Kapitola 1

Uvod

Algoritmus je vseobecne znamy pojem oznacujuci postupnost jednoduchych prikazov, po vykonani ktorych je pozorovatelny nejaky vysledok. Je uvarena kava, upratana izbu, vypocitany $log_5(1000000)$. Vysledok algoritmu mozeme chapat aj ako kriterium uspesnosti a algoritmus, ktory toto kriterium splna, povazovat za uspesny.

Samotnych uspesnych algoritmov s danym nemennym kriteriom sa da najst viacero. Obvykle pouzivaju odlisne pristupy, co sa prejavuje ich rychlostou, spotrebou zdrojov (pristup na disk, pamat, vystup) atd. O uspesnosti algoritmus rozhoduje casto az jeho pouzitie. Preto je nutne naviac definovat aj hodnotiacu funkciu, ktora z mnoziny uspesnych algoritmov vyberie ten najvhodnejsi (optimalny).
Hodnotiaca funkcia sa lisi od kriteria uspesnosti. V kriteriu je v okamihu ukoncenia jednoznacne dane, ci je algortimus uspesny alebo nie (prosta boleavska funkcie). U hodnotiacej funkcie je to len vyber z mnoziny, ktora spravidla nie uplna. Napriklad, ak hladame algoritmus, ktoreho kriteriom je usporiadat postupnost celych cisel, potom existuje niekolko znamych algoritmov (quicksort, bucketsort atd.) V okamihu, ked mame odmedzene mnoztvo pamate, hodnotiaca funkcia vyberie jeden algoritmus, ale nie je zrejme, bez dalsieho dokazu, ze neexistuje vhodnejsi aloritmus. Pod problematikou boja algoritmov budeme v tejto praci rozumiet napisanie uspesnych algoritmov a ich nasledne vyhodnotenie v danom prostredi (v svete konstant a premennych) pri jednom zvolenom kriterii uspesnosti. Tak sme sa dostali k nazvu prace CODEWAR - vojna algoritmov. No v pomerne zuzenom kontexte.

Vymyslanie novych algoritmov zvysuje kretivitu <referencia na nejaky vyskum ASI netreba> ich tvorcov, najma ak ide o riesenie problemu netradicnym sposobom a zlepsenie znamych algoritmov. Napriklad v Strassenovom algoritme pre nasobenie matici sa hladal taky algoritmus, ktory by potreboval co najmenej pamati a kriteriom uspesnosti boli mocniny matic.

1.1 Motivacia

Dobrym prostriedkom, ako vzbudit zaujem o vymyslanie algoritmov, je apelovat na prirodzenu ludsku sutazivost a spravit z problemu hru. Napriklad v hre Herbet (**??**), je svet podobne ako v sachu rozdeleny na cierne a biele policka. Uspesny

algoritmus je taky, ktory dovedie robota kroku po pochodzke po vsetkych bielych policka, neprerusenym sledom na cierne policko. Hodnotiacou funkciou je pocet pismen v zapise algoritmu, cim mensi pocet pismen, tym lepsi algoritmus.

Zmysluplnych kriterii uspesnosti algoritmu je vela. Preto sa tato pracai obmedzuje iba na typ hier so zameranim na prezitie robotov. Prezitie robotov znamena, ze hrac ovlada zvolenymi algoritmami jednu alebo viacero postav. Zakladnym kriteriom bude zostat nazive co najdlhsie spomedzi vsetkych robotov. To znamena, ze treba reagovat na chovanie ostatnym robotom skodim alebo vyhybanim.

Hry, z ktorych praca cerpala inspiraciu, su spomedzi inych tieto:

- ARES je hra dvoch hracov. Odohrava sa na mieste simulujucom pamat pocitaca. Ulohou hraca je naprogramovat robota v tomto prostredi v strojvom kode (aseembleri). Kriteriom uspesnosti je program, ktory sa bude vykonavat (prakticky hocijaky) a hodnotiacou funkcou je cas, za ktory program pobezi. Program skonci v okamihu, ked sa pokusi vykonat neplatnu intrukciu (napr. delenie nulou, skok na adresu nula, prazdnu instrukciu). Cielom je tak prepisat pamat takym sposobom, aby sa druhy program ukoncil. Hrac dopredu nepozna ani program protihraca ani data, ktorymi je inicializovana pamat, ak sa hraci dopredu nedohodnu. Oba programy su podobne ako v realnom svete ulozene v pamati pocitaca a kedze pamat je zdielana, mozu si navzajom prepisovat data alebo dokonca instrukcie.

- POGAMUT je uz vysoko komplexna hra n robotov. Algoritmy sa daju programovat v Jave, cim je dovolene pouzivat specialne znaky jazyka, ako je napriklad pretazenie, dedicnost atd. Prostredie je trojrozmerne a tym maju roboti moznost sirokej skaly hybov, skakania po stena, sklony, pohlady hore a dole. Sposob, akym sa ublizuje dalsim robotov, je daleko viac intuitivnejsi, robot vystreluje obmedzene mnozstvo striel a ma na vyber viac zbrani, ktore sa lisia tym, ako presne zasahuju. Hrac ma dokoca moznost rucne riadit vlastneho robota proti naprogramovanemu a tym otestovat vhodnost jeho algoritmu.

Uvedene hry uspokojujuco splnaju zakladnu problematiku boja algoritmov. V ARESe je kriteriom zostat nazive, v POGAMUTe je naviac vopred dane kriticke mnozstvo protivnikov. Hodnotiaca funkcia je rychlost, kto skor splni ciel, vyhrava.
Zostrojenie takejto hry ale vyzaduje podrobnejsi prehlad o narokoch na jazyk, svet a samotnych robotov. Preto sa zameriame na nasledovne charakteristiky tychto hier:

Priestor hier Kym v ARESe ide o 1D priestor (jedna velka pamat - pole), boj v POGAMUTe sa odohrava v 3D priestore. S tym suvisi pohyb po svete. 3D priestor ma omnoho viac moznosti, ako realizovat pohyb. Je nutne zvazit, ci bude povolene lietanie, padanie, pohlad zhora, zdola, vrhanie zbrani zboku, a v akych smeroch sa objekty sveta odrazaju a podobne. V ARESe sa o pohybe, ako ho pozname (plynuly prechod z miesta A na miesto B) neda ani hovorit, pretoze vsetky akcie suviace so svetom su instrukcie a zmeny v pamati.

Kriteria uspesnosti . V ARESe je jasne, ze hra skonci, ked hrac nedokaze nadalej vykonavat svoj program . V POGAMUTe je situacia o poznanie horsia: Pri programovani robota sleduje programator(hrac) dva ciele (a) bud naprogramovat takeho robota, ktoreho zdolat bude vyzva, alebo (b) naopak takeho robota, o ktorom sa vseobecne vie, ze sice bude porazitelny, ale nie je lahke ho obist. To znamena, naprogramovat takeho robota, ktore zdolavat bude vyzvou, ale ktory bude mat chyby, ktorych sa da vyuzit.

Sposoby boja POGAMUT naviac od ARESa implementuje omnoho viac sposobov, ako ublizit robotovi, od roznych zbrani po odrazenie guliek, rychleho spadnutia a zem atd. Jedinym sposobom, ako je mozne v ARESe poskodit protivnikovi, je prepisat mu tu cast pamati, o ktorej je dovod predpokladat, ze ju bude v dohladnej dobe potrebovat. Vyhodnotenie algoritmu nie je tak mozne po castiach, ale az po skonceni celej simulacie. V POGAMUTe je mozne rozlisit uz v priebehu simulacie, ako a ci robot zasiahol protivnika.

Vykonavanie programu Dalsim pristupom, ktory je v vytvarani hry dolezity, je aj sposob, v akom poradi su akcie hracov interpretovane. Aby ostatni hraci neboli znevyhodneny, je vhodne vykonavat jednotlive casti nezavisle na ostatnych robotov podla jednotnych pravidiel pre vsetkych. V ARESe je to jednoduche, hra prebieha po kolach. Kazde kolo znamena vykonanie aktualnej instrukcie, co je to spravodlive pre vsetkych hracov. V POGAMUTe je nutne zaistit paralelizaciu, aby robot nebol zavisly na vykonavani programu ostanych robotov.

1.2 Ciele prace (iluzie - tvrda realita)

Bakalarska praca sa zaobera vytvorenim vhodneho prostredia, kde moze uzivatel menit svet, v ktorom sa suboj bude odohravat, naprogramovat robota a zistit uspesnost napisaneho algoritmu.

Na zaklade analyzy vyssie uvedenych pristupov k CodeWars boli vytipovane tieto ciele:

Vytvorenie sveta Naprogramovany robot by mal zit vo svete, kde je jednoduche sledovat postup vykonavania jeho algoritmu. Z toho vyplyva narok na prostredie, v ktorom sa bude suboj robotov odohravat. Sucastou sveta budu objekty, ktore interaguju s robotmi a prinasaju tak do vymyslania strategii komplikovanejsie prvky. V ARESe reprezentuju tieto objekty data ulozene vo svete (pamati), v pripade POGAMUTa su to steny, teleporty, priepasti, strely a pod. Treba tiez vymedzit a implementovat take objekty do sveta, ktore prispievaju ku vymyslaniu sofistikovanejsich strategii. To zahrna steny a ich vlastnosti, napr. priehladnost, existencia predmetov na dobijani zdravia, streliva a pod. Predpoklada sa, ze taky svet bude mozne upravovat a vytvarat, aby sa algoritmu boli napisane "na telo" jednej mapy/pociatocnemu stavu sveta. Hrac bude mat mozost ovplyvnit/zmenit spravanie sveta.

Dynamika sveta Naprogramovani roboti budu mat moznost bojovat t.j. si ublizovat a vysledok utoku bude zanmy v okamihu ublizenia pre lahsie vyhodnotenie

programu. Roboti vo svete sa budu pohybovat vsetkymi smermi a interagovat s ostatnymi objektami vo svete (OK rychlost je uz vlastnost, to by som riesila osobitne JJ. Hrac by mal mat tiez volbu utoku robotov na blizko aj na dialku pre lepsie strategicke moznosti, inak by sa hra zvrtla na "najdi robota a kopni ho".

Zivotny cyklus robotov Zivot robotov bude zacinat vstupom do sveta a koncit opustenim sveta. Moznost nejakeho znovuzrodenia ako v hre POGAMUT sa nebude pripustat, ale bude otazkou dalsieho rozsirenia. Vitazny robot ostava zivy. Zivotny cyklus robota sa bude dat naprogramovat pomocou nejakeho programovacieho jazyka (, ktory bude dostatocne rozumitelny aj pre laika tatinka)

Vlastnosti robotov Ani jeden z uvedenych programov ale nema moznost specifikovat, ake budu jednotlive vlastnosti robotov. Ci robotov skoli jedna rana (POGAMUT) alebo tu je aj moznost nejakeho obmedzeneho znovuzrodenia (ARES). V bojovych hrach sa tiez ukazalo vhodne umoznit, aby si hrac pred samotnym vstupom do sveta mohol tieto vlastnosti upravit a tym ovplyvnil priebeh suboja.

Obrazok na (TODO) naznacuje smer, v ktorom sa bude praca uberat.

Kapitola 2

Analyza

Sucastou tejto kapitoly je zdovodnenie jednotlivych rozhodnuti, ktore sme navrhli. Chceme vytvorit svet, v ktorom mozeme pozorovat robotov, ako sa chovaju, pricom kriteriom je zostat nazive a hodnotiacou funkciou je maximalne predlzit cas zivota robota.

2.1 Svet

Neoddelitelnou sucastou hry je svet (prostredie), v ktorom sa bude suboj odohravat. Najskor vysvetlime, co vsetko svet obsahuje, ako sa s tym naraba z hladiska robota a z hladiska uzivatela (narabanie s objektom v programe) a jak to prispieva k uspesnosti algoritmu.
Uvazovany svet bol vybrany dvojrozmerny, pretoze poskytuje dostatok moznosti pre dianie na ploche (smer pohybu, zrozumitelne vykreslovanie stavu a pod.) a sucasne nie je obtiazne implementovatelny.

2.1.1 Sucasti sveta

Robot zije v prostredi a moze ovplyvnovat (utocit) na dalsie roboty, teda sam je objektom sveta. Kedze roboti sa smu pohybovat vsetkymi smermi, bez dalsich objektov by sa jednalo len o najdenie robota, ktory sa potom moze branit pohybom (nic ine by vo svete nebolo). Tento koncept je tiez zaujimavy, ale jednotvarny (rata sa stale s tym istym stavom sveta). Preto uvazujeme aj dalsie objekty.
Nakolko hrac aktivne nevstupuje do vykonavania algoritmu, je nutne popisat svet z hladiska robota. Pod robotovym videnim rozumieme sposob, akym moze ziskavat informacie o svete. Na ne potom moze reagovat. Preto je nutne definovat, ako sa robot orientuje vo svete. V ARESovi sa robot orientuje podla dat v pamati, cita, porovnava a prepisuje. V POGAMUTe reaguje robot na vizualne podnety, ktore vidi aj pozorovatel. Hrac ma naprogramovat algoritmus chovania robota a tak je prirodzenejsie pouzit podobny princip ako v POGAMUTe t.j. zariadit robotovi moznost ziskat informaci zo sveta v nejakom obmedzenom okoli. Obmedzene okolia znamena teda kruhovu vysec. Objekty mimo tejto kruhovej vysece uz nebude robot schony identifikovat.
Objekty uvazovane vo svete vzhladom na obsah sveta su nasledovne:

Robot ako objekt Zakladnou vlastnostou robota je, ze moze ublizovat ostatnym robotom. Ako moc robotovi tieto utoky uskodia. je vyjadrene celym cislom. Tak sa da skoda zistit presne a neobjavia sa problemy s malymi cislami alebo zlomkami, ako je to v pripade realnych cisel (v C je napriklad 0 vyjadrena ako male nenulove cislo). Cim vacsie cislo, tym vacsia skoda sa deje robotovi. V pripade vzdialeneho utoku je tiez dolezitou otazkou, ako daleko moze robot zautocit. Ak je toto cislo vopred dane, mal by o tom robot vediet dopredu, aby mohol svoj algoritmus prisposobit, Utok robotov prebieha na urovni ich tiel a nie programu (viz hra typu ARES). Dalsou otazkou je, kolko takychto zasahov robot vydrzi. Kedze utok je vyjadreny pomocou celych cisel, je vhodne vyjadrit celocislene aj zivotnost robotov.

Dalsou vhodnou vlastnostou je aktivna obrana oproti utokom. Doteraz sa robot mohol branit len dostatocnym poctom zivotov. Moze sa tak stat, ze pri malom pocte zivotov bude stacit jedna rana a robot zahynie. Preto je dalsia vlastnost, ktoru uzivatel moze u robota nastavit je, ake mnozstvo zranenia bude pohltene pred jeho smrtou. Vysledny efekt je ale rovnaky, akoby sa zivotnost zvysila a preto tato vlastnost nebola pouzita.

Strely

Roboti by mali vediet utocit na dialku. To mozno docielit tak, ze robot zautoci z dialky na konkretne miesto a tam okamzite vypocita vysledok utoku. V tom okamihu treba urcit kedy smie robot zautocit tymto sposobom. Ak moze robot zautocit na akekolvek miesto, potom ostatni roboti nielenze nemaju moznost sa utoku vyhnut, ale stracaju sa aj informacie o tom, odkial utok prisiel (na kazde policko moze byt zautocene). Strategie sa zredukuju na dva pristupy - nahodne utocenie zdialky na nejake policka a pohyb dovtedy, pokial sa nenajde ciel a na masivny utok na ciel. Minimalne je teda nutne obmedzit pravidlami, na ktore miesta sa moze utocit z dialky. Najviac intuitivne a lahko jzobrazitelne je vymedzit polomer zasahu. Problemom ale stale zostava nemoznost vyhnut sa utoku na dialku. Nie je tu mozne zaregistrovat utok a adekvatne nan zareagovat, popripade zareagovat na utocnika.

Dalsou moznostou je vytvorenie strely. Strela je objekt, ktoreho jedinou cinnostou je pohybovat sa predvidatelne vpred po dobu dopredu znameho casu (v danom smere vystrelu). Strela vsak skonci svoju cinnost aj v okamihu, ked spravi utok na blizko - zasiahne objekt. Robot teda utoci tym, ze na ciel vystreli a strela spravi potom utok nablizko. Tento pristup poskytuje vacsiu volnost pri utoceni, nakolko staci rozhodnut, v ktorom smere ma strela ist. Dalej je mozne priblizne odhadnut smer, odkial strela prisla. To je dolezite pri rozhodovani sa, kam ist, ci robota napadnut priamo (nablizko) alebo odpovedat strelbou. Strely tak boli pridane ako dalsie objekty, ktore zabezpecuju utok na dialku. To, ze steny poskytuju ukryt pre robota tiez znamena, ze steny obmedzuju vyhlad robota.

Steny Dalsim uvazovanym objektom su objekty, ktore moze robot vyuzivat k svojej obrane. Utocit na robota mozu take objekty len zblizka (robot zblizka

alebo zdialky, co je to vlastne strela zblizka. Vo svete potrebujeme nieco, co zabranuje pohybu. Tymito objektami budu steny. Robot ich moze vyuzivat ako strategicky ukryt podobne ako vojaci vyuzivaju terenne nezrovnalosti.

Nepristupne miesta Strategickym obmedzenim su miesta, na ktore robot za ziadnych okolnosti nesmie stupit. V pripade POGAMUTa su to priepasti, jamy, tekuta lava a pod.. V ARESe zase miesto v pamati, ktore obsahuje neplatnu instrukciu. Existencia nepripustnych miest umoznuje planovat strategiu z informacii o utoku na dialku.

Startovne pozicie robota

Ak roboti nemaju dopredu dane miesto v mape, potom sa ich startovne pozicie musia generovat bud nahodne alebo vypocitat, tak aby umiestnenie bolo nejakym sposobom spravodlive (nebol zvyhodneny ani jeden robot). Rozoberme si podrobnejsie jednotlive ne/vyhody sveta bez startovnych pozicii robotov. Nahodne vygenerovane pozicie:

Roboti vygenerovani vedla seba

Toto nam ale nevadi, pretoze tato situacia moze nastat kedykovek pocas behu programu a tak na nu robot musi vediet zareagovat.

Najdenie pociatocneho miesta

Moze zabrat pomerne vela casu v pripadoch, ked je mapa sveta zlozena z velkeho mnozstva stien a uzkeho priestoru pre pohyb robotov.

Pre zabezpecenie spravodlivosti generovania pozicii robotov bude pouzita heuristika. No je otazne co chapat pod spravodlivostou. Robot ma za ulohu reagovat na kazdu situaciu, nie je preto nutne uvazovat o specialne vypocitanych miestach a tak pojem spravodlivost straca svoj vyznam
Problemom zostava mnozstvo robotov v mape a dlhe generovanie pociatocnych pozicii. Z tohoto dovodu sa pristupilo k moznosi vytvorenia startovacich policok. To prinasa okrem ineho aj moznost definovat, pre maximalne kolko robotov je mapa idealna (rata sa s tymto maximalnym poctom robotov). Ak ale uzivatel zada viac robotov, simulacia sa aj potom moze uskutocnit. Nutne je len uzivatela upozornit, ze prebehol pokus umiestnit robota nahodne a ci bol tento pokus uspesny. Ak aj pokus nebol uspesny, nic vyznamne sa nedeje, pretoze existuje v mape sveta dostatocne mnozstvo robotov.

2.1.2 Zivot v prostredi

Roboti ziju vo svete a snazia sa znicit ostatnych. Kedze ale mapa moze byt rozlahla a roboti nemusia byt na dosah utoku, roboti sa na vyhodnejsie miesto musia dostat. Pohyb moze byt realizovay ako jednodychy presun z miesta A na miesto B (teleportacia), alebo postupny prechod na druhe miesto.
Idealne sa javi postupne vykonavanie pohybu (plynuly prechod na dalsie miesto), pretoze je to prirodzenejsie a takyto pohyb poskytoje priestor pre navrh strategii.

Ak sa robot ocitne pred priepastou, vie, ze robot za nim toute cetou nepride, pretoze by spadol. Podobne ak je za stenou, robot do nej pri pokuse ist za ni narazi a neobjavi sa za nim, takze je robot istym sposobom chraneny. Pri zdani pohybu sa tento pohyb zacne vykonavat po priamke, pretoze podpora algoritmov znamych ako path-finding by znamanalo obmedzenie uzovatela, co sice zvysuje kvalitnost algoritmu (uspesnost) ale sucasne znizuj naroky na napisanie programu pre robota.
Co sa samotneho pohybu tyka, bude zobrazovany tak, aby ho oko vnimalo ako plynuly $TODO - nejakareferencia$ Pohyb, ktory moze urobit robot, je pohyb v lubovolnom smere. Smer je urceny vektorom $[x, y], x, y \in N$, takze sa nim da vyjadrit presne smer pohybu. Konkretny pohyb je potom aproximovany useckou (objekt sa bude pohybovat po ucecke, pricom jeho suradnice budu vypocitane v zavislosti na case). Roboti sa teda mozu hybat vsetkymi smermi vyjadritelymi celymi cislami. Otazkou je, ako sa samotny pohyb tymto smerom realizuje. Robot vidi isty usek pred sebou a tak je rozumne mu tiez dovolit sa otacat a tym pokryt cele svoje okolie. Pohybujuci sa robot bude moct spravit nasledujuce veci. Bud sa bude hybat len smerom, ktorym je otoceny alebo sa bude moct hybat kamkolvek. Druhy pristup je o trochu lepsi lebo umoznuje vyuzit informaciu ktoru robot nazbieral cestou Prvy sposob si mozeme predstavit ako pohyb u kona - ten kde nevidi, nevleze. Druhy sposob sa da prirovnat k pohybu jelena - ten v pripade nudze uhyba efektne vsetkymi smermi. .
V suvislosti s pohybom, je mozne uvazovat o rozsireni sveta nad ramec popisu v ¡STATICKY SVET¿, a to konkretne o pohyblive steny. Prinos pohyblivych sten je konkretne v tom, ze robot potom bude moct zmenit prostredie sveta tak, aby odpovedal jeho konkretnemu algoritmu. Napriklad ak algoritmus pocita s tym, ze robot bude strielat na iste miesto a potom sa okamzite skryje za prekazku, musi naskor vmanevrovat nepriatelskeho robota do priestranstva, kde je takato prekazka, alebo si take priestranstvo vytvorit. Druhy sposob ma naviac oproti vyuzivaniu konkretneho prostredia sveta tu vyhodu, ze algoritmus musi pocitat so zmenenym svetom a tym sa zvysuje narocnost hry. Stale by vsak svet mal moznost mat statickeho prostredia (neposuvne) steny. V suvislosti s pohybom nastava tiez otazka, kedy a ako budu objekty navzajom interagovat. Kolizia nastane vtedy, ked sa obrazy objektov pretnu (maju spolocny neprazdny prienik). To kladie nemale naroky na strukturu sveta, ale sucase to ma tiez vedlajsi efekt. Cim vacsi obrazok bude symbolizovat objekt, tym vacsia je moznost kolizie. To moze byt trochu neprijemne, ale poskytuje to moznost pre dalsie rozsirenia, ked napr. silnejsirobot (viac zivota, vacsi utok) bude mat povinne vacsi aj zodpovedajuci obraz. Kolizia moze nastat prakticky pri akomkolvek malom pohybe. Na samotne ukladanie objektov do mapy existuje jednoduchy trik, rozdelit mapu na male policka a kazde policko obsadit prave jednym objektom. To prinasa asi vacsie naroky na pamat (obzvlast ak bude velka mapa a male policka), ale zato koliziu vieme urcit okamzite. Staci zistit, ci v danom policku, kde lezi vysledok pohybu, je objekt rozny od uvazovaneho. Tento sposob sa dost casto pouziva v bludiskach, kde su objekty rovnako velke. Okrem velkosti policka ma tento pristup ale problem aj s rozhodnutim, kde objekt patri. Ak sa v mape pohne len o niekolko pixelov, bude patrit stale do jedneho policka, pretoze mapa je rozdelena staticky (policka sa nepohybuju s objektom). Stale vsak moze kusok objektu presahovat nad ramec policka. Teda kolizia nemusi presne zodpovedat tomu, ako je zobrazena. Preto

boli uvazovne nasledujuce algoritmy pre detekciu kolizie, ktore pouzivaju iba velkost objektu:

Quadtree ¡priblizny popis , vyhody/nevyhody + referecia¿

Mriezkova metoda ¡priblizny popis , vyhody/nevyhody + referecia¿

Pre lepsiu implementovatelnost bola zvolena mriezkova metoda. Neboli zaznamena vyrazne rozdiely oproti quadtree.
Ked mame vyriesenu koliziu, je treba urcit, ako jednotlive objekty reaguju na koliziu. Rozhodli sme sa takto:

Robot vs. strela Strela ukonci svoj zivot vybuchom a ustedri robotovi nalezite zranenie. Robotovi sa v tomto okamihu prerusi akakolvek cinnost, ktoru predtym vyvijal (nariklad pohyb). To zodpoveda tomu, ze strela zasiahla hocijaky ciel (v tomto ponimani iba roboti) a teda dalej dovod pokracovat (ziadne viacnasobne zranenia) Robot si v tomto okamihu nemui robit starosti, ze strela sa odrazi priamo k nemu. Je vsak na dalsom rozsireni, ako sa strela po zasahu bude chovat.

Robot vs. Robot Roboti nemozu navzajom interagova inak ako ublizenim a pri kolizii sa prijavi utok nablizko. Tento sposob zlahcuje pisanie algoritmu, nakolko sa nemusi explicitne deklarovat utok nablizko (utok na dialku algoritmus musi explicitne vyjadrit, utok nablizko je automaticky). Utok nablizko bude prevedeny robotom, ktory sposobil koliziu.

Robot vs. stena Pokial robot narazi na stenu, ta ho zastavi. Toto chovanie je prirodzene, pretoze stena tohoto typu predstavuje staticky svet, ktory sa nemeni.

Robot vs. posuvna stena Posuvna stena ma schopnost menit svoj miesto. Robot ju ma moznost posunut (musi byt pri stene a pohybovat sa). Stena by sa mala hybat len s robotom, kedze cielom je, aby mu sustavne poskytovala ukryt. Stena sa tak pri kontakt s robotom posunie v smere, v akom ide robot. Ak sa snazia stenu posuvat obaja roboti a smer ich pohybu je vyjadreny dojrozmernym vektorom, potom sa stena pohybuje v smere vektorove suctu tychto dvoch smerom, co dava aj fyzikalne prijatelny zmysel.

Robot vs. prepadlisko Robot by na prepadlisko nemal stupat. Ak by sme sa obmedzili iba mna nestupnutie policka, staci nam stena,. Preto robot musi byt potrestany vstupom na toto policko. Ako najjednoduchsi sposob sa ponuka strata zivotov a nasledne zastavenie alebo prejdenie prepadliska za cenu niekolkonasobnej straty zivotov. Bol implmenetovabny druhy sposob. Pri prejdeni prepadliska aj za ceny toho, ze bude robot polomrtvy, sa pri dostatocnom mnoztve zivota moze este podielat na simulacii no algoritmus, ktory sa bude stale pokusit prestupit prepadlisko isto zahynie.

Robot vs. strela Strela pri kontakte s robotom zautoci svojim utokom na blizko. Nasledne sa robotovi na zaklade tohoto utoku znizi zivotnost. Strela nasledne zmizne z hracieho pola, pretoze zasiahla ciel. Strela moze zasiahnut aj toho

kto ju vystrelil. Je to rozumne z toho dodu, aby robot nemal tendenciu strielat vsade, ale rozmyslal si, ci to neublizi aj jemu. Strela by preto mala mat vacsiu rychlost ako robot. Inak s moze stat, ze po vystreleni spravnym smerom sa robot tym smerom pohne (aby mohol sledovat obet) a zasiahne ho vlastna strela. Potom ale nema zmysel pouzivat strelu na dialku, pretoze rychlejsie by bolo mozne pouzit utok na blizko. Strelu by sa dalo pouzit iba v pripade prepadliska. Ak strela bude ale dostatocne pomala, robot sa jej lahko vyhne a tym sa opat straca vyznam utoku na dialku.

Stena vs. strela Strela sa bude moct od obycajnej steny odrazat. Stena nie je niekdy primarnym cielom. Preto nema zmysel, aby strela ukoncila svoju drahu pri narazeni na stenu. Jediny dosledok by bol, ze robot moze strelu vidiet a tym by bola prezradena pozicia strelca. Napriek tomu, ak strela ukonci svoku drahu len v okamihu, ked zasiahne robota, bude mozne pomocou vlastnosti strely ako dostrel cistit cestu aj za "rohom". Mozne je i umyselne miest protivnika vyslanim strely tak, aby sa odrazila. Preto bolo rozhodnute o odrazani strely od steny.

Stena vs. posuvna stena Robota normalna stena zastavuje a posuvna stena sa hybe robotovym pricinenim a poskytuje mu kryt. Je preto logicke, aby normalna stena zabranovala pohybu aj posuvnej stene. Posuva stena sa teda na rozdiel od normalnej steny strely neodraza, ale zastavuje.

Strela vs. posuvna stena Otazkou je, ci by mala posuvna stena sa hybat aj pri kontakt so strelou. Posuvna stena je vsak primarne urcena pre ukryt robota a jedine, co by mohla strela robit je robotovi zase ukryt zobrat ked vystreli strely. Ak vsak by tento ukryt mal zmiznut po vystreleni strely, smer strely by musel so smerom tejto posuvnej steny zvietar tupy uhol. ¡TODO OBR.¿ To ale znamena, ze strela by sa aj tak odrazila, pretoze, posuvna stena je len druh steny. Vysdedok by v najhorsom pripade nemusel byt okamzity - robot by steny stale posuval tym smerom a tak ciastocne anuloval vysledky strelby. Vysledom by tam bol prinajmensom neisty a tazko vyuzivatelny. Preto sa strela od posuvnej steny iba odraza, namiesto toho, aby ju aj posuvala.

2.1.3 Vytvaranie map

Sucastou prace bolo aj editacia a generovanie map nakolko cielom je odskusanie algortimu v roznych svetoch. Generovanie mapy pozostava z urcenia prvkov, ktore sa budu generovat, urcenie velkosti a samotnym algoritmom na umiestnenie tychto objektov. Uzivatel bude mat moznost upravit vygenerovanu mapu ak zodpoveda jeho predstavam.
Za generovatelne objekty boli vybrane iba mapy. Roboti a strely sa nemozu generovat, pretoze robota zastupuje startovne policko a strely sa nepotuluju svetom, pokial robot nezautoci. A tak zostali startovne policka, obycajne a posuvne steny a prepadliska. Obycajne steny su zakladnym prvkom, s ktorymi robot pocita, takze tie sa generuju.

Prepadliska su obtiazne generovatelna z toho dovodu, ze nahodne generovane prepadliska sa mozu stat uzkym hrdlom nejakeho koridoru. To znamena, ze vzniknu dve oddelene casti, kam robot moze len so znacnout stratou na zivotoch. Je mozne tieto situacie nechat osetrit uzivatela, ale to by musel prejst celu mapau a hladat taketo miesta. To ale nie je uzivatelsky prijemne. Mozne pouzit heuristiku, ktora tieto miesta najde a odstrani. Taka heuristika nie je narocna na implementovanie - stacila ibu kontrola uzkych hrdiel, moznych prepojeni s ostatnumi miestami,
Posuvne steny neboli zahrtnute do genrovania. Ich generovania totiz obsahuje aj to, ci su priesvitne a pohyblive a ci nahodou nevytvori neriesitelny svet. Preto genrovanie posuvnych stien je prenechame uzivatelovi.
Samotny algoritmus generovania vychadza z napadu - nie zaplnit miesto objektami, ale miesto plne objektov postupne vyprazdnovat. Tymto sposobom sa nageneruju steny. Dalsi generovatelny objekt - prepadliska sa potom prigeneruju dodatocne prejdenim vygenerovanej kostry sveta a testovanim, ci existencia prepadliska neodporuje vyssie uvedenym kriteriam.
¡popis pazraveho algoritmu¿ Tymto sposobom vygenerovana mapa obsahuje dostatocne velky priestor pre manevrovanie robota a sucasne sa dostatocne velka krat podari vygenerovat miesta, ktore robot pouzije ako kryt. Pre nase ucely testovania algoritmov to postacuje.