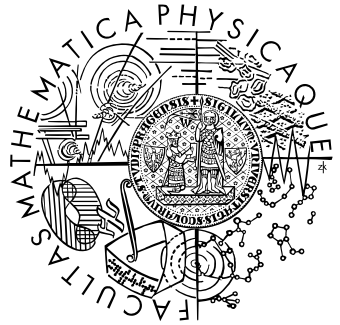
Univerzita Karlova v Prahe
Matematicko-fyzikálna fakulta

BAKALÁRSKA PRÁCA



Michal Danilák

Benchmarky pro projekt Pogamut

Kabinet software a výuky informatiky

Vedúci bakalárskej práce: Mgr. Cyril Brom, Ph.D.

Študijný program: obecná informatika

2010

Chcel by som sa poďakovať predovšetkým svojmu vedúcemu práce, Cyrilovi Bromovi, za jeho trpezlivosť, cenné námety a pripomienky, bez ktorých by táto práca asi nevznikla.

Ďalej by som sa chcel poďakovať Jakubovi Gemrotovi, za jeho rady a pomoc pri práci s platformou Pogamut, Danielovi Toropilovi a doc. RNDr. Romanovi Bartákovi, Ph.D. za prepožičanie plánovacieho softwaru a pomoci pri jeho používaní.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičaním práce.

V Prahe dňa 4. 8. 2010 Michal Danilák

Obsah strana

Názov práce: Benchmarky pro projekt Pogamut
Autor: Michal Danilák
Katedra: Kabinet software a výuky informatiky
Vedúci bakalárskej práce: Mgr. Cyril Brom, Ph.D.
e-mail vedúceho: Cyril.Brom@mff.cuni.cz

Abstrakt: Práca sa zaoberá tvorbou a použitím testovacích prostredí pre počítačom ovládaných agentov (botov) do hry Unreal Tournament 2004. V prvej časti sa snaží identifikovať kľúčové prvky botov, ktoré je možné exaktne testovať v prostredí hry a následne vyhodnocovať ich efektivitu. Druhá časť spočíva v návrhu a tvorbe prostredí, v ktorých by sa vybrané prvky otestovali. Cieľom tretej časti je vytvorenie referenčných botov pre jednotlivé prostredia, ktorí budú slúžiť ako základ pre budúci vývoj.

Kľúčové slová: bot, Unreal Tournament 2004, testovacie prostredie

Title: Benchmarks for the project Pogamut
Author: Michal Danilák
Department: Department of Software and Computer Science Education
Supervisor: Mgr. Cyril Brom, Ph.D.
Supervisor's e-mail address: Cyril.Brom@mff.cuni.cz

Abstract: In the present work we design and develop the benchmarks for the computer controlled agents (bots) into the game Unreal Tournament 2004. In the first part of the work, we try to identify the most important aspects of the bots which should be tested. In the second part we design and develop the environments for testing the selected aspects. In the third part we implement the reference solutions (bots) for each of the created environments.

Keywords: bot, Unreal Tournament 2004, benchmark

1. Kapitola

Úvod

V oblasti umelej inteligencie rozumieme pod pojmom multiagentový systém simulované prostredie, v ktorom dochádza k interakcii určitých typov aktérov, agentov. V prípade, že hovoríme o virtuálnych agentoch, tak za simulované prostredie považujeme počítačový program simulujúci zjednodušený model sveta. Pre takúto simuláciu sa väčšinou používa prostredie už existujúcej počítačovej hry, ktoré je trojrozmerné s fungujúcou fyzikou a má bližšie k širokému publiku. Jednou z najpoužívanejších hier v tejto oblasti je Unreal Tournament 2004 (ďalej len UT2004) [1,2].

Žáner hry UT2004 je first person shooter (FPS), čo v preklade znamená strieľačka videná z pohľadu prvej osoby. Cieľom hráča v takej hre je zastreliť všetkých súperov a ostať nažive. Jeho súperi môžu byť buďto iní hráči, pripojení cez sieť, alebo počítačom ovládaní agenti (boti). A práve tvorba a testovanie týchto botov je kľúčová v našej práci.

V súčasnosti neexistuje štandardná sada nástrojov, ktorá by automaticky otestovala úspešnosť botov v hrách (nie len UT2004), nech už si pod pojmom „úspešnosť“ predstavíme čokoľvek. Väčšinou sa pod tým myslí bojaschopnosť bota, ale taktiež to môže znamenať uveriteľnosť bota, tzn. nakoľko sa nám bot javí ako ovládaný počítačom a nakoľko ako ovládaný človekom.

Cieľom tejto práce je identifikovať kľúčové aspekty botov, ktoré je možné takto testovať (cieľ č. 1) a následne vytvoriť testovacie prostredia vo forme máp do UT2004, ktoré by ich otestovali (cieľ č. 2). Posledným krokom bude tvorba referenčných botov, ktorí by slúžili ako odrazový mostík pre budúci vývoj (cieľ č. 3).

V 2. kapitole si stručne popíšeme princíp hry UT2004 a následne platformu Pogamut, ktorá umožňuje jednoduchý vývoj botov do UT2004 v jazyku Java.

V 3. kapitole identifikujeme kľúčové aspekty botov, na ktoré sa zameriame a pre ktoré v ďalšej časti vytvoríme testovacie prostredia (cieľ č. 1).

V 4. kapitole navrhneme štruktúru samotných testovacích prostredí a ukážeme si nástroj na generovanie náhodných testovacích máp do UT2004, ktorý bol vyvinutý práve za účelom testovania botov (cieľ č. 2).

V 5. kapitole vytvoríme samotných referenčných botov (cieľ č. 3).

2. Kapitola

Programovanie botov a platforma Pogamut

V tejto kapitole si stručne popíšeme proces, ktorý je spojený s vývojom virtuálneho agenta a následne si ukážeme ako vyzerajú takýto agenti v prostredí hry UT2004. Na záver si popíšeme platformu Pogamut, ktorá umožňuje jednoduchý vývoj týchto agentov do hry UT2004 v jazyku Java.

2.1 Programovanie virtuálnych agentov

Pod pojmom „virtuálny agent“ rozumieme počítačový program, ktorý sa snaží simulovať ľudské chovanie vo virtuálnom prostredí (virtuálnom svete) [3]. V tomto prostredí dochádza medzi agentom a jeho okolím k interakcii, pričom agent sa snaží za každých okolností chovať tak, aby čo najviac vyhovel cieľu, ktorý mu bol predpísaný. V prostredí počítačových hier je virtuálny agent často nazývaný *bot*, a tento názov budeme používať aj my. Ak existuje v prostredí viacero agentov súčasne a dochádza medzi nimi k interakcii, hovoríme o multiagentovom systéme.

Aby nejaký bot mohol úspešne simulovať ľudské chovanie, je potrebné aby mal svoje virtuálne telo, ktorým sa bude navonok prejavovať a cez ktoré bude prijímať vnemy z okolia. Aby však toto telo mohlo interagovať so svojím prostredím, je potrebné zasadiť ho do nejakého virtuálneho sveta. Podľa možností by tento svet mal byť čo najviac podobný tomu nášmu a teda trojrozmerný s fungujúcimi fyzikálnymi zákonmi. Ďalej je potrebné tento svet nejako vizualizovať, aby mohol človek posúdiť, či sa bot navonok skutočne chová ľudsky. A až budeme mať všetko toto pripravené, konečne môžeme nášmu botovi naprogramovať „umelú myseľ“, ktorá ho bude riadiť a následne ho môžeme otestovať v našom virtuálnom svete.

Z vyššie uvedeného vyplýva, že človek zaoberajúci sa umelou inteligenciou nemá šancu sám naprogramovať všetky komponenty potrebné k vytvoreniu virtuálneho sveta, ktorých tvorba je nie len časovo veľmi náročná, ale zároveň vyžaduje pokročilé znalosti z oblasti počítačovej grafiky, fyzikálneho modelovania, softwarového inžinierstva atď. Preto sa v takejto situácii väčšinou volí druhá možnosť a to využitie už existujúceho virtuálneho sveta, napríklad počítačovej hry.

V našom prípade sa bude jednať o hru Unreal Tournament 2004. Hra UT2004 bola zvolená hlavne, z dôvodu už existujúcej platforme Pogamut [4], ktorá umožňuje veľmi jednoduché programovanie botov do hry v jazyku Java a taktiež poskytuje vynikajúcu podporu ladenia týchto botov. Navyše sa s využitím platformy Pogamut každoročne koná celosvetová súťaž

v programovaní botov do UT2004, na ktorú sa neskôr detailnejšie pozrieme a pre ktorú, dúfame, bude mať táto práca prínos.

2.2 Unreal tournament 2004

Hra Unreal Tournament 2004 [2] je počítačová hra vyvinutá americkou spoločnosťou Epic games [1] a vydaná v roku 2004. Jedná sa o strieľačku videnú z prvej osoby (FPS – first person shooter), v ktorej je úlohou hráča, v závislosti od zvoleného módu, buďto zastreliť čo najviac súperov za stanovený časový limit (Death Match) alebo ostať posledný nažive (Last Man Standing) alebo ukradnúť súperovi vlajku a doniesť ju na vlastnú základňu (Capture The Flag) atď. Všetky tieto súboje sa odohrávajú na mapách (leveloch), ktoré majú arénový charakter.

Obr. 1: Hra Unreal Tournament 2004 (Copyright © 2010 Epic Games)

Virtuálny svet, ktorý táto hra poskytuje je plne trojrozmerný a s funkčnou fyzikou. Boti existujúci v tomto svete majú svoje virtuálne telá a možnosť prejavovať sa tým, že sa v tomto svete pohybujú (chodia, bežia, skáču), zbierajú predmety ako zbrane, náboje, lekárničky a štíty a strieľajú po svojich súperoch.

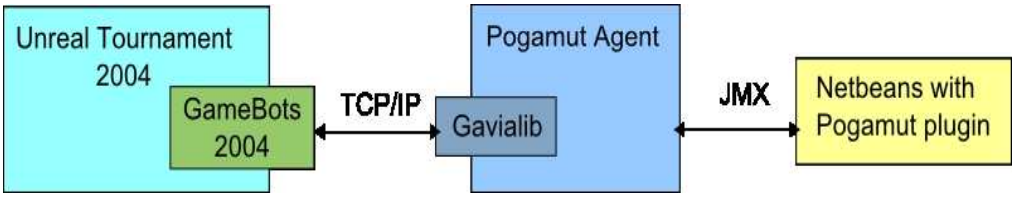
Primárne slúžia boti v tejto hre na to, aby bojovali proti hráčovi a robili tak hru zábavnou. UT2004 je ale ľahko modifikovateľná a z hľadiska umelej inteligencie je možné do nej napísať botov s ďaleko komplexnejším chovaním a prejavovaním sa.

2.3 Platforma Pogamut

Platforma Pogamut [4] je softwarový nástroj umožňujúci programátorovi jednoduchý vývoj a testovanie bota do hry UT2004 v jazyku Java vo vývojovom prostredí NetBeans [5]. Pre našu potrebu platforma poskytuje nasledujúce veci:

- Simulátor virtuálneho sveta – hra Unreal Tournament 2004 rozšírená o komunikačné rozhranie GameBots2004 distribuované pod licenciou GNU.
- Integrované vývojové prostredie - plug-in do prostredia NetBeans, ktoré poskutyje podporu pri programovaní, ladení a experimentovaní s botom.
- Knižnicu tried v jazyku Java – poskytuje API pre programovanie botov a zároveň implementuje sadu nízkoúrovňových algoritmov (napr. pathfinding, streľba atď.).

Pre lepšiu predstavu o tom, v akom vzťahu je platforma Pogamut ku hre UT2004, si ukážeme nasledujúci obrázok:



Obr. 2: Architektúra platformy Pogamut (prevzaté z [4])

Na ňom vidíme, že informácie z hry UT2004 sú exportované cez TCP/IP pomocou textového protokolu GameBots2004. Tieto správy sú následne spracované Javovskou knižnicou GaviaLib, takže pri programovaní samotného bota už pracujeme s objektmi jazyka Java. Pomocou protokolu JMX skrz Pogamut NetBeans plug-in je možné vzdialené ladenie bota.

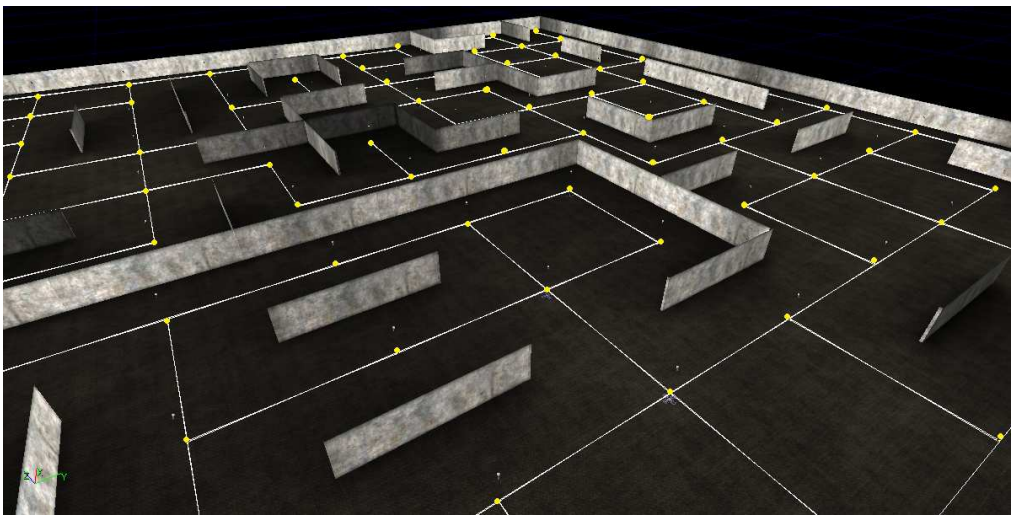
2.4 Princípy používané pri programovaní botov

Pre úplnosť teraz vymenujem niekoľko základných princípov, ktoré sa pri vývoji botov používajú a nemusia byť na prvý pohľad zrejmé. Aby sme tieto princípy lepšie pochopili, pozrime sa najprv na obrázok č. 1 (vyššie), kde sme mali možnosť vidieť, ako vyzerá UT2004 z pohľadu hráča. V pravom dolnom rohu obrazovky vidí hráč zbraň, ktorú má práve aktívnu a uprostred obrazovky je vidieť malý zelený terčík, ktorý presne ukazuje na miesto, kam hráč zbraňou mieri. Len pár metrov od hráča vidíme tesne nad zemou poletovať zbraň, čo označuje miesto, kde si hráč môže vyzdvihnúť novú zbraň. V pravej časti obrazovky vidíme v pozadí na moste malé modré fľaštičky, reprezentujúce

lekárničky, ktoré môže hráč v prípade potreby zobrať a doplní sa mu tak zdravie, ktorého aktuálnu hodnotu má možnosť vidieť v ľavom dolnom rohu. Všetky tieto informácie a pár ďalších vidí hráč na obrazovke a práve vďaka ním je schopný pohybovať sa a interagovať vo virtuálnom svete.

Z principiálneho hľadiska nie je možné poskytovať botovi informácie o svete v rovnakej podobe ako sú poskytované hráčovi – tzn. vo forme obrazu. Z rýchlostných dôvodov je botovi predložená veľmi zjednodušená reprezentácia sveta, ktorá je vo väčšine prípadov plne postačujúca. Na obrázku č. 3 máme možnosť vidieť informácie o svete, ktoré bot má k dispozícii.

Samotná mapa vyzerá ako bludisko. V rámci celej mapy sú rovnomerne umiestnené takzvané navigačné body (žlté bodky na obrázku), medzi ktorými vedú takzvané navigačné hrany (biele čiary na obrázku) a dohromady tvoria navigačný graf. Takýto graf je prítomný v každej mape a vytvára ho dizajnér danej mapy (tzn. nie je vytváraný automaticky). Len podotknem, že tento graf je orientovaný a nie je zaručené, že ak vedie hrana z bodu A do bodu B, tak vedie aj hrana z bodu B do bodu A.



Obr. 3: Navigačné body (žlté bodky) a navigačné hrany (biele čiary) je to jediné, čo bot "vidí" (Copyright © 2010 Epic Games)

Z celého bludiska, ktoré máme možnosť na obrázku vidieť, má bot informácie iba o polohách navigačných bodov a hranách, ktoré medzi nimi vedú. Navigačné body vždy označujú miesta, ktoré sú pre bota bezpečné (tzn. nie sú umiestnené nad priepasťou) a niekedy aj niečím dôležité (napr. označujú miesto, kde je lekárnička). Navigačná hrana vedúca medzi dvoma navigačnými bodmi označuje trasu, po ktorej sa bot dostane z jedného bodu do druhého, pričom sa predpokladá, že je táto trasa pre bota bezpečná. O miestach, ktoré nie sú navigačnými bodmi pokryté nemá bot žiadnu informáciu a je preto dobrým pravidlom, aby sa takým miestam vyhol.

Taktiež sú botovi poskytnuté informácie o presných polohách všetkých predmetov, ktoré sa na mape vyskytujú a s využitím navigačného grafu má bot

možnosť vypočítať si trasu, ktorou sa tam dostane. Ďalej má bot informácie o objektoch, ktoré sa vyskytujú v jeho zornom poli, čo je často jediný spôsob ako zistiť polohu nepriateľa. V neposlednej rade je botovi k dispozícii plná introspekcia – tzn. má informácie o vlastnej polohe na mape, rýchlosti pohybu, počte životov, počte zobratých zbraní atď.

S využitím týchto informácií sa musí bot v každom kroku rozhodnúť, čo urobí ďalej, pričom má na výber z akcií pohybu (premiestnenie sa, výskok, čupnutie, uhnutie strele) a streľby (výstrel, sekundárny výstrel, zmena zbrane).

3. Kapitola

Kľúčové aspekty botov

V tejto kapitole sa pokúsime identifikovať kľúčové aspekty botov, na ktoré sa zameriame a v ďalšej časti pre ne vytvoríme testovacie prostredia (cieľ č. 1).

3.1 Boti v hrách obecne

Väčšina moderných počítačových hier je založená na princípe, kedy medzi sebou dvaja alebo viacerí hráči súperia a každý z nich sa snaží ako prvý dosiahnuť vopred stanovenú víťaznú podmienku. Tieto hry sa hrávajú buďto cez Internet alebo po lokálnej sieti (LAN). Aby však hra mala úspech, snažia sa jej autori do nej implementovať počítačom ovládaných hráčov, botov, proti ktorým si človek môže zahrať v prípade, že nemá pripojenie k Internetu alebo nemá ľudského súpera, proti ktorému by si zahral. Aby mal takýto bot medzi hráčmi úspech, musí spĺňať viacero podmienok:

1) Musí byť schopný hrať hru na úrovni začiatočníka, stredne pokročilého hráča a veľmi pokročilého hráča, aby oslovil čo najširšie publikum – táto možnosť sa v hrách vyskytuje ako voľba obtiažnosti.
2) Rovnako ako ľudský hráč, musí rešpektovať pravidlá hry a hrať za rovnakých podmienok a obmedzení – teda nemal by „podvádzať“, aby sa mohol vyrovnať pokročilému hráčovi.
3) Jeho štýl hry musí do určitej miery zodpovedať štýlu hry ľudského hráča – teda hráč by nemal nadobudnúť pocit, že je botova hra „umelá“.

Splniť tieto podmienky nie je vždy jednoduchá vec a preto sa autori hier často uchyľujú k podvádzaniu (tzn. porušeniu podmienky 2.), aby tak mohli čo najlepšie splniť podmienky 1. a 3. Z hľadiska herného biznisu to funguje veľmi dobre, ale z hľadiska umelej inteligencie takéto riešenie nie je postačujúce a hľadajú sa pokročilejšie a prijateľnejšie metódy.

3.2 Súťaže Botprize a iné

Odpoveď na otázku, kam až je umelá inteligencia v hrách schopná zájsť, hľadajú ľudia aj tým, že každoročne organizujú celosvetové súťaže v tom, kto naprogramuje lepšieho bota do tej, ktorej hry. Asi medzi najvýznamnejšie patria súťaže organizované konferenciou CIG (Computational Intelligence and Games) [6], ktorých je každý rok hneď niekoľko. Len zbežne spomeniem, že pre rok 2010 sú pripravované súťaže botov do hier ako StarCraft, UT2004, Ms. Pac-Man alebo Mario [7].

Z nášho pohľadu je najzaujímavejšia súťaž Botprize [8], ktorá tiež patrí pod konferenciu CIG a súťaží sa v hre UT2004, pričom sa využíva už spomínaná platforma Pogamut. Na rozdiel od ostatných súťaží, kde majú programátori za úlohu vytvoriť čo najlepšie hrajúceho bota, majú v tejto súťaži za úlohu vytvoriť čo najpresvedčivejšie hrajúceho bota, inými slovami bota, ktorý hrá čo najpodobnejšie človeku. Z troch bodov, ktoré sme si uviedli, má teda bot čo najlepšie spĺňať bod číslo 3.

Teraz zbežne popíšem ako táto súťaž prebieha, pretože práve tie aspekty botov, ktoré sú v tejto súťaži dôležité, budú dôležité aj pre našu prácu. Pravidlá súťaže Botprize sa vo veľkej miere podobajú Turingovmu testu [9]. Na jednej mape hrajú vždy traja hráči – rozhodca, ľudský hráč a bot. Všetci traja sa hry aktívne zúčastňujú, ale navzájom nevedia kto je kto. Úlohou rozhodcu je po odohratom časovom limite určiť, ktorý zo zvyšných dvoch hráčov bol človek a ktorý z nich bol bot. Absolútnym víťazom súťaže sa stane ten bot, ktorý „oklame“ všetkých piatich zúčastnených rozhodcov. Ešte ani jednému botovi sa nepodarilo oklamať všetkých rozhodcov, ale v roku 2008 a 2009 sa víťaznému botovi podarilo oklamať dvoch z piatich rozhodcov.

3.3 Dôležité aspekty bota

Hra UT2004 poskytuje hráčovi možnosť vybrať si až z ôsmich obtiažností botov, proti ktorým si môže zahrať. Boti na najťažšej obtiažnosti sa prakticky nedajú poraziť. Môžeme teda povedať, že sú v istom smere dokonalí. Tak prečo sa potom ľudia ďalej snažia vytvoriť botov lepších? Respektíve, v akom smere lepších? Paradoxne slovo *lepších*, v tomto prípade znamená *horšie* hrajúcich...

Bot, ktorý každého porazí, z pohľadu hráča nie je zábavný. Je namieste otázka, ako to bot dokáže. Je príliš rýchly? Je príliš presný? Má viac životov ako hráč? Nepodvádza náhodou? Správny bot by mal tiež robiť chyby, napríklad niekedy spadnúť z budovy, niekedy pri streľbe minúť. Tieto chyby by ale na druhú stranu nemal robiť príliš často a mal by sa z nich poučiť – tzn. robiť ich menej často. A práve tieto požiadavky hráčov na botov nie sú splnené a je na nich stále čo zlepšovať.

Jedným z najzákladnejších aspektov presvedčivého bota je schopnosť učiť sa, resp. prispôsobiť sa neznámym podmienkam. Aby som to uviedol na príklade, tak na súťaži Botprize v roku 2008 [10] mali boti za úlohu naučiť sa efektívne používať dopredu neznáme zbrane. Hlavná myšlienka je v tom, že každá zbraň má odlišne vlastnosti pri streľbe – niektoré zbrane sú rýchlopalné a presné do diaľky, iné majú plošný efekt pri streľbe a keď je bot blízko miesta dopadu strely, tak ho to zraní, niektoré zbrane strieľajú po balistickej krivke a iné zase po priamke. UT2004 poskytuje základnú sadu zbraní, ktorých je asi 15 a nie je ťažké botovi naprogramovať, ktorú zbraň má v ktorej situácií použiť. Lenže v Botprize 2008 boli efekty všetkých zbraní zmenené a boti sa museli naučiť ich používať za behu.

Ďalším významným aspektom presvedčivého bota je schopnosť riešiť netriviálne problémy. Ako si v ďalšej časti predvedieme, tak ani tí najlepší boti UT2004 nie sú schopní riešiť problémy, ktoré vyžadujú naplánovanie si cesty „2 kroky napred“. Konkrétnejšie, ak má bot za úlohu dostať sa z miesta A na miesto B, tak to zvládne bez problémov. Lenže čo ak by bol náš bot otrávený a každú sekundu by stratil niekoľko životov až by nakoniec umrel skôr, než by do bodu B dorazil? Riešením tohto problému by bolo zastaviť sa cestou z A do B po lekárničku, ktorá je na mieste C. Lenže cesta k miestu C je dosť veľká zachádzka a z hľadiska optimálnosti cesty z A do B neprichádza do úvahy. Ale to, že to je jediná možnosť ako prežiť si už drvivá väčšina botov neuvedomí.

3.4 Zhrnutie

Ak by sme teda mali zhrnúť aspekty, na ktoré sa v tejto práci zameriame, tak to bude predovšetkým schopnosť prispôsobovať sa dopredu neznámym podmienkam a schopnosť riešiť netriviálne problémy.

4. Kapitola

Tvorba testovacích prostredí

V tejto kapitole si v úvode stručne vysvetlíme, čo sa myslí pod pojmom testovacie prostredie pre bota do hry UT2004. Následne si ukážeme a popíšeme nástroj UT2004MapsGenerator, ktorý bol vytvorený práve za účelom tvorby testovacích máp pre botov. Bližšie rozoberieme aké typy máp je tento nástroj schopný vygenerovať a ktoré aspekty botov je týmito mapami možné testovať. (cieľ č. 2)

4.1 Motivácia testovacích prostredí

V súčasnosti neexistuje sada nástrojov, ktorá by bola schopná automaticky otestovať a vyhodnotiť správanie bota. Pod pojmom vyhodnotiť máme na mysli buďto číselne ohodnotiť jeho úspešnosť (napr. na stupnici od 1 do 10) alebo ju aspoň nejako jednoducho graficky znázorniť (napr. grafom).

Možnosť automaticky testovať a vyhodnocovať správanie botov je výhodná hlavne z dôvodu časovej nenáročnosti a exaktnosti. Užívateľ získa možnosť otestovať viacero botov súčasne a bez jeho prítomnosti a zároveň pri dobrom systéme vyhodnocovania, bude mať možnosť exaktne porovnávať efektivitu použitých algoritmov v botovi.

Určité pokusy o automatické rozhodovanie toho, či sa jedna o bota alebo o hráča (a teda aj vyhodnocovanie správania bota) môžme nájsť v práci [11, 12].

4.2 Testovacie prostredie a mapy

Samotné testovacie prostredie sa bude skladať z dvoch komponent:

1) Mapa (level) – na ktorej dochádza k testovaniu bota.
2) Observer – program, ktorý sleduje bota počas testovania, vedie si o jeho chovaní štatistiky. Na konci tieto štatistiky vyhodnotí a vráti nám výsledok testovania (číselné ohodnotenie / graf).

V nasledujúcich častiach sa budeme venovať prvému bodu a teda mapám, na ktorých bude dochádzať k samotnému testovaniu. Bod číslo dva bude rozobratý v časti 4.6.

Možnosť vytvárať mapy do hry UT2004 majú užívatelia vďaka nástroju UnrealEd, ktorý je dodávaný spolu s hrou. Vytvorenie mapy obnáša vytvorenie samotnej geometrie mapy (tzn. ako bude mapa vyzerať), nastavenie textúr použitým objektom, vloženie predmetov do mapy ako sú zbrane, náboje, či lekárničky, vloženie ďalších objektov ako sú voda, výťahy, dvere a iné, vloženie osvetlenia do mapy, nastavenie fyziky, ktorá bude na mape platiť

a následne vytvorenie navigačného grafu pre botov. Aj pre skúseného dizajnéra zaberie vytvorenie jednej takej mapy rádovo hodiny.

Ak už sa nám podarí nadizajnovať mapu, ktorá bude schopná testovať nejaký aspekt bota, bolo by dobré mať takú mapu v rôznych obmenách a variantoch. Je to z toho dôvodu, aby sme neladili botovo chovanie pre jednu konkrétu mapu, ale pre jeden konkrétny aspekt.

Ďalej by bolo výhodné mať možnosť mapu v niektorých ohľadoch parametrizovať a skúšať tak vytvárať nové jedinečné podmienky, v ktorých by sme mohli bota ďalej testovať. Vynikajúcim príkladom takéhoto postupu je víťaz súťaže Botprize z roku 2009 [13], ktorý využil práve takéto parametrizovanie mapy k tomu, aby sa jeho boti efektívnejšie pohybovali po mape aj za podmienok riedkeho navigačného grafu [14].

4.3 Nástroj UT2004MapsGenerator

Za účelom tvorby parametrizovateľných testovacích prostredí bol vytvorený nástroj UT2004MapsGenerator (ďalej len MG). Stručne povedané je MG program napísaný v jazyku C#, ktorý je schopný generovať tri rôzne typy máp (pretože budeme vytvárať tri rôzne testovacie prostredia). Tieto mapy sú ľahko parametrizovateľné a pri ich vytváraní sa využíva prvok náhody, takže sú zakaždým iné. Tieto tri typy máp si o chvíľu veľmi podrobne popíšeme, pretože sú pre našu prácu kľúčové. Najprv sa ale pozrieme na to, ako MG funguje a aké sú jeho hlavné výhody.

Ako už bolo spomenuté, oficiálnym nástrojom na tvorbu máp do UT2004 je UnrealEd. Každá mapa vytvorená v tomto editore sa musí, ešte predtým než sa začne používať, najprv skompilovať. Počas tejto kompilácie sa do mapy dopočítava množstvo informácií, ktoré sa počas hry už meniť nebudú a je z časového hľadiska výhodné vypočítať ich len raz a uložiť si ich pre budúce použitie. Patria sem napríklad shadow mapy všetkých statických objektov, ktoré sú osvetľované statickým osvetlením – pretože keď sa svetlo ani objekt počas hry nebudú hýbať, tak nie je dôvod túto informáciu počítať viac než raz, navyše je tento výpočet časovo veľmi náročný. Ďalej sa vypočíta a uloží celý navigačný graf mapy – pretože dizajnér určil iba polohy navigačných bodov, je potrebné dopočítať navigačné hrany. Výsledná mapa sa uloží na disk ako súbor s príponou *.ut2* a je pripravená na použitie.

Pre externý nástroj, akým je MG, je nemožné tento proces kompilácie zreplikovať, pretože jeho presný priebeh nie je známy a formát súboru *.ut2* taktiež nie. Našťastie sa tento proces dá obísť tak, že podobne ako človek vytvárajúci mapu, tak aj program MG, mapu len „popíše“ (tzn. povie, ktorý objekt kam patrí, akú má mať veľkosť a textúru atď.) a na záver nechá UnrealEd, aby proces kompilácie urobil za neho. Takzvané „popisovanie“ pozícií objektov sa realizuje pomocou textového formátu *t3d*, cez ktorý je možné importovať objekty a celé mapy do nástroja UnrealEd. Uvedieme si príklad, ako tento formát vyzerá:

```
   Begin Actor Class=Light Name=Light0
       LightHue=51
       LightSaturation=204
       LightBrightness=99.000000
       LightRadius=128.000000
       Level=LevelInfo'myLevel.LevelInfo0'
       Region=(Zone=LevelInfo'myLevel.LevelInfo0',iLeaf=-1)
       Tag="Light"
   PhysicsVolume=DefaultPhysicsVolume'myLevel.DefaultPhysicsVol
ume0'
       Location=(X=-16.000000,Y=-16.000000,Z=16.000000)
   End Actor
```

Aj človek, ktorý nikdy s nástrojom UnrealEd nepracoval dokáže vyčítať, že horeuvedený kus kódu definuje svetlo umiestnené niekde na mape, ktoré má definované vlastnosti ako `LightHue`, `LightSaturation`, `LightBrightness` a `LightRadius`. Napriek tomu, že pre formát *t3d* neexistuje dokumentácia, boli sme schopní metódou pokusu a omylu prepísať do jazyka C# značné množstvo objektov, s ktorými už následne MG dokáže pracovať.

Zdrojový kód programu MG sa skladá z dvoch samostatných častí:

- Knižnice objektov, ktoré reprezentujú objekty na mape a poskytujú jednoduchý interface k ich manipulácii.
- Troch rôznych generátorov máp, ktoré za pomoci algoritmov rozmiestňujú objekty po mape, následne celú mapu vyexportujú do formátu *t3d*, užívateľ si ju naimprotuje do UnrealEd, nechá si ju skompilovať a tým je mapa pripravená k použitiu.

Pre bližší popis používania nástroja MG viď užívateľskú dokumentáciu na priloženom CD.

4.4 Knižnica objektov nástroja UT2004MapsGenerator

Aby mohol program MG fungovať a vytvárať množstvo variabilných máp, je potrebné aby mal k dispozícií, čo najväčšiu sadu objektov (medzi objekty rátame štetce definujúce geometriu mapy, osvetlenia, navigačné body, predmety atď.), s ktorými môže pracovať a umiestňovať ich do máp. Do základnej verzie MG je ich pripravených približne 25 a nie je problém dodefinovať ďalšie.

Každý takto definovaný objekt musí podporovať vlastné exportovanie do formátu *t3d*. A pretože má tento formát deklaratívny charakter (tzn. len priradzuje objektovým premenným hodnoty), je pre každý objekt vytvorený externý textový súbor, v ktorom je uložený jeho presný *t3d* formát s vyznačenými miestami (názov premennej medzi dvoma znakmi %), na ktoré majú prísť výsledné hodnoty. Objekt tak pri procese vlastného exportovania len prepisuje tento súbor na výstup, pričom za názvy premenných dosádza ich skutočné hodnoty. Napríklad pre vyššie definované svetlo, vyzerá súbor nasledovne:

```
   Begin Actor Class=Light Name=%name%
       LightHue=%lightHue%
       LightSaturation=%lightSaturation%
       LightBrightness=%lightBrightness%
       LightRadius=%lightRadius%
       Level=LevelInfo'myLevel.LevelInfo0'
       Region=(Zone=LevelInfo'myLevel.LevelInfo0',iLeaf=-1)
       Tag="Light"
   PhysicsVolume=DefaultPhysicsVolume'myLevel.DefaultPhysicsVol
ume0'
       Location=(X=%locationX%,Y=%locationY%,Z=%locationZ%)
   End Actor
```

Takýto súbor nazývame šablónový (template file) a jeho presný formát je popísaný v súbore *„Template format documentation.txt“* na priloženom CD. Len doplním, že šablónové súbory podporujú aj polia, ktoré sa využívajú v pokročilejších objektoch.

Systém externých šablónových súborov bol zvolený hlavne kvôli jeho prehľadnosti a jednoduchému používaniu. Zároveň v prípade zmeny formátu *t3d* je možná aktualizácia šablónových súborov bez nutnosti zásahu do zdrojových kódov programu a nutnej rekompilácie.

Pre bližší popis možného rozšírenia knižnice objektov viď programátorskú dokumentáciu na priloženom CD.

4.5 Generátory náhodných máp

Aktuálna verzia MG obsahuje tri typovo odlišné generátory náhodných máp, ktorých názvy sú: **maze**, **lava** a **water**. Úlohou týchto generátorov je vytvárať parametrizovateľné mapy s prvkom náhodnosti, ktoré budú následne exportované do formátu *t3d* a uložené na disk ako súbor s príponou *.t3d*. Užívateľ si následne tento *.t3d* súbor naimportuje do nástroja UnrealEd, kde ho nechá skompilovať a mapa je tak pripravená k použitiu.

Proces generovania náhodnej mapy sa v podstate dá zhrnúť do nasledujúcich fáz:

- Vygenerovanie geometrie mapy.
- Vygenerovanie objektov špecifických pre tú, ktorú mapu (výťahy, jazierka, atď.).
- Vygenerovanie osvetlenia mapy.
- Vygenerovanie štartovných pozícii pre botov (tzn. miesta, na ktorých sa na začiatku hry objavia).
- Umiestnenie predmetov do mapy (zbrane, náboje, lekárničky atď.).
- Umiestnenie navigačných bodov do mapy.
- Definovanie fyziky, ktorá bude na mape platiť.

V závislosti od generovaného typu mapy, je priebeh niektorých fáz parametrizovateľný a v niektorých fázach zasa vystupuje prvok náhody.

Generované mapy je možné parametrizovať prostredníctvom externých *.ini* súborov, ktoré sú pribalené k samotnému MG programu. Formát týchto súborov má jednoduchý deklaratívny charakter. Ako príklad si ukážeme kus kódu, uložený v súbore *UT2004MapsGenerator.ini*, ktorým sa dá parametrizovať fyzika mapy:

```
GravityX = 0.0
GravityY = 0.0
GravityZ = -950.0
TerminalVelocity = 2500.0
GroundFriction = 8.0
DamagePerSec = 3.0
```

Pre bližší význam týchto premenných viď [15].

Užívateľ ma možnosť všetky tieto hodnoty ľubovoľne pozmeniť a ich efekt sa následne objaví vo vygenerovanej mape.

Spomínaný súbor *UT2004MapsGenerator.ini* je spoločný pre všetky tri typy generovaných máp a okrem fyziky, sa v ňom nachádzajú taktiež deklarácie predmetov (tzn. koľko inštancií má mapa, z toho ktorého predmetu, obsahovať) a osvetlenia mapy.

Ďalej sú ku generátoru pribalené tri ďalšie *.ini* súbory, vzťahujúce sa k jednotlivých typom generovaných máp. Sú to súbory:

- *MazeGenerator.ini* – pre mapu typu **maze**.
- *LavaGenerator.ini* – pre mapu typu **lava**.
- *WaterGenerator.ini* – pre mapu typu **water**.

Tieto súbory, majú rovnaký formát ako súbor *UT2004MapsGenerator.ini* a obsahujú zoznam parametrizovateľných premenných špecifických pre jednotlivé typy máp (napr. veľkosť bludiska pre mapu typu **maze**).

Zoznam parametrizovateľných premenných je v každom *.ini* súbore úplný a dostatočne popísaný. V prípade, že užívateľ niektorú z týchto premenných zmaže, použije pre ňu generátor výchoziu hodnotu. V prípade, že užívateľ chce pre niektorú premennú použiť výchoziu hodnotu, ale zároveň nechce túto premennú stratiť tým, že ju vymaže, má možnosť zakomentovať riadok s touto premennou tak, že na začiatok riadku pridá znak ‘#’. V prípade, že užívateľ zle zadá hodnotu niektorej premennej (v zlom formáte), generátor ho na to patrične upozorní. V prípade, že užívateľ zadá nezmyselné hodnoty (napr. šírka levelu = -1), je chovanie nedefinované.

Ak nebude uvedené inak, tak všetky jednotky vzdialenosti sú definované v takzvaných „UT units“, ktorých mierka je približne 1 meter = 52.5 UT units. Pre bližší popis viď [15].

Teraz si podrobnejšie popíšeme jednotlivé typy generovaných máp.

4.5.1 Mapa typu MAZE

4.5.1.1 Testovaný aspekt bota

Pri programovaní úspešného bota do hry UT2004 je jednou z jeho dôležitejších komponent efektívny zber predmetov, ktoré sa na mape nachádzajú. Medzi tieto predmety patria lekárničky, zbrane, náboje, štíty a iné. Po zobratí každého z týchto predmetov získa bot určitý bonus (doplnia sa mu životy, získa novú zbraň, atď.), pričom veľkosť bonusu v niektorých prípadoch záleží od situácie, pretože bot s plnými životmi už ďalšie nezíska a ak už nejakú zbraň má, tak jej opätovné zobratie nemá efekt. Taktiež sa po zobratí stane predmet na mape na určitý čas neaktívnym a bude ho možné opäť zobrať až po uplynutí určitého času (asi 30 sekúnd) – ak si teda chce bot urýchlene doplniť životy, tak mu nestačí stáť len pri jednej lekárničke, ale musí ich obehať niekoľko.

V najzákladnejšej podobe je možné formulovať problém nasledovne:

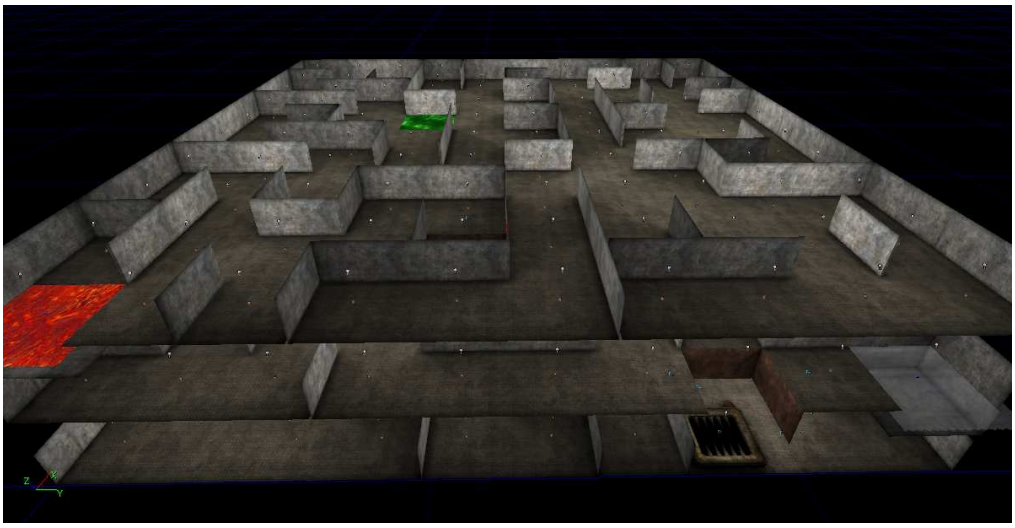
> *Je daný zoznam predmetov umiestnených na mape. Botovou úlohou je ich všetky čo najrýchlejšie pozbierať.*

Táto úloha je známa ako problém obchodného cestujúceho [16] a v súčasnosti neexistuje algoritmus, ktorý by ju pre väčší počet objektov na mape, bol schopný vyriešiť optimálne za rozumný čas. Za rozumný čas budeme v našom prípade považovať 2 sekundy, pretože akékoľvek väčšie zdržanie bota je v akčnej hre neprípustné.

U väčšiny botov sa tento problém rieši rôznymi heuristikami, ktorých riešenia v praxi nemajú od optimálneho príliš ďaleko. Často je ale ťažké povedať, ktorá heuristika je lepšia, preto je vhodné mať možnosť si ich otestovať a porovnať výsledky.

4.5.1.2 Základný popis mapy

Pretože v tejto časti budeme testovať pokročilejšie funkcie botovej navigácie, bude mať mapa, ako už jej názov napovedá, bludiskový charakter (viď obrázok č. 4). Aby sme však donútili bota k zbieraniu čo najväčšieho počtu predmetov rozmiestnených po mape, je potrebné ju správne navrhnúť.



Obr. 4: Pohľad na mapu typu MAZE (Copyright © 2010 Epic Games)

Idea mapy bude v tom, že každú sekundu ubudne botovi určitý počet životov. Na mape budú rozmiestnené iba lekárničky a bot ich bude musieť začať zbierať aby prežil (takéto chovanie je očakávané). Vhodnou mierkou úspešnosti bota môže byť čas, po ktorý prežil.

Do bludiska je taktiež možné pridať rôzne prekážky, s ktorými si bot bude musieť poradiť (napr. jazierka s lávou alebo kyselinou, výťahy, atď.). Užívateľ tak získa zároveň možnosť otestovať bota v interakcii s týmito objektmi.

4.5.1.3 Proces generovania mapy

V súbore *MazeGenerator.ini*, má užívateľ možnosť nastaviť hodnoty nasledujúcim premenným:

```
CellBreadth, CellWidth, CellHeight
LevelBreadth, LevelWidth, LevelHeight
WallHeight, WallThickness
Connectivity
NumPlayerStarts
NumPools
```

Prvým krokom v procese generovania mapy je navrhnutie samotnej geometrie, ktorá bude v našom prípade bludiskom. Základnou požiadavkou na toto bludisko bude, aby z každého jeho miesta existovala aspoň jedna cesta do ľubovoľného iného miesta na mape (teda malo by byť súvislé). Samozrejme nie je vylúčené, že by týchto ciest nemohlo existovať viac. Druhou požiadavkou bude, aby tieto cesty boli dostatočne kľukaté a netriviálne (teda aby to skutočne bolo bludisko).

Pre jednoduchosť si za základ bludiska vezmime kváder obsahujúci `LevelBreadth` buniek na šírku, `LevelWidth` buniek na dĺžku a `LevelHeight` buniek na výšku (teda počet poschodí bludiska). Pod pojmom bunka rozumieme prázdny priestor v tvare kvádra o rozmeroch `CellBreadth x CellWidth x CellHeight.` Našou úlohou bude oddeliť susedné bunky v kvádri stenami tak, aby následne vytvorili medzi sebou bludisko, po ktorom sa bude bot pohybovať. Výšku stien je možné definovať premennou `WallHeight`, pričom jej hodnota môže byť rôzna od hodnoty premennej `CellHeight`, ak nechceme aby steny siahali až po strop. Hrúbka stien je definovaná premennou `WallThickness`.

Pre jednoduchosť budeme bludisko na každom poschodí generovať zvlášť a na záver prepojíme susedné poschodia výťahmi. Proces generovania bludiska na jednom poschodí je plne náhodný, pričom užívateľ má možnosť pomocou premennej `Connectivity` ovplyvniť ako veľmi má byť toto bludisko spojité (tzn. koľko stien sa do bludiska pridá). Hodnota `0.0` znamená silne nespojité bludisko, kedy medzi ľubovoľnými dvoma bunkami vedie práve jedna cesta; hodnota `1.0` znamená bludisko bez stien. Hodnoty medzi sú lineárne interpolované medzi tieto dva extrémy.

Algoritmus generovania bludiska je odvodený od Kruskalovho algoritmu hľadania najlacnejšej kostry v ohodnotenom grafe [17] a prebieha nasledovne:

1. vlož stenu medzi každú dvojicu susedných buniek
2. ulož všetky steny do poľa **S**
3. ak je pole **S** prázdne, choď na krok 8., inak choď na krok 4.
4. z poľa **S** vyber a odstráň náhodnú stenu oddeľujúcu bunky **b1** a **b2**
5. ak neexistuje cesta z **b1** do **b2**, tak stenu zničˇ
6. inak s pravdepodobnosťou **Connectivity** stenu zničˇ
7. choď na krok 3.
8. koniec

Výsledok tohto algoritmu je súvislé bludisko. Ak by totiž nebolo súvislé, tak by to znamenalo, že neexistuje cesta medzi nejakými bunkami `b1` a `b2`. Bez ujmy na všeobecnosti, nech sú tieto bunky susedné. Potom stena, ktorá ich oddeľuje musela byť v nejakom kroku nášho algoritmu spracovaná. Ale podľa podmienky č. 5, by táto stena mala byť zničená, čo je spor.

Taktiež vidíme, že premenná `Connectivity` má silný vplyv na počet stien, umiestnených v bludisku.

V prípade, že má bludisko viac ako jedno poschodie, budú do vygenerovaných priestorov vložené výťahy spájajúce susedné poschodia. Na každých sto buniek poschodia pripadne jeden výťah, ktorého pozícia je zvolená náhodne. Ak je pozícia výťahu na dvoch susedných poschodiach zvolená rovnako, tak sa vygeneruje len jeden výťah jazdiaci medzi viacerými susednými poschodiami.

Premenná `NumPlayerStarts` označuje počet miest na mape, na ktorých je možné aby sa hráč na začiatku hry objavil. Premenná `NumPools` označuje počet jazierok na mape, zaberajúcich plochu celej jednej bunky. Jazierka môžu byť naplnené buďto vodou, kyselinou alebo lávou, pričom každá z možností je rovnako pravdepodobná. Pozície týchto objektov, ako aj objektov definovaných súborom *UT2004MapsGenerator.ini* sa určia náhodne z množiny voľných buniek, pričom na každej bunke môže byť maximálne jeden objekt. V prípade, že je počet objektov väčší ako počet voľných miest na mape, prednostne sa umiestnia dôležitejšie objekty.

Na záver sa na voľné bunky pridajú navigačné body, ktoré bude bot pri navigácii využívať, rovnomerne po celej mape sa rozmiestni osvetlenie a pre celú mapu sa definuje fyzika, ktorú si užívateľ zvolil.

4.5.2 Mapa typu LAVA

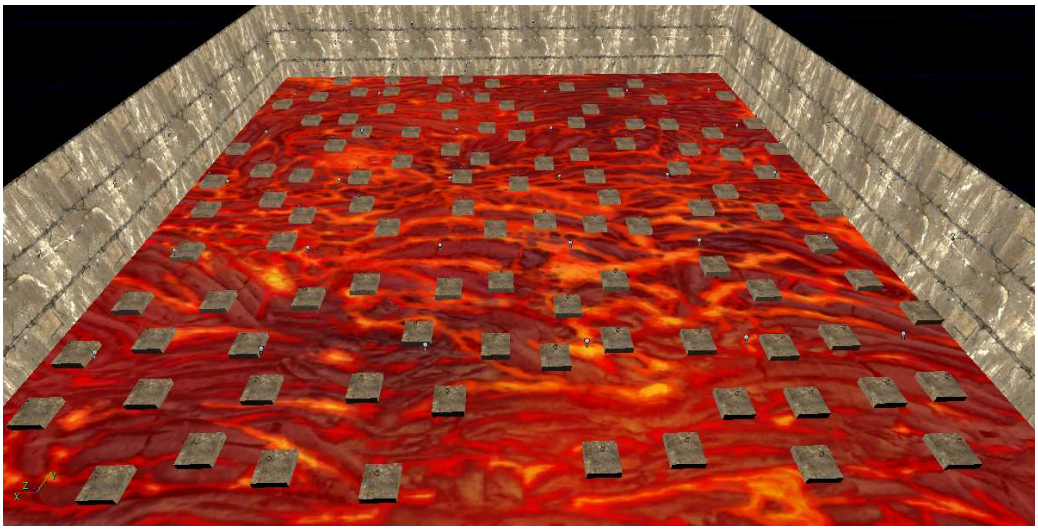
4.5.2.1 Testovaný aspekt bota

Druhým kľúčovým aspektom bota, ktorý sme identifikovali, bola schopnosť prispôsobiť sa dopredu neznámym podmienkam. Túto vlastnosť budeme testovať na schopnosti prispôsobiť sa neznámej sile gravitácie. Samozrejme nie je cieľom práce požadovať po botoch, aby sa prispôsobovali práve gravitácii. Ide skôr o to, poukázať na problém prispôsobovania sa obecne a práve ten po botoch vyžadovať. Gravitácia je len jeden z mnohých príkladov, pri ktorých je potrebné sa prispôsobiť.

4.5.2.2 Základný popis mapy

Mapa vytvorená za účelom testovania botovej schopnosti prispôsobovať sa je typu lava a na prvý pohľad pôsobí až veľmi triviálne. No práve v jej jednoduchosti je krása. Celú mapu tvorí jedna obrovská miestnosť, ktorej podlaha je pokrytá lávou a miestami z nej vytŕčajú malé ostrovčeky, na ktorých môže bot bezpečne stáť a medzi ktorými môže preskakovať (viď obrázok č. 5).

Aby sme donútili bota k pohybu, je možné použiť techniku ubúdania životov a lekárničiek, spomenutú v predchádzajúcom oddiely. Každopádne budeme po botovi požadovať, aby sa pokúšal preskakovať medzi ostrovčekmi. V závislosti od sily gravitácie bude bot schopný doskočiť na rôzne veľké vzdialenosti a vďaka procesu učenia by mal byť schopný nájsť presnú hodnotu vzdialenosti, na ktorú dokáže doskočiť.



Obr. 5: Pohľad na mapu typu LAVA (Copyright © 2010 Epic Games)

Vhodnou mierkou úspešnosti učenia sa bota, môže byť prejdená vzdialenosť v závislosti od počtu úmrtí bota. Istou nevýhodou tejto mierky môže byť to, že bot donekonečna preskakuje medzi dvoma susednými ostrovčekmi, u ktorých ma istotu, že vzdialenosť medzi nimi preskočiť dokáže. Na tento prípad je potrebné dať si pozor.

4.5.2.3 Proces generovania mapy

V súbore *LavaGenerator.ini*, má užívateľ možnosť nastaviť hodnoty nasledujúcim premenným:

```
BlockBreadth, BlockWidth, BlockHeight
LevelBreadth, LevelWidth, LevelHeight
MinimumDistance
NumPlayerStarts
```

Podobne, ako v prípade mapy typu maze, je prvým krokom vygenerovanie samotnej geometrie. Tá je v tomto prípade jednoduchšia a začína tým, že sa vytvorí obrovská miestnosť kvádrového tvaru o rozmeroch `LevelBreadth x LevelWidth x LevelHeight.` Následne sa vytvoria ostrovčeky, ktoré budú rozmiestnené po celej podlahe miestnosti. Každý z týchto ostrovčekov bude taktiež kvádrového tvaru a bude mať rozmery presne `BlockBreadth x BlockWidth x BlockHeight`, pričom hladina lávy na záver bude v troch štvrtinách výšky ostrovčeka. Pre lepšiu kontrolu nad testovacím prostredím bude platiť, že vzdialenosť medzi stredmi ľubovoľných dvoch ostrovčekov bude aspoň `MinimumDistance.` Takto

bude mať užívateľ možnosť vygenerovať si prostredie, kedy sú ostrovčeky tesne vedľa seba, ako aj prostredie, kedy sú od seba ďaleko.

Presné polohy ostrovčekov sú volené s prvkom náhody a teda zakaždým iné. Algoritmus, ktorý ich generuje sa dá popísať nasledovne:

```
1. pole P je na začiatku prázdne
2. premenná pokus je na začiatku 50
3. ak pokus == 0, tak choď na krok 9.
4. pokus := pokus-1
5. vygeneruj náhodné x∈ <BlockBreadth/2,
         LevelBreadth-BlockBreadth/2>
6. vygeneruj náhodné y∈ <BlockWidth/2,
         LevelWidth-BlockWidth/2>
7. ak je pozícia [x,y] vzdialená od všetkých pozícií v poli
   P aspoň o MinimumDistance, tak
   7.1.  ulož [x,y] do P
   7.2.  pokus := 50
8. choď na krok 3.
9. koniec
```

Premenná `NumPlayerStarts` slúži, podobne ako na mape typu maze, k určeniu počtu štartovných pozícií pre hráča a botov. Tieto pozície spolu s pozíciami objektov definovaných súborom *UT2004MapsGenerator.ini* sa určia náhodne na pozíciách ostrovčekov, pričom na jednom ostrovčeku je povolený maximálne jeden objekt. Následne sa na voľné ostrovčeky pridajú navigačné body, ktoré bude bot využívať. Na záver sa do mapy umiestni láva, pokrývajúca celú podlahu, rovnomerne po celej mape sa rozmiestni osvetlenie a definuje sa fyzika, ktorá bude na mape platiť.

4.5.3 Mapa typu WATER

4.5.3.1 Testovaný aspekt bota

V časti 4.5.1 sme si ukázali, že v oblasti plánovania cesty bota je stále čo zlepšovať. V tejto časti sa ešte raz pozrieme na tento problém, ale v inom kontexte. Tentoraz budeme po botoch požadovať, aby si nie len naplánovali optimálnu trasu po zadaných objektoch, ale aby si aj naplánovali, po ktorých objektoch sa majú vydať.

Konkrétne postavíme bota pred problém dostať sa z miesta A do miesta B. Väčšina botov v takomto momente nájde vzdialenostne najkratšiu cestu a vydá sa po nej. Problém nastane v momente, kedy botovi postupne ubúdajú životy a je nutné aby sa bot cestou zastavil v mieste C, kde si ich doplní. Ak je bot C dostatočne ďaleko od bodov A a B, väčšina botov ho ignoruje a úlohu tak nedokážu splniť.

4.5.3.2 Základný popis mapy

Mapa typu water je vo viacerých veciach odlišná od predchádzajúcich dvoch máp. Prvou veľkou odlišnosťou je to, že zatiaľ čo predchádzajúce dve mapy fungujú v hernom móde Deathmatch, kedy je úlohou botov zabiť čo najviac súperov za stanovený časový limit, táto mapa funguje v hernom móde Capture the flag, kedy sú boti rozdelení do dvoch družstiev (modré a červené družstvo) a ich úlohou je ukradnúť súperovu vlajku a doniesť ju na vlastnú základňu. Druhou veľkou odlišnosťou je, že sa pri vytváraní mapy nevyužíva prvok náhody a teda za rovnakých parametrov sa vygeneruje vždy rovnaká mapa. Je to z toho dôvodu, že táto mapa v sebe implementuje určitý scenár, ktorého sa budú musieť boti držať, aby uspeli a prvok náhody by tento scenár mohol negatívne ovplyvniť.

Mapa samotná je tvorená dlhou chodbou, na ktorej protiľahlých koncoch sa nachádzajú základne a vlajky súperiacich tímov. Úlohou botov je teda prebehnúť touto chodbou z jednej strany na druhú a zase späť. Problém je v tom, že podlaha chodby je z veľkej časti pokrytá vodou, ktorá pri dotyku uberá botom životy. Pri správnom nastavení parametrov mapy je toto zranenie natoľko veľké, že boti nemajú šancu takto priamočiaro úlohu splniť. Aby uspeli, musia si počas svojej cesty urobiť, v určitom zmysle neoptimálnu, zachádzku, na ktorej konci nájdu životy a práve tie um umožnia úlohu úspešne dokončiť (viď obrázok č. 6).

Obr. 6: Pohľad na mapu typu WATER (Copyright © 2010 Epic Games)

Vhodnou mierkou úspešnosti bota môže byť dosiahnuté skóre, teda počet úspešných prenesení vlajok.

4.5.3.3 Proces generovania mapy

V súbore *WaterGenerator.ini*, má užívateľ možnosť nastaviť hodnoty nasledujúcim premenným:

```
LevelWidth, LevelHeight
ShaftLength
CorridorLength
DamagePerSec
```

Pretože sa tento typ mapy generuje bez prvku náhodnosti, je proces generovania jeho geometrie priamočiary a jedná sa iba o nastavenia parametrov. Hlavná chodba a vedľajšie chodbičky so životmi majú na šírku `LevelWidth` jednotiek a na výšku `LevelHeight` jednotiek. Ostrovčeky, na ktorých sa nachádzajú vlajky a životy zaberajú vždy celú šírku chodby a na dĺžku majú `islandBreadh` jednotiek (hodnotu tejto premennej nie je možné meniť). Premenná `CorridorLength` určuje vzdialenosť medzi ostrovčekom s vlajkou a najbližšej chodbičky so životmi a taktiež vzdialenosť medzi dvoma chodbičkami so životmi. Celkovú dĺžku hlavnej chodby teda môžeme vypočítať podľa vzorca:

```
3 * CorridorLength + 2 * islandBreadh + 2 * LevelWidth
```

Premenná `ShaftLength` určuje vzdialenosť vchodu do chodbičky so životmi až po samotný ostrovček so životmi. Celkovú dĺžku vedľajšej chodby teda môžeme vypočítať podľa vzorca:

```
ShaftLength + islandBreadh
```

V ďalšom kroku sa do mapy umiestni voda, ktorej sa nastaví aby pri dotyku uberala botovi `DamagePerSec` životov za sekundu. Na záver sa na presne určené miesta vygenerujú vlajky súperiacich tímov, životy do vedľajších chodbičiek, osvetlenie a navigačné body, definuje sa fyzika a mapa je tak pripravená na použitie.

4.6 Program observer

4.6.1 Základný popis

Druhou dôležitou súčasťou testovacieho prostredia je mechanizmus na samotné vyhodnocovanie testovania. Ako výstup tohto mechanizmu si ideálne predstavujeme jedno číslo alebo názorný graf, z ktorého bude možné

jednoducho vyčítať, ako si testovaný bot v experimente viedol. Z dôvodu príliš komplexného výsledku sme si v našej práci zvolili graf.

Základný princíp fungovania vyhodnocovacieho mechanizmu je jednoduchý – v priebehu testovania bude sledovať bota (preto názov „observer“) a v krátkych časových intervaloch (napr. každú sekundu) zaznamenávať do súboru namerané hodnoty vybraných parametrov bota. Je dôležité vybrať takú množinu sledovaných parametrov, aby po ich znázornení do grafu vypovedali o určitom chovaní bota v priebehu času, resp. aby bolo možné posúdiť, či sa bot počas testovania choval správne alebo nesprávne.

Pre naše potreby sme si zvolili nasledujúcu množinu parametrov:

- Počet životov bota (pre mapu typu maze)
- Počet úmrtí bota (pre mapu typu lava)
- Prejdená vzdialenosť bota (pre mapu typu lava)
- Dosiahnuté skóre bota (pre mapu typu water)

Samozrejme nie je problém v prípade potreby doplniť ďalšie.

4.6.2 Implementácia

Pôvodná predstava o programe observer bola, že by sa jednalo o samostatný program, ktorý by sa spúšťal spolu so serverom, na ktorom beží testovacia mapa. Observer by tak sledoval a vyhodnocoval súčasne všetkých botov na danej mape. V čase písania práce však platforma Pogamut tento postup nepodporovala, preto sme museli zvoliť menej elegantné riešenie implementácie.

Program observer je v aktuálnej podobe implementovaný ako vnútorná trieda bota, ktorá sleduje len toho jedného bota. Ak by teda užívateľ chcel vyhodnotiť chovanie nejakého bota, musel by do hlavnej triedy bota vložiť vnútornú triedu *BotObserver*, vo funkcii *botSpawned()* vytvoriť objekt tejto triedy a vo funkcii *logic()* zavolať funkciu observera *update()*. Pri vytváraní objektu observera má užívateľ možnosť nastaviť veľkosť časového intervalu (v sekundách), v ktorom majú byť namerané hodnoty zaznamenávané do súboru.

Výstupom observera je súbor *meno_bota.csv*, obsahujúci namerané údaje ako tabuľku. Súbor *.csv* je podporovaný väčšinou tabuľkových procesorov (napr. MS Excel), v ktorých si následne môžeme nechať namerané hodnoty vykresliť do grafu. Formát súboru si ukážeme na príklade:

Time	Health	Deaths	Distance
0,28	94	0	1,39
2,35	82	0	1,39
3,45	76	0	257,93
3,73	76	0	377,95
4	76	0	498,48
4,27	70	0	618,99
4,55	70	0	739,5
4,82	70	0	860,51
5,1	70	0	982
5,37	64	0	1102,99
5,65	64	0	1221,32

4.7 Zhrnutie

V tejto kapitole sme navrhli a vytvorili tri typy rôznych testovacích prostredí, na ktorých budeme botov testovať. Tieto prostredia otestujú jednak botovu schopnosť prispôsobovať sa neznámym podmienkam a jednak schopnosť riešiť netriviálne problémy. Vďaka nástroju UT2004MapsGenerator máme možnosť tieto prostredia v mnohých ohľadoch parametrizovať a vytvárať tak nové jedinečné podmienky na testovanie. Zároveň vďaka prvku náhodnosti, ktorý sa využíva pri generovaní týchto máp, máme zadarmo k dispozícii celú paletu odlišných máp s podobnými vlastnosťami.

Výsledok každého testovania bota je programom observer zaznamenaný a uložený v externom súbore s príponou *.csv* vo forme tabuľky, ktorú si následne ľubovoľným tabuľkovým procesorom môžeme nechať vykresliť do grafu. V ďalšej kapitole si na referenčných botoch ukážeme, ako namerané výsledky interpretovať.

Cieľ č. 2 tak bol úspešne splnený.

5. Kapitola

Tvorba referenčných botov

V tejto kapitole vytvoríme pre každé testovacie prostredie, navrhnuté v 4. kapitole, referenčného bota, ktorý bude slúžiť jednak na demonštračné účely jednotlivých prostredí a jednak bude slúžiť ako základ pre budúci vývoj v tejto oblasti (cieľ č. 3).

Referenční boti budú naprogramovaní nad, už spomínanou, platformou Pogamut. Každý referenčný bot bude v sebe implementovať algoritmus riešenia zadaného problému, ktorý bude v čo najväčšej miere použiteľný aj pre iné situácie (tzn. nebude samoúčelný). Samotní boti však kvôli jednoduchosti samoúčelní budú – tzn. nebudú použiteľní na iných, ako prezentovaných, mapách. Zároveň v sebe nebudú implementovať iné, než len nevyhnutne potrebné mechanizmy – napr. nebudú vedieť bojovať.

5.1 Referenčný bot maze

5.1.1 Použitý algoritmus

Úlohou referenčného bota typu maze bude v bludisku plnom lekárničiek, v ktorom mu každú sekundu ubudne niekoľko životov, prežiť čo najdlhšie.

Ako referenčné riešenie tohto problému sme si zvolili nájdenie optimálnej trasy (tnz. optimálneho riešenia problému obchodného cestujúceho), ktoré pre N objektov dokáže nájsť optimálnu trasu v čase $O(N^2 2^N)$. Tento algoritmus je bližšie popísaný v [18] a využíva metódu dynamického programovania. Pre $N \leq 20$ beží tento algoritmus menej ako 2 sekundy, čo je podľa našej definície rozumný čas.

Keď sa náš referenčný bot po prvýkrát objaví na mape, z celého navigačného grafu si vypočíta algoritmom Floyd-Warshall [19] maticu najkratších ciest medzi ľubovoľnými dvoma navigačnými bodmi, pričom ako cenu navigačnej hrany medzi dvoma bodmi berie vzdialenosť týchto dvoch bodov. Následne predá tento graf triede *TSPSolver*, ktorá nájde optimálne riešenie problému obchodného cestujúceho. Bot si toto riešenie uloží, nájde trasu k najbližšej lekárničke a po jej zobratí pokračuje podľa nájdeného riešenia stále ďalej po cykle. Až bot dorazí na koniec, tak pokračuje v prechádzaní cyklu od začiatku.

Zároveň v sebe bot implementuje jednoduchý detekčný mechanizmus, ktorým rozpozná, či lekárničku skutočne zobral. Problém je totiž v tom, že ak by bol cyklus lekárničiek príliš krátky, tak by bot dorazil k už navštívenej lekárničke skôr, ako by sa stihla obnoviť.

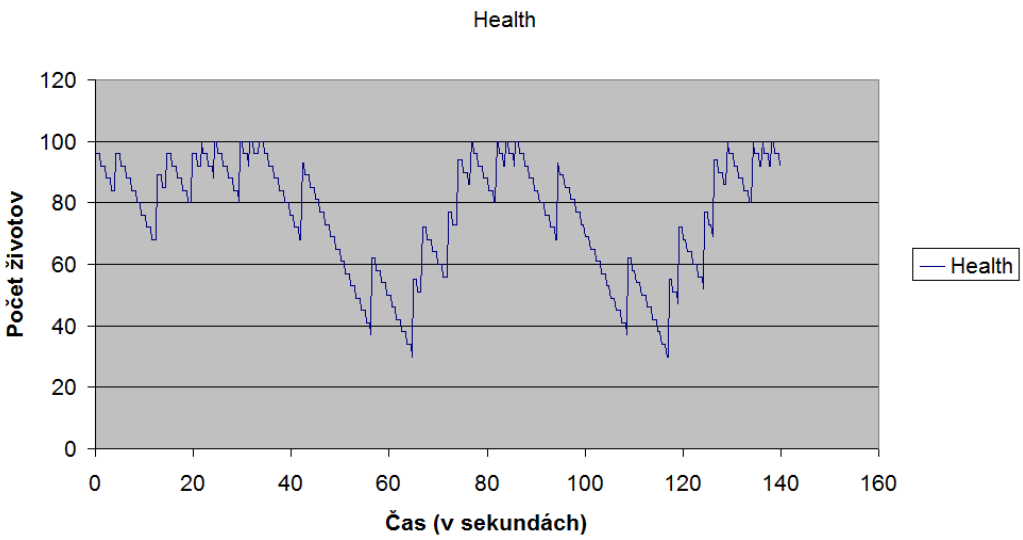
Bot v sebe neimplementuje mechanizmy potrebné na správnu interakciu s výťahom prípadne jazierkom, ktoré sa na mape môže vyskytnúť. Odporúča sa preto testovať bota iba na mapách bez týchto objektov.

5.1.2 Výsledok testovania

Pomocou nástroja UT2004MapsGenerator si necháme vygenerovať mapu typu maze s nasledujúcimi parametrami:

- mapa má 1 poschodie a rozmery 10x10 buniek
- na mape je 10 lekárničiek (lekárnička pridá botovi 25 životov)
- spojitosť mapy (connectivity) je 0.5
- každú sekundu ubudnú botovi 4 životy

Na takto vygenerovanej mape necháme bota dva a pol minúty testovať a výsledok testovania si necháme vykresliť do grafu (viď obrázok č. 7), ktorý zobrazuje počet životov bota v závislosti od času.



Obr. 7: Graf úspešnosti bota typu MAZE

Z takto získaného grafu môžeme vidieť ako úspešný náš použitý algoritmus bol, aké sú jeho slabé a silné stránky. Napríklad si všimnime, že náš použitý algoritmus je optimálny v tom smere, že bot po dobehnutí celého cyklu skončí na 100 životoch a teda vydrží teoreticky na mape bežať donekonečna. Táto optimálnosť samozrejme nie je pravidlom a existujú mapy, na ktorých to neplatí. Druhú vec, ktorú si môžeme všimnúť je, že algoritmus neberie do úvahy vzdialenosť medzi dvoma po sebe idúcimi lekárničkami na trase a preto môžeme na grafe pozorovať veľký pokles v 40. a 100. sekunde testovania. Tento pokles je spôsobený priveľkou vzdialenosťou medzi dvoma

lekárničkami a v prípade, že by bola o trochu väčšia, bola by pre bota osudnou. Preto by bol v tomto prípade výhodnejší algoritmus, ktorý by túto vzdialenosť do úvahy bral.

5.2 Referenčný bot lava

5.2.1 Použitý algoritmus

Úlohou referenčného bota typu lava bude na mape pokrytej lávou naučiť sa efektívne preskakovať medzi ostrovčekmi vytŕčajúcimi z lávy za predpokladu, že bot nepozná silu gravitácie, ktorá je na mape definovaná.

Ako referenčné riešenie tohto problému sme si zvolili veľmi elegantný a jednoduchý proces učenia, kedy si bot pamätá interval vzdialeností <`minDist`, `maxDist`>, ktorý je pre neho bezpečný na skákanie. V podstate znamená, že ak sa bot pokúsi skočiť na kratšiu vzdialenosť ako `minDist` alebo na dlhšiu vzdialenosť ako `maxDist`, tak spadne do lávy (buďto cieľ preskočí alebo k nemu nedoskočí).

Pretože ma bot k dispozícii dva druhy skokov – jednoduchý a dvojitý (používa sa na väčšiu vzdialenosť), bude si takéto intervaly pamätať dva:

```
< minJumpDist, maxJumpDist >
< minDoubleJumpDist, maxDoubleJumpDist >
```

pričom vieme, že platí:

$$\texttt{maxJumpDist} \leq \texttt{maxDoubleJumpDist}$$
$$\texttt{minJumpDist} \leq \texttt{minDoubleJumpDist}$$

Na začiatku testovania majú `minJumpDist` a `minDoubleJumpDist` hodnotu `0` a `maxJumpDist` a `maxDoubleJumpDist` hodnotu 10^{12} (dostatočne veľká hodnota). V každom ďalšom kroku testovania si bot náhodne zvolí typ skoku ktorý použije a následne ostrovček, na ktorý sa pokúsi doskočiť, pričom požadujeme, aby vzdialenosť k tomuto ostrovčeku bola v intervale bezpečnej vzdialenosti pre daný typ skoku. Ak neexistuje žiaden ostrovček v bezpečnej vzdialenosti, pokúsi sa bot použiť druhý typ skoku. V prípade, že ani pre druhý typ skoku neexistuje ostrovček v bezpečnej vzdialenosti, výpočet končí a bot ostane stáť na mieste.

V opačnom prípade sa bot pokúsi na vytýčený cieľ doskočiť. Jednoduché detekčné mechanizmy výsledok skoku vyhodnotia. Podľa toho či bot na cieľ doskočil alebo ho preskočil alebo k nemu nedoskočil upravia hodnoty premenných a celý cyklus sa zopakuje.

Do bota sú ďalej implementované mechanizmy sledujúce, či sa bot pri výskoku odrazil úspešne, pretože sa stáva, že sa bot pri rozbehu nestihne

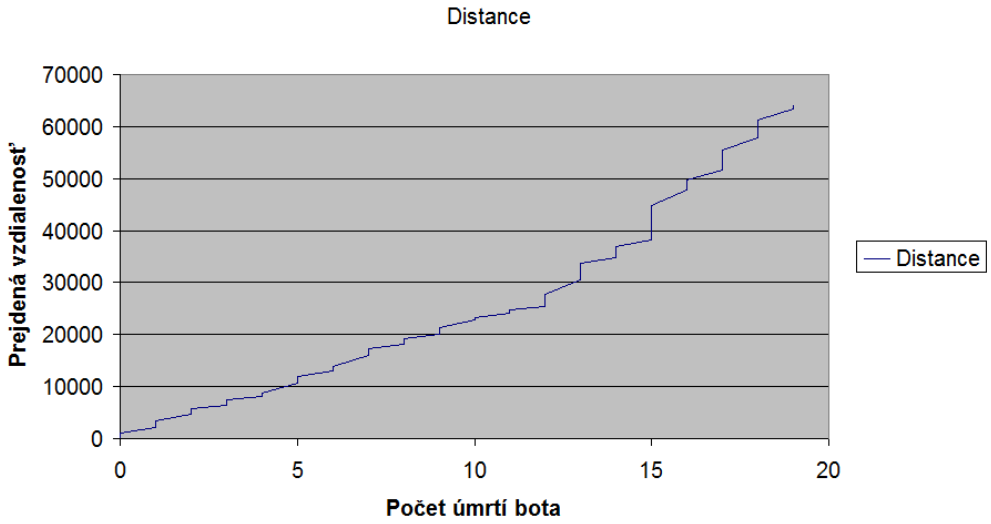
odraziť a spadne do lávy. Samozrejme nechceme, aby takéto pokusy ovplyvňovali proces učenia.

5.2.2 Výsledok testovania

Pomocou nástroja UT2004MapsGenerator si necháme vygenerovať mapu typu lava s nasledujúcimi parametrami:

- sila gravitácie bude 5000 (tzn. veľmi silná gravitácia)
- rozmery ostrovčekov budú 150x150
- minimálna vzdialenosť stredov dvoch ostrovčekov bude 216

Na takto vygenerovanej mape necháme bota tri a pol minúty testovať a výsledok testovania si necháme vykresliť do grafu (viď obrázok č. 8), ktorý zobrazuje prejdenú vzdialenosť v závislosti od počtu úmrtí bota:



Obr. 8: Graf úspešnosti bota typu LAVA

Tento graf je tvorený schodíkmi, u ktorých zmena výšky reprezentuje vzdialenosť prejdenú botom bez úmrtia. Pretože na testovacej mape nie je iná možnosť pohybu ako preskakovanie medzi ostrovčekmi, môžeme usúdiť, že čím väčšiu vzdialenosť bot bez úmrtia prejde, tým lepšie sa prispôsobil aktuálnej sile gravitácie. V nameranom grafe si môžeme všimnúť, že úvode testovania je výška týchto schodíkov minimálna, čo znamená, že bot skoro pri každom pokuse o preskok medzi ostrovčekmi skončil v láve. Pri 12-tom úmrtí však môžeme pozorovať výraznú zmenu vo výške schodíkov a pri 15-tom úmrtí dokonca výnimočne veľkú prejdenú vzdialenosť. V podstate to môžeme považovať za moment, kedy sa bot úspešne prispôsobil sile gravitácie. Každé ďalšie úmrtie je zväčša spôsobené tým, že sa botovi nepodarilo odraziť a spadol do lávy nechtiac.

Jedno z možných vylepšení uvedeného algoritmu by bolo urýchliť proces učenia. Zatiaľ čo sa referenčný bot učí na náhodných preskokoch, cieleným učením by sa celý proces urýchlil.

5.3 Referenčný bot water

5.3.1 Použitý algoritmus

Ako referenčné riešenie tohto problému sme si zvolili, na akčnú hru veľmi neobvyklý, prístup. Formulujeme botovu úlohu ako plánovací problém, ktorý následne necháme vyriešiť obecným plánovačom [20]. Nájdený výsledok interpretujeme a budeme postupovať podľa neho. Podobný prístup riešenia problémov bol už v minulosti na botoch použitý v počítačovej hre F.E.A.R. [21], ktorá sa svojím charakterom veľmi podobá hre UT2004.

Teraz si zbežne popíšeme, ako taký plánovací problém vyzerá. Detaily jeho následného vyriešenia presahujú rámec tejto práce, ale sú k dispozícii v [20].

Vo svete plánovacieho problému existujú takzvané plánovacie premenné, ktoré môžu nadobúdať celočíselné hodnoty z nejakého dopredu zvoleného intervalu (každá premenná má svoj vlastný interval). Ďalej existuje takzvaný počiatočný stav, ktorý určuje hodnoty premenných na začiatku simulácie a koncový stav, ktorý definuje cieľové hodnoty stavových premenných, tzn. ktoré sa snažíme dosiahnuť. Priebeh simulácie je určený takzvanými prechodmi, ktoré transformujú hodnoty stavových premenných. Každý prechod je určený zoznamom a hodnotami stavových premenných, ktoré musia byť splnené, aby mohol byť prechod aplikovaný. Aplikácia prechodu zmení hodnotu niektorých stavových premenných (nastaví novú hodnotu / inkrementuje / dekrementuje o nejakú konštantu). Riešením plánovacieho problému rozumieme nájdenie takej postupnosti prechodov, ktorých postupná aplikácia transformuje počiatočný stav do koncový stavu, pričom v žiadnom kroku nebol aplikovaný nepoužiteľný prechod a zároveň hodnota plánovacej premennej nevybehla mimo definičný obor danej premennej.

Teraz si popíšeme ako z mapy typu water vytvoriť plánovací problém. Začneme definíciou stavových premenných:

- `botPosition` ∈ `<0, numNavPoints-1>` – definuje pozíciu bota na mape, ako index navigačného bodu, na ktorom sa bot nachádza (resp. ku ktorému smeruje). Pre výnimočné pozície, ako sú pozícia základne a vlajky budeme používať názvy `redStart` / `blueStart` a `redFlag` / `blueFlag`.
  Počiatočná hodnota: `botPosition` = `redStart` (predpokladáme, že bot bude v červenom tíme)

- `botHealth` ∈ `<1, 100>` - definuje počet životov bota.
  Počiatočná hodnota: `botHealth` = `100`

- `hasFlag` ∈ `<0, 1>` - definuje, či bot nesie súperovu vlajku. 1, ak áno; 0, ak nie.
  Počiatočná hodnota: `hasFlag` = `0`

- `takenMedKits` ∈ `<0, 15>` - definuje bitovú masku znamenajúcu, ktoré lekárničky už bot zobral. Budeme totiž problém formulovať tak, že na ceste po vlajku môže bot zobrať každú lekárničku maximálne raz. Na ceste späť sa táto premenná vynuluje a bot tak môže opäť zbierať lekárničky. Pretože sú na mape rozmiestnené 4 lekárničky, môže táto premenná nadobúdať 2^4 rôznych hodnôt. N-tý bit je zapnutý práve vtedy, keď je daná lekárnička už zobratá.
  Počiatočná hodnota: `takenMedKits` = `0`

Keďže už počiatočný stav popísaný máme, definujeme teraz koncový stav:

- `botPosition = redFlag` – bot musí byť na základni svojho tímu
- `hasFlag = 1` – bot musí mať súperovu vlajku

Na záver si definujeme množinu prechodov. Ako bolo v úvode spomenuté, každý prechod je definovaný zoznamom a hodnotami stavových premenných, ktoré musia byť splnené, aby mohol byť prechod aplikovaný a vlastnou transformáciu stavových premenných.

- Pohyb bota po mape – pre každú dvojicu navigačných bodov `from` a `to`, medzi ktorými vedie navigačná hrana definujeme prechod nasledujúcim spôsobom.
  Podmienky: `botPosition = from`
  Transformácie: `botPosition = to`
  `botHealth -= damage` (v prípade, že nagivačná hrana vedie cez vodu).
  Presná hodnota `damage` silne závisí od množstva životov, ktoré voda za sekundu uberá a pretože sa jedná o hrubú aproximáciu, je potrebné s ňou trochu experimentovať.

- Zobratie vlajky – to je moment, kedy bot dorazil k nepriateľskej vlajke a má možnosť ju zobrať.
  Podmienky: `botPosition = blueFlag`
  `hasFlag = 0`
  Transformácie: `hasFlag = 1`
  `takenMedKits = 0` (na ceste späť má bot možnosť opäť zbierať lekárničky).

- Zobratie lekárničky – pre každú lekárničku na mape a každú možnú hodnotu bitovej masky zobratých lekárničiek musíme vytvoriť samostatný prechod. Zároveň musíme rozlišovať medzi situáciou kedy má bot menej ako 75 životov a zase naopak, kedy má 75 a viac životov. Je to z toho dôvodu, že lekárnička doplní botovi 25 životov, ale nikdy nie nad 100. Najprv si ukážeme prechod, kedy má bot menej ako 75 životov.

  Podmienky: `botPosition = chosenMedKitPosition`
  `takenMedKits = chosenBitmask`
  `botHealth ∈ <1, 74>`
  Transformácie: `botHealth += 25`

  Teraz si ukážeme prechod, kedy má bot aspoň 75 životov.

  Podmienky: `botPosition = chosenMedKitPosition`
  `takenMedKits = chosenBitmask`
  `botHealth ∈ <75, 100>`
  Transformácie: `botHealth = 100`

Takto pripravený plánovací problém podsunieme obecnému plánovaču, ktorý nám nájde riešenie. Botovu úlohou potom bude obehnúť mapu po naplánovanej trase a tým je úloha úspešne vyriešená.

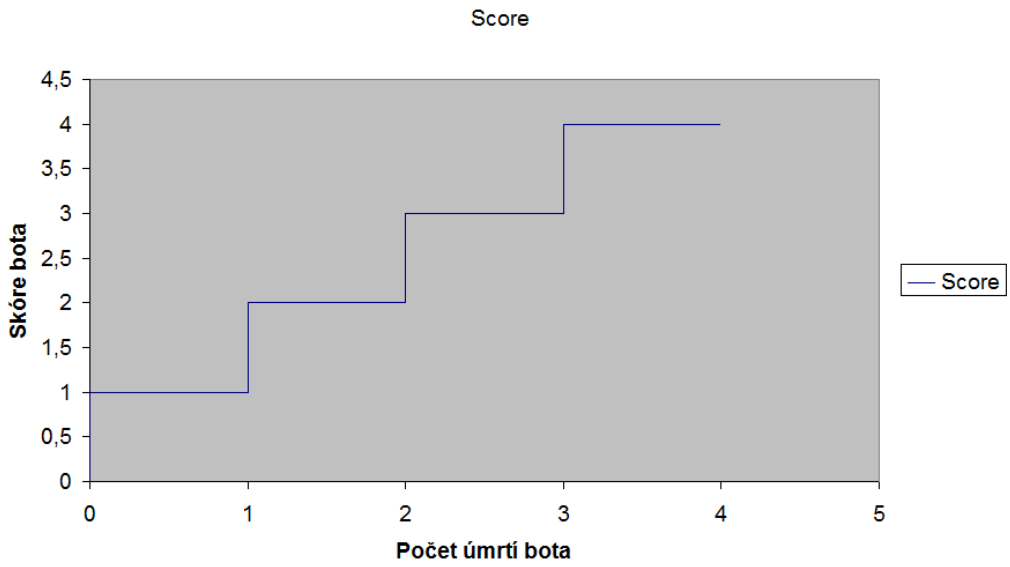
Plánovač, ktorý sme mali k dispozícii bol vytvorený doc. RNDr. Romanom Bartákom, Ph.D [22] a napísaný v jazyku Prolog. Preto bolo nutné vytvoriť interface, cez ktorý by prebiehala komunikácia medzi botom, naprogramovaným v jazyku Java, a samotným plánovačom. Pre viac informácií je tento interface spolu s botom a plánovačom k dispozícii na priloženom CD.

5.3.2 Výsledok testovania

Pomocou nástroja UT2004MapsGenerator si necháme vygenerovať mapu typu water s nasledujúcimi parametrami:

- voda uberie botovi každú sekundu 7 životov
- dĺžka koridoru (`CorridorLength`) je 768
- dĺžka bočnej chodbičky (`ShaftLength`) je 256

Na takto vygenerovanej mape necháme bota dve minúty testovať a výsledok testovania si necháme vykresliť do grafu (viď obrázok č. 9), ktorý zobrazuje skóre bota (koľkokrát zobral súperovi vlajku) v závislosti od počtu úmrtí bota:



Obr. 9: Graf úspešnosti bota typu WATER

Ako môžeme na grafe pozorovať, tak je bot schopný na jedno úmrtie jedenkrát ukradnúť súperovu vlajku. Zadanú úlohu je teda schopný úspešne splniť.

Na druhú stranu, konkrétna inštancia mapy, ktorú sme na testovanie bota použili vyžaduje, aby bot počas svojej cesty zobral aspoň dve lekárničky, teda urobil si jednu zachádzku. Veľmi jednoduchou heuristikou – ak máš málo životov, choď k najbližšej lekárničke – je väčšina botov túto inštanciu schopná vyriešiť tiež. Pri zvýšení počtu životov, ktoré voda uberá, je možné si vynútiť až štyri zachádzky počas cesty (dve cestou tam, dve cestou späť). Túto inštanciu problému však náš bot nie je schopný v rozumnom čase vyriešiť.

Obecný plánovač, ktorý referenčný bot využíva, nie je schopný v rozumnom čase nájsť riešenie nášho problému vyžadujúce viac ako 23 krokov. Pre inštanciu použitej mapy malo riešenie iba 18 krokov. Plánovaču robí taktiež veľké problémy hustý navigačný graf, pretože pre každú hranu grafu musíme vytvoriť plánovací prechod, čím sa stavový priestor nezadržateľne zväčšuje.

Idea použiť obecný plánovač na riešenie problémov v akčnej hre, akou UT200 je, bola skôr len proof of concept. Chceli sme dokázať prípadne vyvrátiť použiteľnosť tohto prístupu, ktorý sa spočiatku javil veľmi zaujímavý a inovatívny. Dospeli sme však k záveru, že bez akýchkoľvek ďalších optimalizácii je tento prístup nepoužiteľný pre zložitejšie problémy.

5.4 Zhrnutie

Referenční boti, ktorých sme v tejto kapitole vytvorili, názorne demonštrujú určité spôsoby riešenia predložených problémov. Zároveň sme sa snažili, aby algoritmy použité na riešenie problémov boli v čo najväčšej miere použiteľné aj v iných ako predložených situáciách – tzn. aby neboli samoúčelné.

Na vytvorených botoch bolo zároveň názorne predvedené, ako funguje testovacie prostredie a ako interpretovať získané výsledky. Cieľ číslo 3 sme tak úspešne naplnili.

6. Kapitola

Záver

V tejto práci sme sa zaoberali návrhom, tvorbou a použitím testovacích prostredí pre botov do hry Unreal Tournament 2004. V úvode sme úspešne identifikovali apekty botov, ktoré sú pre samotný vývoj a úspešnosť botov kľúčové a ktoré je možné automaticky testovať a vyhodnocovať. Následne sme si predstavili nástroj UT2004MapsGenerator, ktorý sme vytvorili za účelom automatického generovania parametrizovateľných testovacích prostredí a s ktorého pomocou sme vytvorili tri rôzne typy máp, na ktorých bude testovanie prebiehať. Na záver sme pre každý typ prostredia vytvorili referenčného bota, na ktorom sme demonštrovali použitie testovacích prostredí v praxi a ktorý taktiež poslúži ako referenčné riešenie, voči ktorému môžeme porovnávať ďalšie riešenia.

Výsledok tejto práce bude mať prínos pre tvorcov botov do už spomínanej súťaže Botprize. Zároveň použitie obecného plánovača do akčnej hry, ktoré sme skúsili v poslednej kapitole pri tvorbe bota typu water, vnieslo určité svetlo na tento netradičný prístup, aj napriek tomu, že výsledok bol skôr negatívny.

V blízkej budúcnosti sa plánuje prerobenie komponenty observer, ktorá by sa stala nezávislým externým programom a bolo by tak možné botov testovať aj bez explicitného zásahu do ich zdrojového kódu.

Literatúra

[1] Epic games: http://www.epicgames.com/, 5. 8. 2010

[2] Unreal Tournament 2004: http://www.unrealtournament2003.com/, 5. 8. 2010

[3] Brom C.: Dizertačná práca, *Řízení virtuálních lidí ve velkých virtuálních světech*, MFF-UK, Praha, 2006

[4] Platforma pogamut: http://diana.ms.mff.cuni.cz/main/tiki-index.php, 5. 8. 2010

[5] Vývojové prostredie NetBeans: http://netbeans.org/, 5. 8. 2010

[6] Konferencia CIG: http://www.ieee-cig.org/, 5. 8. 2010

[7] Konferencia CIG 2010: http://game.itu.dk/cig2010/, 5. 8. 2010

[8] Súťaž Botprize: http://www.botprize.org/, 5. 8. 2010

[9] Turingov test: http://en.wikipedia.org/wiki/Turing_test, 5. 8. 2010

[10] Súťaž Botprize 2008: http://botprize.org/2008.html, 5. 8. 2010

[11] Tencé F., Buche C.: *Automatable evaluation method oriented toward behaviour believability for video games*, Université Européenne de Bretagne, France, 2008

[12] Chen K., Liao A., Pao H., Chu H.: *Game Bot Detection Based on Avatar Trajectory*, Springer Berlin, 2008

[13] Súťaž Botprize 2009: http://botprize.org/2009.html, 5. 8. 2010

[14] Víťaz súťaže Botprize 2009: http://code.google.com/p/sqlitebot/wiki/MapGenerator, 5. 8. 2010

[15] Busby J., Parrish Z., VanEenwyk J.: *Mastering Unreal Technology: The Art of Level Design*, Sams, 2004

[16] Problém obchodného cestujúceho: http://en.wikipedia.org/wiki/Travelling_salesman_problem, 5. 8. 2010

[17] Joseph. B. Kruskal: *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem*, *Proceedings of the American Mathematical Society*, Vol 7, No. 1 (Feb, 1956), pp. 48–50

[18] Bellman, R.: *Combinatorial Processes and Dynamic Programming*, Combinatorial Analysis, Proceedings of Symposia in Applied Mathematics 10, American Mathematical Society, pp. 217–249.

[19] Algoritmus Floyd-Warshall: http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm, 5. 8. 2010

[20] Publikácie o plánovaní: http://ktiml.mff.cuni.cz/~bartak/html/publications.html, 5. 8. 2010

[21] Orkin J.: *Three States and a Plan: The A.I. of F.E.A.R.*, Monolith Productions / M.I.T. Media Lab, Cognitive Machines Group, 2006

[22] Stránky Romana Bartáka: http://kti.mff.cuni.cz/~bartak/index.html, 5. 8. 2010

Príloha

Obsah priloženého CD

Na priloženom CD užívateľ nájde nasledujúce zložky a súbory:

.. - obsahuje podrobný návod na inštaláciu a spustenie platformy Pogamut, testovacích prostredí a referenčných botov

..\Binary\ReferenceBots - obsahuje spustiteľné *.jar* súbory všetkých referenčných botov

..\Binary\UT2004MapsGenerator - obsahuje spustiteľný *.exe* súbor generátoru náhodných máp spolu s užívateľskou dokumentáciou k jeho používaniu

..\Install files - obsahuje inštalačné súbory platformy Pogamut, použiteľné v prípade, že by najnovšia verzia dostupná na Internete nebola s aktuálne použitou kompatibilná

..\Projects - obsahuje projekty a zdrojové kódy všetkých programov vytvorených v tejto práci, tzn. referenčných botov a UT2004MapsGenerator spolu s programátorskou dokumentáciou

..\Testing environments - obsahuje vygenerované a skompilované mapy, na ktorých sme referenčných botov testovali, každú v troch rôznych obtiažnostiach spolu s *.bat* skriptom na ich jednoduché spúšťanie a *.csv* súbormi obsahujúcimi namerané dáta