Evoluce chování agentů v 3D prostředí

Jakub Gemrot[1], Rudolf Kadlec[1], Cyril Brom[1] a Petra Vidnerová[2]

[1] Matematicko-Fyzikální fakulta, UK
jakub.gemrot@gmail.com, rudolf.kadlec@gmail.com, brom@ksvi.mff.cuni.cz,
WWW : http://artemis.ms.mff.cuni.cz/pogamut
[2] Ústav Informatiky, AV ČR
petra@cs.cas.cz

Abstrakt *Článek se zabývá evolučním přístupem k optimalizaci pohybu a vysoko-úrovňového rozhodování virtuální postavy v prostředí komerční hry Unreal Tournament 2004 (UT). Představíme infrastrukturu pro optimalizaci chování pro vyhýbání se překážkám pomocí evolučních algoritmů (EA) a funkcionální architekturu, která je vhodná pro optimalizaci vysoko-úrovňového rozhodování pomocí genetického programování (GP). Možnosti obou přístupů jsou demonstrovány na dvou experimentech.*

1 Úvod

Svět počítačových her prochází v posledních letech rapidním vývojem. Hry jsou stále sofistikovanější, realističtější a odehrávají se zpravidla v dynamických a komplexních 3D světech. Na druhou stranu inteligence počítačem řízených postav *botů* se zlepšuje mnohem pomaleji. Tito boti splňují kritéria agentů tak, jak je popsal Wooldridge [12], i když jejich chování je často naskriptované a pevně spjaté s konkrétní prostředím. Takto naskriptovaný bot pak zpravidla nedokáže proaktivně sledovat cíl v jiných prostředích. Z tohoto pohledu představují moderní počítačové hry zajímavou výzvu pro výzkum umělé inteligence. Cílem tohoto článku je představit problém vývoje chování botů, jejich parametrizaci a námi zvolené řešení.

Vývoj a parametrizaci chování botů řešíme pomocí prostředků genetických algoritmů a genetického programování [6]. Pokud budeme na chování bota nahlížet jako na vrstevnatou architekturu, můžeme rozlišit dvě hlavní vrsty. Vrstvu nižší úrovně, která se stará především o navigaci bota v prostředí a vrstu vyšší úrovně, která plánuje provádění akcí. Navigace prostředím zajišťuje bezpečný pohyb bota ve světě a pracuje nad akcemi nízke abstrakce, např. udělej krok, otoč se doprava. Plánování akcí bota sleduje vždy určitý cíl a provádí rozhodnutí typu, kam jít a co tam udělat. Při plánování se již počítá s akcemi vyšší abstrakce, např. jdi do Fredova domu. Tyto dvě vrstvy chování vyvíjíme odděleně a postupně. V naší práci se zatím nezabýváme dalšími aspekty botů, jako jsou reprezentace znalostí či jejich sociálním chování.

Navigace prostředím je optimalizována za pomocí klasických genetických algoritmů s fixní délkou chromozomu. Naproti tomu pro optimalizaci vysokoúrovňového rozhodování jsme použili genetické programování s proměnlivou délkou chromozomu.

V následující kapitole se seznámíme se specifiky zvoleného prostředí hry Unreal Tournament 2004, po té popíšeme příbuzné práce. V páté kapitole budou představeny softwarové nástroje potřebné pro provádění našich experimentů. V šesté kapitole představíme naše modely a výsledky jejich testování. Na závěr provedeme diskusi možného budoucího vývoje.

2 Boti a prostředí hry Unreal Tournament 2004

Zvoleným 3D prostředím je svět ze hry Unreal Tournament 2004 (UT04), který patří do kategorie first person shooters (FPS) her. Hrač vidí svět z pohledu očí svého avatara (first person), může jím procházet, sbírat předměty a pomocí střelných zbraní bojovat s ostatními avatary ve hře (shooter). Kromě avatarů hráčů se ve světě mohou nacházet ještě avataři kontrolováni počítačem (boti), kteří představují umělé protivníky. Hra UT04 pak nabízí několik typů her, které se liší způsobem, jakým hráči (i boti) dostávají vítězné body. V zakladním typu hry Death match je hráč odměňován pouze za zneškodňování nepřátel. Další typy her jako Capture the flag nebo Domination point pak odměňují hráče za kradení vlajek nepřítele resp. za dobití a udržení určitého území ve světě. V tomto článku se zabýváme vývojem chování pouze pro první typ hry Death match.

Důležitým prvkem UT04 je způsob reprezentace prostředí pro potřeby botů. Vzhledem k výpočetní náročnosti algoritmů analýzy 3D prostředí musí být toto prostředí pro boty předzpracováno a reprezentováno tak, aby v něm bylo možné rychle navigovat. Pro tento účel reprezentuje UT04 prostředí jako orientovaný graf navigačních bodů. Místa, která jsou pro bota dosažitelná chůzí či skákáním v prostředí, jsou rovnoměrně pokryta navigačními body. Dva navigační body jsou pak spojeny hranou tehdy, je-li možné se dostat přímým pohybem od jednoho k druhému. Během pohybu po této hraně je občas nutné vyhnout se drobným překážkám, např. rohu budovy, což řešíme odděleným vý-

vojem nižší vrstvy chování bota, která se stará o navigaci prostředním. Součástí navigačního grafu jsou též veškeré předměty, které se ve světě nachází. UT04 také poskytuje algoritmus na hledání nejkratší cesty v tomto grafu.

Každý bot v UT04 je reprezentován 1) svým humanoidním tělem, 2) množinou sensorů, kterými prostředí vníma, 3) množinou akcí, které může vykonávat, 4) mechanismem pro výběr následující akce. Bot má také svůj cíl, sbírat vítězné body, jejichž význam je závislý na typu hry - v našem případě je získává za zneškodňování nepřátel.

Sensory dále můžeme rozdělit na interní a externí.

Interní sensory poskytují botovi informace o jeho těle a inventáři, jsou to: procento zdraví (při poklesu na 0 bot umírá), množství brnění (snižuje účinek nepřátelských zbrání), pozice v prostředí a aktuální rotace, aktuálně zvolená zbraň a seznam všech zbraní spolu s počtem jejich nábojů, které bot má.

Externí sensory poskytují botovi informace o konfiguraci okolního prostředí, jsou to: seznam viditelných hráčů, navigační graf, seznam míst, kde se nachází předměty (zbraně, náboje, brnění a lékárny), informace z raycastingu [10] a množina asynchronních událostí (bot zemřel, bot uslyšel zvuk, přichází střela a další).

Akce jsou: jdi dopředu, zastav se, vyskoč, pliž se, najdi cestu v navigačním grafu, jdi do strany, otoč se o daný úhel, vystřel na určené místo, změn zbraň.

3 Příbuzné práce

Prakticky všechny příklady genetické optimalizace chování postav v prostředí komerčních her spadají do období posledních deseti let. Zaměříme-li se na doménu FPS her můžeme najít dvě hlavní skupiny takovýchto modelů.

První dovolují genetickou optimalizaci téměř všech aspektů botova chování (pohyb, výběr zbraně, rozhodní zda zaútočit či ne, atd.) [3,8]. Tyto modely jsou většinou inspirovány evoluční robotikou a většinou nevyužívají veškerou informaci poskytovanou herním prostředím. Náš přístup naopak využívá všech dostupných informací, které zvolené herní prostředí UT04 nabízí (např. navigační graf).

Druhá skupina používá genetiku pro optimalizaci vybraného, člověkem většinou obtížně parametrizovatelného, chování v jinak z většiny předprogramovaném algoritmu botova rozhodování. Nejčastěji je optimalizováno chování pro bojové situace [11,2,4]. Výhodou těchto modelů je, že využívají symbolickou informaci poskytovanou herním prostředím.

4 Softwarové nástroje

I přesto, že hra UT04 poskytuje vlastní scriptovací jazyk UnrealScript, pro připojení našich botů k hernímu prostředí jsme použili platformu Pogamut[5]. Pogamut umožňuje mimo jiné psaní logiky botů v jazyce Java, pro který existuje široká nabídka volně šiřitelných knihoven. To je nesporná výhoda oproti použití UnrealScriptu, který takovouto podporu nemá. Platforma Pogamut je již několik let vyvíjena na našem pracovišti[3].

Pogamut zjednodušuje připojení botů do herního prostředí UT04 a poskytuje množství sensorických a motorických primitiv, manažer žurnálů a přídavný modul pro IDE Netbeans^{TM}. Platforma je založena na dobře známém protokolu GameBots [1], který rozšiřuje o mnoho nových možností a zpřístupňuje jej pro jazyk Java. Platofrma Pogamut je volně šiřitelná pro nekomerční a nevojenské účely a může být stažena z našich webových stránek[4].

UT04 je realtimové prostředí, a proto samo o sobě není příliš vhodné jako simulátor pro použití genetických algoritmů. Rychlost běhu času sice může být zvýšena, ale hra neobsahuje možnost "běž, jak nejrychleji je to možné". Změna rychlosti simulace může navíc výrazně ovlivnit fyzikální simulátor a tím i výsledky experimentu.

Abychom obešli tuto nevýhodu prostředi UT04, vyvinuli jsme rozšíření platformy Pogamut, které nazýváme Pogamut GRID. Pogamut GRID umožňuje spouštění experimentů na clusteru počítačů a tím mnohonásobně zkracuje čas potřebný pro běh evoluce.

5 Modely pro vývoj a parametrizaci chování botů

Na chování botů nahlížíme jako na vrstevnatou architekturu. V této kapitole představujeme dvě vrstvy. Nejprve vyvíjíme nižší vrstvu, která je zodpovědná za navigaci v UT04, pomocí genetického algoritmu. Následně ve vyšší vrstvě vyvíjíme mechanismus pro výběr akcí pomocí genetického programování.

5.1 Nižší vrstva chování a vyhýbání se překážkám

Jak již bylo řečeno, nižší vrstva chování se stará o navigaci v prostředí. Tato navigace se opírá o navigační graf, který je poskytován UT04, který také poskytuje implementaci algoritmu pro hledání cesty v tomto grafu. Jakmile je cesta v grafu navigačních bodů

[3] AMIS, http://artemis.ms.mff.cuni.cz/

[4] Pogamut, http://artemis.ms.mff.cuni.cz/pogamut/

známá, zbývá jen vyřešit problém pohybu mezi jednotlivými body. Během tohoto pohybu je občas nutné se vyhnout menším překážkám (angl. obstacle avoidance) [9]. Následující kapitola představuje námi navržený model pro vyhýbání se překážkám, který je parametrizován pomocí genetického algoritmu [7].

Jediným způsobem v UT04, jak zjistit přítomnost překážek v některém ze směru od aktuální pozice bota, je použití raycastingu. Raycasting zde má funkci podobnou infračerveným sensorům robota Khepery, které robot používá na detekci překážek kolem něj. Tedy náš model se sestává s několika takovýchto paprsků. Pokud paprsek narazí do překážky, tak vygeneruje sílu, kterou vychýlí bota z jeho aktuálního pohybu. Problémem je, kolik těchto paprsků má model obsahovat, jakým směrem se mají tyto paprsky vrhat a do jaké vzdálenosti a jakou sílu (směr a velikost) mají generovat. Počet paprsků byl experimentálně stanove na tři. Ostatní parametry byly stanoveny za pomocí genetického algoritmu.

Model pracuje tak, že na bota vždy působí konstantní síla směrem k cíli. Bot s frekvencí 5hz používá raycastingu k detekci překážek. Narazí-li některý z paprsků na překážku, tak vygeneruje dodatečnou sílu, která na bota působí následujících 200ms a odchyluje ho od přímého pohybu k cíli. Směr této síly je stanoven pevně svým úhlem. Velikost vygenerované síly je kvadratickou funkcí $f(x) = Ax^2 + Bx$ vzdálenosti překážky od bota. Viz obrázek 1. I když je použit termín síly, tak bot není chápán jako hmotný bot a nepočítá se jeho setrvačnost. Bot běží vždy směrem výsledné síly maximální možnou rychlostí avatara v UT04.

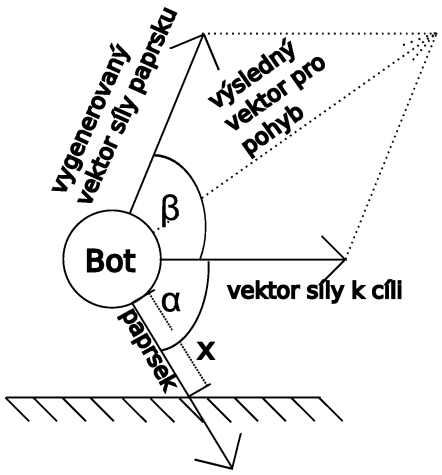
Gen jedince tedy obsahuje těchto 5 parametrů (reálných čísel):

1. úhel paprsku α
2. vzdálenost, do které je paprsek vrhán
3. úhel síly β, kterou paprsek generuje
4. parametry funkce A a B

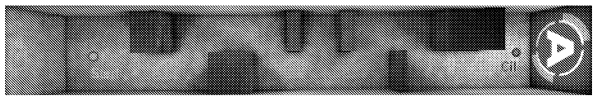
Pro potřeby evoluce parametrů jsme vytvořili následující prostředí (obrázek 2) obsahující dva navigační body, mezi kterými se nachází množství překážek, kterým se však dá jednoduše vyhnout úkrokem doleva či doprava. Bot se vždy objeví na bodu Start a má 15 sekund na dosažení bodu Cíl. Člověkem ovládaný avatar dokáže projít trasu asi za 5 sekund.

Fitness jedince (bota) je pak stanovena podle těchto kritérií:

1. vzdálenost, jak daleko se bot dostal na cestě od startu k cíli — bonus za vzdálenost s
2. jakou průměrnou rychlostí se bot pohybova — bonus za plynulý pohyb v
3. dosažení cíle v časovém limitu —- zbylý čas připočten jako bonus t



Obrázek 1. Pohled na bota shora, na kterém je zobrazen jeden paprsek generující sílu, která bota odpuzuje od stěny, stejně se pak skládá výsledná síla pro více sil. Hodnota x je použita jako vstup funkce f, která udává velikost vektoru síly generovaného paprskem. Vektor síly k cíli a vektor síly paprsku se vektorově sečtou a určí výsledný směr botova pohybu.



Obrázek 2. Prostředí pro evoluci parametrů modelu vyhýbání se překážkám, pohled shora.

4. zda bot narážel do zdí —- postih $a(z)$, kdy z je počet naražení do zdí, $a(0) = 1$, $z \geq 1 : a(z) = 1 + \log_{10}(z)$

Označme maximální průměrnou rychlost bota v_m, vzdálenost mezi startem a cílem s_m a maximální čas t_m. Výsledná fitness (účelová funkce) je pak

$$fitness = \frac{\frac{v}{v_m} + 3(sin(\frac{s}{s_m} * \frac{\pi}{2})) + 5\frac{t_m - t}{t_m}}{a(z)}$$

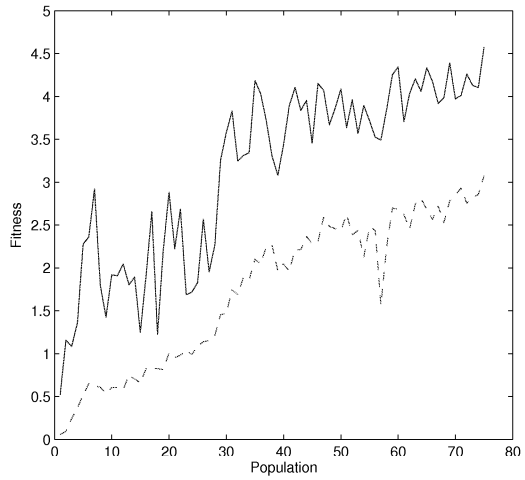
Tento vzorec je výsledkem pozorování jednotlivých běhů genetického algoritmu. Vynecháme-li například bonus za plynulý pohyb, měli boti tendenci do překážky před sebou narazit a pak po ní klouzat dokud se nedostali za její roh. Naopak, když maximální bonus za dosaženou vzdálenost a průměrnou rychlost byl stejný, tak měl genetický algoritmus tendenci uváznout na lokálním maximu, kdy boti běhali co nejrychleji v rohu u stěny druhé překážky.

Další parametry genetického algoritmu:

1. selekce – ruleta
2. mutace – 10% pro jedince, 20% že u mutovaného jedince zmutují parametry toho kterého paprsku

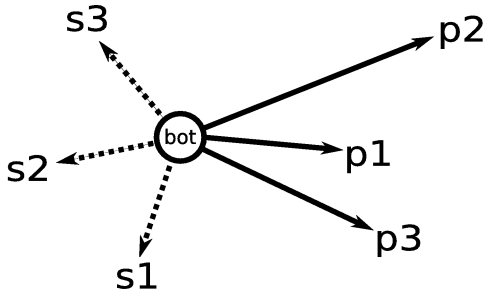
3. křížení – 70%, že se parametry dvou zvolených i-tých paprsků zprůměrují, 30% že se vezme celý i-tý paprsek jednoho z rodičů
4. počet jedinců v populaci – 50
5. počet generací - 75

Následují výsledky genetického algoritmu: výsledná fitness (obrázek 3), rozmístění paprsků nejlepšího jedince 75. generace (obrázek 4), trajektorie cesty nejlepšího jedince 75. generace v mapě (obrázek 5).



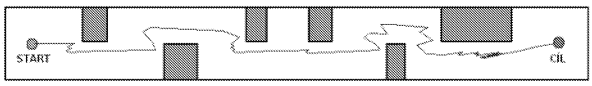
Obrázek 3. Graf fitness evoluce. Plnou čarou je znázorněna fitness nejlepšího jedince v populaci, přerušovanou čarou je znázorněna průměrná fitness celé populace.

Z grafu fitness evoluce (obrázek 3) je patrné, že evoluce neprobíhala hladce a mezi jednotlivými populacemi byly značné rozdíly, nicméně je v něm zřejmá konvergentní tendence. Na obrázku trajektorie cesty nejlepšího jedince 5 vidíme, že zvládl úlohu vyhýbání se překážkám v cestě. Ze stejného obrázku 5 lze také vidět, že pohyb není tak plynulý, jak bychom očekávali. Nabízí se dvě možnosti. Za prvé, buď úplně změnit model vyhýbání se překážkám - například bychom mohli uvažovat bota jako hmotný bot a počítat se setrvačností. Dále bychom mohli více přizpůsobit fitness funkci, která by penalizovala jakýkoli botův pohyb dozadu. Výsledného jedince jsme zatím neotestovali na komplexnějších mapách UT04. Ukázalo se však, že zvolený model lze parametrizovat pomoci genetických algoritmů a je schopen zvládnout problém vyhýbání se překážkám v cestě.



paprsek	α	délka paprsku	β	A	B
1	-11,8°	100	107,4°	-0,05	0,79
2	20,8°	174	164,8°	0,09	-0,31
3	-31,7°	132	-136,9°	0,14	1,11

Obrázek 4. Rozmístění a parametry paprsků nejlepšího bota. P jsou vektory paprsků, S jsou příslušné vektory síly.



Obrázek 5. Trajektorie cesty prostředím nejlepšího bota.

5.2 Vyšší vrstva chování - evoluce mechanismu výběru akcí

Druhou oblastí, na kterou jsme se zaměřili, je vysokoúrovňová optimalizace botova chování. Vyhýbání se překážkám je jen jedním z mnoha úkolů, které musí bot v prostředí řešit. Tím nejzákladnějším problémem je takzvaný problém výběru akcí (action selection problem – ASP). Řešením problému výběru akcí je mechanismus výběru akcí (action selection mechanism – ASM). ASM na základě botova stavu a informací ze sensorů vybírá nejvhodnější možnou akci, t.j. je to funkce ze stavu bota a jeho okolí do množiny akcí.

Námi navržená funkcionální architektura pro popis algoritmu botova chování je inspirována konceptem stromů chování (behavior trees). Stromy chování jsou snadno pochopitelné a v posledních letech se společně s konečnými automaty staly jedním z nejčastěji používaných konceptuálních modelů pro popis chování herních postav. Listy stromu chování odpovídají atomickým chováním, která mohou být přímo vykonána v prostředí herního simulátoru. Odpovídají buď přímo jedné z atomických akcí (vystřel) nebo používají akcí nižší vrstvy chování (jdi k lékárničce). Vnitřní uzly odpovídají arbitrům, kteří rozhodují o vhodnosti akcí navrhovaných jimy podřízenými uzly. Takovýto strom může být vyhodnocen dvěma způsoby: od kořene k lis-

tům, nebo od listů ke kořeni. Naše implementace vyhodnotí nejdříve všechny listy. Výsledkem výpočtu v listu je dvojice [navrhovaná akce, její vhodnost]. Výsledky jsou v dalším kroku postoupeny rodičovským uzlům, které na základě hodnot vhodnosti vyberou tu nejvhodnější akci. Tento postup se opakuje až ke kořeni, akce vybraná kořenem je nakonec provedena v prostředí. Architetura podobná této byla již s úspěchem testována v prostředí simulátoru Robocode [13].

Stromy chování mohou být přímo přeloženy do funkcionálního popisu. Na takto vzniklý program pak můžeme aplikovat metody genetického programování. Genem je v tomto případě samotný strom chování. Genetický operátor křížení zamění v rodičovských stromech dva podstromy stejného typu a operátor mutace nahradí podstrom náhodně vygenerovaným stromem požadovaného typu. Náhodné stromy jsou generovány rekurzivní procedurou jejíž vstupem je požadovaný návratový typ stromu a jeho maximální hloubka. Pro každý typ listu existuje zvláštní randomizační procedura (např. hodnota reálné konstanty je nastavena na náhodné číslo v intervalu $\langle 0, 1 \rangle$).

Program vzniklý překladem stromu chování může obsahovat tři typy funkcí:

- **Funkce chování** —- funkce jejichž návratovým typem je [navrhovaná akce, její vhodnost], tomuto typu říkáme BehResult.
- **Sensorické funkce** — parametrizují funkce chování, jejich návratový typ je buďto reálné číslo normalizované na interval |0,1| (např. distanceTo(Location), heatlh() atd.) nebo typ specifický pro herní prostředí (např. funkce nearestEnemy() vrací ukazatel na objekt reprezentující nejbližšího nepřítele).
- **Matematické funkce** — +, *, $1 - x$, sin, min, max, konstanta.

Teď popíšeme jednotlivé skupiny funkcí podrobněji.

Funkce chování Primární funkce jsou předprogramovaná chování zaměřená na splnění základních úkolů, které může bot chtít v prostředí provést.

- ***stay****(double)* — Zůstaň stát na místě.
- ***wanderAround****(double)* — Procházej po předmětech v mapě.
- ***goTo****(Location, double)* — Jdi na danou lokaci.
- ***pickHealth****(Health, double),* ***pickAmmo****(Ammo, double),* ***pickWeapon****(Weapon, double),* ***pickArmor****(Armor, double)* — Dojdi k nejbližšímu předmětu daného typu a seber jej.
- ***turnLeft****(double)* — Otoč se doleva.
- ***attackPlayer****(Player, double)* — Zaútoč na daného hráče. Toto chování je zodpovědné i za výběr nejvodnější zbraně. S ohledem na vzdálenost nepřítele je vždy vybrána nejvhodnější zbraň s nenulovým počtem nábojů.

Všechny primární funkce mají alespoň jeden parametr typu double, hodnota předávaná v tomto parametru reprezentuje vhodnost akce navrhované chováním. Některé funkce mohou hodnotu vhodnosti ještě modifikovat. Pokud je například chování goTo zavoláno na neexistující lokaci, tak bude vhodnost chování nastavena na nula (goTo(null, 0.27) vrací dvojici [prázdná akce,0]).

Sekundární funkce umožnují vznik složených chování.

- ***sequenceArbiter****(BehResult, BehResult)* — Pokud je vhodnost prvního chování vyžší než 0.1 vrátí akci navrhovaou tímto chováním a její vhodnost, jinak vrací dvojici z druhého chování. Tento arbitr umožňuje vznik sekvenčních chování.
- ***highestActivation****(BehResult, BehResult)* — Porovná vhodnosti obou dvojic, jako výsledek vrátí tu dvojici, jejíž vhodnost je větší. Tento arbitr je vhodný pro vyjadření alternativních řešení daného problému.
- ***suitabilityWeighter****(BehResult, double)* — Vynásobí vhodnost v BehResult dvojici hodnotou druhého parametru.

Sensorické funkce Sensorické funkce slouží jako parametry pro funkce chování.

Interní sensory poskytují informace o botově vnitřním stavu.

- ***health****()* — míra zdraví normalizovaná na interval $\langle 0, 1 \rangle$
- ***ammo****()* — počet nábojů pro aktuální zbraň normalizovaný na interval $\langle 0, 1 \rangle$
- ***ammoForWeapon****(WeaponType)* — počet nábojů pro danou zbraň normalizovaný na interval $\langle 0, 1 \rangle$
- ***hasWeapon****(WeaponType)* — 1 pokud bot má zbraň daného typu, 0 jinak.
- ***pain****()* — změna zdraví za poslední iteraci / 30

Externí sensory zprostředkovávají informace o okolním prostředí.

- ***time****()* — Herní čas v sekundách.
- ***nearestAmmo****(),* ***nearestArmor****(),* ***nearestHealth****(),* ***nearestWeapon****()* — Nejbližší předmět daného typu.
- ***nearestEnemy****()* — Nejbližší hráč z nepřátelského týmu.
- ***see****(Viewable)* — 1, pokud bot vidí daný Viewable objekt (Viewable objekty jsou hráči, místa na mapě atd.), 0 jinak.
- ***seeAnyEnemy****()* — Zkratka za **see**(**nearestEnemy**()).

– ***distanceTo****(Location)* — Euklidovská vzdálenost bota od dané lokace namapovaná na interval $\langle 0, 1\rangle$ funkcí arctan. Vrací -1, pokud zadaná lokace neexistuje.

Matematické funkce Na funkce *max*, *min* a $1 - x$ může být v některých kontextech nahlíženo jako na logické funkce *and*, *or* a *not* protože mnoho sensorů jejichž hodnota je pravda/nepravda jsou kódovány čísly $\{0, 1\}$.

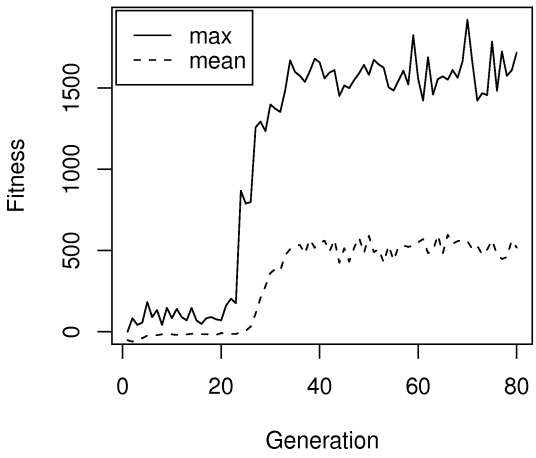
Experiment Pro otestování navržené sady primárních a sekundárních funkcí bylo provedeno několik experimentů. Detaily těchto experimentů mohou být nalezeny na blogu *Genetic bots* [5]. Jeden z těchto experimentů teď bude popsán detailněji.

V tomto experimentu hrál evolvovaný bot hru typu DeathMatch proti předprogramovanému botovi Hunterovi, který je řízen pomocí if-then pravidel napsaných v Javě (bot je součástí instalace platformy Pogamut). Podrobnosti nastavení genetického algoritmu následují.

- Mapa — dm-TrainingDay
- Generací — 80
- Jedinců v každé generaci — 150
- Iniciální generace — náhodné funkce s maximální hloubkou 5
- Elita — 16 jedinců
- Pravděpodobnost křížení — 0.2
- Pravděpodobnost mutace — 0.2
- Selekční metoda — vážená proporcionální deterministická distribuce
- Fitness — $dc + 50pk - (5d + \frac{ds}{10})$, kde dc je zranění způsobené evolvovaným botem, pk počet hráčů zabitých botem, d počet úmrtí bota ds zranění, které bot utrpěl
- Přesnost střelby — 0.5, tato hodnota dává napadenému botovi ještě šanci opětovat palbu
- Výpočet fitness — Každý bot bojoval po 90 sekund proti botovi Hunter

Mapa dm-TrainingDay je nejmenší z map oficiálně dodávaných se hrou UT. Funkce fitness zvýhodňuje útočné chování oproti vyhýbání se střetům, protože to je požadované chování ve hře typu deathmatch. Odměna za zranení nepřítele je desetkrát větší než penalizace za vlastní zranění. Přesnost střelby nastavená na 0.5 dává botovi, na kterého byla zahájena střelba, ještě šanci utéct.

V prvních dvaceti generacích zkoušela evoluce náhodné strategie, mezi dvacátou a čtyřicátou generací se populace začala zlepšovat a bylo dosaženo lokálního optima. V dalších generacích se fitness již nezvyšovala. Obrázek 6 ukazuje, jak se fitness měnila v průběhu evoluce.



Obrázek 6. Průměrná a maximální fitness v každé generaci

Obrázek 7 ukazuje strom chování jedince z poslední generace. Bot sbíral lékárníčky, když neviděl nepřítele, a zaútočil na nejbližšího nepřítele, pokud nějakého uviděl.

Následující tabulka ukazuje, jaká je výsledná akce toho stromu v závislosti na externím stimulu *SeeAnyEnemy*.

Výsledné chování není příliš komplexní, ale na takto malé mapě při hře typu deathmatch i lidští hráči používají podobnou vysokoúrovňovou strategii.

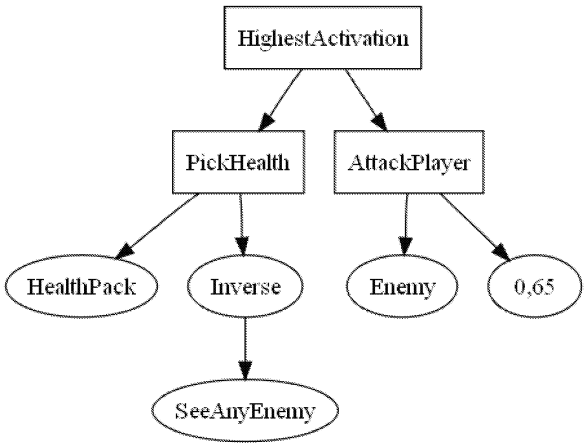
6 Budoucí práce

V budoucí práci se zaměříme na detailnější rozbor obou vrstev chování a na přidání podpory pro nové herní módy. U nižší vrstvy plánujeme otestování navigace prostředím v jiných konfiguracích světa, případně vytvoření dalších modelů pro vyhýbání se překážkám a jejich porovnání. Vyšší vrstvu chování bychom chtěli rozšířit o podporu pro další typy her jako Capture the flag (CTF) a Domination point. Tyto herní módy nabízejí botovi více možných cest k dosažení úspěchu. Například v módu CTF se může bot specializovat na kradení nepřátelské vlajky, na zabíjení nepřátelských

[5] Genetic bots blog, http://artemis.ms.mff.cuni.cz/pogamut/tiki-view_blog.php?blogId=4

Podmínka	Hodnota SeeAnyEnemy	Aktivace PickHealth	Aktivace Attack	Vítězné chování
Bot vidí nepřítele	1	0	0.65	Attack
Bot nevidí nepřítele	0	1	0	PickHealth

Tabulka 1. Jak externí stimuly ovlivňují botovo rozhodování



Obrázek 7. Strom chování nejlepšího jedince z osmdesáté generace

botů nebo může zvolit kompromisní strategii. Na základě strategie nalezéné nejlepšími jedinci by se dalo zjišťovat, zda je daná mapa pro ten který herní mód vhodná a zda příliš nezvýhodňuje jednu strategii na úkor ostatních.

7 Závěr

V článku jsme představili funkcionální model pro popis vysoko-úrovňového chování bota a model umožňující popis vyhýbání se překážek. Oba modely jsme implementovali a otestovali v prostředí hry Unreal Tournament 2004.

Experimenty ukázaly, že námi navrhované metody přinejmenším dokáží najít suboptimální smysluplná řešení zadaných problémů a potencionálně tím mohou ušetřit práci herních vývojářů.

Poděkování

Tato práce byla podpořena grantem GA UK 1053/2007/AINF/MFF. Práce byla podpořena částečně Ministerstvem školství, mládeže a tělovýchovy (projekt MSM0021620838) a částečně projektem "Information society" pod projektem 1ET100300517.

Reference

1. R. Adobbati, A. N. Marshall, A. Scholer, and S. Tejada. Gamebots: A 3d virtual world test-bed for multi-agent research. In *Proceedings of the 2nd Int. Workshop on Infrastructure for Agents, MAS, and Scalable MAS. Montreal, Canada*, 2001. URL: http://www.planetunreal.com/gamebots.
2. Alex J. Champandard. *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors.* New Riders, Indianapolis, IN, USA, 2003.
3. N. Chapman. Neuralbot. URL: http://homepages.paradise.net.nz/nickamy/neuralbot/nb_about.htm, 1999.
4. J. Holm and J. D. Nielsen. Genetic programming - applied to a real time game domain. Master's thesis, Aalborg University, 2002.
5. R. Kadlec, J. Gemrot, O. Burkert, M. Bída, J. Havlíček, and C. Brom. Pogamut 2 - a platform for fast development of virtual agents' behaviour. In *Proceedings of CGAMES 07, La Rochelle, France*, 2007.
6. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.
7. M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1996.
8. S. Priesterjahn, O. Kramer, A. Weimer, and A. Goebels. Evolution of human-competitive agents in modern computer games. pages 777–784, 2006.
9. C. W. Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference, San Jose, California.*, pages 763–782, 1999.
10. Scott D. Roth. Ray Casting for Modeling Solids. 18(2):109–144, February 1982.
11. J. Westra. Evolutionary neural networks applied in first person shooters. Master's thesis, University Utrecht, 2007.
12. Jennings N. R. Wooldridge, M. Intelligent agents - theories, architectures and languages. In *Volume 890 of Lecture Notes in Artificial Intelligence. Springer-Verlag*, 1995.
13. B. G. Woolley and G. L. Peterson. Genetic evolution of hierarchical behavior structures. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation. London, England*, pages 1731–1738, 2007.