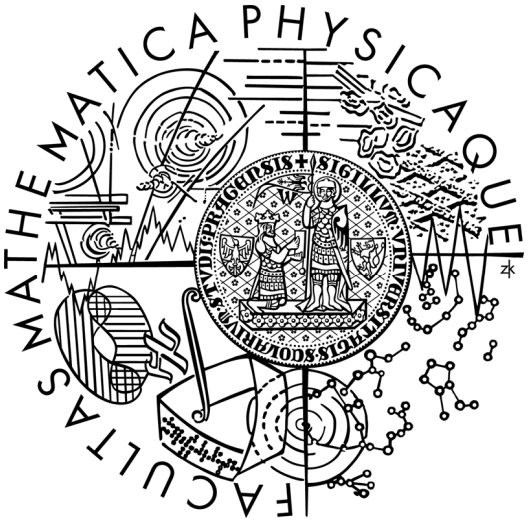
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# BAKALÁŘSKÁ PRÁCE




Pavel Matyska

# Simulace bitvy robotů

*Katedra softwarového inženýrství*

Vedoucí bakalářské práce: RNDr. Jiří Adámek, Ph.D.

Studijní program: programování

2007

Děkuji svému vedoucímu za jeho neskonalou trpělivost při přípravě podkladů této práce.

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

V Praze dne 12.12.2007 Pavel Matyska

# Obsah

Název práce : Simulace bitvy robotů
Autor : Pavel Matyska
Katedra (ústav) : Katedra softwarového inženýrství
Vedoucí bakalářské práce : RNDr. Jiří Adámek, Ph.D
e-mail vedoucího : adamek@nenya.ms.mff.cuni.cz

Abstrakt :
V této práci se budu nejprve zabývat obecným pohledem na agenty, jakožto obecné objekty vědeckého zkoumání. Z jednoho druhu agentů jsou zkonkretizováním vytvořeni a implementováni roboti v mém sofwarovém projektu. Stejně tak se budu zabývat prostředím, ve kterém jsou agenti umístěni, tedy nejprve z obecného hlediska a poté na mém konkrétním prostředí, ve kterém probíhá samotná simulace bitvy robotů. V práci také předvedu, jak lze navrhnout systém pro vidění softwarových robotů, jaké situace mohou na bitevním poli nastat a jak je bitevní pole, tedy prostředí pro roboty, navrženo. V závěru práce zdůvodním výběr typu agenta pro roboty porovnáním výhod a nevýhod pro záměr projektu.
Klíčová slova: Agent, Prostředí, Robot, Vjem, Akce

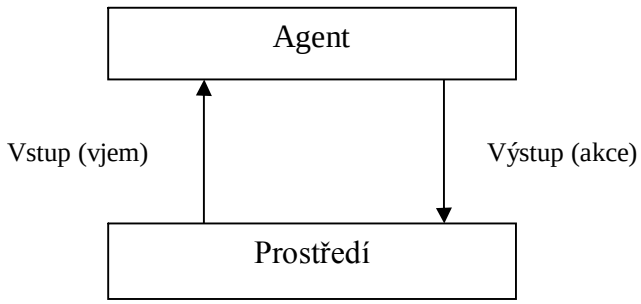
Title : Robot Battle Simulation
Autor : Pavel Matyska
Department : Department of Software Ingeneering
Supervisor : RNDr. Jiří Adámek, Ph.D
Supervisor’s e-mail address : adamek@nenya.ms.mff.cuni.cz

Abstract :
In this work I will discuss general view on agent as objects of science research. One of the discussed type of agent is used for implementing robots in my software project. I will also discuss environment in which agents exist, for the first time from general view, then in my software project, in which the robot battle simulation proceed. In this work I will propose, how a software robot view as percept can be proposed, which situation can occur on the battle field, and how is the battle field as environment for robots designed. At the end of this work I justify my decision to choose the type of agent by comparing theirs advantages and disadvantages for the aim of the project.
Key words : Agent, Environment, Robot, Percept, Action

# Kapitola 1

# Inteligentní agenti a jejich prostředí

Nejprve bych chtěl objasnit, co to je agent. Agent je „výpočetní systém, který je umístěn do nějakého prostředí a je schopen provádět akce takové, které vedou k jeho navrženým cílům“, jak je uvedeno v [1]. Tato definice, jak autor uvádí, není jediná a ani není úplně správná. Není správná hlavně proto, že nehovoří o „inteligenci“ agenta. Pro naše účely však postačí.
Vzhledem k tomu, že agent má nějaký úkol, který se snaží splnit, a že je umístěn do nějakého prostředí, musí na dané prostředí reagovat. Prostředí se tak stává jeho vstupními informacemi. Informace od prostředí získává pomocí takzvaných senzorů. Jaké senzory agent může mít, bude uvedeno později. Zpracováním těchto informací se agent snaží o takovou změnu prostředí, která bude pro agenta výhodnější, tedy aby dosáhl svých cílů. Po změně prostředí však agent tuto změnu zachytí, zpracuje a zareaguje. Je to takový nikdy nekončící proces. Schéma agentovy činnosti ilustruje Obrázek 1.




Obrázek 1. : Schéma agenta umístěného do prostředí.

Aby agent dosahoval svých cílů, musí být kromě senzorů na prostředí vybaven také sadou akcí, kterými lze dosáhnout agentových cílů. Při zpracovávání informací o prostředí vybere jednu z těchto akcí a provede ji. Definice však nehovoří o tom, jak má vypadat sada akcí, kterými je agent vybaven, ani mechanismus, jakým se konečná akce vybírá. Také v ní není uvedeno, jak má prostředí vypadat. Prostředí by tedy mohl být nějaký jiný agent, pro kterého by byl původní agent prostředím, definici by totiž splňoval také.

Výše uvedené vede k logickému rozdělení agentů do skupin podle mechanismu výběru akce z agentových akcí. Dříve, než se dostanu k jednotlivým skupinám (kapitola 1.3), bych zde

chtěl uvést možné prostředí, do kterých může být agent umístěn (kapitola 1.1) a dále abstraktní pohled na agenty, ale bližší, než nám dává výše uvedená definice (kapitola 1.2).

## 1.1. Prostředí pro umístění agenta

Jak jsem se již zmínil, agent může být umístěn do různých prostředí. Prostředí lze rozdělit podle několika faktorů, jestli je pro agenta plně přístupné nebo nepřístupné, jestli je deterministické nebo nedeterministické, jestli se agent může rozhodnout pouze podle současnosti – epizodické, nebo se rozhoduje podle situací minulých či budoucích - neepizodické, jestli je statické nebo dynamické, či jestli je stavové nebo bezestavové, nebo diskrétní či kontinuální. Všechny tyto možnosti jsou diskutovány v [1].

Plně přístupné prostředí je takové, o kterém má agent veškeré informace. Nepřístupné prostředí agentovi poskytuje jen částečné, omezené informace. Nemusí některé informace poskytovat vůbec. Přístupným prostředím může být například desková hra, kde je vše veřejně viditelné a nezáleží na náhodě. Takovou hrou je třeba dáma nebo šachy. Nepřístupné prostředí je každodenní reálný svět.

Deterministické prostředí je takové, ve kterém každá akce má právě jednu reakci. Deterministické prostředí by mohlo být prostředí fyzikálního pokusu, kde změna fyzikální veličiny má za následek přesně definovanou změnu prostředí. Nedeterministické by opět mohl být každodenní reálný svět.

Epizodické prostředí umožňuje agentovi rozhodovat se pouze ze současného stavu prostředí, je rozdělené na epizody bez vzájemné vazby mezi sebou. Neepizodické buď nelze rozdělit, a nebo jednotlivé epizody jsou mezi sebou provázány nějakými vazbami. Epizodické prostředí by splňovaly opět šachy, neepizodické by mohl být opět fyzikální pokus, kde se vědec o dalším postupu rozhoduje na základě dříve zjištěných informacích.

Statické prostředí je takové, u kterého nastane změna až po té, co agent provede svojí reakci na prostředí. Dynamické se naopak mění bez ohledu na činnost a existenci agenta. Statické prostředí mohou reprezentovat opět šachy a dynamické opět reálný svět.

Stavové prostředí je definováno konečnou množinou konkrétních stavů a v každém okamžiku se prostředí nachází v jednom z těchto stavů. U bezestavového prostředí stavů není konečně mnoho. Příkladem stavového prostředí mohou být opět deskové hry, příkladem bezestavového opět reálný svět.

Domnívám se, že nejkonkrétnější prostředí, tedy takové, ve kterém se agent nejsnáze rozhoduje je přístupné, deterministické, epizodické, statické a stavové. Nejobecnějším by pak mělo být nepřístupné, nedeterministické, neepizodické, dynamické a bezestavové.

V další části práce se budu pohybovat po prostředí, které je nepřístupné, nedeterministické, epizodické, dynamické a stavové.

## 1.2. Obecná konstrukce agenta

Než budu moci zkonstruovat nějakého agenta, musím si odpovědět na otázku, jestli ho musím konstruovat s ohledem na prostředí, do kterého ho umístím. Myslím si, že kdybych konstruoval agenta do nejobecnějšího prostředí, bude se umět chovat i v konkrétnějších prostředích, i když specializovanější agent bude mít efektivnější postupy a rychleji dosáhne svých cílů. Stejně tak si musím odpovědět na otázku, jak takový agent má „vypadat“, tedy co ho reprezentuje. Agent musí „vypadat“ stejně jako prostředí, do kterého bude umístěn. Nelze umístit softwarového agenta do fyzického světa a naopak. Tedy jestliže chci agenta, který bude umístěn do reálného světa, jeho senzory budou muset být také fyzické. Například meziplanetární vozítko bude mít například kameru, nebo nějaký gyroskop pro vnímání terénu pod sebou. Naopak postava v počítačové hře bude moci „vnímat“ jen to, co se děje v rámci hry, nikoli to, jaká je teplota v pokoji s počítačem. Podobné otázky si pokládal i Franklin v [2].

Agent tedy musí mít nějaké senzory, kterými monitoruje své prostředí a musí mít nějakou sadu akcí, ze které vybírá tu nejvhodnější pro danou situaci. Pro výběr akce musí mít i nějaký systém, pomocí něhož akci vybírá. Celý agent pak musí mít nějakou řídící část, která rozhoduje, jestli bude agent právě vnímat prostředí, zpracovávat údaje, či provádět akce. Takto se dívá na agenta Franklin v [2].

Z výše popsaného vyplývá, že kromě samotného agenta musím zkonstruovat prostředí, ve kterém bude. Protože agenta umísťuji do prostředí, nejprve nadefinuji prostředí. Nadefinuji prostředí i agenta tak, jak ho definoval Wooldridge v [1]. V dalším textu budu používat následující značení: nechť $A$ je množina, potom $A^*$ je množina posloupností prvků z množiny $A$ a $exp(A)$ je množina všech podmnožin množiny $A$. Prostředí je nějaká množina stavů prostředí

$$S=\{s_1, s_2, \ldots\}$$

Prostředí se v každém okamžiku nachází v nějakém stavu $s_i$. Sadu agentových akcí můžeme definovat jako množinu akcí

$$A= \{a_1, a_2, \ldots\}$$

Potom se na chování agenta můžeme dívat jako na funkci

$$VyberAkci : S^* \rightarrow A$$

která na základě vstupních stavů prostředí vybírá akci agenta z jeho sady akcí. Tato definice agenta, respektive chování, umožňuje agentovi rozhodovat se i podle minulých stavů prostředí. Nedeterministické prostředí lze charakterizovat jako funkci

$$VyberStavProstředí : S \times A \rightarrow exp(S)$$

která ze vstupního stavu $s \in S$ a akce $a \in A$ vybírá množinu stavů, ve kterých se prostředí po agentově akci může nacházet. Jestliže budou všechny množiny z oboru hodnot funkce *VyberStavProstředí* jednoprvkové množiny, definujeme tak deterministické prostředí. Další definice, které se týkají práce s historií agenta a jsou uvedeny v [1], potřebovat nebudeme.

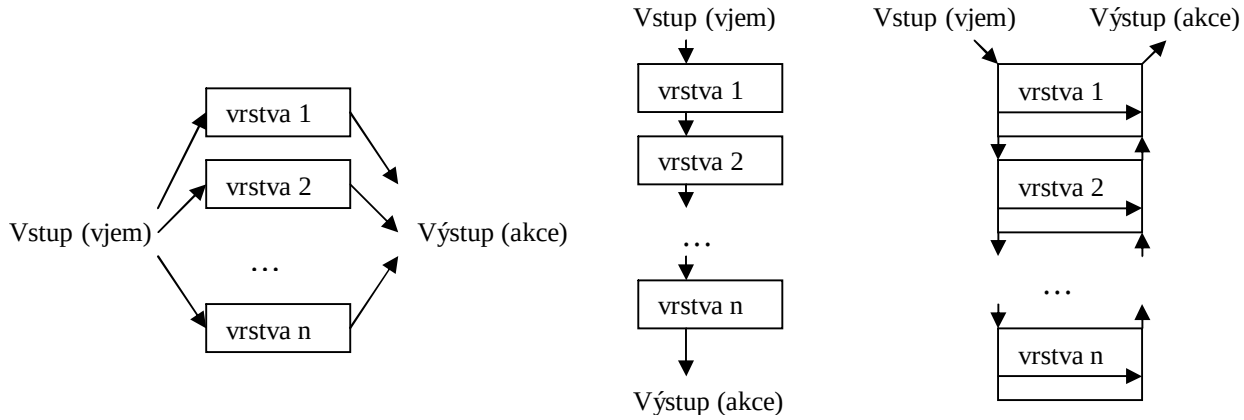
## 1.3. Dělení agentů do skupin

Podobně, jako rozlišujeme několik typů prostředí, stejně tak lze rozdělit agenty do několika skupin. Takové dvě hlavní skupiny agentů jsou čistě reaktivní agenti a stavoví agenti.

Čistě reaktivní agenti vyhodnocují své „vjemy“ prostředí přímo. To znamená, že se rozhodují pouze na základě techto vjemů, neudržují si žádné další informace. Čistě reaktivní agent proto musí neustále vnímat své prostředí, aby neprováděl určité akce v případech, kdy by bylo lepší provádět akce jiné. Toto je jeho hlavní nevýhodou, musí obětovat nemalý čas na neustálé vnímání, místo aby prováděl akce. Může se i stát, že než provede nějakou akci, několikrát přehodnotí situaci, aby provedl tu nejlepší akci. V prostředích silně dynamických, tedy prostředích, která se rychle mění, je toto chování poměrně žádoucí.

Stavoví agenti, jak jejich název napovídá, si v sobě udržují nějaký stav. Může to být jejich minulý stav a tak si udržovat historii chování, může to být posloupnost těchto stavů, nebo třeba dvojice stav agenta a stav prostředí. Uchovávání stavu agentovi umožňuje rozhodovat se na základě „zkušeností“ a vybírat tak akce, které se osvědčily. Uchovávání stavů umožňuje agentovi především plánovat své akce dobudoucna. Avšak procházení své historie a zpracovávání „zkušeností“ zabírá jistý čas, ve kterém nemůže vnímat své prostředí. Takový agent se pak hodí do prostředí spíše statického, tedy takového, které se mění málo, nebo se mění až s agentovou akcí.

Oba druhy agentů mají své výhody i nevýhody. Jistý kompromis mezi oběma přístupy nám dávají tzv. hybridní agenti. Abychom dosáhli obou druhů chování, tedy reaktivního a plánovacího, dojdeme k potřebě agenta složeného z několika částí rozložených do vrstev. Vrstvy, jako subsystémy agenta, mohou být implementovány také jako nějaký agent. Vrstvy lze rozložit horizontálně, nebo vertikálně. Rozložení vrstev ilustruje obrázek 2.




Obrázek 2. : Možné rozdělení jednotlivých vrstev hybridních agentů a způsob zpracování vstupu.

Horizontální rozložení ( na obrázku 2. vlevo ) má každou vrstvu přímo spojenou se vstupy i s výstupy. V horizontálním rozložení se každá vrstva chová jako samostatný agent, který ze vstupu určuje výstup [3], [1]. Z obrázku je patrné, že agent musí mít nějaký vnitřní systém, který vybírá pro konkrétní vstup konkrétní vrstvu pro zpracování. Pro další diskusi o rozhodovacím systému již odkážu na [3], [1]. Toto uspořádání jsem zvolil i ve svém projektu, o tom ale později.

Pro úplnost jen popíšu fungování hybridního agenta s vertikálním rozložením vrstev. U vertikálního rozložení vrstev jednotlivé vrstvy komunikují mezi sebou, avšak každá jen se svou sousední. Proces rozhodování o další akci tak může probíhat dvěma způsoby.

První způsob ( na obrázku 2. uprostřed ) zpracovává vstup první vrstva a svůj výstup předá vrstvě druhé. Druhá vrstva již nemusí znát původní vstup, musí rozumět výstupu první vrstvy. Druhá vrstva zpracuje vstup od první vrstvy, a tak dále, až n-tá vrstva vstup od n-1 vrstvy zpracuje na výstup celého agenta. Vrstvy tak definují několik transformačních funkcí, jejichž složením dostaneme funkci *VyberAkci* z kapitoly 1.2 .

Druhý způsob ( na obrázku 2. vpravo ) zpracovává vstup stejně jako první, jen poslední n-tá vrstva nedává svůj výstup jako výstup agenta, ale zpět vrstvě n-1 jako vstup. Každá vrstva $v_i$ se navíc může rozhodnout, že poskytované informace od nižší vrstvy $v_{i-1}$ jí stačí k tomu, aby vytvořila výstup pro nižší vrstvu $v_{i-1}$ přímo, bez nutnosti postoupení zpracování do vyšší vrstvy $v_{i+1}$. O výstup, tedy o výběr akce, se postará vrstva první. První vrstva se tak stává agentovým rozhraním pro prostředí.

# Kapitola 2

# Využití agentů v projektu

V dalším textu se budeme zabývat jen těmi případy z výše diskutovaných, pro které jsem našel uplatnění ve svém projektu simulace bitev robotů. Ačkoli by se mohlo zdát, že tato simulace je spíše multiagentní systém, nemyslím si to. V multiagentních systémech, kromě agenta a prostředí samotných, funguje jistá komunikace mezi agenty. Agenti se na základě prostředí a této komunikace rozhodují pro další akce, konkrétněji, jestli budou spolupracovat nebo nespolupracovat s jinými agenty. V této simulaci se nevytváří žádné aliance nebo jiné spolky, proto zde není nutné vytvářet žádnou komunikaci. Ostatní agenti se stávají součástí prostředí, na které agent reaguje. Jelikož jsou ostatní agenti zahrnuti do prostředí, zbývá pro další zkoumání jeden agent v prostředí bitevního pole.

Projekt byl inspirován deskovou hrou a jako ke hře k projektu bylo přistupováno při vývoji a nejinak tomu bude i nadále. Převzatá myšlenka hry je následující: robota ovládá pilot, který v robotovi přímo sedí a rozhoduje, co se stane za dané situace. Původní desková hra umožňuje hráči vybrat si z několika druhů robotů s různými speciálními vlastnostmi, tak i můj projekt nabízí výběr ze dvou robotů se svými speciálními vlastnostmi. Typy robotů jsou typ SOBOL a typ WASP. SOBOL má speciální zbraň, WASP má speciální pohyb. Pilota nahrazuje uživatel, který programuje obsluhu událostí robota. Reálný pilot může plánovat, proto je to dovoleno i uživatelům mého projektu.

## *2.1 Robot jako agent v prostředí bitevního pole*

Roboti použití v projektu odpovídají hybridnímu agentovi s horizontálním členěním, umístěnému do dynamického, nepřístupného, nedeterministického, neepizodického, kontinuálního a stavového prostředí.

Prostředí bitevního pole je dynamické kvůli zahrnutí ostatních agentů – robotů. Nepřístupné je díky robotovým vizuálním senzorům, které mají omezený dosah. O tom, co robot vidí viz kapitola 2.2. Nedeterministické je kvůli ostatním agentům, neboť se každý chová jinak. Bitevní pole umožňuje robotovi pamatovat si minulý stav, proto je neepizodické. Kontinuálnost prostředí je způsobena různou dobou provádění akcí robota. Stav prostředí je aktuální rozmístění robotů na bitevním poli. Bitevní pole je rozděleno na čtvercovou síť, kde robot může stát právě v jednom čtverci a na jednom čtverci může stát nejvýše jeden robot.

Čtvercová síť bitevního pole má omezenou šířku a výšku, proto je množina všech možných rozmístění robotů na bitevním poli omezená.

Robot přímo reaguje na události, které mohou nastat, a každá tato událost je obsloužena samostatnou částí robota. Tím je dáno horizontální členění a reaktivní chování robota. Protože na určitou událost může reagovat sekvencí akcí, pamatuje si nějaký stav, který je definován akcemi, které má naplánovány. Události, na které robot reaguje, jsou přijímány jak z bitevního pole, tak i od robota samotného, resp. jeho výbavy. Události výbavy reprezentují hlášení systému pro pilota robota.

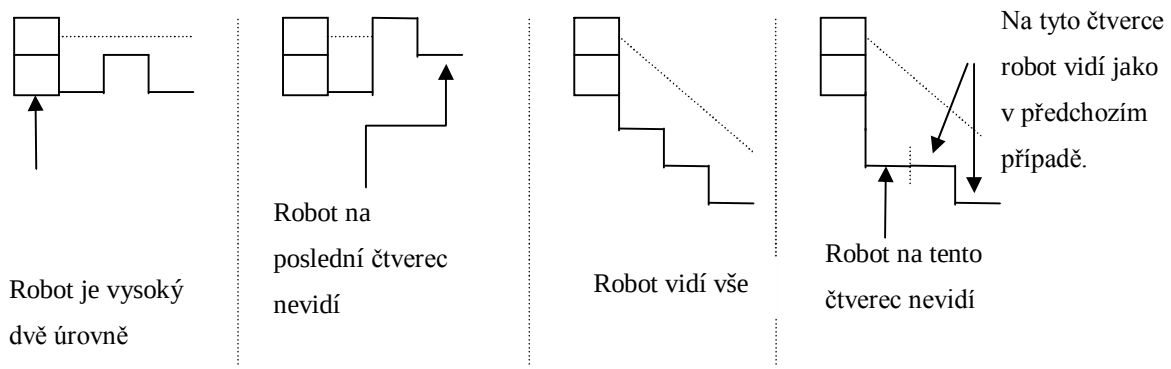
Krajina bitevního pole typicky obsahuje nějaké přírodní úkazy, jako kopec, údolí nebo svah.

Bitevního pole je členěno na úrovně. Každý čtverec krajiny má nějakou úroveň z rozsahu úrovní a definuje tak převýšení pro ostatní čtverce. Pro lepší znázornění uvedu příklad. Představme si, že máme dva čtverce. Jeden čtverec má úroveň rovnu 0, druhý úroveň 2. Mají tedy mezi sebou převýšení rovno 2. Formálně, nechť $B$ je množina čtverců bitevního pole a nechť $i, j \in B$. Nechť množina úrovní $U = \{-3, -2, -1, 0, 1, 2, 3\}$. Nechť úroveň čtverce $i$ je $u_i$ a úroveň čtverce $j$ je $u_j$ kde $u_i, u_j \in U$. Potom převýšení $p$ je definováno jako

$$p = | u_i - u_j |$$

## 2.2 Vjemy robota

Robotovy vjemy z bitevního pole reprezentují vizuální vjemy pilota. Pro úplnost si představme, že robot je dvě úrovně vysoký a pilot nevidí nad robotovu výšku, pod sebe však s jistým omezením vidí. Robot rozezná krajinu bitevního pole do vzdálenosti tří čtverců a jen tu, co má v zorném poli před sebou. Je-li v zorném poli robota kopec alespoň s převýšením 2 od úrovně čtverce, na kterém robot stojí, přes takový kopec nevidí, nevidí ani na vrchol kopce. Jsou-li v zorném poli robota čtverce takové, které směrem od robota klesají s převýšením 1, robot vidí na všechny políčka. Je-li však mezi dvěma sousedními políčky převýšení alespoň dvě, robot na nízké políčko nevidí. Situaci ilustruje obrázek 3.

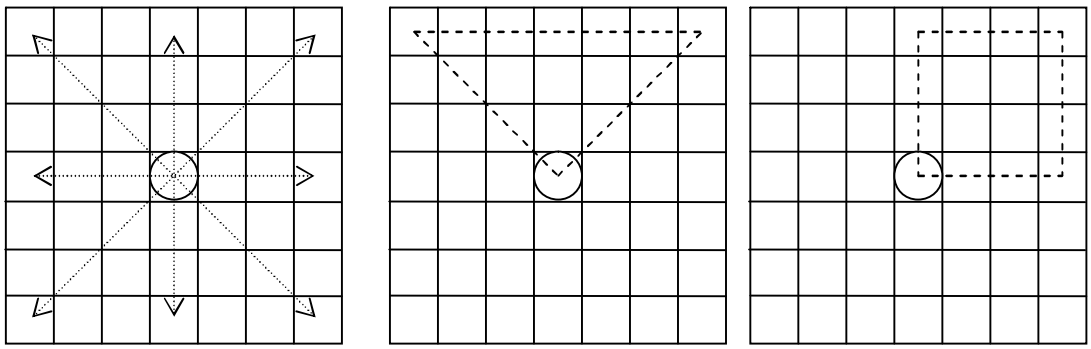



Obrázek 3. : Znázornění viditelnosti čtverců pro robota v závislosti na reliéfu terénu

Robot nevidí jen čtverce přímo v řadě před sebou, ale vidí nějakou výseč z bitevního pole. V zhledem ke čtvercové povaze bitevního pole, je na tomto poli definována vzdálenost čtverců jako maximová vzdálenost. Formálně, nechť $B$ je množina čtverců bitevního pole. Nechť $\check{S}$ je šířka bitevního pole a $V$ je výška bitevního pole. Nechť každý čtverec $b \in B$ má souřadnice $b_x$ , $b_y$ takové, že $0 \leq b_x < \check{S}$ a $0 \leq b_y < V$ . Potom pro každé čtverce $i, j \in B$ je vzdálenost $d$ definována jako

$$d = max( |i_x - j_x| , |i_y - j_y|)$$

Robot může být orientován jedním z osmi směrů, buď po hlavních osách čtvercové sítě, nebo úhlopříčně. Robotova orientace udává, jakou z osmi výsečí prostředí vidí. Výseč, kterou robot vidí, je definována jako čtvrtkruh o poloměru 3. Vzdálenost je však maximová, takže viditelná výseč je spíše čtvrtčtverec. Situaci orientace a plné vidění ilustruje obrázek 4.

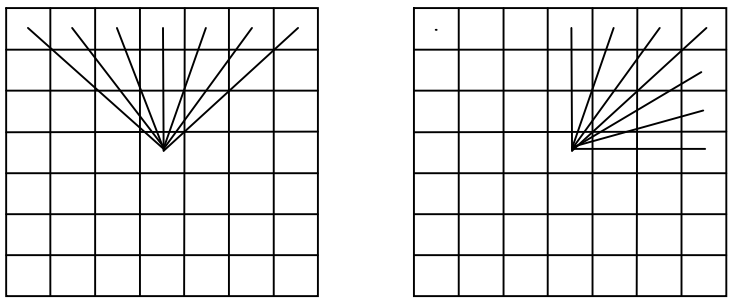


Jak může být robot orientován

Robotovo zorné pole při orientaci podle hlavní osy a při diagonální orientaci.

Obrázek 4. : Znázornění robotovy orientace a tvaru zorných polí při různých orientacích

Na obrázku 5 je znázorněno, přes které čtverce se robot dívá na vzdálenější čtverce. Přesněji, robot se dívá přes čtverce, které jsou vybrány Bresenhamovým algoritmem, který by maloval jednotlivé úsečky znázorněné na obrázku 5.



Obrázek 5. : Znázornění závislosti vidění robota po směru osy a v diagonálním směru.

Robot přijímá vizuální vjemy jen z těch čtverců, které splňují výše definované zákonitosti viditelnosti. Základní vizuální vjem, na který robot reaguje, není nic složitějšího, než že vidí na každý čtverec, který je ve výseči robotovy viditelnosti. Na vjem plné viditelnosti reaguje pouze v případě, že nemá žádné další vjemy od vnitřních systémů a nemá nic naplánováno. Vizuální vjem, kde robot již na nějaký čtverec z výseče nevidí, říká robotovi, že vlastně vidí překážku, tedy potencionální úkryt. Poslední vizuální vjem upozorňuje robota na přítomnost jiného robota v jeho zorném poli.

Vjem, který není příliš vizuální, ale tak trochu souvisí s viděním robota, je signalizace, že robot nemůže provést pohyb vpřed. Tento signál robot dostane vždy, když se pokusí pohnout na čtverec, na kterém stojí jiný robot a v pohybu chůze a běhu, když se pokouší vstoupit na čtverec s převýšením vyšším, než dvě. O pohybech bude diskuse v další kapitole s ostatními akcemi robota.

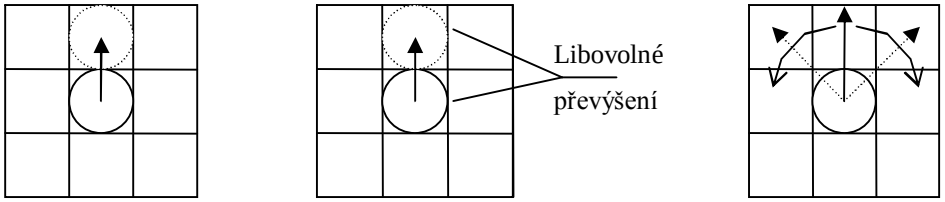
Další vjemy už jsou hlášení od vnitřních systémů robota a jeho vybavení. Důležitým hlášením je hlášení o lokalizaci, tj. o tom, že se robot nachází v zorném poli jiného robota. Robota tak hlášení o lokalizaci varuje před hrozícím nebezpečím. S lokalizačním hlášením souvisí i hlášení o zásahu, neboť hlášení o zásahu často přichází hned po hlášení o lokalizaci.

Je-li robot vybaven radarem, radar robotovi hlásí přítomnost jiných robotů kolem něj.

Pro splnění svého cíle, tedy přežít jako poslední robot, mohou být roboti vybaveni zbraněmi. Každá zbraň má svůj dostřel a počet výstřelů. Když se robot pokusí vystřelit ze zbraně, která nemá munici, dostane hlášení, že použitá zbraň nemá munici. Při pokusu střelby na robota mimo dostřel zbraně dostane robot hlášení, že cílový robot je pro danou zbraň příliš daleko.

## 2.3 Akce robota

Robotovy akce by se dali opět rozdělit na část, která patří přímo robotovi a část, kterou získá pomocí vybavení. Každá akce trvá nějaký čas a až po uplynutí tohoto času je robot opět schopen reagovat. Akce, které patří robotovi přímo jsou akce pohybu. Robot tak může jít o čtverec v před, běžet o dva čtverce v před tím, že přes oba skutečně projde. Jeden druh robota může i skákat, což mu umožňuje ignorovat výškové rozdíly mezi čtverci. Aby se robot mohl pohybovat skutečně libovolně, musí kromě pohybu vpřed zvládat i rotace, tedy otáčení. Robota lze otočit o osminu kruhu po směru hodinových ručiček nebo proti směru. Možnosti pohybu znázorňuje obrázek 6.




Chůze Běh Skok Přípustné rotace, když se robot dívá nahoru.

Obrázek 6. : Pohyby, které robot může vykonávat.

Množinu akcí, které robot získá z vybavení, obsahuje jen jednu akci, a to střelbu z každé zbraně, kterou byl vybaven.

V projektu je snaha o jakési ztvárnění reálného světa, proto každá akce trvá robotovi určitý čas, který je měřen na kola, a robot, resp. pilot by s tím měl počítat. Například, jen provedení akce chůze robotovi trvá 32 kol. Bohužel, měření času na kola stírá rozdíly mezi robotem, který se rozhoduje okamžitě a robotem, který dlouho vyhodnocuje situaci. Nelze tedy rozlišit robota spíše reaktivního od robota spíše stavového.

## 2.4 Proces zpracování vjemů a cíle robota

Protože k projektu bylo přistupováno jako ke hře, musí mít nějaký herní prvek. Herním prvkem je vývoj vlastního robota, tedy naprogramováním jeho chování. Hráči, tedy pilotovi robota, je nabídnut seznam vjemů, na které robot umí reagovat a na hráči – pilotovi je, aby navrhl chování pro daný vjem. Ve vrstevnatém modelu agenta to znamená, že hráč – pilot navrhuje, jakým stylem bude každá vrstva vybírat akce. Stejný robot, stejný ve smyslu typu robota a jeho vybavení, se tak na bitevním poli může vyskytovat v několika verzích, jen s jiným chováním.

Cílem robota je přežít jako poslední. Toho robot dosáhne jen tím, že bude vybaven zbraní, kterou bude schopen vyřadit ostatní roboty ze hry. Měl by být naprogramován tak, aby dokázal rozpoznat, kdy je dobré útočit, a kdy z boje ustoupit.

# Kapitola 3

# Poznatky z vývoje projektu

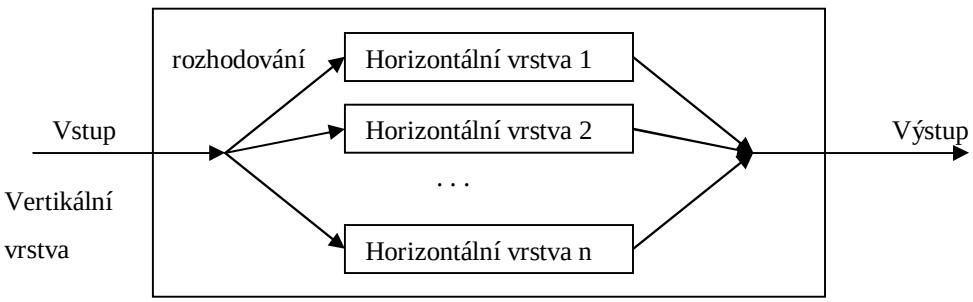
Celý projekt začal myšlenkou naprogramovat hru, kde by hráč na základě pravidel hry naprogramoval svého reprezentanta, který by hru hrál a hráč člověk by už pak jen sledoval, jak dobře zvládl naprogramovat vlastní umělou inteligenci pro svého reprezentanta. Bitva robotů byla z tohoto důvodu vhodnou volbou.

Projekt prošel vývojem, kdy robot byl implementován jako stavový agent. Umožňoval sice hráči reagovat na více podnětů současně, ale robot jako stavový agent byl v rozporu se simulací reálného času, neboť nejprve zpracoval všechny vjemy a až potom reagoval. Stavový agent by se dobře hodil do hry, kde čas určují kola hry a ve hře se nepočítá s reálným časem.

Robot jako čistě reaktivní agent nepřipadal v úvahu, protože by neuměl plánovat sekvenci akcí. Plánování totiž vyžaduje „udržet myšlenku“, tedy pamatovat si nějaké stavy, přes které agent chce projít, a to čistě reaktivní agent neumí.

Robot, který by se pohyboval v reálném čase, tedy musí být implementován jako hybridní agent, aby uměl reagovat na podněty okamžitě a aby uměl plánovat své akce dopředu. Reálný čas lze simulovat i časem počítaným na kola, ale každá akce potom musí trvat několik kol.

Vertikálně členěný agent dle mého názoru musí mít v každé vrstvě nějaký rozhodovací systém, který by agentovi určil, které vjemy přijal a jestli je lze zpracovat přímo a nebo jestli se zpracování pošle do vyšších vrstev. To mě ale přivádí k myšlence, že takové rozhodování odpovídá horizontálnímu rozložení vertikální vrstvy, jak je znázorněno na obrázku 7.




Obrázek 7. : Horizontální rozdelení do vrstev vrstvy vertikální

Z toho vyplynulo, že jediným správným agentem pro simulaci bitev robotů, který by uměl reagovat na podněty okamžitě, uměl plánovat a dovolil nabídnout hráči rozhodnout, jakou akci provést v jaké situaci, je robot jako hybridní agent s horizontálním členěním. Jeho

jedinou nevýhodou je, že neumí reagovat na více podnětů současně, ale musí si vybrat jeden podnět, na který bude reagovat.

Netvrdím, že ostatní agenti jsou špatní, nebo nedokonalí, jen zatím neexistuje agent, který by dokonale odpovídal „lidksému myšlení a reagování“. Každý agent, samozřejmě i robot v mém projektu, má proti lidem nedostatky, které z agentů dělají systémy, jenž lze využít jen ve speciálnějších prostředích.

# Literatura

[1] M. Wooldridge. Intelligent Agents In G. Weiss, editor (1999): Multiagent Systems, The MIT Press.

[2] S. Franklin(1997): Autonomous Agents as Embodied AI. Cybernetics and Systems, special issue on Epistomological Issues in Embodied AI, 28:6 (1997) 499-520.

[3] M. Wooldridge(2002): An Introduction to MultiAgent Systems, John Wiley & Sons Ltd.

# Příloha A

# ROBOTI 2005 PRO .NET 2.0

Uživatelská příručka

Pavel Matyska

## Obsah

# Úvodem

Tento program vznikl jako bakalářská práce na Matematicko-Fyzikální fakultě Univerzity Karlovy v Praze. Cílem práce je navrhnout robota a prostredí pro robota tak, aby uživatel mohl ovlivnit robotovo chování naprogramováním jeho vnitřní logiky. Program je vyladěn na rozlišení 1024 x 768. Program zpustíte zpuštěním souboru „Roboti 2005 BC.exe“ v adresáři .\Roboti 2005 BC\Roboti 2005 BC\bin\Debug .

## *1.4. Požadavky pro zpuštění programu*

Abyste mohli program zpustit, potřebujete mít na počítači, kde program chcete zpustit, nainstalován .NET Framework 2.0.

# Přehled částí programu

Celý program se zkládá z několika částí, které tvoří celé grafické rozhraní programu a ze knihovny pro roboty a prostředí. Podívejme se na jednotlivé části.

## *2.1 Úvodní okno*

Při zpuštění programu se otevře okno, které má jen v horní části menu a panel s rzchlými tlačítky. V tomto okně se bude odehrávat celý zbytek práce na vývoji robota a jeho prostředí.

Přes položku menu File můžeme otevřít okno pro tvorbu nového robota, položka New Robot, nebo tvorbu nového prostředí, položka New BattleField. Stejnou funkci, jako položka New Robot má i tlačítko New na panelu s tlačítky, když není žádné okno otevřené. Položka Open otevírá prázdné okno pro vývoj chování robota. Volbou Exit ukončíme celý program. O všech zmiňovaných oknech se více dozvíte v kapitolách, které se jimi zabývají.
V položce View můžeme zobrazit nebo schovat panel s tlačítky vzbráním položky Toolbar.
V položce menu Windows, kromě nastavování, jak se mají otevřená okna zobrazovat, je položka New Window. V této položce jsou uvedena všechna okna, která můžete otevřít. Okno otevřete jednoduše tím, ho v této položce kliknutím vyberete.

## *2.2 Okno vytváření prostředí*

Toto okno lze otoveřít pomocí voleb menu File-New BattleField, nebo Windows-New Window-BattleField Editor. Je-li toto okno aktivni, mění se funkce tlačítek na panelu s nástroji. Tlačítko New otevírá dialog pro novou mapu, Open otevírá dialog pro otevření již uložené mapy, a Save otevírá dialog pro uložení právě tvořené mapy. Tyto funkce zajišťují i tlačítka ve spodní části okna.

### 2.2.1 Vytváření nové mapy

Při prvním otevření okna se vytvoří mapa o minimálních rozměrech šířky a výšky. Mapa je čtvercová síť, na které jsou vynesené hodnoty o výšce terénu. První otevření okna tedy zobrazí mapu s minimálními rozměry a nulovou výškou na každém políčku. Je-li rozměr nedostačující, lze libovolný rozměr zvýšit na „libovolnou“ hodnotu, minimální rozměr mapy v daném směru je ten počáteční. Jak na to? Zmáčknutím tlačítka New nebo tlačítkem New BattleField si zobrazím dialog pro nové rozměry mapy, navolím požadované a potvrdím tlačítkem OK. Tento postup smaže původní práci!
A teď k vlastnímu editování. Máte-li malou mapu a velké okno, vidíte zelenou plochu s černým okrajem. Naopak, máte-li velkou mapu a malé okno, vidíte černý rám jen nahoře a nalevo od zelené plochy. Políčka s černou barvou už jsou „mimo“ mapu a nedají se editovat. Zelená plocha zobrazuje původní úroveň políčka, tedy nulovou výšku. Výšku políčka lze zvýšit nebo snížit, v obou případech však maximálně o tři stupně. Výsledný rozsah úrovní je tedy od -3 do +3. Zvýšení úrovně o jeden stupeň provedu kliknutím levým tlačítkem myši na

políčko, které chci zvýšit. Snížení úrovně provedu kliknutím pravým tlačítkem myši. Pro rychlou změnu úrovně podržím tlačítko myši a tažením měním políčka, přes které přejedu myší, podle toho, které tlačítko držím. Je to, jako bych maloval.
Jestliže černá políčka nevidíte, pravděpodobně máte otevřenou větší mapu, než se vám na obrazovku vejde a díváte se někam doprostřed mapy. Pohyb po mapě, která se nevejde na obrazovku, zajišťují scrollbary pod mapou a napravo od mapy. Scrollbary se zobrazí pouze v případě, že mapa v daném rozměru je větší než zobrazovací plocha okna.

#### 2.2.1.1 Co znamenají barvy na mapě

Při otevření nové mapy vidíte mapu jako jednolitou světle zelonou plochu. To je v pořádku, touto barvou je zobrazena 0 úroveň terénu. Při zvyšování terénu levým tlačítkem myši dané políčko tmavne. Čím víc tmavší zelená, tím je políčko výš. Naopak při snižování terénu pravým tlačítkem myši úrovně nižší než úroveň 0 modrají přes světle modrou až po nejtmavší. Opět platí, čím více tmavá modrá, tím nižší úroveň než úroveň 0.

### 2.2.2 Editace existující mapy

Pro editaci existující mapy ji musím nejprve otevřít. Provedu to zmáčknutím tlačítka Open a výběrem mapy, kterou chci otevřít. V editování postupuji, jako při tvorbě nové mapy.

### 2.2.3 Uložení mapy

Když jsem s prací na mapě hotov, svou práci uložím zmáčknutím tlačítka Save. V dialogu pro uložení napíšu nové jméno pro mapu a potvrdím tlačítkem Save. Mohu také přepsat existující mapu tak, že vyberu soubor, který chci přepsat. Program se mě zeptá, jestli skutečně má přepsat daný soubor. Myslím-li to vážně, potvrdím Yes a soubor se přepíše, nebo odmítnu zmáčknutím No a napíšu jméno mapy, nebo vyberu jiný soubor.

## *2.3 Okno výběru nového robota*

Teď se podíváme, jakým způsobem lze navrhnout nového robota.

Začneme tím, že otevřeme okno Design New Robot, nejrychleji asi přes menu File-New Robot (klávesová zkratka CTRL+N). V tomto okně provedeme výběr robota a jeho vybavení.

### 2.3.1 Výběr robota a jeho vlastnosti

Můžeme si vybrat mezi dvěma typy robotů, mezi typem SOBOL a typem WASP. Oba typy mají svojí jednu speciální vlastnost. Robot typu SOBOL má v sobě zabudovanou jednu zbraň, naproti tomu robot typu WASP vlastní zbraň nemá, ale má navíc jeden druh pohybu, skákání, a je odolnější než robot typu SOBOL. Odolnost lze ovlivnit brněním a udává, kolik poškození robot snese, než bude vyřazen. Záleží tedy na tom, zda chceme mít výhodu zbraně navíc, nebo lepší pohyblivost. Výběr provedeme tlačítky Next a Previous.

### 2.3.2 Výběr vybavení

Robotovi můžeme přiřadit až čtyři kusy vybavení. Na výběr máme ze tří zbraní, třech druhů brnění a radaru. Je tedy jen na nás, co robotovi dáme do vybavení. Jen ještě připomínám, že jde o kusy, tedy můžu přidat dva kusy stejného vybavení. Přidávané vybavení se zobrazuje pod obrazkem robota. Aby to ale nebylo tak jednoduché, každý robot unese jen nejaké zatíženi. Každé vybavení má nějakou váhu a jeho přidání sníží robotovu nosnost. Může se tedy stát, že vyberu příliš těžké vybavení a čtvrté, nebo dokonce už třetí vybavení robotovi nepřiřadím, protože by robot byl přetížen.

#### 2.3.2.1 Odebrání vybavení

Staneli-se, že chci robotovi přiřadit jiné vybavení, než jsem mu právě vybral, stačí jen poklepat dvojklikem na již přiřazené vybavení. Tím odstraním vybavení z robotova inventáře. Tento postup se může hodit v případě, že robot již více neunese, ale má možnost ještě jednoho vybavení. Odeberu tedy nejaké těžké vybavení a nahradím ho dvěma lehčími.

#### 2.3.2.2 Popis vybavení

Všechny důležité informace o vybavení, tedy váha, u zbraní počet výstřelů a dostřel, u brnění ještě odolnost, a u radaru dosah jsou uvedeny v tabulkách v pravé části obrazovky.

### 2.3.3 Potvrzení výběru robota

Když jsem s výběrem typu robota a jeho vybavení spokojený, dám robotovi jméno. Toto jméno musí být jiné, než již vytvořených robotů, abych nejakého nepřetvořil. Teď jsem ovšem prozradil, že když s nějakým robotem nebudu spokojený už od výběru vybavení, mohu ho takto změnit.
Když tedy mám vybraný typ robota, jeho vybavení a dal jsem mu jméno, stisknu tlačítko Submit. Jestliže jsem robotovi přeci jen, třeba omylem, dal jméno již existujícího robota, program se mě zeptá, jestli ho chci skutečně přepsat. Když nechci, akci zruším tlačítkem No a robota pojmenuji jinak. Při chtěném přepisu potvrdím akci stiskem tlačítka Yes a pokračuji v dalším okně.

## *2.4 Okno vývoje robota*

Výběr typu robota a vybavení v pozadí vygeneroval soubor se zdrojovým kódem v jazyce C#. Tento kód je připravená struktura na obsluhu událostí, které robot může zpracovávat. Doporučuji tento kód měnit jen vyplněním těl metod pro obsluhu událostí, přidáním vlastních metod a proměnných. V žádném případě neměňte kód konstruktoru ani inicializačních metod. Toto okno lze otevřít i pomocí položky menu Windows, New Window, Edit Robot, okno však bude prázdné. Další postup naleznete v kapitole Ukládání a otevírání zdrojových souborů robota.

### 2.4.1 Programování robota

Zdrojový kód upravím v okně s textem nagenerovaného kódu. Protože tuším, že nevíte, co můžete použít za příkazy, v pravé části okna jsou tři tabulky s nápovědou.

#### 2.4.1.1 Práce s nápovědou

Práce s nápovědou je poměrně jednoduchá a téměř intuitivní. Nejprve vyberu třídu, ze které chci použít metodu nebo vlastnost. Třídu vyberu ve spodní tabulce v pravé části okna. Výběr třídy provedu klepnutím na její jméno. Tím se mi v prostřední tabulce, tedy nad tabulkou s třídami, objeví seznam všech použitelných vlastností třídy a ve vrchní tabulce se mi zobrazí seznam metod vybrané třídy. Pak už stačí najet myší nad řádek s metodou nebo vlastností a v nápovědném okně se mi zobrazí text s použitím, hlavičkou a pod. Druhá varianta je otevřít si externí nápovědu, kterou najdete v adresáři doc v hlavním adresáři programu.

#### 2.4.1.2 Editace robota pomocí Visual Studia 2005

*Tato kapitolka je určena skušenějším programátorům a těm, kteří trochu ovládají Visual Studio 2005, proto nebudu zabíhat do podrobností a čtenář si jistě konkrétní cesty najde sám.*
Robota lze editovat i pomocí Visual Studia 2005 od firmy Microsoft. Robota budete tvořit jako knihovnu DLL, a do projektu si musíte přidat referenci na knihovnu se jménem RobotLibrary.dll. Hotovou knihovnu pak nahrajte do adresáře UserData v adresáři Roboti 2005 BC v hlavním adresáři programu. Tam je uložen i zdrojový soubor robota.

### 2.4.2 Ukládání a otevírání zdrojových souborů robota

Svou práci na robotovi mohu kdykoli uložit pomocí tlačítka save, nebo pomocí ikonky s disketkou v panelu nástrojů na hlavním okně programu. V dialogu vyberu soubor svého robota a potvrdím. Kdybych svou práci uložil do nového souboru, přijdu při příští editaci tohoto nového souboru o integrovanou nápovědu.
Otevření zdojového souboru robota provedu zmáčknutím tlačítka Open a výběrěm příslušného souboru. Zdrojový kód edituji jako normální program se všemi zákonitostmi jazyka C#.

### 2.4.3 Překlad hotového robota

Ve chvíli, kdy jsem royhodnut, že robot je již dostatečně naprogramován, musím ho přeložit, abych ho mohl později poslat do arény. Zdrojový kód nestačí pouze uložit. Co to je překlad

zde vysvětlovat nebudu, postačí, když řeknu, že se překlad robota provede sám po zmáčknutí tlačítka Build. Je-li robot syntakticky správně, tzn. překladač nenajde žádnou chybu, v další práci pokračujte stiskem tlačítka Choose Robots. Co když přeci jen nějaká ta chyba byla nalezena? Překladač všechny chyby eviduje a program vám je ukáže v okýnku pod zdrojovým textem. Opravte nahlášené chyby a stisknětě tlačítko Build. Co dělat dále je doufám jasné.

## *2.5 Okno výběru robotů a bitevního pole*

V tomto okně už se schyluje před velkou bitvou robotů. Jsou zde všichni úspěšně přeložení roboti a lze je odsud poslat na libovolné bitevní pole. Toto okno lze otevřít přes již snad známou volbu menu Windows, new Window a Choose Robots.

### 2.5.1 Výběr robotů

Tento krok je velmi jednoduchý. V okně vidíme jména robotů a u každého jména je čtvereček. Ty roboty, které chceme poslat do arény, označíme zaškrtnutím čtverečku u jeho jména. Ti roboti, kteří mají zaškrtnutý čtvereček u svého jména, budou posláni na bitevní pole. Zbývá tedy vybrat bitevní pole a podívaná může začít.

### 2.5.2 Výběr bitevního pole

Výběr bitevního pole provedu stlačením tlačítka Choose BattleField a v dialogu otevření souboru vyberu mapu bitevního pole. Pořadí tohoto kroku se s krokem výběru robotů může s klidným svědomím prohodit, tedy můžete nejprve vybrat bitevní pole a potom vybírat robota, které tam pošlete. Nebo mohu vybrat nejprve roboty, pak mapu, pak zase roboty, ... , dokud nebudu s výběrem spokojen. Svůj výběr potvrdíte tlačítkem Start.

## *2.6 Okno bitevního pole*

Zde bych chtěl především upozornit na skutečnost, že toto okno lze otevřít pouze z předchozího dialogu Výběru robotů. Toto okno slouží k zobrazení naprogramovaného chování vybraných robotů z předchozího dialogu.

Při otevření okna se nám zobrazí vybraná mapa, nebo aspoň její část, a roboti na ní. I když jsme robotovi naprogramovali, že se má nějak hýbat, zatím tak nečiní. Musíme jejich pohyb zpustit. Jak na to se dozvíte v další části.

### 2.6.1 Jednotlivé prvky okna

Toto okno je tím nejdůležitějším z celého programu. Nejprve se podíváme na prvky, kterými můžete ovlivňovat běh programu, pak se podíváme na vlastní zobrazení.

#### 2.6.1.1 Menu

V menu na horním okraji okna máte dvě položky na výběr. První položka pod názvem Start spouští a vypíná běh simulace, nebo ji ukončuje zavřením okna. Druhá položka pod názvem Zoom vám dává možnost přiblížit a oddálit se od mapy. Tuto možnost využijete tehdy, když budete sledovat simulaci na velké mapě. Položka Zoom In přibližuje dění na mapě, na obrazovku se vejde menší část mapy, položka Zoom Out oddaluje pohled na simulaci, máte tak širší rozhled po bitevním poli.

#### 2.6.1.2 Informace o robotech

Důležité informace o robotovi se zobrazují přímo u obrázku robota na bitevním poli. Pod robotem je uvedeno jeho jméno, nad ním pak akce, kterou právě provádí. Při otevření okna roboti ještě neprovádějí žádnou akci, proto není žádná nad robotem uvedena. Napravo od robota v horním řádku je uvedena jeho odolnost, ve spodním je uvedeno, zda je živý, nebo vyřazen z boje.

#### 2.6.1.3 Zobrazení bitevního pole

Hlavní částí okna je zobrazení bitevního pole. Jestli jste viděli okno vytváření prostředí (editor map), a vytvářeli jste mapu, kterou jste zadali, vidíte, že je stejná jako v editoru map. Pro význam jednotlivých barev viz odstavec Co znamenají barvy na mapě v kapitole Okno vytváření prostředí. Kromě zelených a modrých barev zobrazující mapu, ještě zde přibyly ikonky robotů. Pro odlišení typu je robot typu SOBOL zobrazen šedým robotem, robot typu

WASP černým robotem. Pohyb po mapě opět umožňují scrollbary napravo od mapy a pod mapou. Scrollbary se objeví pouze tehdy, je-li mapa příliš velká na zobrazovací plochu okna.

# Pravidla, jimiž se roboti řídí

V této kapitole se podíváme na pravidla, která roboti dodržují. Obsah kapitoly by vám měl pomoci k dobrému naprogramování robota, aby zrovna ten váš byl nejlepší. Roboti se pohybují „kolově“, každá akce trvá několik kol.

## *3.1 Akce robota*

Každý robot má základní sadu akcí, které může provádět. Tuto sadu má robot typu WASP rozšířenu o jednu akci, ale k tomu se ještě dostaneme. Robot typu SOBOL má vlastně také o akci navíc, ale je stejná jako akce spojená s vybavením. Ale o tom také později.

### 3.1.1 Pohyb

Každý robot může vykonávat tři různé pohyby, jít, běžet nebo otáčet se. Robot typu WASP navíc může skákat. Každý pohyb něco stojí. Ke každé akci tedy uvedu počet kol, kolik daná akce stojí.

#### 3.1.1.1 Chůze

Chůze je pohyb, který robota posune o jedno políčko dopředu a robota stojí 32 kol. Jestliže nemůže jít, vyvolá událost CantGo a bude reagovat podle ní. To, jestli může jít, robot zjistí v prvním kole, pak už na pole před sebou může. Robot nemůže jít na políčko, na kterém stojí jiný robot, nebo které je o dvě úrovně výš nebo níž, než úroveň políčka, na kterém robot právě teď stojí. Roboti se mohou potkat na stejném poli, jestliže se ve stejném kole rozhodli, že na dané pole vstoupí. V libovolném kole z dalších 31 kol se roboti „potkat“ mohou, stojí totiž „stále na stejném poli“. Teprve po dokončení pohybu se jsou skutečně na cílovém poli.

#### 3.1.1.2 Běh

Běh je to samé, jako chůze, jen přesun na pole před robotem trvá jen 16 kol. Běh je tedy „dvakrát“ rychlejší, než chůze. Nemůže-li robot běžet, vyvolá se opět událost CantGo a robot bude vykonávat instrukce podle této události.

#### 3.1.1.3 Otáčení neboli rotace

Rotace je pohyb, který robota neposouvá na nové políčko, ale otáčí ho na současném políčku o nějaký úhel a mění mu tak zorné pole. Robot se může otočit o osminu kruhu vlevo nebo vpravo. Otočení o osminu kruhu trvá 1 kolo.

#### 3.1.1.4 Skákání

Pohyb skákání u robota typu WASP je časově nejnáročnější, trvá 43 kol. Skákání posunuje robota o jedno políčko stejně jako chůze. Zatím to vypadá, že to je naprosto zbytečný pohyb. Výhoda tohoto pohybu spočívá v možnosti přesunu na libovolné políčko, na kterém není jiný robot. Skákání tedy umožnuje překonat libovolnou terénní překážku.

### 3.1.2 Zbraně

Základní akce lze rozšířit o vybavení robota zbraněmi. Zbraně jsou určeny k likvidaci ostatních robotů. Všechny zbraně jsou střelné, proto rozšiřují akce robota o střelbu z dané zbraně. Jestliže robot bude chtít vystřelit na robota, který bude dál, než je dostřel dané zbraně, střelba se neprovede a vyvolá se událost RobotTooFar.
Každá zbraň má své výhody i nevýhody. Robot typu SOBOL má v sobě zabudovanou zbraň, která útočí sonickým útokem. Její přední výhoda je v tom, že robot je jí automaticky vybaven a nezabere mu žádnou položku dalšího vybavení. Robot typu SOBOL může tedy mít až pět kusů vybavení.

#### 3.1.2.1 Laser

Laser je zbraň určená k útoku na blízko. Její dostřel je pouze jedno políčko, laserem robot může útočit pouze na políčka těsně sousedící s jeho políčkem. Je to ale zbraň nejlehčí a způsobuje střední zranění. Další výhodou je neomezený počet střel. Jedno vystřelení trvá 25 kol.

#### 3.1.2.2 Canon

Canon je zbraň středního dostřelu. Lze ji použít na roboty vzdálené až dvě políčka, tedy o jedno dál, než laser. Canon je o něco těžší, než je laser a má omezen počet výstřelů. Jeho zásobník je naplněn na 100 výstřelů, každý výstřel trvá 10 kol.

#### 3.1.2.3 Bazooka

Bazooka je zbraň dlouhého dostřelu a těžkého poškození. Její dostřel jsou tři políčka. Je ale z uvedených zbraní nejtěžší a má jen deset výstřelů. Každý výstřel trvá 50 kol, ale zato účinek má strašlivý.

#### 3.1.2.4 Sonický útok

Touto zbraní disponují pouze roboti typu SOBOL. Díky tomu, že je tato zbraň přímo v robotovi, nepociťuje robot její váhu. Protože její útok je pomocí zvuku a ne projektilů, má neomezený počet výstřelů. Účinný dosah této zbraně jsou také tři políčka. Její účinek je ale proměnlivý, závisí na zbývající výdrži robota. Způsobuje poškození podle vzorce

*poškození = zbývající výdrž / 5*

Každý útok touto zbraní trvá 35 kol.

## 3.2 Události, na které roboti reagují

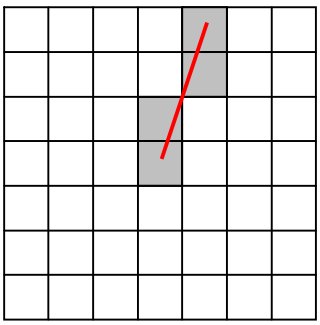
Kromě základní sady akcí má každý robot i základní sadu událostí, na které reaguje. Na některé jsme narazili v minulých kapitolách. V této kapitole se na ně podíváme blížeji a o každé si něco povíme. Základní sada se rozšiřuje podle použitého vybavení.

### 3.2.1 Události pohybu

Události pohybu jsou spíše hlášení, že robot daný pohyb provést nemůže. Jak bylo řečeno výše, událost CantGo nastává v případě, že robot nemůže jít, přestože chůzi měl naprogramovanou. Událost CantGo také nastává v jednom ze dvou kroků běhu. Pro skákání může nastat podobná událost CantJump. Tato událost, jak vyplývá z pohybu skákáním, může nastat jen v případě, že na políčku, kam robot chce skočit, stojí jiný robot. Otáčení žádnou událost nevyvolává, protože celý pohyb se odehrává na stejném políčku, kde robot stojí, a není tedy důvod pro žádné hlášení. Mohou nastat jiné události, ale ty se již netýkají pohybu.

### 3.2.2 Události vidění

Události vidění jsou události, které informují robota o tom, co vidí. Robot standardně předpokládá, že vidí na všechny políčka ve svém zorném poli. Jeho zorné pole má dosah 3 políčka vpřed a rozptyl do čtvrt kruhu (na čtvercové síti to ale vypadá jako čtvrt čtverce). Robot vidí před sebe podle závislostí polí. Robot musí nejprve vidět na pole blíže k sobě, aby viděl na pole dál od sebe. Viditelnost polí je určena úsečkami o délce 4 čtverce, které se položí na čtvercovou síť. Čtverce, nad nimiž se úsečka nachází, definují závislost polí na viditelnosti. Jedna taková úsečka je zobrazena na obrázku. Jestliže však nevidí na všechny, znamená to, že je na nějakém políčku terénní nerovnost.



#### 3.2.2.1 Proč roboti nevidí vše

Terénní nerovnost může být „údolí“ nebo „kopec“. Kopec je taková nerovnost, že políčko tvořící kopec je alespoň o dvě úrovně výš, než políčko, na kterém stojí robot. Přes kopec robot samozřejmě nevidí, takže co je za ním, je schováno. Přes kopeček, který je nejvýšo o jednu úroveň výš než robot, robot vidí, jako by tam byla rovina. Podobné je to s viděním přes údolí. Robot nevidí do velké hloubky pod sebou, ale do daleké hloubky vidět může. Do hloubky robot vidí stylem políčko dál, úroveň níž. Nejlépe si to vysvětlíme na příkladu. Řekněme, že robot stojí na políčku s úrovní 1, před sebou má dvě políčka úrovně -1 a nejvzdálenější políčko je opět úrovně 1. Protože plíčko těsně před robotem je o dvě úrovně níž, ale je jen o jedno políčko vzdálené, robot tam nevidí. Druhé políčko je ale vzdáleno od robota o dvě políčka, jeho úroveň je o dvě nižší, takže robot již na toto políčko vidí. Poslední políčko, které má stejnou úroveň, jako políčko, na kterém stojí robot, je viditelné stejně, jako by tam žádné údolí nebylo. Stejně by bylo vidět políčko s úrovní 2.

#### 3.2.2.2 Robot hlásí „nevidím“

V případech, které jsem popsal v minulém odstavci, robot nahlásí událostí SeeShelter, že jsou políčka, na která není ze současné pozice vidět. Může se tam tedy ukrývat robot, ať už cizí, nebo můj.

#### 3.2.2.3 Robot hlásí „vidím“

Důležitou událostí je samozřejmě událost, kterou robot hlásí, že vidí jiného robota. Jiného robota může vidět jen na políčkách, kam skutečně vidí. Když vidí robota, zahlásí to událostí SeeRobot.

#### 3.2.2.4 Událost robot na radaru

Je-li robot vybaven radarem, může nastat situace, že se do dosahu radaru dostane jiný robot. V takovém případě nastane událost RobotOnRadar. Zaměření robota přes radar má dvě výhodz. Cílový robot není varován, že je zaměřen a robot nemusí být v zorném poli robota,

aby na cílového robota mohl střílet. V případě, že je cílový robot i v zorném poli, tato událost nastává před událostí vidím robota.

### 3.2.3 Události boje

#### 3.2.3.1 Robot hlásí „jsem zaměřen“

S minulou událostí SeeRobot souvisí událost Located. Tato událost nastává u robota, jestliže se ocitl v zorném poli jiného robota.

#### 3.2.3.2 Robot hlásí „zasažen“

Po události Located často nastává událost Hitted. Tato událost oznamuje robotovi, že byl právě zasažen. Události nastávají v tomto sledu. Nejprve se vyvolá událost u zaměřeného robota, že je v zorném poli, hned za ní následuje událost, že robt vidí jiného robota. Většinou, v rámci vlastního přežití, asi robot bude rovnou střílet, bude li mít nějakou zbraň, a tak u zasaženého robota nastane událost zasažen. Tato událost může nastat i v případě, že robot byl zaměřen a zasažen pomocí radaru.

#### 3.2.3.3 Události zbraní

Je-li robot vybaven libovolnou zbraní, vždy může nastat alespoň jedna událost při jejím použití. To, že při střetu nejdříve nastává událost Located před událostí SeeRobot dává robotům šanci utéci. Cílový robot uteče, ale robot, který ho viděl, si stále pamatuje, kterého robota viděl a může na něj chtít vystřelit. Jestliže se cílovému robotovi podaří utéct, zbraň, kterou robot chtěl použít nahlásí, že je cílový robot daleko událostí RobotTooFar. Tato událost nastává i v případě, že cílový robot zůstane v zorném poli robota, ale robot použije zbraň s příliš krátkým dostřelem.
Zbraně, které mají omezený počet výstřelů, tedy Canon a Bazooka, mohou při použití nahlásit, že jim došla munice událostí OutOfAmmo.

### 3.2.4 Speciální událost

Speciální událost AllOk vlastně ani není událost. Je to chování, když žádná předchozí událost nenastává. Nadneseně bychom mohli říct, že to je robotovo chování v krásný klidný slunečný den.

### 3.2.5 Zpracování událostí

Události se zpracovávají funkcemi, které mají jméno skoro stejné, jako událost, kterou zpracovávají, jen začínají slovíčkem On. Například událost CantGo zpracovává funkce OnCantGo, událost Hitted zpracovává funkce OnHitted. Na všechny události, které váš robot může vyvolávat a přijímat, vám program vygeneruje hlavičky obslužných funkcí. Vaším úkolem už je jen naplnit těla funkcí požadovaným chováním a pak už se jen bavit.

# Příklad programování robota

Již víme, podle jakých pravidel se robot chová. V této kapitole si ukážeme konkrétní metody, které obsluhují jednotlivé události a které metody můžeme použít k „oživení“ robota při programování jeho chování. Nejprve ale musíme mít nějakého robota, kterého budeme programovat.
Celý zdrojový kód naleznete v adresáři „Roboti 2005 BC\UserData“ v souboru Vzor.cs.

## *4.1 Návrh vzorového robota*

Jako vzorového robota si vybereme robota typu WASP. Otevřeme tedy okno výběru nového robota z kapitoly 2.3. Tlačítkem „next“ přepneme typ robota na robota typu WASP. Můžeme si všimnout, že náš robot unese 200 váhových jednotek. Tento údaj je v pravém dolním rohu okna. Aby náš robot byl odolnější, vybereme ze seznamu brnění položku „heavy amor“. Je to sice nejtěžší brnění, ale také nejodolnější. Ikonka brnění se objeví nejlevějším čtverci pod obrázkem robota. Dále vybereme ze seznamu zbraní položku „bazooka“, protože je to nejúčinnější zbraň, a položku „canon“, neboť má velkou palebnou kadenci. Nyní, protože robot unese již jen 20 jednotek váhy a jen jedno vybavení, mohli bychom ho vybavit jen

„laserem“, lehkým brněním „light armor“, a nebo radarem. Vybereme si radar, neboť dává robotovi lepší přehled o jeho okolí. Robota pojmenujeme podle svého uvážení, já ho pojmenuji „vzor“. Svůj výběr vybavení, typu robota a jeho jména potvrdíme tlačítkem „submit“.

## 4.2 Programování robota

Jak bylo řečeno výše, výběr z předchozí kapitoly nám negeneroval zdrojový kód pro tuto část vývoje robota. Zde opět upozorňuji, že neopatrný zásah do negenerovaného kódu může způsobit nefunkčnost robota, kterého edituji. Proto neměňte kód v metodě InitWeapons()! Ale teď už k programování chování. Všimněte si, že program negeneroval hlavičky funkcí, které souvisí s chováním robota. Jsou to funkce, které obsluhují události, na které robot umí reagovat.

### 4.2.1 Obsluha události AllOk

První metodou, která obsluhuje nějakou událost, je metoda OnAllOk. Tato metoda obsluhuje událost AllOk, tedy událost, kdy se pro robota neděje nic zajímavého. To, co bude uvedeno v této metodě, bude robot provádět vždy, když může nastat tato událost a nemá naplánovány akce z minulých událostí.
Tak tedy, náš robot bude v situaci, kdy se nic neděje pouze chodit dopředu. Chůze se provede příkazem Go(). Celý kód metody tedy bude vypadat takto:

```
public override void OnAllOk(RobotEventArgs re)
{
      Go();
}
```

### 4.2.2 Obsluha události CantGo

Událost CantGo nastává, když má robot napláováno provést krok, ale z nějakého důvodu krok udělat nemůže. Tuto událost obsluhuje metoda OnCantGo. V této metodě se nejprve podíváme, jestli nemůžeme jít kvůli tomu, že na políčku před námi stojí jiný robot, nebo jestli

je tam terénní nerovnost. Robot tedy nejprve zjistí, jestli na políčko před sebe vidí. Jestli ano, zkontroluje, že na něm opravdu stojí robot a střelí po něm ze zbraně „canon“. Jestli na políčko neuvidí, otočí se robot o čtvrt kruhu doprava. Metodu tedy naprogramujeme takto:

```
public override void OnCantGo(RobotEventArgs re)
{
      Field f = FieldInDirection();
      if (f != null)
      {
            if (f.Robot != null)
                  ((Canon)Items[2]).Shoot(f.Robot);
      }
      else
      {
            Rotate(RobotRotation.Right);
            Rotate(RobotRotation.Right);
      }
}
```

### 4.2.3 Obsluha události SeeRobot

Událost SeeRobot je událost, která robotovi říká, že vidí alespoň jednoho robota a může tedy po něm zkusit střelit. Robot tedy nejprve zkontroluje, jak je cíl daleko, a podle toho zvolí zbraň. Jestliže bude cíl tři pole daleko, vystřelí ze zbraně „bazooka“, bude-li blíž, pokusí se vystřelit třikrát ze zbraně „canon“. Robot bude vždy střílet na prvního, kterého uvidí. Zdrojový kód metody bude vypadat následovně:

```
public override void OnSeeRobot(RobotEventArgs re)
{
      if (Distance(re.Robots[0]) > 2)
            ((Bazooka)Items[1]).Shoot(re.Robots[0]);
      else
      {
            ((Canon)Items[2]).Shoot(re.Robots[0]);
            ((Canon)Items[2]).Shoot(re.Robots[0]);
            ((Canon)Items[2]).Shoot(re.Robots[0]);
      }
}
```

Samozřejmě lze napsat volání metody Shoot zbraně Canon v cyklu.

### 4.2.4 Obsluha události Located

Událost Located má pro robota značný význam. Signalizuje mu, že ho vidí jiný robot a že na něj může střílet. Náš robot se rozhodne, co udělá, v závislosti na tom, jestli také vidí nějakého robota. Jestli uvidí, bude střílet z Canonu, jestli ne, bude utíkat pryč. Celý kód metody:

```
public override void OnLocated(RobotEventArgs re)
{
      if (RobotsInView.Count > 0)
            ((Canon)Items[2]).Shoot(RobotsInView[0]);
      else
            Run();
}
```

### 4.2.5 Obsluha události Hitted

Každý programátor robota si určitě bude přát, aby tato událost nenastávala často, neboť robotovi říká, že byl zasažen. V takovém případě náš robot bude utíkat jako zajíc, tedy, poběží, změní směr, a zase poběží, a na konec se obrátí do směru, ze kterého přišel. Metoda obsluhující tuto událost by mohla vypadat třeba takto:

```
public override void OnHitted(RobotEventArgs re)
{
      Run();
      Rotate(RobotRotation. Left);
      Run();
      Rotate(RobotRotation.Right);
      Rotate(RobotRotation.Right);
      Run();
      Run();
      Rotate(RobotRotation.Left);
      Rotate(RobotRotation.Left);
      Run();
```

```
        Rotate(RobotRotation.Left);
        Rotate(RobotRotation.Left);
        Rotate(RobotRotation.Left);
}
```

### 4.2.6 Obsluha události SeeShalter

Událost SeeShalter nastává v případě, že robot nevidí na každé políčko, které by mohl vidět. Necháme opět robota rozhodnout podle toho, co skutečně vidí. Jestli bude mít před sebou volno a jestli bude mít naplánovány jiné akce, nebude dělat nic nového a bude provádět dál naplánované akce. Jestliže bude mít před sebou volno, ale nic naplánováno mít nebude, půjde vpřed a nakonec, jestli bude mít před sebou terénní překážku, skočí na ni. Následuje opět kód metody:

```
public override void OnSeeShelter(RobotEventArgs re)
{
        if (FieldInDirection() != null)
        {
                if(HasAction)
                        return;
                else
                        Go();
        }
        else
                Jump();
}
```

Připomínám, že akce Jump je speciální akcí pro robota typu Wasp, u robota typu Sobol žádnou takovou akci nenajdeme.

### 4.2.7 Obsluha události CantJump

Událost CantJump, podobně jako událost CantGo, robotovi říká, že se nemůže na pole před sebou přesunout. Když robot nemůže na pole ani skočit, nezbývá mu nic jiného, než stát, nebo se otáčet. Protože kdo se nehýbá, je snadný terč, našeho robota v takovém případě otočíme, třeba na druhou stranu, než by se otáčel, když nemůže jít, tedy doleva.

```
public override void OnCantJump(RobotEventArgs re)
{
      Rotate(RobotRotation.Left);
      Rotate(RobotRotation.Left);
}
```

### 4.2.8 Obsluha události OutOfAmmo

Událost OutOfAmmo robotovi říká, že ze zbraně, ze které chtěl střílet, střílet nelze, neboť již nemá náboje. Tuto zbraň bez nábojů dostane v metodě k dispozici. Náš robot se zachová tak, že když mu dojdou náboje ve zbrani „bazooka“, rozeběhne se, bude-li to zbraň „canon“, otočí se čelem vzad a rozeběhne se.

```
public void OnOutOfAmmo(WeaponEventArgs re)
{
      if (re.Item.Name == "bazooka")
      {
            Run();
            return;
      }
      if (re.Item.Name == "canon")
      {
            Rotate(RobotRotation.Back);
            Run();
      }
}
```

### 4.2.9 Obsluha události RobotTooFar

Událost RobotTooFar je opět událost spojená s bojem a použitou zbraní. Říká robotovi, že cílový robot je pro danou zbraň mimo dostřel. Nastává v momentě, kdy cílový robot je ještě vidět, ale než náš robot stačí vystřelit, zmizí mu cílový robot z dostřelu. Takovou situaci vyřešíme opět rychlým pohybem vpřed, tedy během.

```
public override void RobotTooFar(WeaponEventArgs re)
{
      Run();
}
```

### 4.2.10 Obsluha události RobotOnRadar

Na událost RobotOnRadar může reagovat jen robot, který byl vybaven radarem. Náš robot radar ve své výbavě má, proto může jeho výhod využít. Náš robot nebude čekat, až si ho některý z robotů v jeho okolí všimne, ale rovnou bude střílet ze zbraně, kterou způsobí nejvíce škody, tedy ze zbraně „bazooka“.

```
public void OnRobotOnRadar(RobotEventArgs re)
{
      ((Bazooka)Items[1]).Shoot(re.Robots[0]);
}
```