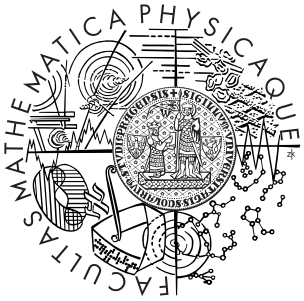
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Pavel Nohejl

Boj robotů ve virtuálním prostředí

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Jaromír Malenko
Studijní program: Informatika, Programování

2007

Na tomto místě bych rád poděkoval Mgr. Jaromíru Malenkovi za vedení projektu, konzultace a cenné rady při psaní této práce a aplikace Tank Battle.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 09.08.2007 Pavel Nohejl

Obsah

Název práce: Boj robotů ve virtuálním prostředí
Autor: Pavel Nohejl
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: Mgr. Jaromír Malenko
e-mail vedoucího: Jaromir.Malenko@mff.cuni.cz

Abstrakt: Cílem projektu je vytvořit knihovnu pro programování softwarových agentů (robotů), virtuální prostředí pro simulaci jejich boje včetně vyzualizace. Aplikace je implementována pomocí technologie .NET, knihovnu pro programování robota je možné využít v libovolném jazyce podporujícím tuto technologii. Ve virtualním prostředí je možné nastavit mnoho parametrů, na kterých závisí optimální strategie robota. Robot má dostupné omezené zdroje, které jeho autor může využívat.
Klíčová slova: souboj programovatelných agentů, virtuální multitasking, technologie .NET, správa mnoha vláken a dynamicky načtených knihoven

Title: Robots Fighting in an Virtual Environment
Author: Pavel Nohejl
Department: Department of Theoretical Computer Science and Computional Logics
Supervisor: Mgr. Jaromír Malenko
Supervisor's e-mail address: Jaromir.Malenko@mff.cuni.cz

Abstract: The goal of this project is to create library for programming software agents (robots), virtual environment for simulation of their battle, including visualization. Aplication is implemented by .NET technology, library for programming robot is possible to use in any language which support .NET technology. Optimal robot strategy depends on many parameters of virtual environment, which can be adjusted. Robot has available limited resources, which his autor can use.
Keywords: duel of programmable agents, virtual multitasking, .NET technology, management of many threads and dynamic loaded libraries

Kapitola 1

Úvod

1.1 Cíle projektu a možné využití v praxi

Cílem této práce bylo vytvořit aplikaci sloužící především programátorům, pro které je programování koníčkem a zábavou. Aplikaci mohou používat jak profesionální programátoři, tak i naprostí začátečníci, pro které je aplikace *Tank Battle* vhodná učební pomůcka.

Vytvořit robota je de facto jen pár řádků kódu, i přesto se již zde začátečník – programátor učí jednomu ze základních principů OOP a tím je dědění. Přemýšlením nad tím jak vylepšit robota, aby byl úspěšný a vyhrával bitvy, si začátečník osvojuje další aspekty programování jakou jsou například podmínky a cykly. Využíváním složitejších funkcí z knihovny robota se učí dalšímu principu OOP a tím je polymorfismus.

Pro pokročilé programátory nabízí aplikace události, díky nimž se dozví co se v bitvě děje. Roboti mohou např. díky radaru získat souřadnice a úhly letících střel, vypočíst si jejich dráhy a vyhýbat se jim. Před spuštěním samotné bitvy je možno nastavit mnohou parametrů prostředí, které zásadně ovlivňují celou bitvu a tím mění požadavky na logiku a chování robota. Veškeré nastavení a průběžné výsledky bitvy si robot může zjistit a přizpůsobovat tomu tak svojí taktiku. Pokud je například jako kritérium vítězství pořadí umístění robota v celé bitvě, není vůbec nutné, aby robot uměl přesně střílet, důležité je přežít. Naopak pokud je kritériem vítězství množství zničených tanků, tak by předchozí taktika přežití pravděpodobně nebyla úspěšná.

Protože je využita platforma .NET a knihovny pro programování robota splňují kritéria CTS (Common Type System – společný systém typů) a

CLS (Common Language Specification – společná specifikace jazyků), nejsou programátoři robotů vázáni na konkrétní jazyk. Je možné využít libovolný jazyk, který podporuje platformu .NET.

V aplikacích podobného zaměření jsou softwaroví agenti (roboti) nazýváni také jako tanky, hlavně z důvodu jejich grafického znázornění. V této práci budu tyto pojmy také zaměňovat.

Při čtení této práce důrazně doporučuji pracovat i s uživatelskou a programátorskou dokumentací aplikace *Tank Battle*, ve kterých se podrobně dozvíte o tom, jak vytvořit tank a o tom, jaké byly největší problémy při psaní aplikace a jejich řešení, například správa velkého množství vláken, ochrana aplikace před chybami v uživatelské kódu jako jsou nekonečné cykly, nebo načítání a odstraňování dynamicky načítaných knihoven za běhu aplikace. V dodatku A se dozvíte, kde naleznete zmíněné dokumentace a aplikaci *Tank Battle* včetně okomentovaných zdrojových kódů.

Kapitola 2

Robot a jeho programovaní

2.1 Anatomie robota a jeho vlastnosti

Obrázek 2.1: Znázornění robota/tanku

Tank se skládá z těla, hlavně a radaru, které jsou na sobě nezávislé a mohou se tedy otáčet nezávisle na sobě. Vzhledem k tomu, že tank zpravidla střílí tam kam míří jeho radar (radarem může detekovat soupeře), tak lze nastavit, aby se radar otáčel současně s hlavní a hlaveň současne s tělem tanku.

Tank může jet vpřed a vzad. Otáčet tělo, hlaveň a radar nezávisle na sobě a s různou silou střílet. Je–li tank zasažen střelou, sníží se mu energie, dle toho jak silná střela ho zasáhla. Když tanku dojde veškerá energie, tak je zničen a v aktuálním kole bitvy skončil.

2.2 Abstraktní třída Tank

Chce–li uživatel naprogramovat svůj tank, musí vytvořit třídu zděděnou z abstraktní třídy *Tank*, implementovat abstraktní metodu *start* (toto je tzv. hlavní funkce tanku/robota, více v podkapitole 2.5) a vytvořit dynamickou knihovnu (DLL). Tato knihovna s tankem, kterých může být v jedné knihovně i více, je načítána aplikací.

Členy této třídy se dají rozdělit do několika skupin. Na funkce přidávající blokující akce a jejich přetížené varianty přidávající neblokující akce (více o akcích v podkapitole 2.4). Dále na vlastnosti, které dovolují tanku zjistit svoji pozici, rotaci a různé nastavení. Dotazovací funkce pro zjištění nastavení aktuální bitvy, průběžných výsledků apod. A nakonec virtuální funkce pro události tanku, které pokud autor tanku překryje, bude dostávat informace o průběhu bitvy (více o událostech v podkapitole 2.6).

V dodatku A naleznete informace o tom, kde se na přiloženém CD nacházejí příklady tanků, včetně jejich zdrojových kódů.

2.3 Cyklus bitvy

Bitva se skládá z cyklů. Čím více cyklů během jednotky času proběhne, tím samozřejme bitva probíhá rychleji . Cykly jsou analogií k frame–rate, neboli počtu snímků za sekundu (v každém cyklu dojde k překreslení grafiky). Jeden cyklus se skládá z těchto fází:

- Probuzení a spuštění některého z vláken robota, je–li to třeba
- Smazání událostí
- Provedení akcí robotů a generování nových událostí
- Provedení pohybu střel, detekce jejich srážky s roboty, generování nových událostí
- Scan radarem (detekce ostatních robotů a jejich střel), generování nových událostí
- Předání dat grafickému enginu a překreslení grafiky

Robot může během jednoho cyklu bitvy provést omezené množství akcí (může se pouze o určitý počet stupňů otočit, o určitý počet pixelů změnit

polohu,...), dle toho jak je prostředí bitvy nastaveno. S tím musí uživatel při programování robota počítat.

2.4 Akce robota a jejich přidávaní

Pomocí akcí se robot může pohybovat, otáčet a střílet. Akce se přidávají do fronty akcí. Pokud chce jet robot například vpřed, zavolá funkci, která přidá příslušnou akci do fronty akcí (nedojde vlastně k okamžitému provedení). Jakmile všechny roboti provedou svůj kód v aktuálním cyklu (doba provádění jejich kódu je omezená), tak se projdou fronty akcí všech robotů a tyto akce se provedou (jízda vpřed, otočení robota, ...).

Akce jsou dvojího druhu: *blokující* a *neblokující*.

Blokující akce
Je–li přidána blokující akce, dojde okamžitě k uspání vlákna robota. Vlákno robota je znovu probuzeno, až když je celá akce provedena. Pokud je například přidána akce pro pohyb robota o 200 pixelů a nastavení je takové, že robot může změnit polohu max. o 10 pixelů za jeden cyklus, tak jeho vlákno bude znovu probuzeno po 20 cyklech. Během této doby se toho v bitvě může mnoho změnit, proto není vhodné používat dlouhotrvající akce. Vlákno robota může být také znovu probuzeno (i přesto že ještě není dokončena blokující akce) pokud nastala nějaká událost (více v podkapitole 2.6).

Neblokující akce
Rozdíl mezi blokujícími a neblokujícími akcemi z hlediska jejich provádění není žádný. Jestliže ale přidáte neblokující akci, tak nedojde k uspání vlákna robota. K uspání dojde až tehdy, když je přidána nějaká blokující akce nebo přidána akce *execute*, která slouží pro provedení neblokujících akcí.

Další rozdíl oproti blokujícím akcím je ten, že i když je ve frontě akcí i více neblokujících akcí, tak je vlákno robota přesto probuzeno.

Neblokujicí akce jsou vhodné především pro pokročilé a složitější roboty a pro naprogramování propracovanějších pohybů robotů. Například naprogramování pohybu robota po kružnici. S blokujícími akcemi by to bylo obtížné (robot nejdříve popojede, v dalším cyklu se o pár stupňů otočí),

výsledkem je buď nepříliš hladká kružnice, nebo hladká kružnice, ale robot musí jet pomalu (kružnice je hladká, protože robot popojede o malou vzdálenost a v dalším cyklu se málo otočí). Při použití neblokujících akcí není problém pohyb robota po kružnici, protože robot se může zároveň otočit i popojet během jednoho cyklu bitvy.

2.5 Hlavní funkce robota

Aby mohl uživatel naprogramovat svého robota, musí implementovat abstraktní funkci *start*, která se nazývá hlavní funkce robota. Tato funkce je volána Enginem a nesmí skončit. Skončí–li, je robot z bitvy vyřazen.

Typická konstrukce hlavní funkce tanku v jazyku C#:

```
protected override void start() {

    /* nastavení robota , inicializace proměnných */

    while ( true ) {

       /* zde se programuje chování robota */

    }

}
```

Jak bylo řečeno výše, hlavní funkce robota nesmí skončit, jinak je z bitvy vyřazen, proto nekonečný while cyklus uvnitř funkce *start*. Hlavní funkce robota běží ve vlastním vlákně, které se vytváří a spouští znovu každé kolo bitvy.

2.6 Události

Události jsou funkce, které se volájí v robotu Enginem aplikace, aby mohl zjistit co se v bitvě právě děje a mohl na to adekvátně zareagovat (např. aby mohl ujet, když po něm střílí soupeř). Funkce událostí dostanou jako parametr objekt události, ve kterém jsou uloženy data týkající se nastálé

událostí, dle kterých může robot reagovat na nastálou situaci.

Druhy událostí:

bulletHitTankEvent Událost je vyvolána pokud robot zasáhne svojí střelou soupeře.

hitByBulletEvent Událost je vyvolána pokud robota zasáhne střela.

hitTankEvent Událost je vyvolána pokud se robot srazí s jiným robotem.

hitWallEvent Událost je vyvolána pokud robot narazí do okraje arény.

scannedBulletEvent Událost je vyvolána pokud robot radarem detekuje střelu.

scannedTankEvent Událost je vyvolána pokud robot radarem detekuje soupeře.

opponentDestroyedEvent Událost je vyvolána pokud je soupeřův robot zničen.

penalizedEvent Událost je vyvolána pokud byl robot penalizován za příliš dlouhé provádění svého kódu.

Funkce událostí jsou definovány v abstraktní třídě *Tank* jako virtuální a standartně pouze zavolají funkci, která signalizuje, že funkce události není překryta. Událost se pak již znovu nevolá a je ušetřena značná režie, která vzniká při spouštění funkcí událostí, protože pro každé spuštění funkce události se musí vytvořit nové vlákno z důvodu ochrany aplikace před kódem robota, který může obsahovat např. nekonečné cykly.

Pokud autor robota překryje některou z funkcí událostí (není tedy zavolána funkce, která signalizuje, že robot funkci nepřekryl), přihlásí se tak vlastně k odběru těchto událostí. Které události autor robota implementuje a jak je implementuje je jen zcela na něm a může používat stejné funkce jako v hlavní funkci robota.

Každý cyklus bitvy jsou vygenerovány nové události, které se týkají akcí prováděných právě v tomoto cyklu, ale robot s nimi může pracovat až v dalším cyklu bitvy (dozvídá se zpětně co se stalo a může na to adekvátně

zareagovat). Po–té co robot dostane možnost tyto události zjistit a zareagovat na ně, dojde k jejich smazání a opětovnému vygenerování nových udalostí, které již nastaly v novém cyklu bitvy.

Vzhledem k tomu, že během boje může nastat více událostí, které robot implementuje, je nutné rozhodnout, které se opravdu zavolají. K tomu slouží *priority* událostí. Logicky se spustí událost s nejvyšší prioritou.

Je nutné si také uvědomit, že provádění jedné události může trvat přes více cyklů bitvy. Nová událost (kterou robot implementuje) se tedy spouští, pokud má větší prioritu než právě prováděná událost. Když se nově spuštěná událost s vyšší prioritou dokončí, Engine se vrátí zpět k dokončení uspané (nedokončené) události.

Situace se komplikuje mají–li dvě události stejnou prioritu. Zde se přihlíží k *přerušitelnosti* událostí. Není–li událost přerušitelná, tak nová událost stejné priority nebude spuštěna (spouští se pouze události vyšších priorit). Je–li událost přerušitelná, tak se starší událost okamžitě přeruší (později se už nedokončí) a je spuštěna nová událost stejné priority. Přerušitelnost je užitečná například v této situaci: narazí–li robot do jiného silnějšího robota a implementuje událost *hitTankEvent*, která je naprogramována tak, aby odjel od silného soupeře do bezpečené vzdálenosti a zaútočil na slabého soupeře. Ale při tomto manévru (odjetí do bezpečené vzdálenosti) může narazit do jiného robota. Je tedy znovu vyvolána událost *hitTankEvent*. Pokud událost není přerušitelná, robot se stále bude řídit kódem původní události (odjetí do bezpečné vzdálenosti). Pokud je přerušitelná, původní událost se ukončí a robot se řídí novou událostí. Jestliže tenktokrát narazil do slabšího soupeře, může na něj zaútočit, což by v případe nepřerušitelné události neudělal.

Kapitola 3

Nastavení prostředí a jejich vliv na průběh bitvy

Aplikace byla od počátku programována tak, aby uživatelé měli možnost nastavit maximum vlastností aplikace, resp. maximum vlastností prostředí bitvy. Díky tomu lze dosáhnout velmi odlišných a zajímavých podmínek souboje tanků, které kladou zcela odlišné nároky na jejich logiku a taktiku. Tank, který v jedné bitvě s převahou vítězí, může v jiném nastavení prostředí bitvy zcela propadnout. Uživatelé si mohou nastavit prostředí bitvy a tedy i nároky na programování tanku, dle svých požadavků. Někdo preferuje raději taktickou bitvu, ve které je důležité přežít co nejdéle, jiný zase agresivní bitvu, ve které je důležité zničit co nejvíce soupeřů.

Nastavení jsou rozděleny do dvou kategorií a to jsou nastavení prostředí bitvy a nastavení vlastností tanku. Veškeré nastavení, které si uživatel zvolí, je možné uložit do souboru.

3.1 Nastavení prostředí bitvy

V nastavení bitvy lze kromě základních parametrů jako je velikost arény, ve které tanky bojují, a počet kol bitvy, ještě nastavit například podmínky vítězství v bitvě, které zásadně ovlivňují požadavky na chování tanků, protože podmínky pro vítězství v bitvě jsou velmi odlišné. V aplikaci jsou definovány 3 rozdílné podmínky pro vítězství v bitvě:

Vyhrává tank, který má v součtu nejnižší umístění Sečtou se umístění tanků ve všech kolech bitvy a tank, který má tento součet nejnižší, vyhrává. Zde je tedy nutné přežít co nejdéle a nezáleží na tom kolik ostatních soupeřů tank zničí.

Vyhrává tank, který vezme ostatním více enegie Vždy když tank zasáhne střelou soupeře, tak soupeři tento zásah vezme nějakou energi, dle nastavení. Vyhrává tank, který zásahy svých střel vezme soupeřům nejvíce energie (sčítají se hodnoty ze všech kol bitvy). Nezáleží tedy na tom, zda je tank zničen jako první v každém kole bitvy.

Vyhrává tank, který zničí nejvíce tanků Zde se počítá, kolik soupeřů tank zničí během celé bitvy. Tank zničí soupeře, když právě jeho střela způsobí, že soupeř nemá již žádnou energii a je tedy zničen. Zde se také naskýtá možnost taktizování a vyhledávání soupeřů, kteří již nemají téměř žádnou energii.

Dále je možné pro pokročilé uživatele nastavit, jak dlouho smí tank provádět svůj kód a případné penalizace za překročení nastaveného limitu. Toto nastavení je zde v zájmu spravedlivé bitvy a také z důvodu ochrany aplikace před nekonečnými cykly v kódu tanků.

Provádí–li tank svůj kód příliš dlouho, předpokládá se, že v jeho kódu je chyba (nekonečný cyklus) a tank je zrušen a vyřazen bitvy, resp. je zrušeno vlákno provádějící jeho kód a tank v aktuálním kole bitvy skončil.

Pokud tank provádí svůj kód déle, ale ne dost dlouho, aby musel být zrušen, tak může být pouze penalizován – několik kol bitvy nebude mít možnost provádět svůj kód. Toto nastavení je zde pro zajištění spravedlivosti bitvy a také nutí psát uživatele svůj kód co nejefektivněji a nevyužívat pouze hrubou sílu CPU.

3.2 Nastavení vlastností robota

Některá z nastavení robota mají také velký vliv na taktiku, která je potřebná pro vítězství v boji, jako je například nastavení radaru. Radar je možno nastavit tak, aby robot musel jezdit po bitevní aréně a aktivně hledat soupeře, nebo tak, že stále ví o všech soupeřích. Pak si například může vybrat nejslabšího, na kterého zaútočí.

Nastavení počáteční energie robota a rychlost obnovování energie má vliv na způsob, jakým bude probíhat bitva. Pokud roboti mají dostatek energie,

která se navíc rychle obnovuje, tak bitva bude agresivnější a plná střelby. Mají–li roboti minimum energie, která se navíc neobnovuje, tak bitva bude spíše taktická, opatrná a roboti se budou muset vyhýbat střelám, aby v bitvě uspěli.

S energií a s rychlostí obnovování energie robota úzce souvisí síla střel a nastavení jak často je možné střely různé síly vystřelit. Jak bylo napsáno výše, pokud mají roboti dostatek energie a tedy se předpokládá agresivní bitva, je vhodné nastavit, aby roboti mohli střílet často a střely byly silnější (vzaly při zásahu více energie). Naopak, když mají roboti minimum energie, je vhodné nastavit, aby nebylo možné střílet příliš často a roboti tedy museli taktizovat a střílet jen v případech, kdy je téměř jisté že střela zasáhne cíl.

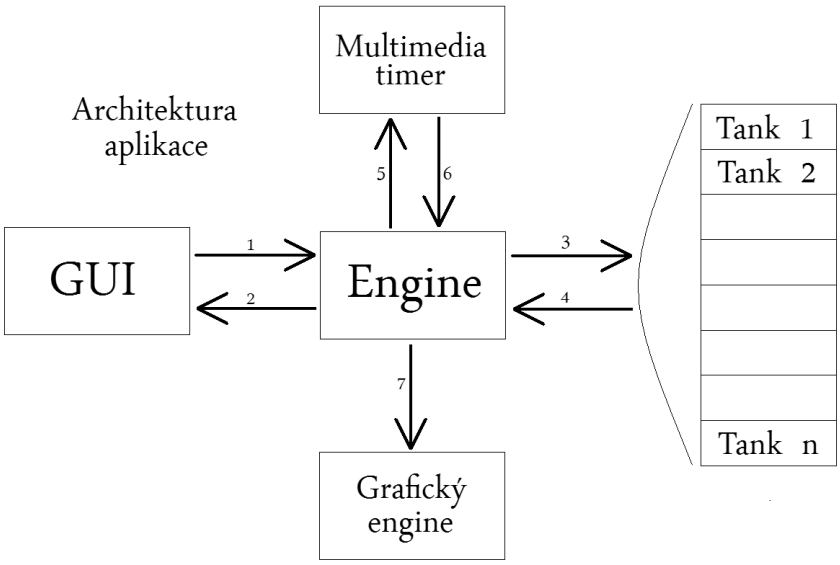
Dále je možné nastavit rychlost pohybu robotů a střel, nebo rychlost otáčení hlavně a robota.

Kapitola 4

Architektura aplikace

V této kapitole bych se rád stručně zmínil o architektuře aplikace a o tom jak funguje. Podrobnější informace se nacházejí v programátorské dokumentaci na CD, více v dodatku A.

4.1 Schéma architektury aplikace



1. GUI předává informace o nastavení bitvy a vybraných tancích Enginu. Také signalizuje ukončení/pozastavení/krokování bitvy.

2. Engine updatuje výsledky o průběhu bitvy a posílá zprávy od tanků, popř. zprávy v jaké fázi se bitva nachází.

3. Engine probouzí postupně jednotlivá vlákna hlavních funkcí a událostí tanků, aby mohly provést svůj kód.

4. Tanky přidávají své akce do fronty akcí, které pak Engine provede. Tanky se také dotazují Enginu na veškeré informace o sobě (pozice, úhel otočení radaru,...).

5. Engine spouští/pozastavuje/vypíná nebo mění periodu timeru.

6. Timer dle nastavené periody volá funkci provádějící cyklus bitvy.

7. Engine předává grafickému enginu informace o tancích a střelách, poté dává pokyn pro vykreslení grafiky, dle předaných dat.

4.2 GUI

Slouží především pro nastavení prostředí bitvy a výběru tanků pro bitvu. Tyto data pak předá Enginu aplikace, který podle nich spustí bitvu. Nastavení bitvy a tanků je možné ukládat a později otevřít.

Důležitým úkolem GUI je načítání knihoven s tanky a kontrola, zda se jedná opravdu o knihovnu s tankem. Když uživatel vybere tanky a spustí bitvu, jsou načteny příslušné soubory a Enginu jsou předány typy tanků. Engine dle těchto typů vytvoří instance tanků s nimiž pracuje.

4.3 Engine

Provádí simulaci celé bitvy. Uspává a probouzí vlákna tanků. Provádí akce tanků a s nimi spojené úkony, např. u změny polohy tanků je to detekce kolizí. Dále detekuje kolize střel s tanky nebo simuluje detekci ostatních tanků radarem. Předává grafickému enginu data o tancích a dává pokyn pro vykreslení grafiky. Pravidelně updatuje výsledky průběhu bitvy v GUI.

Engine pracuje s velkým množstvím vláken (řádově desítky), protože hlavní funkce tanků a všechny spouštěné události musí běžet ve vlastním

vlákně z důvodu ochrany aplikace před kódem v tanku, ve kterém mohou být chyby – nekonečné cykly. Pokud nějaké vlákno spuštěné na funkci tanku po nastavenou dobu nereaguje, je ukončeno. Funkce je také nutné průběžně uspávat dle toho, zda přidaly blokující akci. To by samozřejmě nebylo možné, kdyby vlákno Enginu přímo volalo hlavní funkce tanků nebo události a nespouštělo ty funkce ve vlastních vláknech.

4.4 Knihovna Multimedia Timer

Multimedia Timer [7] je externí knihovna pro timer s vysokou přesností a nízkou periodou. Je implementována pomocí volání API funkcí systému Windows. Bez této knihovny by bylo obtížné implementovat ovládání rychlosti bitvy, protože standartní timer obsažený v knihovnách .NET má minimální garantovanou periodu zhruba 15ms, což odpovídá asi 67 cyklům bitvy za vteřinu, navíc není tolik přesný a docházelo by k častému kolísání rychlosti bitvy. Díky externí knihovně je možné dosáhnout až 1000 cyklů za sekundu, minimální perioda je 1ms. Ale v aplikaci je maximální rychlost omezena na 200 cyklů za vteřinu, větší hodnota již nemá smysl pro sledování průběhu bitvy.

V aplikaci lze nastavit maximální rychlost bitvy, zde se již nevyužívá timer, ale funkce provádějící simulaci bitvy se volají přímo uvnitř while cyklu. Tak je možné dosáhnout rychlosti řadově tisíců až desetitisíců cyklů za vteřinu. Toto nastavení je vhodné, pokud uživatel chce znát pouze výsledky bitvy a nezajímá ho její průběh, pak je vhodné zavřít okno s vizualizací bitvy pro zrychlení průběhu simulace. Knihovna je volně šiřitelná a je možné ji upravovat. Upravil jsem ji tak, aby byla maximálně jednoduchá, jen pro potřeby této aplikace.

4.5 Grafický engine

Grafický engine slouží pro vizualizaci průběhu boje tanků. Je načítán aplikací za běhu z pevně daného souboru. Rozhraní grafického enginu je veřejné a jeho dokumentace je součástí uživatelské dokumentace na CD, více v dodatku A. Je tedy možné napsat nový a přepsáním souboru standartního grafického enginu jej aplikace začne využívat (pokud samozřejmě korektně implementuje definované rozhraní).

Grafický engine dostává z Enginu informace o tancích (jejich souřadnice,

rotace, ...) a střelách. Dostane–li pokyn, tak tyto tanky a střely vykreslí dle dříve předaných dat.

4.6 Abstraktní třída tank

Tato třída definuje funkce a vlastnosti tanku, pomocí kterých uživatel programuje svůj tank. Při programování svého tanku musí z této třídy zdědit.

V této třídě probíhá zachycování vyjímek, které přicházejí z kódu tanku napsaného jeho autorem. Při spouštení vlákna hlavní funkce tanku nebo události, není vlákno spuštěno přímo na těchto funkcích. Tyto funkce jsou spuštěny uvnitř try–catch bloku, který se nachází ve funkci, na které bylo vlákno skutečně spuštěno. Příklad v jazyce C#:

```
//vlákno je spuštěno na této funkci
internal void startTank() {

    try {

        //toto je zavolání hlavní funkce tanku
        start();

    } catch {
        /*
            informování Enginu o tom, že v tanku nastala vyjímka
        */
    }

}
```

Touto konstrukcí je aplikace chráněna před chybami v kódu tanku.

Tato třída se na veškeré informace o tanku (pozice, úhel otočení radaru,...) dotazuje Enginu, který má uložené všechny informace o tancích.

Příklady tanků včetně zdrojových kódů naleznete na přiloženém CD, více v dodatku A.

Kapitola 5

Problémy při vývoji

Podrobněji o problémech při vývoji a jejich řešení se dozvíte v programátorské dokumentaci, viz dodatek A.

5.1 Vícevláknovost a problém souběžného zpracování vláken

V aplikaci se pracuje s velkým množstvím vláken, jak bylo uvedeno v podkapitole 4.3. Při práci s vlákny je nutno myslet na *problém souběžného zpracování* vláken. Když se například zadá pokyn k spuštění vlákna, je zadán pouze požadavek OS, aby vlákno spustil, jakmile to bude možné. Někdy OS může spustit vlákno téměř okamžite, jindy musí čekat až vyprší časové kvantum přidělené jinému vláknu. Mohou tedy nastat situace a chyby v běhu programu, které je velmi těžké zreprodukovat a tím i odladit celou aplikaci. Situace se ještě zkomplikuje využíváním vícejádrových procesorů, kde spuštění dalšího vlákna je podstatně rychlejší (není–li další jádro vytížené). Na jednojádrovém procesoru se musí vlákno, ze kterého probouzíte jiné vlákno, nejprve uspat a až po–té dojde k probuzení jiného vlákna. Toto na vícejádrovém procesoru neplatí, k probuzení jiného vlákna dojde podstatně rychleji na dalším jádru procesoru, protože se nemusí čekat na uspání původního vlákna (toto samozřejmě platí na nevytíženém procesoru).

Programování aplikace tohoto typu klade velké nároky na programátora a hlavně na důkladné testovaní na jedno i více jádrových procesorech. Jak bylo uvedeno výše, chyby se velmi těžko reprodukují a hledají.

I přesto, že jsem se snažil jakoukoli práci s vlákny důkladně promyslet,

vyhnout chybám jsem se vždy nedokázal. Nejhorší chyba, kterou jsem musel hledat byla následující. Vývoj probíhal na dvoujádrovém procesoru a aplikace fungovala ve více než 99% případů. Když jsem našel po velmi dlouhé době místo v kódu, kde byla chyba, nemohl jsem pochopit jak to, že aplikace mohla fungovat. Řešení bylo nakonec jednoduché. Kdyby vývoj probíhal na jednojádrovém procesoru, na chybu by se přišlo podstatně rychleji – aplikace by totiž vůbec nefungovala a místo chyby by šlo jednoduše identifikovat. Ale na vícejádrovém procesoru bylo poměrně složité chybu zreprodukovat a ještě složitejší nalézt místo v kódu, kde by se chyba mohla nacházet, protože aplikace se chovala absolutně nepředvídatelně.

5.2 Vlákna vytvořená programátorem robota, vyjímky v těchto vláknech

Při programování aplikace jsem se snažil, aby byla maximálně robustní a odolná proti chybám v kódu robotů. Tento přístup vede ke složitejšímu a pomalejšímu kódu aplikace, ale dle mého názoru by se aplikace neměla spoléhat na to, že kód robotů bude v pořádku. Jak bylo napsáno v předchozích kapitolách, aplikace je odolná vůči nekonečným cyklům v kódu robotů včetně konstruktoru a také proti nezachyceným vyjímkám. Pokud nějaká z těchto situací nastane, tak je robot vyřazen z celé bitvy nebo z právě probíhajícího kola bitvy.

Bohužel ale není možné zabránit uživatelům, aby vytvářeli vlastní vlákna. Pokud v těchto vláknech nastane vyjímka, která není zachycena, dojde ke ukončení celé aplikace totou vyjímkou. Je možné zjistit, že taková vyjímka nastala, ale není možné ji zachytit.

Další problém s vytvářením vláken v robotech je ten, že aplikace neskončí, dokud nejsou ukončeny všechny vlákna na popředí. Pokud tedy uživatel vytvoří vlákno na popředí a bude v něm nekonečný cyklus, aplikace se sama neukončí. Jedinou možností je pak ukončit aplikaci přes správce úloh systému Windows.

Z vláken vytvořených programátory robotů není možné volat funkce z knihovny robota, resp. tyto volání jsou ignorovány a funkcemi jsou navráceny nesprávné hodnoty. Každé vlákno má unikátní identifikační číslo přidělené aplikací a vždy se kontroluje, zda funkce z knihovny robota nejsou volány neznámým vláknem. Uživatelé tak mají o důvod méně užívat vlastní vlákna.

5.3 Načítání knihoven s roboty

Knihovna načtená do programu je chráněná proti zápisu. Což v aplikaci *Tank Battle* není žádoucí, protože uživatel chce pracovat na svém robotu a vylepšovat ho. Pokud by ale soubor s jeho robotem byl chráněn proti zápisu, vždy by musel aplikaci vypnout, zkopírovat svůj soubor s robotem do adresáře, kde aplikace hledá soubory s roboty, aplikaci znovu spustit a znovu nastavit prostředí bitvy dle svých požadavků. A to není pro uživatele aplikace příliš komfortní.

Proto se procházení adresáře s roboty a hledání knihoven s roboty provádí v jiné aplikační doméně, uloží se pouze názvy typů robotů a soubory, kde se nachází. Po–té se aplikační doména odstraní a tím se odstraní i knihovny v ní načtené a soubory knihoven tak již nejsou chráněné pro zápisu – aplikace je již nepoužívá. Když uživatel vybere roboty, které se budou účastnit bitvy a spustí ji, soubory s vybranými roboty se zkopírují do cache adresáře a až tyto zkopírované soubory se načtou, původní soubory tak nejsou chráněny proti zápisu. Programátor robota tak nemusí při jeho vylepšování vždy restartovat aplikaci.

Kapitola 6

Obdobné aplikace

V této kapitole se stručně zmíním o dvou aplikacích stejného zaměření. Budu zdůrazňovat hlavně jejich rozdíly oproti aplikaci *Tank Battle* a nakonec uvedu jejich klady a zápory.

6.1 Robocode

Aplikace Robocode [4] je napsána v javě a její roboti se také programují v javě. Aplikace je poměrně populární, o čemž svědčí množství robotů, které se dají z internetu stáhnout a také mnoho článků a tipů o tom, jak naprogramovat úspěšného robota. Existuje mnoho soutěží, ve kterých je za úkol naprogramovat co nejlepšího robota. Také existují příklady robotů, jejichž kód je vylepšován genetickými algoritmy. Aplikace je zejména v komunitě java programátorů poměrně populární.

Klady:

- populární, lze nalézt spoustů článků o programování robotů
- možnost napsat a zkompilovat robota přímo v aplikaci, editor podporuje syntax highlighting
- týmová bitva

Zápory:

- programátoři robotů jsou omezeni pouze na jazyk java
- velmi omezené možnosti nastavení prostředí bitvy, vlastností robota a podmínek vítězství v bitvě

6.2 Robot Battle

Aplikace Robot Battle [5] je koncipována jako počítačová hra. Má vlastní programovací jazyk a není tedy nutná znalost určitého programovacího jazyka či technologie pro naprogramování robota. V průběhu bitvy mohou roboti sbírat různé předměty, které jim poskytnou nějakou výhodu v bitvě.

Klady:

- vlastní jazyk – není nutná znalost nějaké konkrétní technologie. Aplikace je tak více odolná oproti chybám, které nastanou v kódu robota.
- týmová bitva, možnost nastavení vzhledu robotů
- dobře zpracovaná dokumentace
- poměrně široké možnosti nastavení bitvy

Zápory:

- vlastní jazyk – toto je zároveň i nevýhoda této aplikace. Uživatelé, kteří mají zkušenosti s programováním se musí učit nový jazyk. Navíc jazyk, ve kterém se programují roboti této aplikace, nemusí být tak mocný jako například java či některý z jazyků platformy .NET a nemůže využívat obrovské množství knihoven, které jsou k dispozici právě v javě či platformě .NET.
- vždy stejné podmínky pro vítězství v bitvě a z tohoto důvodu stále stejné nároky na programování robota

Kapitola 7

Budoucí práce

Aplikace má velký potenciál pro rozšiřování a vylepšování. V následujících podkapitolách uvedu ty, dle mého názoru, nejpřínosnější vylepšení.

7.1 Variabilita prostředí

Pro zpestření bitvy a zvětšení možností pro taktizování robotů jsou vhodné různé překážky v aréně, popř. různé tvary arény. Pro jednoduchost by byly vhodné 3 druhy překážek: překážky přes které radar nemůže detekovat soupeře, o tom co se děje za nimi robot nic neví. Překážky přes které radar může detekovat soupeře, ale nelze přes ně střílet. A překážky přes které radar může detekovat soupeře, lze přes ně střílet, ale robot se přes ně nemůže dostat. Další možností zpestření bitvy jsou různé tvary arény – například kruhová či různé polygony. U obou nápadů by se jako hlavní problém jevilo, jaká data by se robotu předávaly, ze kterých by rozpoznal tvar arény, popř. překážky která se v aréně nachází. S těmito nápady také úzce souvisí vytvoření 3D izometrického enginu, ve kterém by byly pro uživatele sledujícího průběh bitvy lépe rozeznatelné různé překážky (v 2D enginu se mohou lišit pouze barvou).

Díky překážkám uvnitř arény by bylo možné vytvořit herní módy, ve kterých není nutné vůbec střílet. Roboti by mezi sebou mohli soupeřit např. v rychlosti průchodu bludištěm.

7.2 Bitva po síti

Pomocí technologie .NET Remoting by bylo možné implementovat bitvu po síti. Jeden z účastníku bitvy by spustil aplikaci v režimu serveru. Ostatní účastníci by se k této bitvě připojili, vybrali své roboty a ty by se přenesly na počítač, kde běží aplikace v režimu serveru. Jednotliví účastníci bitvy by se dorozumívali pomocí chatu. Aplikace, která by běžela v režimu serveru, by měla právo nastavovat a ovládat bitvu, ostatní by pouze viděli její průběh a výsledky.

7.3 Editor zdrojových kódů robota a kompilátor přímo v aplikaci

Toto rozšíření by bylo vhodné proto, aby uživatelé programu nemuseli užívat jiný nástroj pro programování robotů. Editor by měl umět alespoň základní syntax–highlighting a jednoduchý našeptávač. Standartní knihovny .NET obsahují třídy, pomocí kterých lze zkompilovat za běhu zadaný textový řetezec. Kompilace robotů za běhu aplikace by tedy nebyla problém.

7.4 Týmová bitva

V týmové bitvě bojuje více tanků v rámci jednoho týmu proti tankům patřících po jiných týmů.

V současné době aplikace nepodporuje týmovou bitvu. Není umožněno si mezi roboty s pomocí funkcí v knihovně robota vyměňovat informace. Výměna dat mezi roobty je ale možná, například pomocí statických proměnných a lze tedy omezeným způsobem simulovat bitvu týmů. Problémem je, že bodování bitvy nepodporuje bodování celých týmů robotů. Navíc bitva není ukončena pokud zůstanou v bitvě pouze roboti stejného týmu.

Pro podporu bitvy týmů by bylo nutné do programu implementovat mechanismus pro bodování celých týmů, ukončování kola bitvy, když by v aréně zůstali pouze roboti stejného týmu, podporu pro komunikaci mezi roboty v knihovně robota a nakonec upravit GUI, aby bylo možné vybírat roboty do týmu a vytvářet týmy.

Kapitola 8

Závěr

Výsledkem této bakalářské práce je knihovna pro programování softwarových agentů (robotů) a aplikace *Tank Battle* simulující virtuální prostředí, ve kterém naprogramovaní roboti mohou mezi sebou bojovat. Knihovna robota je navržena tak, aby ji bylo možné použít v libovolném jazyce podporujím technologii .NET.

Virtuální prostředí pro simulaci boje je velmi flexibilní, je možné nastavit mnoho parametrů prostředí, které kladou na logiku a chování robota zcela jiné požadavky. Uživatel si tak může zvolit prostředí, které mu vyhovuje, např. prostředí ve kterém vítězí agresivní robot, nebo prostředí, ve kterém je nutné taktizovat a agresivní robot by v něm neuspěl.

V průběhu vývoje bylo nutné vyřešit několik důležitých problémů jako je přesné měření času, správa mnoha vláken a jejich postupné probouzení a uspávání. Dále bylo nutné vyřešit, jak chránit aplikaci před chybami v kódu robota, jako jsou nekonečné cykly nebo nezachycené vyjímky. Nakonec bylo nutné zajistit, aby knihovny s roboty nebyly po načtení do aplikace chráněny proti zápisu a uživatel tak nemusel při vylepšování svého robota aplikaci vždy restartovat.

Aplikace je volně šiřitelná a v budoucnu bude dostupná na internetu včetně veškeré dokumentace pro vytváření robotů, popřípadě nového grafického enginu. Aplikace bude také dále postupně rozšiřována o další rysy, zejména o ty zmíněné v kapitole 7 .

Dodatek A

Obsah přiloženého CD, instalace aplikace

Na přiloženém CD naleznete tyto složky :

Application – obsahuje aplikaci *Tank Battle*, kterou můžete spustit přímo z CD souborem *tankBattle.exe*. V aplikaci je pro vyzkoušení několik robotů, nemusíte tedy programovat nějaké vlastní.

Bc – obsahuje tuto práci, včetně zdrojového kódu

Doc – obsahuje programátorskou a uživatelskou dokumentaci. Uživatelská dokumentace je k dispozici ve formátu chm a html. Nápověda ve formátu html se otevře souborem *index.html*.

Install – obsahuje instalační soubory *Microsoft .NET Frameworku 2.0* a prostředí *SharpDevelop* pro vývoj robotů v jazyce C#.

Sources – obsahuje zdrojové kódy aplikace *Tank Battle* včetně knihoven, které používá. Naleznete je v podadresáři **Application**. V podadresáři **Tank Examples** naleznete zdrojové kódy tanků, které jsou obsaženy v aplikaci.

Postup instalace aplikace :

1) Pro spuštění aplikace je nutné mít nainstalovaný *Microsoft .NET Framework 2.0*, který naleznete na přiloženém CD. Na CD je také instalační soubor

prostředí *SharpDevelop* pro vývoj robotů v jazyce C#.

2) Aplikaci je možné spustit přímo z CD souborem *tankBattle.exe*. Nebude ale možné ukládat nastavení aplikace. Chcete–li aplikaci spustit z pevného disku zkopírujte složku **Application** na disk.

Literatura

[1] Andrew Troelsen: *C# a .NET 2.0 profesionálně*, Zoner Press, 2006.

[2] Chris Sells: *C# a WinForms*, Zoner Press, 2005.

[3] Microsoft Development Network: *http://www.msdn.com*, [2007]

[4] Robocode: *http://robocode.sourceforge.net*, verze aplikace: 1.3.5

[5] Robot Battle: *http://www.robotbattle.com*, verze aplikace: 1.4.05

[6] The Code Project - Free Source Code and Tutorials: *http://www.codeproject.com*, [2007]

[7] The Code Project - Multimedia Timer: *http://www.codeproject.com/cs/miscctrl/lescsmultimediatimer.asp*, [09.08.2007]