Genetické algoritmy v hrách

Daniel Bendík[1]*

Odbor Aplikovaná informatika, FI MUNI, Botanická 68a, 602 00 Brno

Abstrakt: Jedným z kľúčových prvkov, ktoré tvorí hru zábavnou je rovnováha medzi hratelnosťou a obtiažnosťou. Nájisť tuto hranicu býva tažkou úlohou pre herných dizajnérov. Genetické algoritmy (GA) a hybridné použitia evolučných algoritmov s inými technikami sa ukazujú ako možný spôsob prístupu hľadania tejto hranice. Dielom článku je ukázať ich aplikácie v hernom priemysle a priblížiť problém hľadania *flow*[4] stavu hráča.

Kľúčové slová: Genetické algoritmy, neuro-evolučné algoritmy, flow, herný dizajn

1 Úvod

Súčasťou práce herného dizajnéra je vytvoriť náročné herné prostredie pre hráča. To sa dosahuje aj tým, že herný svet je vyvážený. Virtuálny svet ponúka hráčovi výzvu, ktorú považuje za splniteľnú. Úzkym miestom tohoto problému je práve umelá inteligencia protrihráčov reps. hry samotnej. Častokrát býva len naskriptovaná, prípadne fixne pred–dizajnovaná bez možných budúcich variácii. Čo môže viesť k veľmi rozdielnému tempu hrania pre hráčov s rozdielnou úrovňou schopností[5]. Môžeme na to nazerať ako na dynamický systém. Problematiku udržania hráča v rovnovážnom stave popisuje teória tokov(Flow theory)[4]. V kontexte s danou teóriu sa ukazujú GA a ich použitie s inými technológiami(napr. neuronové siete), ako zaujímavá metóda.

2 Teoria tokov (Flow theory)

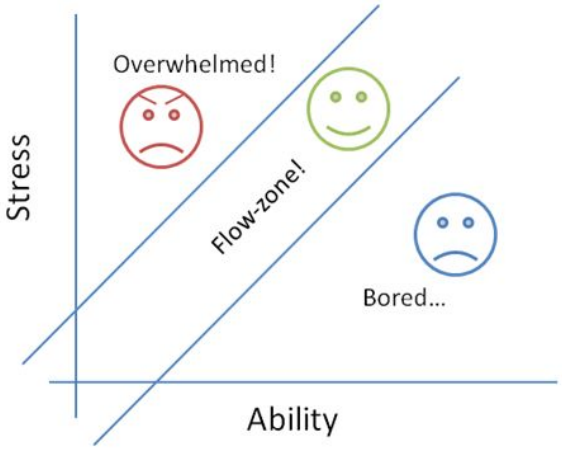
Zadefinovať vzťah medzi hrateľnosťou a obtiažnosťou sa dá pomocou *Flow theory*. Ide o empirické pozorovanie, ktoré sa ukázalo užitočné v kontexte herného dizajnu.

Teóriu založil Dr. Mihaly Csikszentmihalyi, ktorý na základe empirických pozorovaní popísal závislosť medzi výzvou a schopnosťou. Vychádza z myšlienky, že človeka ovládajú emocie, ktoré ovplyvňujú jeho výkon pre danú úlohu. Ak človek prežíva negatívne emócie (úzkosť, nudu), tak s veľkou pravdepodobnosťou nedosiahne zónu toku(Flow-zone).

Flow-zone je stav optimálneho sústredenia, nadšenia, pozitívnej odozovy, ktorý môžme súhrne opísať ako zábavu.

Ak je výzva väčšia ako sú schopnosti jedinca,tak ten upadá do stavu úzkosti a naopak do stavu nudy. Udržovanie dynamickej rovnováhy medzi nimi je kľúčom k zábave[4].

Dizajnéri pozorujú rovnaké javy pri hrách(pozri obrázok 1).



Obr. 1: Upravením konceptu toku dostávame daný diagram.[8]

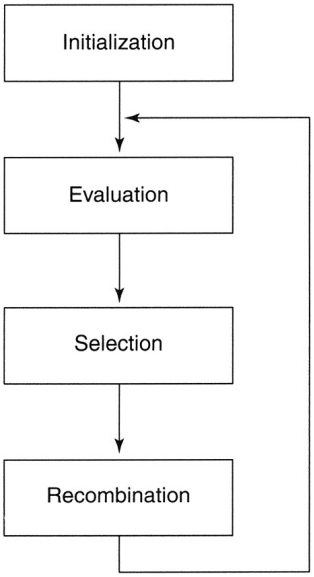
3 Genetické algoritmy

Genetické algoritmy, sú jednoduchou implementáciou Darwinovho princípu prirodzenej selekcie, prežívania najsilnejšieho, ako metódy na prehladávanie stavového priestoru. Z tohoto pohľadu je život ako taky istou konvergenciou k dokonalému prispôsobeniu sa k prostrediu. Každá oddelitelná životná forma predstavuje istý stupeň riešenia. [1]. Charakteristiky najschopnejších riešení sú zakódované do chromozómov. Pomocou kríženia viacerých riešení sa propagujú tieto charakteristiky do dalších generácií[7].

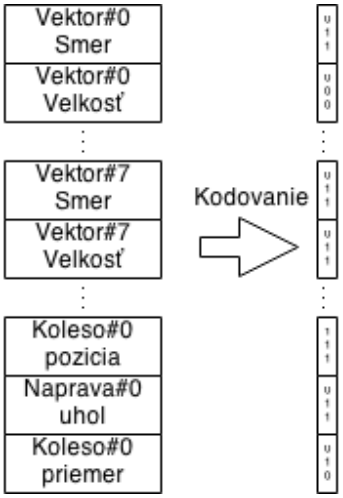
GA pozostáva z troch fundamentálnych krokov (pozri obrázok 2)

*418118@mail.muni.cz

[1] Kvalita riešenia je známa aj pod anglickým slovom *fitness*.



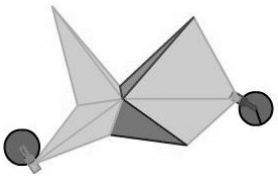
Obr. 2: Proces evolúcie.[7]



Obr. 4: Chromozóm autíčka. Dokopy sa počet premenných môže zvysit až na 16(počet vektorov) + 3 * 8(kolies + náprav) = 40.[1]

3.1 BoxCar2D

Na popis jednotlivých procesov z obrázku 2 použijeme veľmi jednoduchú aplikáciu BoxCar2D(http://boxcar2d.com/index.html). Ide o webovú hračku ktorá sa učí skladať dvojrozmerné autíčka(zložených z trojuholníkov).



Obr. 3: Autíčko z BoxCar2D.[1]

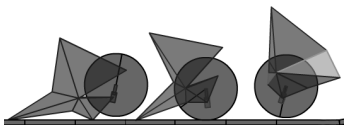
Kódovanie do chromozómov GA pracujú s generáciamia „organizmov“. V našom prípade je organizmom auto. Telo autíčka je tvorené množinou 8 vektorov vychádzajúcich z jedného bodu spojených do trojuholníkov. Kolesá autíčka sú zavesené na náprave. Ich počet variuje od 0 – 8. Pre nápravy sa určuje pozícia v trojuholníku, priemer kolesa a uhol nápravy. Poradie v akom sú atribúty kolies usporiadané korešpodujú s poradím trojuholníkov[1]. (pozri obrázok 3)

Každý atribút je určený metrikou, ktorej hodnota je zakodóvaná do binárneho čísla. Postupnosť atribútov tvorí chromozóm (pozri obrázok 4).

Inicializácia Východiskovým bodom pre algoritmus je počiatočná (prvá) generácia riešení. Typicky, je vygenerovaná nahodnými chromozómami[2]. Pre urýchlenie konvergencie optimálneho riešenia sa môžu využiť aj „zdravé“ chromozómy[3]. V našom príklade sa chromozómy vyberajú náhodne. Dôležitým obmedzením na počiatočnú populáciu je diverzita[5].

Evaluácia V tomto kroku sa snažíme identifikovať najúspešnejších jedincov populácie. To typicky docielime pomocou *fitness* funkcie. Cielom funkcie je ohodnotiť/evaluovať jednotlivcov[7].

V našom prípade *fitness* predstavuje dosiahnutú vzdialenosť v jednotkách (pozri obrázok 5).



Obr. 5: Najlepšie *fitness* skóre dosahuje autičko najviac vpravo.[1]

[2]Odpovedajucích jednej schéme(rovnaky počet a typ parametrov; randomizuje sa len ich poradie).

[3]Predstavujú čiastočné riešenia, známe už pri špecifikacií problému.

Selekcia Na konci každej generácie je potrebné vybrať pár rodičov na vyprodukovanie novej generácie potomkov. V selekcií sa chromozómy vyberajú na základe ich *fitness* skóre. Je doležité, aby tu bol istý faktor náhody a zároveň aby rodičia pochádzali zo širšej množiny riešení (napríklad prvý 4 najlepší). Pokial by do výberu vstupovali len a len najlepší jedinci strácala by sa diverzita populácie. Naopak, čisto náhodny proces by negarantoval vylepšovanie *fitness* skóre[7].

V boxcar2D bola použitá metóda selekcie *Ruleta* (Roulette–Wheel). Tá využíva relatívnych hodnôt z *fitness* skór jedincov[4]. Následne sa vyberie náhodné číslo od 0 – 100. Na základe prislusnosti výsledku do intervalu sa vyberie jedinec (pozri obrázok 6) . Toto sa udeje n/2 krát. Pre zaručenie konvergencie sa z času na čas využuje náhodný výber bez použitia rulety.[1]

Skóre	Pravdepodobnosť	Roulette-Wheel
3.9	35.1%	0-35,1
0.2	1,8%	35,2-36,9
6.0	54.1%	37 až 91
1.0	9,0%	91-100

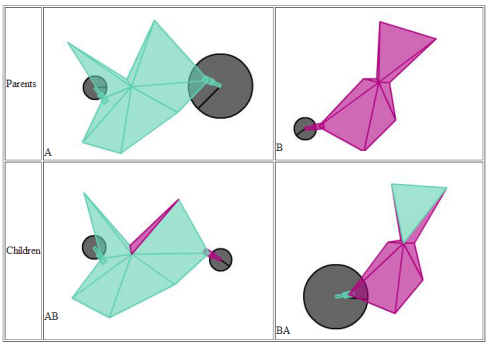
Obr. 6: Príklad pre 2. generáciu o velkosti populácie 4. Vylosované číslo 50 by zodpovedalo rodičovi na 3-ťom riadku.

Rekombinácia(Kríženie) V rekombinácií sú jednotlivé páry chromozómov rekombinované/zkombinované (pozri obrázok 7). Páry chromozómov predstavujú rodičov dalšej generácie potomkov. Do kríženia, z času na čas, vstupuje aj operátor mutácie, čo odpoved konceptu prirodzenej selekcie. Mutácia plní doležitú úlohu pri rozširovani stavového priestoru. Nuž musí sa s ňou narábať opatrne. Napríklad pri nízkej úrovni mutácie môže GA vrátiť ako optimálne riešenie lokálny extrém a pri vysokej úrovni sa degraduje na hladové prehladávnie[7].

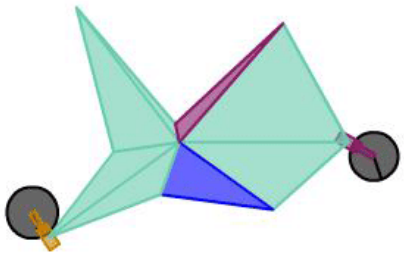
3.2 Aplikácie GA za behu hry

3.2.1 invAIders

InvAIders (pozri obrázok 9) je nezávislá Indie[5] hra, ktorá je z rady remake–ov Space Inviders strielačky

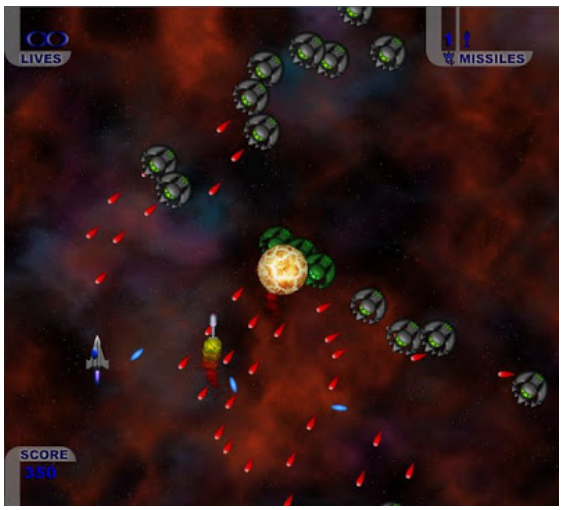


Obr. 7: Rodičia A a B tvoria potomkov AB, BA.[1]



Obr. 8: Mutácia(označená hnedou farbou) nastala na časti chromozómu, kde sa nachádza informácia o pozicii osy kolesa.[1]

z konca 70. rokov minulého storočia. Z časti experimentom uviesť adaptívnu AI na GA. Hra vrhá hrača proti generáciam nepriateľských lodičiek. Keď je porazená jedna vlna GA ohodnotí dané AI lodiek a použije ju na vytvorenie novej generácie nepriateľov.



Obr. 9: Pohľad do hry invAiders[8]

[4] *fitness* skóre každého jedinca sa predelí sumou všetkých *fitness* hodnôt. Z teorie štatistiky to poznáme aj ako relatívne, kumulatívne relatívne početnosti

[5] Nízkonákladové hry vydávané= bez publisher–a

Zameriam sa hlavne na to ako autor pristúpil k implementacií GA a či je AI dostatočná zábava. Vo všeobecnosti sa dá proces rozdeliť do nasledujúcich krokov[8]:

- Definovanie možných riešení.
- Vytvorenie počiatočnej generácie nepriateľov.
- Vyhodnocovanie *fitness* funkcie.
- Vytvorenie potomkov.

Definovanie možných riešeni Každé riešenie pozostávalo z dvoch hlavných prvkov: parametre určujúce pozíciu narodenia, a rozkazy.

Všetci nový nepriatelia prichádzajú z hornej časti obrazovky s rozdielnou rychlostou. Od seba sú oddelený nahodným offsetom, tak aby sa nezrazili. Kdežto rozkazy sú vlastne zoznam správaní. Spravanie sa skladá z pohybu(doprava, doľava, rovno), natáčanie smerom k hráčovi a strielania(smerom k dolnému okraju obrazovky, strielania na hráča). Každé zo správani pre lodičku je spojené s časom, v ktorom je aktívne. Keď tento čas vypršal prešlo sa na iný druh správania. Jeden celý rozkaz pre jednu loď spočíval z približne 40 týchto správaní dokopy usporiadaných do zoznamu[8].

Vytvorenie počiatočnej generácie nepriateľov Pre inicializáciu GA sa použili len dve štartovacie generácie. Prvé generácie v invAideroch su náhodne vygenerované s náhodnymi parametrami počas behu programu. Nové AI sa dajú ľahko rozpoznať podľa zelených kokpitov.

Teoreticky takýto náhodný proces mohol už priamo na začiatku hry priniesť najlepšie riešenie problému hráča[8]. Verím, že cielom takto zvoleného prístupu bolo prekvapiť oponenta a navodiť hráčovi pocit, neistoty pri každom spustení hry.

Vyhodnocovanie *fitness* funkcie Najskôr si je dobré ujasniť čo je problém a čo je riešenie. Problémom v tomto prípade je hráč(jeho chovanie, schopnosť odolavať nepriateľom) a riešením je AI. Tá má za úlohu zničiť daného hráča. To čo v najkratšom čase. Pri kalkulovaní *fitness* funkcie je nutné pracovať s atribútami AI. V tomto prípade predstavujú: počet zabití oponenta danou AI, priemerná prežitia– schopnosť danej lodičky.

Na ich základe sa určuje *fitness* skóre riešenia. Okrem toho každá individuálna AI si nosila zo sebou aj históriu predchodcov. Relatívny výkon bol porovnávaný naprieč jej celou genalógiou predchodcov. Zovelný prístup zaviedol do generovania nasledujucej AI stabilitu[8].

Vytvorenie potomkov Pre propagovanie "dobrých"vlastností AI ďalej sa pri vytvárni hry aplikovali rôzne druhy známych heuristik v GA ako selekcia, mutácia a kríženie.

V **selekcii** sa pre výpočet *fitness* skóre využívala relatívna kumulativna početnosť naprieč celou históriou ohodnotení AI. V konečnom dôsledku to znamenalo, že najnebezpečnejší nepriateľ v danej generacii nemusel byť nutne najnebezpečnejší vo všeobecnosti. Dôvod prečo to takto autor naimplementoval, bolo dosiahnutie určitého stupňa konzistencie medzi rôznorodymi AI.

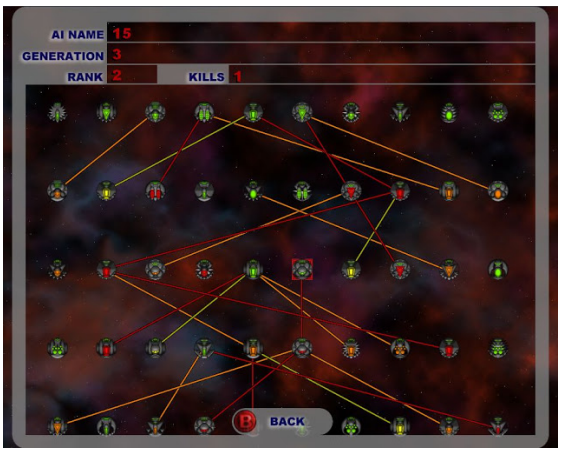
Ako to teda celé prebieha ? V danej generacii sa vyberú traja najlepší kandidati na riešenie. Z nich sa metódou rulety vyberu dvaja na nasledovné kríženie. V hre je ich možno indentifikovať ako lodičky s červeným kokpitom a označenim „Dominant“.

Na prehladanie širsieho spektra AI v priestore riešení autor využil **mutáciu**. Najskôr vygeneroval náhodne zovlené číslo v rozmedzí počtu daných parametrov (pozicie, zoznamom chovaní). Zmenenil daný počet prvkov medzi rodičom a potomkom. Tento proces zahŕňa vytvorenie kópie rodičovej AI a následné zmenenie (zmutovnie) jej niektorých faktorov, ktoré ju definujú. Týmto lodičkam (mutantom) bola priradená oranžova farba kokpitu[6]. V každá vlna obsahuje aspoň troch mutantov, zložených s najlepšie chovajucej AI a najlepších krížencov.[8].

Kríženie zahrňuje proces separovania dvoch idividuálnych lodičiek a z nich vytvorenie kríženca. Chromozómy kríženca sú zložené z príkazov AI s najvyšším skóre v danej vlne a parametrov určujúcich pozíciu narodenia z náhodne vybranej AI. V hre sa označujú žltým kokpitom[8].

Na zachovanie diverzity celú ďalšiu generáciu dopĺňajú náhodne vygenerované AI. Na konci hra ponukne hráčovi výčet všetkých generacií a historie riešení (pozri obrázok 10).

[6] zmiešanie červenej(dominantnej AI) so žltou(AI získanou krížením)



Obr. 10: Genalógia nepriateľských AI.[8]

Obr. 11: Tower of Reus. V ľavo dole je vidiet vstupnú bránu, v pravo hore bázu a kusok nad ňou bielym písmom jej život.[3]

Z pohľadu teorie tokov Autorovi sa v určitom rozahu[7] podarilo implementovať adaptibilnú AI. Tá mení svoje chovanie počas celej hry a je unikátna pre každeho hrača. A teda je tu istý predpoklad udržovania hratelnosti vo *Flow-zóne*. Avšak takto použité GA sa ukazalo byť podľa autora veľmi pomalé. Argumentuje tým, že GA na nájednie optima potrebuje stovky až tisíce generácií. Nuž v hre boli použité maximálne desiatky. Čo neprináša úplne uspokojivé výsledky[8].

3.3 Aplikácie GA v behu dizajnu

3.3.1 Towers of Reus

Towers of Reus (ToR) (pozri obrázok 10)[3] je interakívnou *tower defense*[8] hrou, ktorá využíva komponenty GA v návrhu[9]. Komponenta je v istom zmysle nástroj na vytváranie vyváženej mapy a testovania hratelnosti nadefinovaných typov veží. Súčasťou je aj editor máp.

Narozdiel od predchádzajúcej prípadu spadá ToR do kategórie offline riešení získaných za pomoci GA. Tie dali možnosť analyzovať isté dizajnérske prvky pri vývoji levelov. To bola aj motivácia vývojarov. Pri tomto druhu hry ma dizajnér úrovni za úlohu hrať jednotlivé levely a haldať pre ne riešenie. Znamená to zodpovedať otázky: Da sa daný level vyhrať? Aké veže sa dajú použiť? A aké nie? Ak to nejede, tak treba prehodnotiť dizajn, buď sily vezí, typ vezí alebo zloženie mapy. Z pravidla sa na toto využívajú beta testery, čo sú dalšie naklady na vývoj. V podobe času a peňazí. Cieľom vývojarov bolo zautomatizovať túto analýzu a preto prišli práve s implementáciou GA ako AI, ktorá hrá rolu beta testera.

Generálny proces Prvé čo by sa malo spomenúť skôr ako sa budeme rozpravat o GA použitých v tomto nástroji, je dobré si zadefinovať čo vlastne tvorí jednotlivých jedincov generácie. Generácie sú tvorené celými mapami. Ak sa do nich zavedu aj veže vytvorí sa celok – layout. Mapa pozostáva z gridov – sietí štvorcov. Tieto štvorce môžu byť vyplnené rôznym druhom políčok: tráva (jediný typ, na ktorom sa dá stavať veža), cesta (rovna rohova)– po ktorej prechádzaju nepriatelia, prekážka (skala, strom), miesto kde sa rodia(spawn-uju) nepriatelia, báza – miesto kam smeruju nepriatelia (po ceste)[3].

Inicializácia Vytvorenie štartovacieho layout-u nie je úplne náhodny proces ako pri invAideroch. Je do neho zanesená určitá forma heuristiky hľadajúca optimálne miesto umiestnenia veže. To tak, aby mala v dostrelu cestu.[10] Cieľom je urychliť GA, vylúčením úplne hlúpych riešení. Inak chromozom layout-u tvorí zoznam párov: umiestnenia, typu veže. Až na vyššie spomenuté obmädzenie je výber úplne náhodny[3].

[7]Pre nedostatočné dáta, môžem toto konštatovať len na základe subjetivneho pocitu. Nadobudnúteho hraním hry počas len niekoľkých hodín.

[8]Cieľom hry je zastaviť nepriateľov v prekročení mapy stavaním útočiacich vežičiek a pascí.

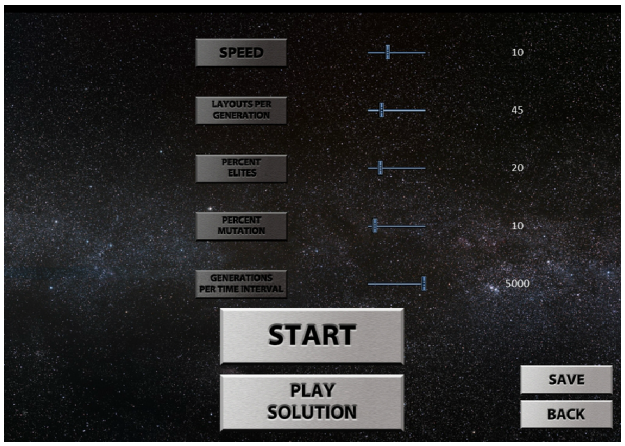
[9]V behu dizajnu

[10]Príklad „zdravšieho“ chromozomu.

Selekcia V štandardnom nastavení je jedna generacia tvorená 45 layoutami. Tieto layouty sú hodnotené *fitness* funkciou, ktorá berie do úvahy hodnoty úspešnosti ako: vyhra/prehra a čas počas, ktorého bola schopná daný stav dosiahnuť. Do selekcie vstupuje elitarstvo. Jedinci sa vyberajú so stanovenou pravdepodobnosťou. Dané skóre je ukladané do textových dokumentov. Layout je uloženy v binárnych súboroch s príponou analysis[3]. Je to jediný možný prístup k medzivýsledkom.

Kríženie a mutácia Kríženie beží na rulete s jedným bodom. To znamena, že každy rodič prispeje novému potomkovi zhruba polkou chromozomu. Do toho vstupuje aj operátor mutácie. Mutujú sa tri veci: pozicia veže, typ veže, vymenenie pozície dvoch veži. Mutáciu je možne nastaviť od 0 - 100 % z empirického hľadiska je najlepšie nastavovať rozmedzie medi 5 - 20% [3].

Rozhranie nástroja je rozdelené do 5 slidrov (pozri obrázok 10). Rýchlosť s akou beži simulácia. Počet layoutov pre jednu generaciu. Percento elitárov použitých v procese kríženia, percento výskytu mutacií a časový limit pre jednu generáciu (pokial vyprší nedopočitavajú sa zostavájúce layouty). Akonále hladanie optimálneho riešenia dobehne do konca. Je možne si prehrať simuláciu. Nasledne na jej základe postaviť analýzu dizjnu.



Obr. 12: Rozhranie vyvojarského nástroja. [3]

Z pohľadu teórie tokov Ako bolo už spomenuté nejde o adaptívnu AI bežiaciu v run-time ale viac–menej o analytický nástroj používaný hlavne v čase návrhu levelov. Zvyčajne štúdia zamestnávajú extra zamestnancov na beta-testing. Berú v úvahu ich osobné pozorvania počas programovania hry, napríklad či nejaka veža nebola priliš silná, alebo nejaký typ nepriateľa neumiera príliš ľahko. Vyvojarmi ToR-u zvolený prístup šetrí čas poskytuje konzistentnejší, nesubjektívný pohľad na vyavažovanie aj dynamicky meniacej sa hry. To sa rovna rýchlejšie získaným datám a flexibilnejším úpravam (patch–om). A tým s oneskorením udržiavať komunitu bližšie k *flow-zone*.

Nevyhody Vhodné riešnie môže pri východzích nastaveniach zabrať aj niekoľko hodín. Na dosiahnutie optima je zhruba nutné analýzu spustiť 2 - 3 krát[3]. Tento typ dizajnovania levelov vyžaduje istú úroveň pochopenia evolučných algoritmov dizajnérom. Napríklad nízko zvolená hodnota mutácie môže viesť k zaseknutiu sa v lokálnom extréme.

4 Hybridné riešenia

4.1 NERO

NERO 2.0 alebo Neuro-Evolving Robotic Operatives je unikátnou hrou. Dovoluje hráčovi konštruovať adaptívnych inteligentných virtuálnych agentov (IVA). Za pomoci využitia nového druhu strojového učenia vyvynutého na Fakulte Informačných techonologií Texaskej Univerzity v Asutine U.S. Cielom projektu je demonštrovať použitelnosť technologie strojového učenia v hre a poskytnúť robustnú platformu pre vývoj, testovanie a výskum chovaní IVA[9].

Okrem hrania hlavného príbehu, hra / aplikácia poskytuje uživatelovi dva hlavné módy:

1. Trénovací mód
2. Bojový mod (Battle mod)

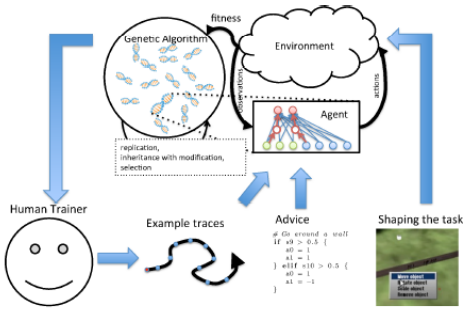
Trénovacia fáza Užívateľ pred samotným testovaním / hraním hry v *battle mode* potrebuje vytrénovať vlastné skupiny robotov s vlastným „mozgom“ beziacim na neuroevolucnych algoritmoch (NE). Robot alebo skupina robotov je vhodených do hry za účelom vykonania hráčom/dizajnérom daných cielov. NE potom hodnoti a modifikuje *mozgy* jedincov na základe ich úspešnosti. Jednym z možnych cielov môže byť priblíženie k nepriateľovi (pozri obrázok 13) alebo útočenie v skupine . Ciele sú z pra-

vidla definované rôznorodnými kombináciami primitív z množiny chovaní: priblíženie, agresivita, udržiavanie sa v skupine, udržiavanie si odstupu, vyhybanie sa strelbe, zotrvanie na mieste, priblíženie sa k vlajke. Hráč jednotlivým primitívam udáva pozitívne (resp. negatívne) reálne hodnoty, ktoré tvoria určitý system odmien (resp. trestov) a teda ide o istú formu strojového učenia s učitelom[9].

Obr. 13: Trénovací mod. Populácia robotov sa snaží dostať k statickému nepriateľovi (vpravo) [2]

Vytvaranie zložitejších taktických chovaní vyžaduje od dizajnéra viac vstupu a premyslenejší plán. Napriklad vytváraním scenarov pre rôzne problemy v strategickych hrach. Toto poskytuje mocny a relatívne pohodlný nástroj na vývoj adaptabilných rozhodovacích plánov pre IVA pohánaných NEAT(Neuro-Evolution of Augmenting Topologies)[9]. Dokonca aj tento funkčny prototyp[11] je schopný pokryť väčšinu hypotetických problémov spojenych dizajnovaním AI v dnesnych a možno aj budúcich realtimeových hrách (S určitou modifikaciou aj tahovych strategickych hrach). Dalšie úpravy by mohli rozšíriť aplikovatelnosť takmer do každeho žánru v hernom priemysle.

Neuro-evolučné algoritmy Ide o využitie dvoch konceptov na najdenie optimálneho riešenie pre daný optimalizačný problém. Prvým konceptom je neurónový mozog IVA a druhým GA. Pomocou GA sa zo sady „mozgov“ vyberajú naschopnejší. V prípade NERO 2.0 ide o metódu učenia IVA so spätnou väzbou a to pomocou učitela (človeka). Odmeňuje úspešných agentov a vice–versa. GA tu posúva do dalšej generacie jedincov najlepšie pasujúcich do schémy požadovaného chovania. Okrem toho hra ešte využíva vylepšenú formu NE– rtNEAT(real-time NEAT) algoritmus[9].



Obr. 14: Diagram fungovania NE s učitelom v NERO 2.0[6]

Väčšina genetických algoritmov používa offline počitanie s pred-špecifikovaným ukočovacím počtom generácií. V rtNEAT sa vyvíja len malá populácia v reálnom čase za účasti (trenéra). Ten môze počas výpočtu meniť atribúty vhodného správania. Predstavuje to adaptabilný system. Okrem toho sa nečaká kým daná generácia evaluje všetky IVA, ale rovno vyhadzuje najslabších jedincov a nahradzuje ich novými. Simulácie ukazujú, že tento prístup vie produkovať vhodné správania i pre malé populácie o 30 jednotkách robotov[9].

5 Záver

5.1 Výhody

Prečo využívať GA v hrách ? Zda sa, že v dnešnom svete počítačových hier je AI veľmi nie dobre preskúmanou zložkou. I tie najväčšie tituly častokrát majú pozlátko v podobe dobrej grafiky, ktorá zaujme hráča na pár hodín, dokým tento menší podvod neprekukne. Hranie sa tak stáva repetitívnou rutinou a v našom kontexte teórie tokov upadá človek do stavu znudenia. Preto si myslím, že viac času by sa malo venovať vytvoreniu znovuhratelnej hry aj za pomoci adaptívnych AI, nepriateľov IVA alebo v podobe dynamickeho game-designu. GA poskytujú priestor na experimntovanie v tejto oblasti. Je to vidieť aj na snahe hier ako sú invAIders. Trochu ďalej sa dostavajú nástroje na analýzu herných mechanizmov. Benchmarking dizajnu: testovanie konflikov determi-

[11] NERO 2.0

nujúcich pravidlá hier/citePAOGD. Veľmi pekne to ukázali tvorci Tower of Reus. Tí tiež demonstrovali, že implementácia GA nieje zdaleka tak časovo náročna[12] ako beta-testing.

5.2 Nevýhody

GA vo všeobecnosti pomaly produkujú optimálne riešenie. Za tento čas pre špecifické optimalizačné problémy iné heuristické algoritmy môžu najisť lepšie riešenia[13][5]. Za behu programu ich využitie častokrát nestíha riešiť problémy. AI sa tak nedokonalo adaptuje hráčovi a ten vybieha z *Flow-zony*. Tiež GA podliehajú veľkej kritike:

- Mechanizmus konvergecie nie je dobre pochopený. GA pracuju dobre empiricky, ale chýba im solídne postavená teoria.
- GA môžu spadnuť do lokálnych miným.

5.3 Budúcnosť

Buducnosť vidím v spájaní rôznych druhov techonologií dokopy[14], ako aj nasadenie parciálnych výsledkov do cloudov (napríklad steam, xBoxLive). Odkial by sa vysledky používali na urýchlovanie výpočtov GA. Hrači by mali tiež možnosť hodnotiť tieto riešenia. V istom zmysle substituovať samotnu evaluacnu funkciu hráčom.

Dalším dobrým príkladom spojenia dvoch technologií sú NE, špeciálne rtNEAT. Jeho devízu vidím v riešení problemu pracnej tvorby behaviorálnych stromov. To sa dnes robí pre taktické chovania, len na základe skúsenosti dizajnera úmelych inteligencií. Takýto proces vývoja zahŕňa v sebe aj ich testovanie a zavedenie do prototypu hry. NERO 2.0 poskytuje celý sandbox[15] na ich tvorbu.

Literatúra

[1] Boxcar 2d. `http://boxcar2d.com/about.html`.

[2] Nero 2,0. http://nerogame.org/.

[3] Heath Wickman Vincent Espinoza James Jenkins Dan Glosier Cameron Ferguson, Wendi Kidd. Towers of reus – developing games with genetic algorithms. `http://www.cameronferguson.net/projects/games/towers-of-reus`.

[4] Mihaly Csikszentmihalyi. Ai for game developers. In *AI for Game Developers*. O'Reilly, 2004.

[5] Glenn Seeman David M. Bourg. Ai for game developers. In *AI for Game Developers*. O'Reilly, 2004.

[6] Vinod K. Valsalam Igor V. Karpov, Leif Johnson and Risto Miikkulainen. Evaluation methods for active human-guided neuroevolution in games. In *Evaluation Methods for Active Human-Guided Neuroevolution in Games*. Dept. of Computer Science, The University of Texas at Austin 1616 Guadalupe, Suite 2.408, Austin, TX, 78701 USA.

[7] M. Tim Jones. Al application programming. In *Al Application Programming*. Charles River Media, 2003.

[8] Michael Martin. Using a genetic algorithm to create adaptive enemy ai. `http://gamasutra.com/blogs/MichaelMartin/20110830/8325/Using_a_Genetic_Algorithm_to_Create_Adaptive_Enemy_AI.php`.

[9] Kenneth O. Stanley, Bobby D. Bryant, Igor Karpov, and Risto Miikkulainen. Real-time evolution of neural networks in the nero video game. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006)*, pages 1671–1674, Boston, MA, 2006. Meno Park, CA: AAAI Press.

[12] Tvorba celej hry zabrala nieco cez 8 tyzdnov.

[13] Napriklad A* algoritmus.

[14] GA spolu s komplementárnymi algoritmami ako evolučné programovanie a lineráne programovanie

[15] Pieskovisko nastrojov a materialov s minimalnymi obmedzeiami na uzivatela.