje práci jiných tříd, které reprezentují jednotlivé složky neuronové sítě jako jsou různé druhy vrstev, neurony a aktivační funkce.

4.4.1. Struktura

Neuronová síť si v sobě drží pole tříd implementujících interface Layer. Jednotlivé položky tohoto pole reprezentují jednotlivé vrstvy neuronové sítě. Využít interface na zastřešení vrstev v neuronové síti jsem se rozhodl především proto že si jsou všechny vrstvy co se týče celkové struktury a veřejných metod podobné a proto abych je všechny mohl uložit do jednoho pole. Tento interfae implementují třídy InputLayer, HiddenLayer, OutputLayer, které jsou naprosto klíčové pro fungování sítě. V budoucnosti bych rád rozšířil dosavadní vrstvy i o jiné druhy vrstev.

V teoretické rovině by pro tento projekt stačilo aby vrstvy měli propojení pouze s vrstvou následující a nikoliv s vrstvou předchozí protože hodnoty zde putují pouze jedním směrem, ale protože je možné že v budoucnosti budu chtít implementovat backpropagation na učení neuronové sítě tak budu potřebovat mít spojení i s předchozí vrstvou.

Co se týče třídy Neuron tak tady to začíná být doopravdy zajímavé, protože zde dochází k veškerým výpočtům, které se odehrávají v neuronové síti. Implementace neuronu je prakticky totožná s teoretickými podklady, neuron totiž obsahuje instanci třídy implementující interface ActivationFUnction, pole vstupních vah, bias, výstupní hodnotu a metodu compute(double[] inputValues) ve které se odehrají kalkulace.

Interface Activation function obsahuje pouze jednu metodu a to compute(double sum). Tento interface v současné chvíli implementují dvě třídy - SigmoidFun a ReLU. Obě dvě reprezentují často používané aktivační funkce.

Ještě je dobré zmínit že třída Network implementuje interface VisualizableFullyConnectedNetwork který umožňuje třídě NetworkVisualiser přistupovat k hodnotám

nutným pro vykreslení sítě.

5. Teorie

5.1. Umělá neuronová síť

Neuronová síť anglicky (Arteficial neural network, Neural network) je výpočetní model využívaný v oblasti strojového učení. Vyznačuje se velmi dobrou schopnosti rozpoznávat složité vzory (patterns) ve vstupních datech, která jsou často obrovských rozměrů.

Mezi přední využití neuronových sítí se řadí rozpoznávání obrazu (strojové vidění), rozpoznávání řeči, optimalizace doporučování obsahu uživatelum na základě nasbíraných informací o uživateli a nyní i experimentálně lékařské diagnozy. Obecně se dá říct že neuronové sítě excelují ve vyhledávání vzorů ve velkém množství dat.

5.1.1. Struktura

Základní stavební/výpočetní jednotkou neuronové sítě, jak už název napovídá, jsou umělé neurony. Tyto neurony jsou shlukovány do vrstev, kde každé dvě sousedící vrstvy jsou navzájem propojené (každý jeden neuron z jedné vrstvy je propojený s každým jedním neuronem vrstvy následující).

Každá spojeni mezi dvěma neurony nese tzv. váhu, která representuje senzitivitu daného neuronu na daný parametr.

Rozlišujeme v základu 3 druhy vrstev - Vstupní vrstva (input layer), skrytá vrstva (hidden layer) a výstupní vrstva (output layer).

- Vstupní vrstva - role této vrstvy je pouze předat vstupní hodnoty do skrytých vrstev. Neprobíhají v ní žádné výpočty, defacto neobsahuje ani neurony.
- Skryté vrstvy - Zde se odehrávají veškeré výpočty neuronové sítě a dá se zde tedy nejvíce experimentovat s různým rozložením neuronů, počtem skrytých vrstev, použitím různých aktivačních funkcí (často se i kombinují), potažmo je zde i možné implementovat konvoluční vrstvy (Convolutional layers).
- Výstupní vrstva - Svým chováním je totožná se skrytou vrstvou s tím rozdílem že výstup jednotlivých neuronů v této vrstvě je označován za výstup neuronové sítě (měl být v rozsahu 0-1) a může být použit jinde v programu.

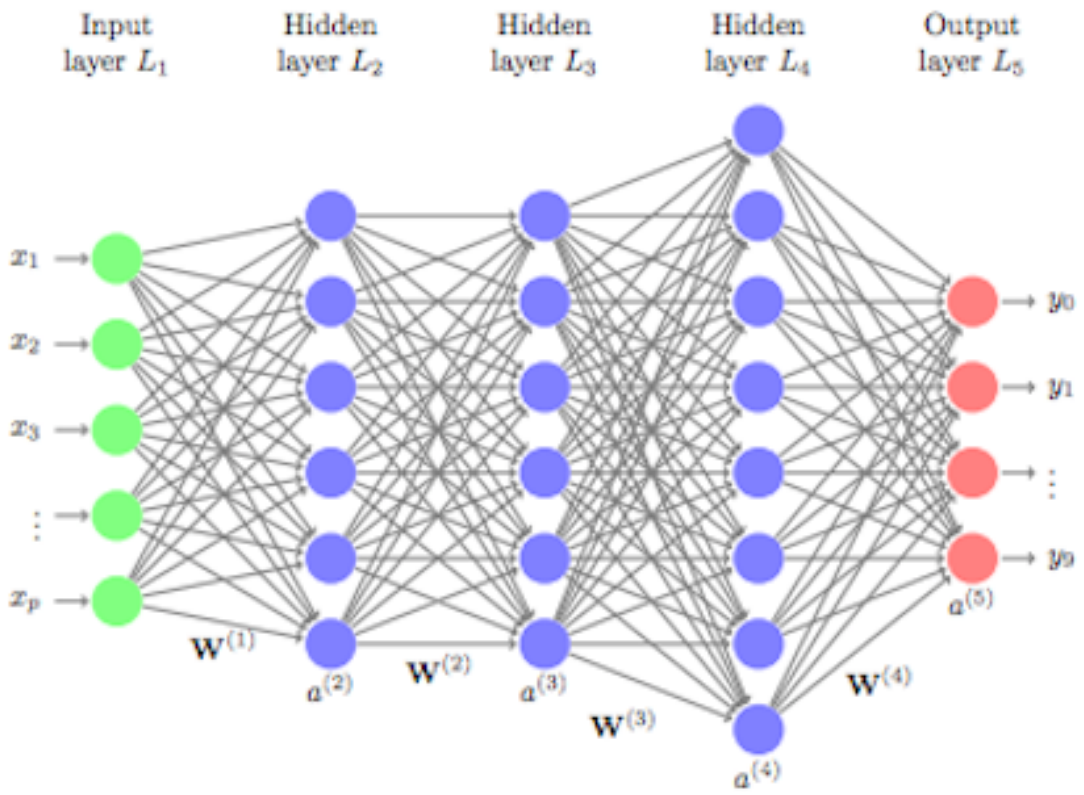

Figure 9: Ilustrace neuronové sítě

5.1.2. Neuron

Princip fungování umělých neuronů, byť s odlišnostmi, vychází z neuronů živých organizmů. Neuron musí mít alespoň 1 vstup (input) a právě jeden výstup (output). Vstupy do neuronu jsou výstupy jiných neuronů kde každý z nich je vynásoben korespondující váhou (weight). Následně neuron sečte všechny vstupy vynásobené váhami a přičte k celkové sumě tzv. bias. Bias je konstanta v každém neuronu (pro každý neuron je zpravidla jiná a může s ní být manipulováno pro dosažení lepších výsledků) díky které dosahují neuronové sítě dramaticky lepších výsledků. Suma je poté předána tzv. aktivační funkci (activation), která vypočítá finální výstup neuronu.

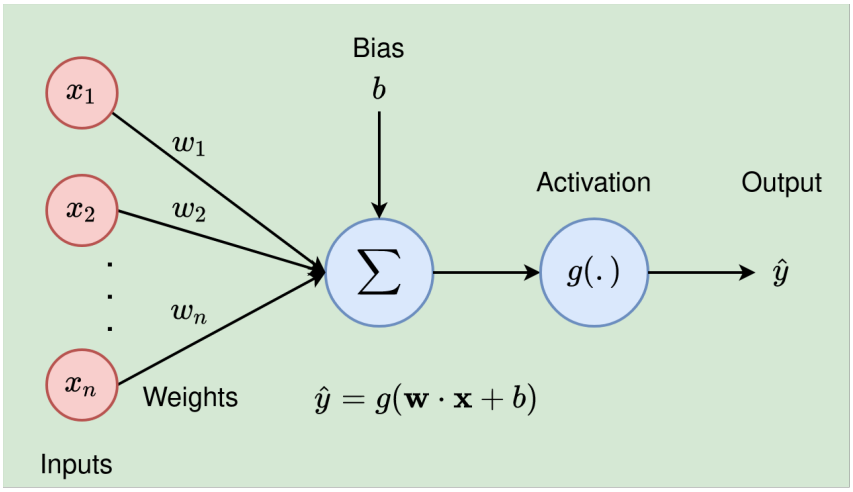

Figure 10: Ilustrace neuronu

5.1.3. Aktivační funkce

Aktivační funkce je klíčová součást neuronu, její vlastnosti totiž silně ovlivňují vlastnosti neuronové sítě. Jako příklad uveďme schopnost rozpoznávat složité vzory ve vstupních datech při klasifikačních úlohách a rychlost a efektivitu učení. Aby tedy bylo dosaženo maximálních schopností neuronové sítě při klasifikaci dat tak se využívají aktivační funkce s nelineárním průběhem. Využití takovýchto funkci umožňuje modelovat složitější než lineární vztahy mezi vstupními daty a výstupem neuronové sítě.

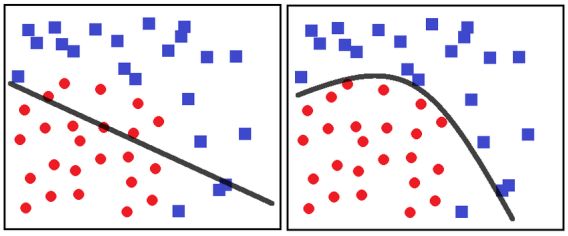

Figure 11: Lineární vs nelineární rozhodovací hranice (decision boundary)

5.2. Normalizace dat a obor hodnot

Již výše v projektu jsem se zmínil, že hodnoty, určené ke vstupu do neuronové sítě, jsou normalizované do intervalu $\langle 0; 1 \rangle$. Takže zde bych nyní rád lehce nastínil motivaci normalizace hodnot před vstupem do neuronové sítě. Hlavním problémem by bylo to, že vstupy, které se měří v řádově jiných hodnotách by zprvu dominovali v

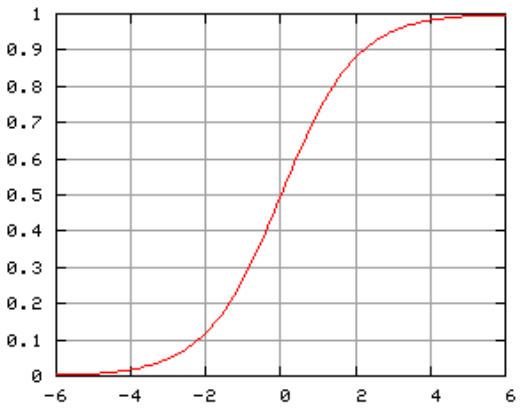

Figure 12: Sigmoid

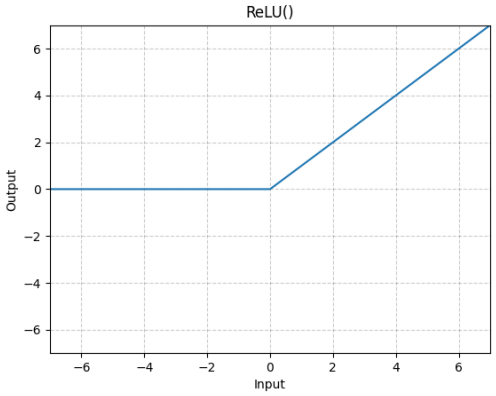

Figure 13: ReLU

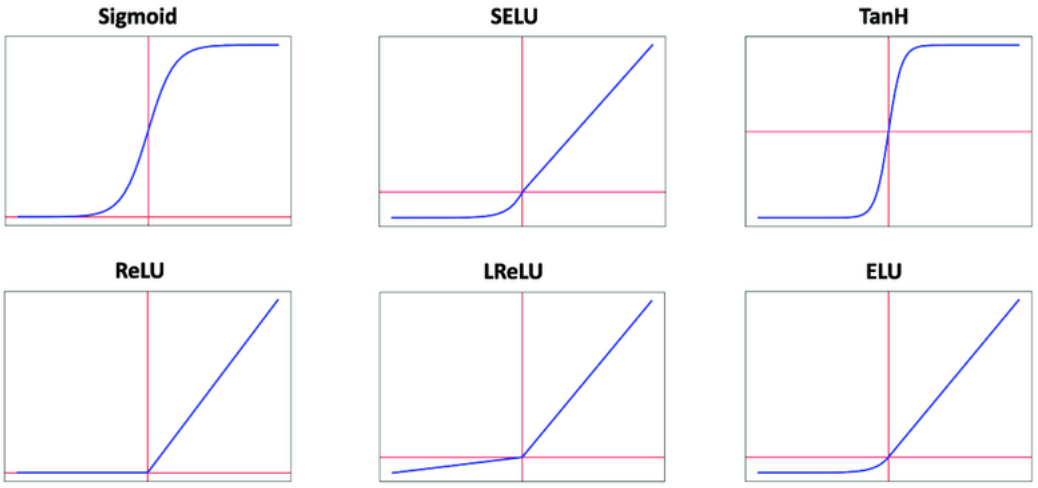

Figure 14: Některé podobné funkce

rozhodovacích procesech neuronové sítě a ta by se potom nerozhodovala správně. V teoretické rovině by se ale neuronová síť časem naučila tyto velké vstupy kompenzovat/normalizovat sama a nakonec by se naučila řešit daný problém tak nebo onak - přiřazováním velmi slabých vah diametrálně silnějším vstupům a velmi silných vah diametrálně slabším vstupům. Toto chování je možné pozorovat, když se rozhodneme použít backpropagation jako učící algoritmus.

Na toto ovšem nelze spolehnout s lokální implementací neuronové sítě a genetického algoritmu. Váhy v neuronové síti jsou totiž v základu inicializovány jako hodnoty v intervalu $\langle -2; 2 \rangle$ což považuji za docela rozumě velké hodnoty. Genetický algoritmus potom prostřednictvím mutací upravuje tyto hodnoty řádově maximálně o setiny, ale ve většině případů o podstatně menší hodnoty. Z toho vyplývá že proces učení, při použití tohoto genetického algoritmu by trval nesmírně dlouho,

protože docílit prostřednictvím takto malých mutací třeba hodnoty -6 by trvalo hrozně dlouho. Samozřejmě by se dalo namítat že by se dala zvýšit síla mutací apod. ale to by negativně ovlivnilo celkový průběh a stabilitu tréningu.

Jinak co se týče oboru hodnot výstupů neuronové sítě, tak ten by se měl rovnat následujícímu intervalu $\langle 0; 1 \rangle$. Tento interval je oborem hodnot aktivační funkce sigmoid, která se zpravidla používá ve výstupních vrstvách. Tyto hodnoty se dají interpretovat jako na kolik procent si je neuronová síť jistá že pro dané vstupy je správným výstupem právě tento neuron.

6. Genetický algoritmus

Genetický algoritmus se využívá na optimalizaci řešení problémů, při aplikaci na neuronovou síť je jeho největší výhoda že na rozdíl od jiných algoritmů na učení neuronových sítí (např. backpropagation), nepotřebuje data set s předem klasifikovanými daty, vůči kterým by porovnával úspěšnost neuronové sítě. Jediné co genetický algoritmus potřebuje je mít opakovaně možnost vyhodnotit celkovou úspěšnost nového řešení. Je inspirovaný přirozeným výběrem, který se odehrává mezi živými organizmy. Či-li v počátku špatná nebo neoptimální řešení se náhodně vyvíjejí krok po kroku k lepšímu a lepšímu řešení.

6.1. Pojmy

V této sekci si dovolím ustanovit význam následujících pojmů:

- Klient - jeden jedinec (jedno řešení), který je hodnocen a vyvíjen. Obsahuje genom, je součástí generace.
- Genom - určitá sekvence znaků, nejčastěji čísel, která nějakým způsobem reprezentuje řešení problému.
- Generace - sada jedinců která je pozorována a hodnocena.
- Mutace - alternace genomu jedince.
- Elita - vybraný počet nejlepších jedinců, kteří budou základem pro další generaci.

6.2. Průběh

Genetický algoritmus tedy krok po kroku funguje tak že:

1. Vytvoření první generace - Vytvoříme první generaci klientů, například náhodně, o dané velikosti.
2. Následně se vypočítá úspěšnost každého jedince (fitness function)

3. Výběr elity a rozmnožení - Vybereme X nejlepších jedinců a zbytku se zbavíme. Z nejlepších vybereme vždy 2 "rodiče" (parent) a z těch pomocí různých funkcí křížení (crossover function) vytvoříme potomka (offspring). Vytvoříme tolik potomků abychom dorovnali původní množství jedinců.

4. Mutace - Následně každého jednoho jedince v populaci podrobíme mutaci. Tyto mutace umožňují představit nové variace/možnosti, které by prostým křížením nemohli vzniknout.

5. A nyní se celý cyklus opakuje od 2. bodu s novou skupinou jedinců.

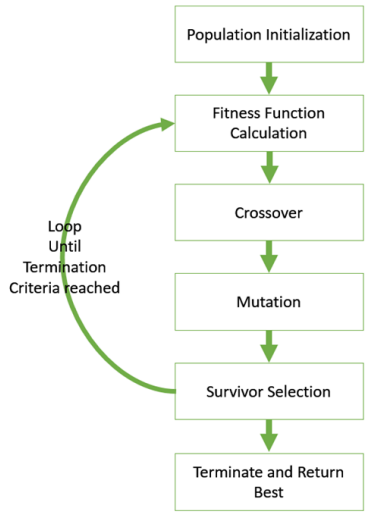

Figure 15: Genetický algoritmus

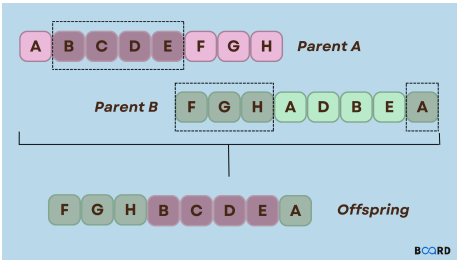

Figure 16: Křížení

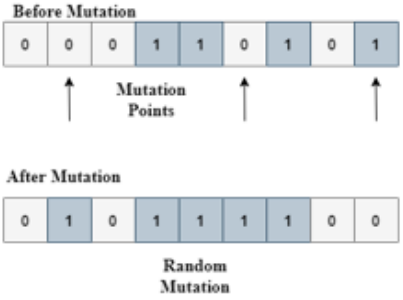

Figure 17: Mutace

7. Závěr

Závěrem bych rád řekl že si myslím že jsem zadáni projektu splnil. Myslím si totiž že se mi podařilo využít genetický algoritmus na tvorbu neuronové sítě, která byla schopná hrát hru flappy bird určitě lépe než. V průběhu práce jsem si rozšířil obzory o mnoho nových poznatků, jak o neuronových sítích, tak třeba i o genetickém algoritmu. Nejsou to však jen teoretické znalosti, kterých jsem při práci nabyl, nýbrž i praktické schopnosti pracovat s programovacím jazykem Java a jeho knihovnami. Obecně tento projekt považuji za cennou zkušenost.

Zdroje budou doplněny

Seznam příloh