VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

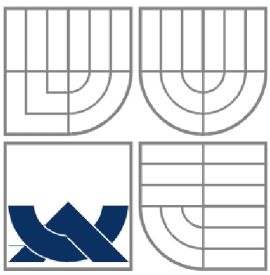
IMPLEMENTACE PRAVIDEL A STRATEGIÍ SOUTĚŽE REDBOT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

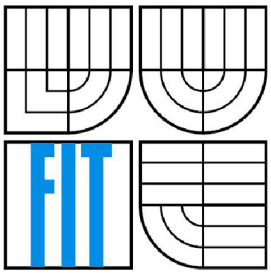
AUTOR PRÁCE
AUTHOR

MARTIN ORAVA

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE PRAVIDEL A STRATEGIÍ SOUTĚŽE REDBOT

RULES AND STRATEGIES IMPLEMENTATION OF REDBOT COMPETITION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Martin Orava

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Šárka Květoňová Ph.D

BRNO 2013

Abstrakt

Tato práce se zabývá vytvořením zadání programovací soutěže RedBot. Jedná se o soutěž pro studentské týmy, pořádanou společností RedHat. Práce popisuje postup tvorby zadání soutěže a implementaci vyhodnocovacího serveru, který bude sloužit k vyhodnocení soutěže. Tato práce se dále zabývá implementací vizualizační utility k zobrazení průběhu hry a implementací obslužných rutin, které mohou soutěžící využít pro rychlejší tvorbu strategií.

Abstract

The bachelor thesis deals with creating rules of RedBot programming competition. This competition is organized by RedHat company and is designed for students' teams. The objective of thesis is to describe process of making competition rules and implementation of evaluation server for assessing results of the competition. The other objective is to implement visualization utility for view of the game and implementation of basic routine which is useful for faster strategy creation.

Klíčová slova

RedBot, soutěž, pravidla, hra, strategie, vyhodnocení

Keywords

RedBot, competition, rule, game, strategy, evaluation

Citace

Martin Orava: Implementace pravidel a strategií soutěže RedBot, Brno, FIT VUT v Brně, 2013

Implementace pravidel a strategií soutěže RedBot

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Mgr. Pavla Moravce, Ph.D a Ing. Šárky Květoňové, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Orava
15.5.2013

Poděkování

Chtěl bych poděkovat Mgr. Pavlu Moravci, Ph.D. ze společnosti RedHat za neustálou podporu při tvorbě práce a za užitečné podměty a rady, které mi poskytl.
Dále bych chtěl poděkovat Ing. Šárce Květoňové, Ph.D. za to, že se ujala vedení této bakalářské práce a umožnila mi tak vytvořit dílo, které má reálné využití.

Obsah

1 Úvod

RedBot je soutěž studentských týmů v programování virtuálních robotů. Tuto soutěž organizuje společnost RedHat v pravidelných půlročních intervalech. Koncem dubna 2013 byl ukončen třetí ročník této soutěže. Soutěž RedBot navázala na soutěž FIbot, která byla dříve organizována na Fakultě informatiky Masarykovy univerzity pro její studenty.

Úkolem soutěžících je implementace strategie pro zadanou logickou hru. Každý ročník soutěže přichází s novou, originálně vymyšlenou logickou hrou, která se zaměřuje na jiné způsoby tvorby strategií.

V průběhu soutěže probíhá pokusné odevzdání, které soutěžícímu umožní ladit strategie na základě přátelských zápasů s jinými strategiemi. Po ukončení soutěže probíhá její vyhodnocení podle vzájemného souboje vytvořených strategií ve hraní zadané logické hry. *[1]*

Podobnou programátorskou soutěží světového rozsahu je soutěž Rock Paper Scissors Programming Competition. Cílem soutěžících je napsat program vítězící ve známé hře kámen, nůžky, papír. Soutěžící při tvorbě svých programů vytváří algoritmus umělé inteligence, který se učí a reaguje na rozhodovací strategii soupeře. Algoritmy vyhrávající soutěž jsou algoritmy nenáhodné, které jsou schopny predikovat soupeřovo jednání. Náhodné algoritmy mají 50% úspěšnost při hře a umístí se vždy uprostřed pořadového žebříčku. *[5]*

Další známou programovací soutěží je soutěž Battlecode, která kombinuje strategii boje, softwarové inženýrství a umělou inteligenci. Cílem soutěžích je napsat nejlepší program, který hraje počítačovou hru Battlecode. Jedná se real-time strategickou hru, ve které spolu bojují dva týmy robotů. Roboti k boji využívají různé typy zbraní a rádio, kterým spolu roboti jednoho týmu komunikují, aby mohli útočit koordinovaně. Každý robot je ovládán samostatně programem, který mu soutěžící vytvoří. Tento program je nahrán do všech robotů jednoho týmu.

Před finále soutěže mají soutěžící možnost ladit své strategie pomocí domluveného souboje s jiným soutěžícím. Při finále probíhají napínavé živé komentované zápasy před publikem. *[6]*

Podobnou soutěží ke hře Battlecode je soutěž AI Challenge. Jedná se o soutěž pořádanou společností Computer Science Club na University of Waterloo. Tuto soutěž podporuje společnost Google propagací a sponzorováním.

V době psaní této práce probíhá soutěž ve hře Mravenci. Jedná se o strategickou hru pro více hráčů, ve které každý hráč ovládá své mraveniště. Cílem hry je najít a zničit nepřátelské mraveniště a zároveň ochránit své mraveniště před zničením. Ve hře hráči hledají jídlo, které jim umožní plodit další mravence.

Pro vývoj strategií mají soutěžící k dispozici startovací balíčky, které urychlují vývoj strategie. Vytvořenou strategii hráči nahrají na server, na kterém se tvoří průběžný pořadový žebříček soubojem nahraných strategií proti sobě, po celou dobu konání soutěže. *[7]*

Programátorské soutěže jsou velmi populární, protože umožňují rozvoj kreativního a logického myšlení zapojených soutěžících a jejich schopností v programování. Pořádající společnosti získají díky těmto soutěžím kontakty na aktivní potenciální zaměstnance. Tyto soutěže také slouží k rozvoji algoritmů určených k tvorbě umělé inteligence.

2 Cíl práce

Cílem této bakalářské práce je vytvoření **zadání soutěže** pro některý z příštích ročníků soutěže RedBot. Součástí práce je také tvorba **vyhodnocovacího serveru**, který spustí souboj jednotlivých hráčských strategií proti sobě. Tímto způsobem dojde k vyhodnocení nejlepší strategie.

Programovací jazyk pro tvorbu soutěžních strategií je libovolný. Pro nejrozšířenější programovací jazyky, použité v předchozích ročnících soutěže, jsou vytvořeny **obslužné rutiny** hry. Obslužné rutiny slouží ke zpracování vstupních dat, aby se soutěžící mohli věnovat pouze implementaci samotné strategie.

Pro usnadnění vývoje strategií je soutěžícím dodána **vizualizační utilita**, která zobrazuje průběh hry.

2.1 Požadavky na zadání soutěže

Zadání soutěže musí být jednoduché, snadno pochopitelné a jednoznačné. Musí být snadné vymyslet a vytvořit primitivní strategii, a také by měl být neomezený prostor tuto strategii zdokonalovat. Hra musí využívat interakce mezi hráči a nesmí existovat jediná výherní strategie. Dále je třeba, aby hra byla dostatečně složitá, aby nebylo možné prohledat celý stavový prostor hry. Posledním požadavkem je možnost vizualizovat průběh hry.

2.2 Implementační požadavky

2.2.1 Vyhodnocovací server

Úkolem vyhodnocovacího serveru je provést zápas dodaných soutěžních strategií a vyhodnocení, která z nich je nejlepší. Porovnání strategií bude probíhat na základě výsledků hry, kterou hrály strategie proti sobě.

Vyhodnocovací sever bude v každém kole hry spouštět soutěžní strategie, předávat jim informace potřebné k vyhodnocení tahu, a získávat od nich odpovědi. Tyto odpovědi zpracuje a vypočítá změny na hrací desce. Změny budou promítnuty do informací předávaných hráčským strategiím při dalším spuštění. Tyto informace musí být uloženy pro pozdější přehrání hry ve vizualizační utilitě.

Vyhodnocovací server také musí dbát na dodržování pravidel hry a nesmí hráčům povolit provést zakázaný tah.

Dalším úkolem vyhodnocovacího serveru bude ukončit hru a udělit body jednotlivým strategiím.

2.2.2 Vizualizační utilita

Vizualizační utilita bude sloužit k prohlížení zápasů zaznamenaných vyhodnocovacím serverem. Hráči ji budou moci využívat pro ladění svých strategií. Vizualizační utilita bude také sloužit k vytvoření videozáznamu zápasu, který bude dostupný spolu s výsledky na stránce soutěže.

2.2.3 Obslužné rutiny

Obslužné rutiny jsou kostrou programu hráčských strategií. Jejich úkolem je komunikace s vyhodnocovacím serverem. Není nutné aby je hráči využili, ale jejich použití jim značně ulehčí vývoj strategií, a tak se mohou věnovat pouze samotné logice strategie. Obslužné rutiny budou vytvořeny pro nejpoužívanější programovací jazyky z minulých ročníků soutěže: jazyk Python a C. Obslužná rutina jazyka C bude využitelná i pro strategie využívající jazyk C++.

3 Analýza a návrh

3.1 Zadání hry

Při tvorbě zadání hry jsem se inspiroval u předchozích ročníků soutěže, aby mělo přiměřenou obtížnost. Níže jsou uvedeny popisy zadání předchozích ročníků soutěže, a to Soupeřící červi a Války království. *Popisy zadání byly převzaty z [1].*

3.1.1 Soupeřící červi

Cílem hry je v libovolném programovacím jazyce implementovat strategii pro robota/červa. Červ se pohybuje na hracím plánu spolu se 3 soupeřícími červy. Na hracím plánu jsou rozmístěny zdi a bonusové předměty (kytičky, krystaly ledu a bonusy). Pokud červ vstoupí na pole s kytičkou, prodlouží se a získá body. Led pak znamená zmražení ostatních červů na několik následujících kol. Bonus zvyšuje efekt kytičky či ledu. Více bonusů za sebou má násobící efekt. Pokud červ vstoupí na políčko, kde je zeď anebo jiný červ, umírá. Vítězí červ, který získal po daném počtu kol nejvíc bodů.

3.1.2 Války království

V této hře je proti sobě postaveno několik království, která soupeří o území. Budují si armádu a dobývají protivníkova území. Najímají rolníky na produkci jídla, zlepšují si efektivitu svých akcí a můžou provádět speciální akce. Hraje se na předem daný počet kol a na konci hry vítězí království s největším územím.

Základní parametry každého království:

- území,
- jídlo,
- vojáci,
- úroveň zbrojení,
- rolníci,
- úroveň farmaření.

Království může provést každé kolo právě jednu z těchto akcí:

- verbovat vojáka,
- vyučit rolníka,
- vylepšit úroveň zbrojení,
- vylepšit úroveň farmaření,
- sklízet úrodu,
- útočit na vybrané království,
- vylepšit úroveň tajných služeb,
- získat informace o ostatních královstvích, která mají nižší úroveň tajných služeb,
- loupit svitky se znalostmi daného království.

Pořadí vyhodnocení akcí:

- naverbují se vojáci, vyučí rolníci, zlepší se úrovn vlastností a provede se sklizeň,
- vyhodnotí se útoky,
- provede se loupež svitků,
- provede se akce zjisti informace.

Automaticky se na konci kola odečtou zásoby jídla nutné na provoz království (každý rolník a voják spotřebuje za kolo 1 jednotku jídla). Pokud království dojdou zásoby, obyvatelé, které nebylo možné nakrmit, umírají hlady. Nejdříve vojáci, pak rolníci.

3.1.3 Tvorba vlastního zadání hry

Při tvorbě vlastního zadání hry jsem se inspiroval v hlavním principu deskové hry Quoridor. Je to hra pro 2-4 hráče, jejímž cílem je projít s figurkou na opačnou stranu hrací desky. Ve hře se taktizuje pokládáním kamenů zdí do cesty soupeřovým figurkám. Položení kamene zabere tah, kterým je možné posunout vlastní figurku blíže cíli. Hráč nesmí položením kamene nesmí zablokovat poslední možnou cestu do cíle.

Další inspiraci jsem spatřil v deskové hře RoboRally. V této hře ovládá každý hráč jednu figurku - robota, u níž je dán směr, kterým je otočena. Hrací plán je síť čtvercových polí, na nichž jsou rozmístěny praporky s čísly 1 až n. Cílem hry je projít svým robotem všechna pole s praporky v daném pořadí. Hráč, který to dokáže jako první, vyhrává.

Na rozdíl od mnoha jiných her se hráči nestřídají jeden po druhém, ale hrají více méně současně. Hra se dělí na kola a tato kola na tahy neboli takty. Na začátku každého kola hráči sestavují program pro svého robota na pět kroků dopředu. V jednotlivých tazích se potom tyto kroky vykonávají, vždy jedna instrukce v jednom taktu, ovšem se všemi roboty najednou.

Smysl hry je v tom, že při plánování pohybu robota na 5 kol dopředu, je těžké počítat se všemi vlivy, které mohou jeho pohyb ovlivnit. Robot svým pohybem posunuje ostatní roboty, kteří mu stojí v cestě. Dále existují speciální pole, která robota otáčí a pohybují sním. *Volně převzato z [8].*

Tvořené zadání soutěže počítá s hrací deskou o rozměrech 25x25 polí. Každý hráč bude ovládat dva roboty. Jeho úkolem je dorazit s jedním z nich na cílové pole na na druhou stranu hrací desky dříve než soupeři. Pohyb robota půjde ovlivnit jen třemi typy příkazů: pohyb vpřed, otočka vpravo, otočka vlevo. Podobně jako ve hře Quoridor lze taktizovat pokládáním zdí soupeřům do cesty. Oproti hře Quoridor, kde je pokládání zdí vyřešeno zablokováním dvou hran pole, položení zdi zablokuje přístup na určené pole. Toto řešení bylo zvoleno pro zjednodušení algoritmizace úlohy.

Dobrá kombinace položených zdí by umožnila zablokovat hráče tak, že neměl možnost přístupu do cíle. Aby tato situace nevznikla, bylo potřeba přidat jedno z těchto pravidel:

a) Zákaz stavění zdi na pozici, která zablokuje poslední možnou cestu robota do cíle.
b) Umožnění střelby, kterou si roboti požadovanou cestu prostřílí.

Řešení a) by přineslo, že vítězství ve hře plně závisí na úspěšné taktice pokládání zdí a dobrého hledání nejkratší cesty. Zatímco řešení b) umožňuje vyhrát více různými způsoby. Jedná se o vhodnou kombinaci pokládání zdí, jejich obcházení a prostřílení cesty. Střelba může hru ozvláštnit o snahu zasáhnout soupeřovy roboty. Proto bylo zvoleno právě toto řešení.

Střelba

Aby střelba měla větší uplatnění při souboji robotů, byla doplněna o zranění všech objektů (robot nebo zeď) v osmi okolí zasaženého objektu. Definice osmi okolí lze vidět na obrázku 1. Toto pravidlo sebou přineslo možnost zastavění robota na jednom poli s tím, že při pokusu o prostřílení ven, se robot sám zničí rozptylem střelby. Hráč, který by tuto strategii využíval, by vyhrál poměrně snadno. Proto bylo potřeba vymyslet nové pravidlo, které tuto situaci vyřeší. Nabízela 3 různá řešení:

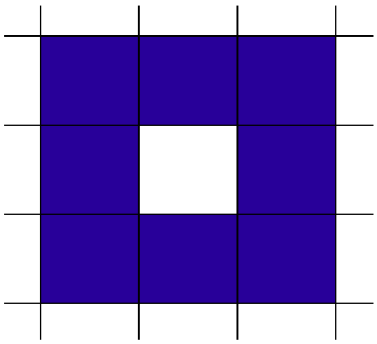
a) Robot zraní objekty, které mu brání v provedení akce pohyb vpřed.
b) Zákaz stavění zdi do osmi okolí robota.
c) Robot střelbou nezraní sám sebe.

Řešení a) nabízí mimo jiné vyřešení situace, kdy dva hráči neustále chtějí vstoupit robotem na stejné pole. Řešení naopak ale vede k podporování příliš jednoduchých strategií, kdy robot čeká před zdí, než bude zbořena.

Řešení b) naproti tomu by mělo zpestřit vizuální pojem hry, protože více motivuje hráče hledat nové cesty a vybízí je k větší kreativitě i při stavění zdí.

Řešení c) by hru ochudilo o složitou úvahu, kdy se ještě hráči vyplatí střelbou zranit soupeře i sebe, která se u propracovanějších strategií může vyskytnout.

Po této úvaze byla zvolena možnost varianty b). Varianta a) však nebyla za vrhnuta ale byla využita k řešení zablokovaných robotů dvou hráčů pohybem na stejné místo.



Obrázek 1 - Osmi okolí pole

Dále bylo potřeba zajistit, aby se stavbou zdí nezaplnil hrací plán natolik, že by se hra stala nehratelnou. Proto bylo rozhodnuto, že půjde zastavit pouze 1/8 hrací desky. K tomuto poměru se přišlo při porovnávání vizualizací zastavěných částí.

Další pravidlo vzniklo při simulaci hry. Jedná se o zákaz stavění zdi na místo, kde již dříve stála zeď a byla zbořena. Cílem tohoto pravidla je zvýšit dynamiku hry, tím že se zabrání opakování stejných akcí. Příkladem může být situace, kdy je robot obestavěn zdmi a ven se musí prostřílet během 10 kol. Po těchto 10 kolech by jej soupeř mohl znovu uvěznit stavbou zdi na původní místo.

Pořadí vyhodnocení akcí:

- Pohybová akce
- Střelba
- Stavění

Úmyslně byla první zvolena pohybová akce. To hru zpestřuje o to, že roboti po sobě střílí na základě předpokládané pohybové akce ostatních hráčů, a ne na základě známé pozice robota.

Provedení střelby před stavěcími akcemi naopak podporuje jednoduší zasáhnutí robota, protože hráč by měl možnost při vstupu na ohrožované pole se chránit stavbou zdi do cesty mezi ohrožujícího robota a pole, na které chce vstoupit.

Volba počtu životů a účinnosti střelby

Pro navržená pravidla je potřeba správně rozvrhnout počty životů robotů, zdí a o kolik životů přijde zasažený objekt střelbou. Hra by měla být vyrovnaná a nabízet různé možnosti řešení situace, přičemž žádné z nich není zcela správné nebo chybné.

Existují tři způsoby zranění objektu. Přímou střelbou, zraněním v osmi okolí od zasaženého objektu a zranění objektu pohybem robota, pokud mu objekt brání vstupu na pole. Tyto způsoby zranění byly seřazeny dle obtížnosti jejich dosažení a body jim byli přiděleny v exponenciální závislosti.

- Zranění pohybem: 1 život.
- Zranění nepřímou střelbou: 2 životy.
- Zranění přímou střelbou: 4 životy.

Od těchto hodnot se odvíjí určení životů zdí a robotů. Počet životů zdi je závislý na tom, že obejití zdi trvá robotu 8 kol, což by mělo být výhodnější řešení, než zůstat stát na místě a zdí se prostřílet. Proto bylo rozhodnuto, že k prostřílení skrz zeď přímou střelbou bude trvat 10 kol + 2 kola na přejití zničené překážky. Počet životů zdi byl z toho důvodu určen na 40. Tento počet životů zároveň vylučuje praktické využívání zničení zdi pohybem, které bylo vyhodnoceno jako příliš snadné řešení.

Počet životů robota byl určen tak, aby 3 přímé zásahy robota nevyřadily ze hry, ale jakékoliv další zranění jej už vyřadilo. Z tohoto důvodu byl počet životů robotů určen na 13. Toto číslo je dostatečně malé, aby zranění pohybem mělo na roboty citlivý dopad a po několika opakování stejné kolize umožnilo pokračování v pohybu zdravějšímu robotu.

Bodování

Závěrečné bodování se skládá ze tří částí: body za zbylé životy robotů, body za zásah nepřátelských robotů a body za příchod do cíle.

Snažil jsem se docílit takového poměru mezi střelbou a příchodem do cíle, aby specificky zaměřené strategie na jednu z těchto oblastí neměly výhodu.
Body za střelbu byly dány do rovnováhy s body za zbylé životy. Za každý život robota na konci hry hráč získá 1 bod. Za každý zásah robota přímou střelbou získá střílející hráč 4 body a za každý zásah robota nepřímou střelbou získá střílející hráč 2 body. Tyto body jsou přiděleny i za zásah vlastního robota.

Počet bodů za příchod do cíle je určen pořadím robotů v cíli, rozdíl mezi jednotlivými pořadími je 10 bodů. Jedná se o počet udělených bodů za dva přímé zásahy a jeden nepřímý, kterým se bodově vyrovná. Také to značí to, že po jednom přímém zásahu, vzniká mezi hráči bodový rozdíl 8 bodů, který je menší než body udělené za dřívější příchod do cíle.

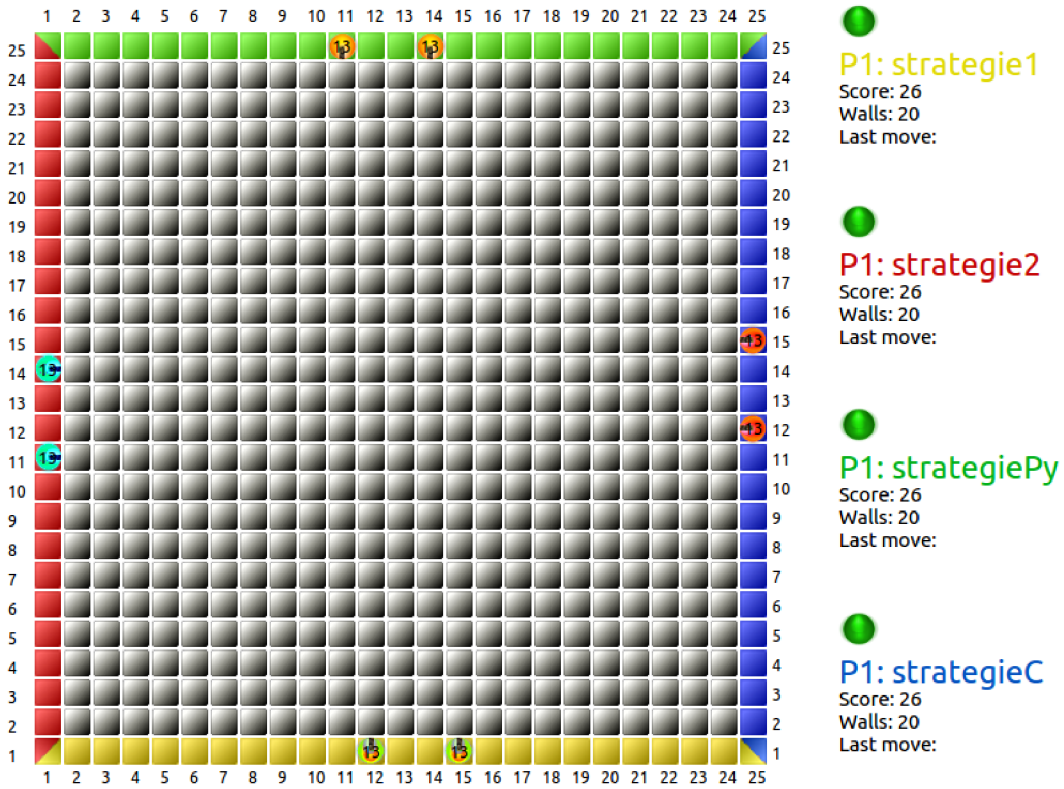
4 Návrh řešení

4.1 Zadání soutěže

Pravidla hry

Hra je určena pro čtyři hráče. Hraje se na šachovnici o rozměrech 25x25 polí. Každý hráč má k dispozici 2 roboty a v průběhu hry může postavit 20 zdí. Počáteční rozestavení lze vidět na obrázku 2.

Cílem hry je přivést co nejrychleji jednoho ze svých robotů do cíle, který tvoří celá protější strana od počáteční pozice hráče na hrací desce. Druhým ne méně důležitým cílem hry je střílet na roboty soupeřů, protože za úspěšné zásahy se získávají bonusové body. Robot má 13 životů a zeď 40. Hra se hraje na předem určený počet kol, nebo dokud všichni hráči neukončí hru příchodem do cíle, nebo ztrátou obou svých robotů.



Obrázek 2 - Počáteční rozestavení

Průběh hry

V každém kole hráč získá informace o dění na celé hrací desce a určí každému svému živému robotu pohybovou a speciální akci. Tento příkaz strategie musí podat do 1 sekundy běhu strategie.

Pohybové akce:

- Pohyb vpřed – posune robota o jedno pole vpřed. (Ve směru jeho pohledu) Tato akce se provede tehdy, pokud robot vstupuje na pole neobsazené. Pokud je pole obsazené zdí, nebo jiným robotem, nebo ve stejném kole žádá jiný robot o přístup na stejné pole, je tento blokující objekt zraněn o 1 život a hráčův robot se vpřed neposune. Podle stejného principu je zraněn o 1 život pohybující se robot A, na jehož původní pozici se snaží vstoupit robot B. Robot B zůstává na své původní pozici.
- Otočka vlevo – otočí pohled robota o 90° vlevo (v protisměru hodinových ručiček).
- Otočka vpravo – otočí pohled robota o 90° vpravo (ve směru hodinových ručiček) .
- Žádný pohyb – robot nezmění svou pozici ani směr.

Speciální akce:

- Střelba – Robot vystřelí střelu letící z jeho pozice ve směru jeho pohledu. Střela naráží na nejbližší objekt (robot nebo zeď) a vybuchuje. Zasažený objekt je zraněn za 4 životy. Všechny ostatní objekty v osmi okolí zasaženého objektu (obr. 1) jsou zraněny za 2 životy. Za každý zásah soupeřova nebo svého robota získá střílející hráč do závěrečného hodnocení body. Za zásah robota přímou střelbou získá hráč 4 body a za každý zásah robota nepřímou střelbou získá hráč 2 body. Pokud střela při své cestě k okraji hrací desky nenarazí na žádný objekt, byla vystřelena zbytečně a nic se neděje.
- Stavba zdi – Hráč může postavit zeď na libovolnou pozici na hrací desce s výjimkou:
  - pole, na kterém stojí některý z živých robotů a v jeho osmi okolí,
  - pole, na kterém již stojí zeď nebo stála zde zeď, která je již zbořená.
- Žádná akce – Robot se vzdává své speciální akce.

Vyhodnocení kola

Nejdříve se provedou pohybové akce všech robotů naráz. Pokud dva roboti žádají o přístup na stejné pole, jsou oba dva zraněni o 1 život (viz pohyb vpřed).

Poté se vyhodnotí střelba. Při vyhodnocování střelby jsou roboti již přesunuti na nové pozice a otočeni dle své pohybové akce. Nejdříve se provede střelba všech robotů

a pak se teprve zjišťuje, který robot nebo zeď byli zničeni a je třeba je odstranit z hrací desky. Body za zásah se přidělují vždy v plném rozsahu střelby.

Poslední část je vyhodnocení staveb zdí. Kontroluje se, zda zeď na daném místě může být postavena, a zda hráč nepřekročil svůj stavební limit. Odpočet zdi se provede, pouze pokud stavba zdi proběhla úspěšně. Když v jednom kole staví dva nebo více hráčů zeď na stejné pole, zeď je zde postavena jen jedna, ale odpočítána je všem hráčům, kteří zde chtěli stavět.

Konec hry

Hra končí po maximálním počtu kol, nebo pokud všichni hráči již dorazili do cíle nebo ztratili oba dva své roboty.
Každému hráči jsou vypočítány body, které za hru získal:

1. Za každý život, který zbyl jeho robotům 1 bod.
2. Za střelbu. Za každý přímý zásah robota (i svého) 4 body, za nepřímý zásah 2 body.
3. Za pořadí příchodu do cíle. Pokud žádný hráčův robot do cíle nepřišel, nezískává žádný bod. Pokud více hráčů přišlo ve stejném kole, získávají z možných ohodnocení to vyšší. Hráč, který dorazí do cíle po nich, získává následující bodové ohodnocení:
    1. 40 bodů,
    2. 30 bodů,
    3. 20 bodů,
    4. 10 bodů.

4.2 Vyhodnocovací server

Programovací jazyk vyhodnocovacího serveru byl dán v zadání práce. Vyhodnocovací server bude napsaný v jazyce C++. Tento jazyk je vhodný pro možnost objektově orientovaného návrhu, a protože se jedná kompilovaný jazyk, umožňuje rychlejší běh programu.

Vyhodnocovací server bude ke svému běhu potřebovat vstupní parametry: cestu ke čtyřem strategiím, složku, kam ukládat výstupy z jednotlivých kol, a číslo, které udává maximální počet kol, po kterém hra ukončí.

Po zpracování parametrů a inicializaci vnitřních struktur bude vyhodnocovací server ve smyčce provádět:

1. Generování souboru s popisem kola.
2. Spuštění soutěžních strategií a převzetí odpovědí.
3. Vyhodnocení pohybových akcí.
4. Vyhodnocení střelby.
5. Vyhodnocení stavby zdí.

Tato smyčka bude ukončena po maximálním počtu kol, nebo pokud již ve hře nebude žádný z hráčů.

Generování souboru s popisem kola

Do složky zadané na vstupu programu bude vytvořen soubor, který bude obsahovat všechny všechny informace o aktuálním dění na hrací desce a hráčích. Struktura tohoto souboru bude podrobněji popsaná v kapitole Návrh komunikačního protokolu.

Spuštění hráčských strategií a převzetí odpovědí

Odkaz na hráčské strategie má vyhodnocovací server k dispozici z argumentů při spouštění. Tyto strategie potřebují ke svému běhu odkaz na soubor s aktuálním popisem kola a určení jeho pozice na hrací desce. Po spuštění strategií je třeba počkat na odpověď. Pokud strategie neodpoví v rozumném časovém intervalu, hráčovi roboti dané kolo neprovedou žádnou akci. Tento časový interval byl zvolen na 4 sekundy, po které jsou všechny čtyři strategie puštěny paralelně.

Vyhodnocení pohybových akcí

Akce otočky robota doleva a doprava se provede vždy. Akci pohyb vpřed je třeba kontrolovat, zda je možné jej provést. Akce pohyb vpřed se neprovede, pokud na místě, kam by se měl robot posunout, stojí zeď, jiný robot nebo jiný robot žádá o pohyb na stejné místo. V závěru této akce je třeba zkontrolovat, zda některý z robotů nedorazil do cíle.

Vyhodnocení střelby

Vyhodnocení střelby probíhá pro každého živého robota, který má nastavenou akci střelba. Je třeba najít nejbližší živý objekt (robot nebo zeď), který leží rovně ve směru pohledu robota. Od zasaženého objektu bude provedena nepřímá střelba, která zraní všechny objekty v 8 okolí pole. Zasaženým objektům je třeba odečíst životy, a hráči, který střelu provedl, je třeba přičíst body za střelu, která zasáhla robota.

Po proběhnutí všech střel, je třeba ověřit všechny zdi a roboty. Ty roboty a zdi, kteří mají méně než 1 život, označit za zničené.

Vyhodnocení stavby zdi

Zde se kontroluje, zda stavěnou zeď je možno na vybraném místě postavit. Nesmí se jednat o pozici, kde již stojí nebo stála zeď. Dále také se nesmí jednat o pozici nebo 8 okolí pozice, kde stojí některý z živých robotů.

Dále zde také bude probíhat kontrola, zda více hráčů nestaví na stejném místě. V tom případě bude zeď odečtena všem hráčům, kteří zde v tomto kole chtějí stavět.

4.3 Návrh komunikačního protokolu

Při spouštění hráčských strategií je třeba jim dodat informace, které popisují aktuální dění na hrací desce, a identifikátor hráče. Protože informací popisujících dění na hrací desce je hodně, budou předány prostřednictvím odkazu na soubor. Tento soubor bude mít využití i pro vizualizaci hry ve vizualizační utilitě.

V tomto souboru je třeba uvést informace: číslo aktuálního kola, maximální počet kol hry, rozměry hrací desky a umístění zdí, a jejich životy. Dále budou popsány informace o každém hráči, zda je stále ve hře, v kterém kole dorazil do cíle, kolik bodů získal za střelbu a kolik zdí může ještě postavit. Je také nutné popsat oba hráčovy roboty. Ke každému robotu je třeba uvést, kolik má životů, kterým směrem je natočený, a na kterém poli se nachází. Dále jsem se rozhodl uvést zde i cílové pole. Tato informace by se dala vypočítat i z jiných informací, ale její uvedení umožňuje drobné změny v pravidlech. Další informace, která se využije převážně ve vizualizační utilitě, je odpověď jednotlivých strategií z předchozího kola.

Dále by bylo možné doplnit textovou vizualizaci hry pro rychlejší orientaci při tvorbě strategií. Textová vizualizace nakonec nebyla doplněna, protože pro vizualizaci hry lépe slouží grafická vizualizační utilita.

Výstupem hráčských strategií bude pohybová akce a speciální akce pro oba dva roboty.

4.4 Návrh vizualizační utility

Vizualizační utilita má umožnit prohlížení hry pro usnadnění vývoje strategií. Bude zobrazovat dění na hrací desce v jednotlivých kolech. Jejím cílem je také umožnit nahrávání celých záznamů zápasů.
Pro vývoj vizualizační ulity jsem vybral programovací jazyk C++ s využitím frameworku Qt. Grafická část aplikace bude vytvořena pomocí jazyka QML.

Jazyk C++ byl vybrán pro korespondenci s vyhodnocovacím serverem a možné znovuvyužití kódu. Knihovny Qt byly zvoleny pro jejich rozšířenost na linuxových distribucích. Jazyk QML jsem zvolil z důvodu snadné tvorby grafických aplikací s možností animací.

Vizualizační utilita bude umožňovat výběr složky se záznamem zápasu a procházení popisujících souborů každého kola hry. Vizualizační utilita by měla přehledně zobrazit informace o dění na hrací desce. Musí zobrazovat pozice všech zdí, rozlišovat mezi živými a zbořenými zdi, pozice živých robotů a jejich směr pohledu.

5 Implementační nástroje

Výběr implementačních nástrojů byl ovlivněn požadavkem ze zadání, aby implementační server a vizualizační utilita fungovaly na všech běžných linuxových distribucích.

5.1 C/C++

Programovací jazyky C a C++ jsou si velmi blízké. Je třeba zdůraznit, že C++ není pouhé rozšíření jazyka C, neboť při návrhu C++ byla kompatibilita s jazykem C brána v potaz, nikoliv však za každou cenu. V jazyce C existují konstrukce, které nejsou v jazyce C++ dovoleny. Existují také konstrukce, které mají v obou jazycích různý výraz.

Jazyk C navrhl a implementoval na počátku sedmdesátých let 20. století Dennis Ritchie a použil ho při implementaci jedné z verzí operačního systému Unix. Brzy se ukázalo, že jde o velice příjemný a přitom i mocný programovací jazyk a začal být používán k nejrůznějším účelům. S tím ovšem procházel řadou změn. V roce 1978 vydal D. Ritchie spolu s B.W. Kernighanem knihu The C Programming Language, která se stala na dlouho dobu neoficiálním standardem jazyka C. V roce 1990 byl jazyk C poprvé standardizován na mezinárodní úrovni.

Jedním z prvních kroků na cestě k jazyku C++ byl Objektiv C. Jazyk C++ vznikl v roce 1985 a od té doby dozrál a je stejně úspěšný jako jazyk C. Jedním z důvodů úspěchu je jistě téměř úplná kompatibilita C++ s C, která programátorům pracujícím v C usnadnila přechod k objektově orientovanému programování a zaručila použití již existujících kódů napsaných v jazyce C. *Tento odstavec byl převzat z [2].*

5.2 Qt Creator

Qt Creator je vývojové prostředí umožňující vývoj aplikací v programovacím jazyce C++ s využitím knihoven Qt. Výhodou Qt knihoven je umožnění lokalizace aplikací, správa vláken, přístup k souborům a práce s grafikou.

Vývojové prostředí Qt Creator zahrnuje textový editor, napojení na verzovací systém, debugger, dissigner pro návrh grafické rozhraní aplikací a mnohé další nástroje. *[4]*

5.3 QML

Jazyk QML (Qt Markup/Modeling Language) je součástí frameworku QtQuick. Byl vyvinut hlavně pro použití na mobilních zařízeních a práce v něm je rychlejší a jednodušší než při použití nativního C++. Jedná se o obdobu tvorby rozhraní na webu pomocí CSS a JavaScriptu, která nabízí možnost konstrukce 2D i 3D dynamických uživatelských rozhraní a výstupů.

K definici chování elementů je použit JavaScript. Pro začlenění do Qt aplikace se pak využije widget QDeclarativeView do kterého se pouze načte obsah definovaný v QML. QML a QDeclarativeView je vystavěno nad standardními Qt třídami a lze tedy použít mechanismus signálu a slotu pro zasílání zpráv a využít dalších vlastností meta-object systému. V QML pak lze přistupovat k nativním datům z jazyka C++ a naopak QML objekty jsou přístupné z C++ kódu. QtQuick se tedy využívá hlavně pro definici vzhledu jednotlivých prvků uživatelského rozhraní, zatímco pro ostatní logiku a hlavně jádro aplikace se kvůli výpočetní efektivitě používá backend napsaný v jazyce C++. *[4]*

5.4 Python

Python je pravděpodobně nejsnadněji osvojitelný programovací jazyk, který se snadno používá. Kód jazyka Python je srozumitelný pro čtení i zápis a k tomu je stručný bez jakéhokoli nádechu tajemna. Python je velice expresivní jazyk, což znamená, že obvykle stačí napsat daleko méně řádků kódu jazyka Python, než kolik by jich bylo zapotřebí pro ekvivalentní aplikace napsané třeba v jazyku C++. Jednou ze silných stránek Pythonu je, že se dodává se skutečně kompletní standardní knihovnou. Kromě této standardní knihovny existuje velké množství knihoven třetích stran, které jsou k dispozici v seznamu balíčků pro jazyk Python.

V jazyku Python lze programovat v procedurálním, objektově orientovaném a v menší míře též funkcionálním stylu, i když v jádru je Python objektově orientovaným jazykem. Python je dynamický interpretovaný jazyk. *Tento odstavec byl převzat z [3].*

6 Popis implementace

6.1 Komunikační protokol

Komunikační protokol je tvořen dvěma částmi - předáváním informací jednotlivým soutěžním strategiím a přebíráním jejich odpovědí vyhodnocovacím serverem.

6.1.1 Předávání informací soutěžním strategiím

Soutěžním strategiím je třeba předat informace popisující celou situaci na hrací desce, informace o průběžném skóre všech hráčů a počtu zdí, které mohou postavit, a také pro kterého hráče má strategie vytvářet odpověď. Jako vhodné řešení se projevilo spouštět strategie pouze se dvěma argumenty příkazového řádku.

Prvním argumentem je **identifikátor hráče**. Jedná se o číslo 0 až 3, které specifikuje hráčovu startovní pozici na hrací desce:

- 0 - Vrch hrací desky, v informačním souboru má hráč označení P1,
- 1 - pravá strana hrací desky, hráč je označen P2,
- 2 - spodek hrací desky, hráč je označen P3,
- 3 - levá strana hrací desky, hráč je označen P4.

Druhým argumentem je odkaz na **soubor popisující dění na hrací desce**. Tento soubor má následující podobu:

```
aktuální kolo/Poslední kolo
Rozměry hrací desky
P1: InGame FinishRound Skore_za_strelbu Počet_zdí_ke_stavbě R1
životy směr X Y R2  životy směr X Y
P2: InGame FinishRound Skore_za_strelbu Počet_zdí_ke_stavbě R1
životy směr X Y R2  životy směr X Y
P3: InGame FinishRound Skore_za_strelbu Počet_zdí_ke_stavbě R1
životy směr X Y R2  životy směr X Y
P4: InGame FinishRound Skore_za_strelbu Počet_zdí_ke_stavbě R1
životy směr X Y R2  životy směr X Y
P1_finish: 1 1,2 1,3 1,4 1,5 1,6 1,7 1,8 1,9 1,10 1,11 1,12 1,13
1,14 1,15 1,16 1,17 1,18 1,19 1,20 1,21 1,22 1,23 1,24 1,25 1,
P2_finish: 1 1,1 2,1 3,1 4,1 5,1 6,1 7,1 8,1 9,1 10,1 11,1 12,1
13,1 14,1 15,1 16,1 17,1 18,1 19,1 20,1 21,1 22,1 23,1 24,1 25,
P3_finish: 1 25,2 25,3 25,4 25,5 25,6 25,7 25,8 25,9 25,10 25,11
25,12 25,13 25,14 25,15 25,16 25,17 25,18 25,19 25,20 25,21 25,22
25,23 25,24 25,25 25,
P4_finish: 25 1,25 2,25 3,25 4,25 5,25 6,25 7,25 8,25 9,25 10,25
11,25 12,25 13,25 14,25 15,25 16,25 17,25 18,25 19,25 20,25 21,25
22,25 23,25 24,25 25,
Walls: X Y životy, X Y životy,
P1_move: odpověď hráče P1 z minulého kola
```

```
P2_move: odpověď hráče P2 z minulého kola
P3_move: odpověď hráče P3 z minulého kola
P4_move: odpověď hráče P4 z minulého kola
P1: Skore name
P2: Skore name
P3: Skore name
P4: Skore name
```

Položka `InGame` udává, zda je hráč stále ve hře. Hodnota 1 značí, že hráč je ve hře (nedorazil do cíle ani jeho roboti nejsou zničeni). Hodnota 0 značí, že hráč hru ukončil.

`FinishRound` udává kolo v kterém hráč dorazil do cíle. Pokud tam nedorazil, je hodnota `FinishRound` rovna `INT_MAX`.

`Skore_za_střelbu` je součet všech bodů, které hráč získal při akci střelba za přímé a nepřímé zásahy robotů.

`Počet_zdí_ke_stavě` značí, kolik zdí může daný hráč ještě postavit. Na začátku hry jich má k dispozici 20.

Každý robot má definovanou hodnotu `směr`, která nabývá hodnot: `UP`, `RIGHT`, `LEFT` a `DOWN` dle směru na hrací desce, kterým je robot natočený. Položka `životy` udává počet životů robota. Na začátku hry jich má 13. Pokud počet jeho životů klesne na 0 a méně, je robot vyřazen ze hry, a pole, na kterém se nachází, je považováno za neobsazené.

Řádek `Px_finish:` udává cílové pole daného hráče, pokud na toto pole dorazí jeden z jeho robotů, získává hráč body za příchod do cíle. Body jsou určeny dle pořadí, v kterém hráč do cíle dorazil. Tímto hra pro hráče končí.

Řádek `Walls:` popisuje postavené a zbořené zdi na hrací desce. Položky `X` a `Y` udávají pozici zdi. Položka `životy` určuje kolik zdi zbývá životů a dle této hodnoty lze zjistit, zda je zeď zbořená. Zeď má na začátku hry 40 životů. Pokud její životy klesnou na 0 a méně, je považována za zbořenou. Zbořená zeď se herně chová jako prázdné pole, na kterém však nelze postavit novou zeď.

Řádky `PX_move:` popisují odpověď hráče v minulém kole.

Položka `Skore` udává průběžný celkový součet bodů hráče. Položka `name` udává jméno hráče.

Příklad informačního souboru:

```
1/30
25x25
P1: 1 2147483647 0 20 R1 13 DOWN 11 24 R2 13 DOWN 14 24
P2: 1 2147483647 0 18 R1 13 LEFT 24 12 R2 13 LEFT 24 15
P3: 1 2147483647 0 19 R1 13 UP 12 2 R2 13 UP 15 2
P4: 1 2147483647 0 19 R1 13 RIGHT 2 11 R2 13 RIGHT 2 14
P1_finish: 1 1,2 1,3 1,4 1,5 1,6 1,7 1,8 1,9 1,10 1,11 1,12 1,13
1,14 1,15 1,16 1,17 1,18 1,19 1,20 1,21 1,22 1,23 1,24 1,25 1,
P2_finish: 1 1,1 2,1 3,1 4,1 5,1 6,1 7,1 8,1 9,1 10,1 11,1 12,1
13,1 14,1 15,1 16,1 17,1 18,1 19,1 20,1 21,1 22,1 23,1 24,1 25,
```

```
P3_finish: 1 25,2 25,3 25,4 25,5 25,6 25,7 25,8 25,9 25,10 25,11
25,12 25,13 25,14 25,15 25,16 25,17 25,18 25,19 25,20 25,21 25,22
25,23 25,24 25,25 25,
P4_finish: 25 1,25 2,25 3,25 4,25 5,25 6,25 7,25 8,25 9,25 10,25
11,25 12,25 13,25 14,25 15,25 16,25 17,25 18,25 19,25 20,25 21,25
22,25 23,25 24,25 25,
Walls: 12 9 40, 15 18 40, 5 24 40, 7 13 40,
P1_move: R1 UP FIRE R2 UP FIRE
P2_move: R1 UP BUILD 12 9 R2 UP BUILD 15 18
P3_move: R1 UP BUILD 5 24 R2 UP FIRE
P4_move: R1 UP FIRE R2 UP BUILD 7 13
P1: 30 strategie1
P2: 20 strategie2
P3: 32 strategiePy
P4: 22 strategieC
```

Struktura tohoto souboru byla zvolena tak, aby umožňovala případné změny v herních pravidlech a tvorbu nových herních módů. Proto jsou zde použity duplicitní informace o rozsahu hrací desky a definice cílových polí každého hráče. Také je umožněno přidávání a ubírání robotů. Tyto změny pravidel by přinesly značné zesložitění implementace soutěžních strategií, a proto nebyly využity.

6.1.2 Odpověď strategií

Soutěžní strategie musí odpovídat jednotným způsobem, který dokáže vyhodnocovací server zpracovat. Za odpověď server považuje první řádek na standardním výstupu. Tento řádek musí být v požadovaném tvaru:

`R1 pohybová_akce strategická_akce R2 pohybová_akce strategická_akce`

kde za pohybovou_akci se dosadí jedno z následujících:

- `UP` – pro pohyb vpřed,
- `LEFT` – pro otočení robota doleva,
- `RIGHT` – pro otočení robota doprava,
- `NONE` – robot neprovede žádnou pohybovou akci.

Za strategickou_akci se dosadí:

- `FIRE` – pro provedení akce střelba,
- `BUILD X Y` - pro provedení akce stavba zdi na pozici `X Y`, kde `X` a `Y` jsou souřadnice pole na hrací desce.
- `NOTHING` – robot neprovede žádnou speciální akci.

Příklad výstupu soutěžní strategie:

```
R1 UP BUILD 10 15 R2 LEFT FIRE
```

Ve výstupu je důležitá velikost písmen. Pokud výstup není v očekávaném tvaru, server jej zpracovává jako výchozí výstup:

```
R1 NONE NOTHING R2 NONE NOTHING
```

6.2 Vyhodnocovací server

Vyhodnocovací server byl vytvořen v programovacím jazyce C++. Ve funkci `main` se pracuje s objektem třídy `tparser` a objektem třídy `tgame`.

Objekt třídy `tparser` slouží ke kontrole a zpracování parametrů z příkazové řádky.

6.2.1 Třída tGame

Objekt třídy `tgame` slouží k cyklickému spouštění jednotlivých strategií, zpracování jejich odpovědí a na konci hry určí vítěznou strategii. Tato třída je definována v souboru game.h a game.cpp. V souboru game.h jsou definovány důležité konstanty, kterýma lze měnit dílčí pravidla hry a nastavení vyhodnocovacího serveru. Všechny uvedené konstanty musí obsahovat celé číslo.

```
#define STRAIGHT_FIRE 4
#define SPLASH_FIRE 2
#define STEP_ATTAK 1
#define MAX_FIN_BONUS 50
#define FIN_BONUS_STEP 10
#define CHECK_BOTS_INTERVAL 100
#define ONE_ROUND_DELAY 4000
#define NUM_PLAYERS 4
```

Konstanta `STRAIGHT_FIRE` udává počet ubraných životů při zásahu objektu přímou střelbou a počet bodů, které střílející hráč získá, pokud zasáhl robota.

Konstanta `SPLASH_FIRE` udává počet životů ubraných objektu nepřímým zásahem, vzniklým rozptylem od zasaženého objektu a také počet bodů, které střílející hráč získá při zásahu robota.

Konstanta `STEP_ATAK` udává velikost zranění, způsobená pohybem robota zdi nebo robotu, který mu stojí v cestě.

Pomocí konstanty `MAX_FIN_BONUS` a konstanty `FIN_BONUS_STEP` lze spočítat bodový zisk udělený za příchod hráče do cíle. Výpočet se provede tak, že od konstanty `MAX_FIN_BONUS` se odečte tolikrát konstanta `FIN_BONUS_STEP`, kolikáté je pořadí hráče v cíli. U hráče, který dorazil do cíle, se konstanta `FIN_BONUS_STEP` odečte jednou, u čtvrtého hráče se tato konstanta odečte čtyřikrát.

Konstantou `CHECK_BOTS_INTERVAL` se definuje počet milisekund, jak často vyhodnocovací server po spuštění soutěžních strategií zjišťuje, zda již odeslaly odpověď. Tato konstanta musí být v rozsahu 1-1000.

Konstantou `ONE_ROUND_DELAY` se definuje maximální časový interval v milisekundách, který mají všechny čtyři soutěžní strategie dohromady na odeslání odpovědi.

Konstantou `NUM_PLAYERS` se určuje počet soupeřících strategií. Změna této konstanty by však žádala větší změny ve zdrojových kódech vyhodnocovacího serveru.

Základní algoritmus programu:

```
while(game.round_counter < game.max_round
&&game.someone_ingame()){

        //pusteni vsech strategii
        game.gen_file();
        game.run_strategi();

        //vyhodnoceni akci
        game.step_move();
        game.step_fire();
        game.step_build();

        //posun kola
        game.round_counter++;

    }//while(round << max_round)

    game.gen_file();
    game.print_score();
```

Algoritmus hry je inspirován algoritmem pro řízení spojité simulace. Jedná se o cyklus, který končí po maximálním počtu kol, nebo když všichni hráči ukončí hru.

V těle cyklu se vygeneruje informační soubor pro dané kolo metodou `gen_file`. Poté se spustí soutěžní strategie a převezme se od nich odpověď v metodě `run_strategi`. Dále následuje vyhodnocení pohybových akcí, střelby a stavění zdí v metodách `step_move`, `step_fire`, `step_build`. V závěru proběhne posun počitadla kol inkrementací atributu `round_counter`. Po skončení hry se vytiskne soubor s výsledky hry pomocí metody `print_score`.

Metoda gen_file

Metoda `gen_file` vytváří informační soubor dle výše popsaného komunikačního protokolu. Všechny vypisované hodnoty se nachází ve vnořených objektech třídy `tplayer` a `tdesk`. Atribut `name` udávající jméno hráče je získán z atributu `strategy`, který obsahuje cestu k soutěžní strategii. Atribut `name` tvoří první adresář této cesty.

Metoda run_strategi

Metoda `run_strategi` slouží ke spuštění soutěžních strategií a převzetí jejich odpovědí. Větší část této metody byla převzata z implementace vyhodnocovacího serveru k předchozímu ročníku soutěže RedBot. Metoda využívá pole struktur typu

thread_data_t, ve kterém je jedna struktura pro každou soutěžní strategie. V této struktuře je třeba nastavit identifikátor vlákna v položce tid a spouštěný program (soutěžní strategii) v položce binary_to_exe. Následně je možné program pustit ve vlákně použitím funkce pthread_create.

Po spuštění vláken se soutěžními strategiemi běží v hlavním programu smyčka, která se každých 100 milisekund dotazuje, zda je v položce response[0] struktury thread_data_t odpověď. Pokud odpověď nepřijde do 4 sekund, jsou všechny doposud běžící vlákna zastaveny.

Po přijetí odpovědi se nakopíruje její první řádek do atributu response objektu hráči a jeho metodou actualize se odpověď zpracuje.

Metoda step_move

Tato metoda prochází všechny roboty a provádí jejich pohybovou akci. Pohybová akce otočka doleva a otočka doprava se provede vždy.

Akce pohyb vpřed používá metodu go_up pro zjištění, na které pole se hráč chce posunout a zda je toto pole za hranicí hrací desky. Pokud toto pole je za hranicí hrací desky, metoda go_up vrátí pole, na kterém robot stojí. Dle toho se rozhodne, že pohyb vpřed se neprovede. Dále se zjišťuje, zda je toto pole prázdné. Pokud se jedná o prázdné pole, robot si vytvoří žádost o přístup na toto pole metodou push_request objektu desk. Pokud je pole obsazené, je objekt na tomto poli zraněn o STEP_ATTAK životů.

Po provedení výše uvedeného pro všechny roboty se prochází vektor request objektu desk. V tomto vektoru se nachází pole, na které chce některý z robotů vstoupit a vektor ukazatelů na roboty, kteří na něj chtějí vstoupit.

Pokud je v tomto vektoru pouze jeden ukazatel na robota, pohyb se provede. Pokud je zde více ukazatelů na roboty, všichni roboti jsou zraněni o STEP_ATTAK životů.

V závěru metody probíhá kontrola, zda některý z hráčů přišel se svým robotem do cíle.

Metoda step_fire

Tato metoda prochází všechny živé roboty, kteří mají nastavenou speciální akci střelbu. Nejdříve se metodou count_strange_fire zjistí zasažený objekt a provede odpočet životů zasaženého objektu. Pokud se jedná o robota, přičte se střílejícímu hráči příslušný počet bodů, když metoda vrátila NULL, žádný objekt nebyl zasažen. Pokud se vrátil ukazatel na některý objekt, jsou všechny objekty v osmi okolí toho zasaženého objektu zraněny metodou count_splash_fire, která se také postará o odpočet životů objektu a při zasažení robota přičtení bodů střílejícímu hráči.

Po provedení střelby všech robotů se metodou check_dead_walls zjišťuje, které zdi jsou zbořeny. Dále se metodou check_dead_robot vyvolanou každému

hráči zjišťuje, zda nebyl zničen některý z jeho robotů a zda hráče není třeba vyřadit ze hry, protože přišel o oba své roboty.

Metoda step_build

Metoda `step_build` prochází všechny živé roboty, kteří mají nastavenou speciální akci stavění zdi. Nejdříve se zjišťuje, zda se zeď staví na prázdném poli metodou `who_here` a také zda se nejedná o osmi okolí některého robota metodou `isRobotArea`. Pokud na tomto poli lze stavět, uloží se pole do vektoru `new_walls` pro pozdější postavení a hráči se zeď odečte.

Po průchodu všech robotů se prochází vektor `new_walls` a pokud metoda `who_here` označí pole za neobsazené, zeď se zde postaví. Tím je docíleno toho, že pokud více hráčů staví ve stejném kole zeď na jedno pole, je zeď postavena jen jedna, ale je odečtena všem stavějícím hráčům.

Pokud hráč staví zeď mimo hrací pole, je tato zeď postavena a hráči odečtena, nikomu však nebrání v pohybu.

Metoda print_score

Metoda `print_score` vytváří soubor score.txt do složky s popisem hry. V tomto souboru jsou uloženy výsledky hry.

Nejdříve se metodou `sort` seřadí ukazatele na objekt hráče vložené do pomocného vektoru `sort_players` dle atributu `fin_round`. Poté se objekty hráčů prochází podle seřazeného pořadí, a těm hráčům, kteří dorazili do cíle (mají v atributu `fin_round` jinou hodnotu než `INT_MAX`), jsou přiděleny body za příchod cíle do atributu `fin_bonus`. Přidělená hodnota je spočítaná z proměnné `bonus`, která je na začátku rovna `MAX_FIN_BONUS`. Od této proměnné se odečítá konstanta `FIN_BONUS_STEP`, pokud předchozí hráč dorazil v jiném kole než hráč, pro kterého se právě počítá proměnná `bonus`. Pokud hráč nedorazil do cíle, hodnota jeho atributu `fin_bonus` je rovna 0.

Po vypočtení atributu `fin_bonus` všem hráčům dochází k tisku porovnávaných hodnot. Jedná se o součet počtu životů hráčových robotů, bodů, které hráč získal za střelbu a právě spočítaného bonusu za příchod do cíle.
Pokud je počet udávaných životů robota menší než 0, je započítána 0.

6.3 Vizualizační utilita

Vizualizační utilita byla vytvořena v programovacím jazyce C++ s využitím knihoven Qt a jazyka QML. V rámci implementace lze samostatně mluvit o tvorbě uživatelského rozhraní a o implementaci obslužných funkcí, které zpracovávají události v něm vyvolané.

6.3.1 Uživatelské rozhraní

Hlavním souborem uživatelského rozhraní je soubor main.qml. Tento soubor obsahuje popis rozmístění ovládacích tlačítek, popisku s číslem zobrazovaného kola a umístnění hrací desky v qml souboru GameWiew.

Ovládací tlačítka jsou definovaná v souboru button.qml. Tento soubor je šířen pod licencí Creative Commons "BY-SA", která umožňuje použití kódu pod stejnou licencí a se zveřejněním autora QUItCoding. *[9]*

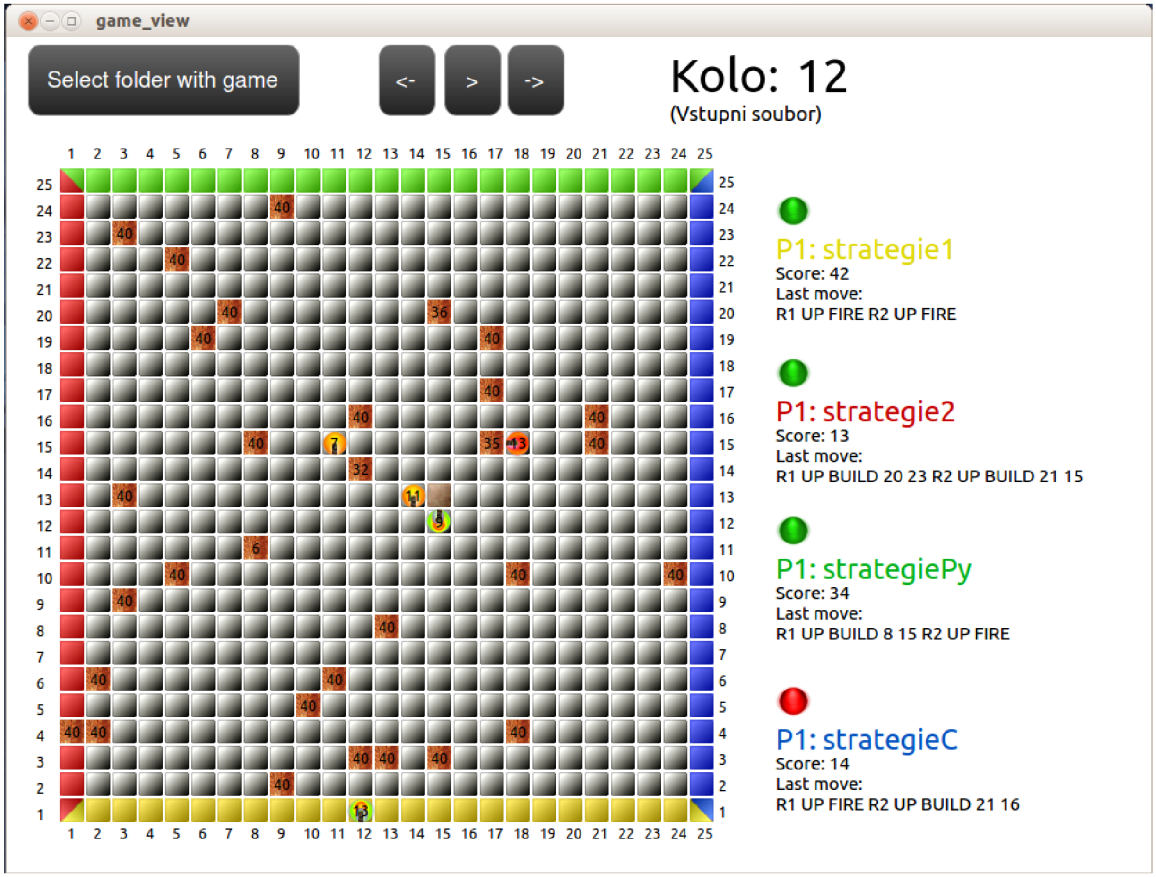
Soubor GameWiew.qml slouží k popisu hrací desky. Obsahuje `grid` tvořený 25 sloupci. V tomto `gridu` je pomocí repeatru vykresleno 625 polí hrací desky. Tyta pole jsou definovány v Qlistu `m_tiles`, který se nachází v objektu `GameData`. Pomocí tohoto objektu lze k jednotlivým polím přistupovat a měnit jejich hodnoty.

Samotná pole jsou popsána v qml souboru tile.qml. Pole je tvořeno obrázkem na pozadí, obrázkem v popředí, textovým popiskem udávajícím životy objektu na poli a u krajních polí se také nachází textový popisek udávající souřadnici pole. Všechny tyto položky se zobrazují v závislosti na hodnotách atributů objektu daného pole. Toto nastavení se zjišťuje dotazem na `modelData`, který zde zastupuje ukazatel na objekt pole.

Příklad:

```
Image {
source: {
    if (modelData.hasLive<1 && modelData.hasWall)
                "images/dead_wall.png"
```

Tento příkaz volá metody `hasLive` a `hasWall` objektu pod zástupným jménem `modelData`. Jedná se o příslušný objekt třídy `tile` odkazovaný v Qlistu `m_tiles`. Dle získaných hodnot z tohoto objektu je zobrazen obrázek.



Obrázek 3 - Vizualizační utilita

6.3.2 Obslužné funkce

Ve funkci main je instanciován objekt `MainWidget`. Tento objekt má mezi atributy objekt třídy `gameData` `m_gameData`, který tvoří jádro aplikace. V konstruktoru objektu `MainWidget` se dále nachází pouze nastavení kontextu pro propojení grafické části s objektem `m_gameData`.

Třída gameData

Třída `gameData` slouží k odchytávání uživatelských událostí vyvolaných z grafického rozhraní a jejich zpracování. Jedná se o tyto události:

```
void find_folder();
void bforward_click();
void bback_click();
void bplay_click();
```

Událost `find_folder` je napojena na tlačítko načíst složku. Jejím úkolem je umožnit uživateli výběr složky se hrou a zobrazit první soubor hry. Výběr složky se hrou je proveden pomocí metody `QfileDialog::getExistingDirectory`. Po té je nastaveno `id_file` na 0, což znamená zobrazovat první vstupní soubor hry. Následně jsou zavolány metody `clear_desk`, `parse_file` a `show_desk`. Úkolem těchto metod je vymazání všech objektů z hrací desky, získání dat z nového souboru a jejich zobrazení na hrací desce. Tyto metody budou popsány níže.

Procházení informačních souborů

Metoda `bforward_click` slouží k zobrazení následujícího vstupního souboru. Je v ní provedena inkrementace proměnné `id_file` udávající zobrazovaný soubor, vymazání objektů z hrací desky, zpracování nového vstupního souboru a jeho zobrazení na hrací desce.

Metoda `bback_click` funguje obdobně jako metoda `bforward_click`, ale zobrazuje předchozí soubor.

Metoda `bplay_click` slouží k přehrání celé hry ve smyčce, nebo naopak k zastavení probíhajícího přehrávání. Toto rozhodnutí probíhá pomocí atributu `_is_play_loop`, do nějž se ukládá stav aplikace. Na jeho základě se rozhoduje, zda se pustí cyklus přehrávání hry pomocí zaslání zprávy `start` objektu `myThread`, nebo se přehrávání ukončí zasláním zprávy `therminate` stejnému objektu.

Úkolem objektu `myThreadu` je spuštění přehrávací smyčky v novém vlákně. Kdyby se přehrávání pouštělo ve stejném vlákně jako zbytek aplikace, nebylo by jej možné přerušit.

Objekt `myThread` je vytvořen zděděním od třídy `Qthread`, která umožňuje práci s více vlákny. Objekt `MyThread` vyžaduje v konstruktoru ukazatel na objekt třídy `GameData`, protože bude volat jeho metody. Objekt `MyThread` obsahuje jedinou metodu `run`, která se invokuje při obdržení zprávy `start`. Metoda `run` zasílá zprávu `game_play` objektu třídy `GameData`, jehož reference se získala v konstruktoru objektu.

Metoda `game_play` cyklicky volá metodu `bforward_click` a uspání na 1 sekundu, aby si uživatel stačil prohlédnout změny na herní desce. Cyklus je ukončen, když metoda `set_next` vrátí `false`. To značí, že neexistuje další vstupní soubor k zobrazení.

Dobu mezi jednotlivými snímky lze upravit změnou konstanty `ROUND_DELAY`, která je definována v souboru gamedata.h. Očekává se zde celé číslo, které udává počet sekund.

Vymazání objektů z hrací desky

Metoda `clear_desk` slouží k odstranění všech objektů z hrací desky. Tato metoda využívá vnořených objektů třídy `tdesk` a `tplayer` převzaté z implementace vyhodnocovacího serveru. Metoda `clear_desk` pomocí těchto objektů prochází všechny zdi a roboty a vymazává je z hrací desky. Při vymazávání se nejdříve se zjistí

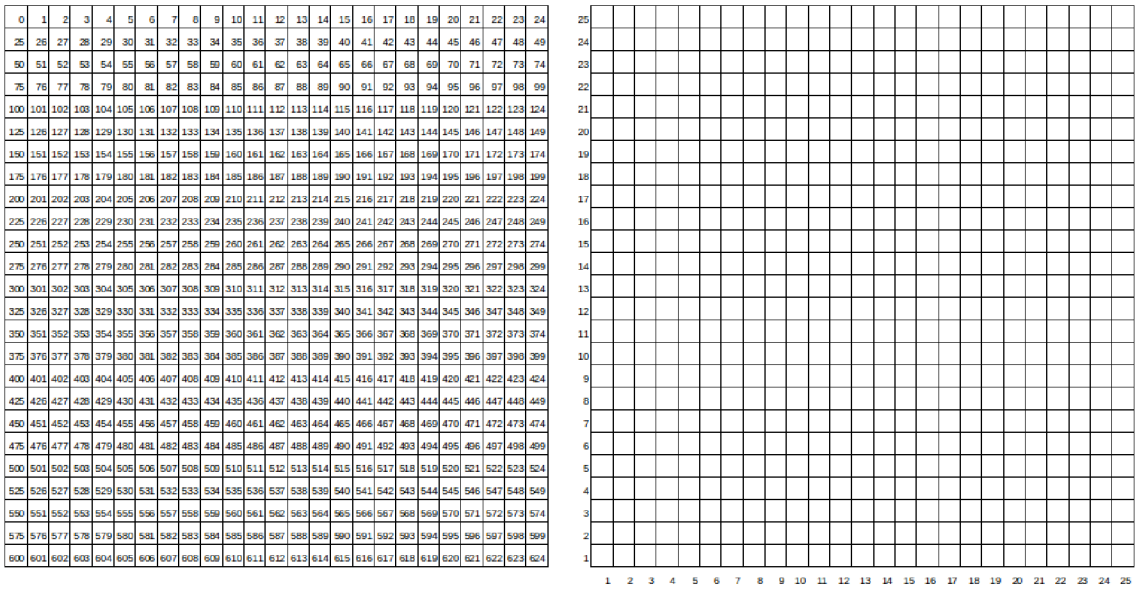
`index` vymazávaného pole ve Qlistu `m_tiles` z pozice zdi zadané souřadnicemi `x y` metodou `pos_to_Index`.

Metoda pos_toIndex:

```
if(x<1 || x>25 || y<1 || y>25)
                return 0;
return ((DESK_SIZE-y)*DESK_SIZE)+x-1;
```

Nejdříve probíhá kontrola, zda je validní vstupní pozice, jinak se vrátí index 0. Z toho vyplývá, že na toto pole jsou vykresleny zdi, které hráč postavil mimo hrací desku.

Pro validní souřadnice se zde vypočítá index v rozsahu 0 až 624 začínající vlevo nahoře 0 ze souřadnic v 1. kvadrantu kartézského souřadného systému začínající od 1. (Viz obrázek 4)



Obrázek 4 - Převáděné tabulky v metodě pos_toIndex

Po získání indexu, se nastaví správnému poli zrušení zdi:

```
m_tiles[index]->setHasWall(false);
```

Aby se změna provedla do grafického rozhraní, musí být ve třídě `tiles` na implementované metody a atributy:

```
bool m_hasWall;
Q_PROPERTY(bool hasWall READ hasWall WRITE setHasWall NOTIFY
hasWallChanged);
bool hasWall() const { return m_hasWall; }
void setHasWall(bool state){ if(state==m_hasWall) return;
m_hasWall=state;emit hasWallChanged();}
```

```
signal: void hasWallChanged();
```

Atribut `m_hasWall` udává stav, zda pole má vykreslit zeď. Tento stav měníme metodou `setHasWall`. Tato metoda při změně stavu zasílá signál `hasWallChanged`. Protože tento signál je v `Q_PROPERTY` propojen s metodou `hasWall`, která je využita v qml souboru tile pro určení, zda vykreslovat zeď na pozadí pole (viz výše), změní se dle této hodnoty pozadí pole.

Zpracování informačního souboru
Metoda `parse_file` slouží k načtení vstupního souboru do objektů, jenž usnadní následné vykreslování. Celá funkce se nachází v bloku `try`, aby chyba při zpracovávání nevalidního vstupního souboru neshodila aplikaci.

Vstupní soubor se čte po řádcích. Tyto řádky se rozdělují metodou `string.split` do vektoru, který obsahuje jednotlivé informace, které se ukládají do vektoru objektů třídy `tplayer hráči` a do objektu `desk` popisujícího hrací desku.

Vykreslení objektů na hrací desku
Metoda `show_desk` prochází všechny zdi a roboty v objektech `desk` a ve vektoru `hráči` a zobrazuje je na hrací desce stejným způsobem, jaký byl popsán v metodě `clear_desk`.

Dále jsou zde volány metody `set_next` a `set_before`, které ovládají funkčnost tlačítek pro přechod na následující a předchozí snímek hry. V těchto metodách se zkouší otevřít následující, nebo předchozí soubor k aktuálně zobrazovanému souboru. Podle něj se nastaví povolení, nebo zakázání příslušného tlačítka.

6.4 Obslužné rutiny

6.4.1 Obslužná rutina pro C

Tato obslužná rutina je navržena a implementována tak, aby byla použitelná také pro strategie implementované v jazyce C++. Kvůli tomuto požadavku je explicitně zadáván typ textových řetězců na char *, protože tímto způsobem lze reprezentovat řetězce znaků v C i v C++.

Hlavním účelem obslužné rutiny je zpracování vstupního souboru s popisem dění na hrací desce do vhodných struktur, se kterými bude soutěžící pracovat při tvorbě strategie. Strukturou, přes kterou lze přistupovat ke všem atributům, je **struktura Game**:

```
typedef struct sGame{
    int my_position;
```

```
    int akt_round;
    int finish_round;
    player players[4];
    wall walls[MAX_WALLS*4];
    int walls_counter;
}game;
```

Položka `my_position` udává pozici hráče na hrací desce. Položka `Akt_round` udává číslo aktuálního kola, `finish_round` udává číslo posledního kola. Položka `players` je pole struktur popisující informace vztahující se k jednotlivým hráčům, tato struktura bude popsána níže. `Walls` je pole struktur popisující postavené a zbořené zdi na hrací desce. `Walls_counter` udává počet zdí popsaných v poli `walls`.

Struktura wall:

```
typedef struct sWall{
    int x;
    int y;
    int lives;
}wall;
```

Struktura `wall` popisuje každou postavenou zeď. Její položky `x` a `y` označují pozici zdi na hrací desce, položka `lives` udává počet životů zdí.

Struktura players:

```
typedef struct sPayer{
    bool in_game;
    int finish_round;
    int score;
    int walls;
    robot robots[2];
    tplace places[25];
}player;
```

Struktura `player` slouží k popsání informací o hráči. Položka `in_game` udává, zda je hráč stále ve hře. Hru hráč mohl opustit tím, že přišel o oba své roboty nebo jedním ze svých robotů dorazil do cíle. Položka `finish_round` udává kolo, ve kterém hráč přišel do cíle. Pokud do cíle zatím neodrazil, je zde uložena hodnota `INT_MAX`. Položka `score` popisuje počet bodů, které hráč získal za střelbu, položka `walls` udává, kolik zdí může hráč postavit. Pole struktur `robots` slouží k popisu hráčových robotů. Pole struktur `places` udává cílová pole hráče. Struktura `tplace` je tvořena položkami `x` a `y`, které udávají pozici pole.

Struktura robot:

```
typedef struct sRobot{
    int x;
    int y;
    int lives;
    torient orient;
```

```
    torient last_move;
    torient move;
    actions last_action;
    actions action;

    int build_x;
    int build_y;
    int last_build_x;
    int last_build_y;
}robot;
```

Struktura `robot` slouží popsání pozice robota a jeho tahu z minulého kola. Položky `x` a `y` popisují jeho pozici na hrací desce. Položka `lives` určuje počet životů robota. Položka orient udává směr, jakým je robot natočený.

Položky `last_move` a `last_action` udávají pohybovou a speciální akci robota v minulém kole. Pokud speciální akcí byla stavba zdi, je v položkách `last_build_x` a `last_build_y` uložena pozice této zdi.

Položky `move`, `action`, `build_x` a `build_y` jsou definovány funkcemi `setBuild`, `setFireAction` a `SetBuildAction` a slouží k nastavení tahu hráče a vytisknutí odpovědi strategie ve správném formátu pomocí funkce `printResponse`. Tyto položky jsou definovány jen u robotů hráče, jehož strategie je zpracovávána.

6.4.2 Obslužná rutina pro Python

Tato obslužná rutina byla vytvořena pro jazyk Python 2.7. Pro soutěžící by však neměl být problém upravit si ji pro python3. Účelem obslužné rutiny je zpracování vstupního souboru do vhodné objektové struktury, se kterou se bude soutěžícímu pohodlně pracovat při implementaci strategie.

Objektem, přes který soutěžící přistupuje ke všem informacím ze vstupního souboru, je objekt `game`. Tento objekt obsahuje gettry `akt_round`, `finish_round` a `my_id` pro získaní čísla právě probíhajícího kola, posledního kola a hráčovy pozice na hrací desce. Hodnota `my_id`, udávající pozici hráče na hrací desce, nabývá hodnot 0-3.

Dále tento objekt obsahuje pole zdí `_walls`. Toto pole je tvořeno objekty třídy `tWall`, která díky dědění od objektu `tPlace` zdědila metody pro získání atributů udávající pozici zdi na hrací desce `x` a `y`. Dále tato třída obsahuje gettr `lives`, který vrací počet životů zdi.

Objekt `game` dále obsahuje pole hráčů `_players`, toto pole je tvořeno objekty třídy `tPlayers`. Třída `tPlayers` obsahuje metody `in_game`, `finish_round`, `score` a `walls` pro získání informací, zda je hráč ve hře, v kolikátém kole hráčův robot dorazil do cíle, kolik bodů získal hráč za střelbu a kolik zdí může ještě postavit. Dále také obsahuje pole `_robot`, které popisuje hráčovy roboty a pole `_finish_zone` popisující cílová pole hráče. V poli `_finish_zone` jsou uloženy objekty třídy `tPlace`.

Roboti jsou popsáni třídou `tRobot`. Třída vznikla děděním ze třídy `tPlace`, takže také umožňuje prostřednictvím metody `x` a `y` získat robotovu pozici. Třída

`tRobot` dále obsahuje metodu `orient` pro získání směru pohledu robota. Dále lze získat informace o minulém tahu robota pomocí metod `last_move` a `last_action`. Pokud `last_action` vrací hodnotu `BUILD`, lze získat pozici stavěné zdi metody `last_build_x` a `last_build_y`.

Uložení informací ze vstupního souboru probíhá při vytváření těchto objektů. Konstruktor objektu `game` požaduje odkaz na zpracovávaný soubor, z nějž si po řádcích získává informace. Celý řádek s popisem hráče začínající `PX:` je předán konstruktoru třídy `tPlayer`, která informace z něj získává pomocí funkce `split`. Konstruktory ostatních objektů vyžadují již zpracované hodnoty. Ke všem atributům objektu se přistupuje pouze pomocí metod getrů.

7 Práce s programem

Ovládání obslužných rutin

Hráč, který chce implementovat svou strategii, může využít připravených obslužných rutin. Jedná se kostry kódu strategie v jazyce C a Python, které za hráče vyřeší zpracování vstupních informací do lépe přístupných struktur nebo objektů a výstup aplikace v požadovaném tvaru.

7.1 Obslužná rutina C

Tento program je možno použít pro implementace strategií v jazyce C nebo v C++. Je zde implementováno zpracování parametrů z příkazového řádku a na parsování informačního souboru do struktur. Ke všem položkám se přistupuje pomocí ukazatele `Game`, který ukazuje na strukturu `game`. Přístup k jednotlivým položkám struktury je ukázán v kódu rutiny. Struktura obsahuje zanořenou hierarchii podstruktur obsahující popis zdí a hráčů a jejich robotů. Struktury jsou podrobně rozepsány v kapitole implementace. Uživatel implementuje vlastní strategii a poté použije funkce:

- `setMove(Game, Id_robota, pohybová akce)` – Pro nastavení pohybové akce vybranému robotu.
- `setFireAction(Game, Id_robota)` – Nastavení speciální akce střílení vybranému robotu.
- `setBuildAction(Game,Id_robota, X , Y)` – Nastavení speciální akce stavění na pozici X Y vybranému robotu.

Pokud hráč robotu akci nenastaví, bude pro pohybovou akci odpovídat `NONE` a pro speciální akci `NOTHING`.

Poté co hráč nastaví odpověď pro oba své roboty, použije funkci `printResponse(Game)` pro tisk odpovědi ve správném tvaru. Pro vyhodnocovací server je důležitý pouze první řádek standardního výstupu aplikace.

7.2 Obslužná rutina Python

K jednotlivým položkám po zpracování informačního souboru se přistupuje přes objekt game. Postup je opět ukázán ve zdrojovém kódu obslužné rutiny.

Pro uživatele jsou nachystány metody objektu `game` pro nastavení pohybu a speciální akce robota. Nastavení pohybové akce robota:

- `Set_move(id_robota,pohybové akce)`

Nastavení speciální akce robota se děje pomocí přetížené metody `set_action`:

- `set_action(id_robota,“FIRE“)`
- `set_action(id_robota,“BUILD“,X,Y)`

Tisk odpovědi v požadovaném tvaru se provede pomocí metody `print_response`. Stejně jako u obslužné rutiny C, pokud hráč robotu akci nenastaví, bude pro pohybovou akci odpovídat `NONE` a pro speciální akci `NOTHING`.

7.3 Ovládání vyhodnocovacího serveru

Přeložení vyhodnocovacího serveru bude provedeno příkazem:
`make`

Spouštění vyhodnocovacího serveru:
`./redbot_paintball strategie1 strategie2 strategie3 strategie4 složka maximálni_počet_kol`

Parametry `strategie1` až `strategie4` jsou odkazy na soubor s hráčskými strategiemi.
`Složka` je místo, kam se budou ukládat vstupní soubory jednotlivých kol. Pokud tato složka neexistuje, bude vytvořena s názvem `složka`. Pokud obsahuje záznam předchozí hry, tak bude vymazán. Parametr maximální `počet_kol` udává, po kolika kolech bude hra ukončena.

Pro častější pouštění vyhodnocovacího serveru při vývoji strategie je možné použít příkaz:
`make run`

Tento příkaz spustí vyhodnocovací server s parametry odkazující na ukázkové strategie. Mezi nimi je spuštěna i ukázka zpracovaní obslužných rutin C. Proto je třeba předem tuto strategii přeložit příkazem:
`make strategiC`

Hra spuštěná příkazem `make run` končí po 30 kolech a její průběh bude uložen do složky hra1. Příkaz `make run` lze v souboru Makefile jednoduše upravit nahrazením ukázkových strategií vlastním řešením.

7.4 Ovládání vizualizační utility

Vizualizační utilitu lze přeložit příkazem:
`make gui`

Aby se překlad provedl, je třeba mít na svém počítači nainstalovaný balíček qt-libraies nebo Qt creator verze 4.8 a novější. Dále je potřeba mít v proměnné PATH uvedenou

cestu k utilitě qmake. Toto nastavení lze uložit do makefilu pro usnadnění častějšího překladu.

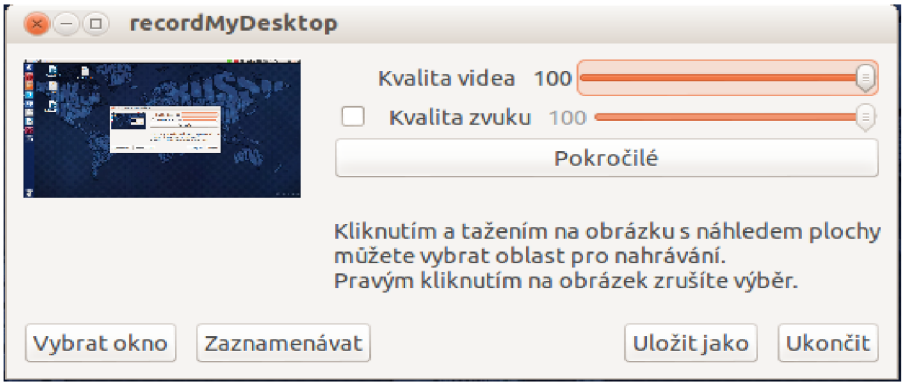
Na distribuci Fedora 18 může dojít k problémům při překladu, které lze vyřešit nainstalováním balíčku qt3-devel a následným upravením proměnné PATH, aby v ní byl uveden odkaz na qmake verze 4.8 dříve než na qmake z qt3-devel.

Vizualizační utilitu lze spouštět pouze ze složky game_view. Po spuštění aplikace je třeba klepnout na tlačítko výběr hry a v dialogovém okně vybrat složku se záznamem hry, vytvořenou vyhodnocovacím serverem. Po načtení hry lze tlačítky ← a → přepínat mezi soubory zobrazujícími situaci na hrací desce na začátku jednotlivých kol. Tlačítkem > lze přehrát celou hru se sekundovou pauzou mezi každým snímkem. Napravo jsou uvedeny informace o jednotlivých hráčích. Zelená tečka značí, že hráč je stále ve hře, červená tečka značí, že hráč již hru opustil. Položka Score udává průběžný součet všech hráčových bodů.

7.5 Nahrávání záznamu hry

Pro tvorbu videa zaznamenávajícího soutěžní zápasy byla vybrána aplikace RecordMyDesktop. Tato aplikace se nachází v centru softwaru ve většině dnes rozšířených linuxových distribucí. Také ji lze stáhnout z webových stránek autorů: http://recordmydesktop.sourceforge.net/. Aplikace Record myDesktop umožňuje nahrávání obrazovky s možností výběru nahrávaného okna. Video vytvořené aplikací je ve formátu Theora v kontejneru ogg.

Záznam zápasu se provede současným spuštěním vizualizační utility a aplikace RecordMyDesktop. Ve vizualizační utilitě se vybere složka s hrou a spustí se automatické přehrání hry. V aplikaci RecordMyDesktop si zvolíme kvalitu nahrávaného videa. Zde bych doporučil ponechat 100, což znamená, že se nebude snižovat rozlišení nahrávaného videa oproti rozlišení aplikace. Nahrávání zvuku lze vypnout, protože nahrávaná vizualizační utilita žádné zvuky nevydává.
V pokročilém nastavení aplikace v záložce výkon je možnost změnit počet snímků za sekundu. Protože vizualizační utilita mění snímek po 1 sekundě, k nahrání videa stačí nahrávat 2 snímky za sekundu. V záložce ostatní je vhodné nastavit kurzor myši na žádný pro nezobrazování kurzoru myši v záznamu. Dále v této záložce je vhodné odškrtnout položky „zahrnout dekoraci oken“ a „vybrat oblast na obrazovce“ pro nahrávání pouze užitkové plochy aplikace.



Obrázek 5 - Aplikace RecordMyDektop

8 Možnosti rozšíření

Tato práce by se dala rozšířit ještě o další úkony, ale na ty již v této bakalářské práci není prostor. Naskýtá se možnost dalšího testování vyhodnocovacího serveru, animování tahů ve vizualizační utilitě, vytvořit více obslužných rutin nebo lépe podpořit vývoj strategií na OS Windows. Na bakalářskou práci by se dalo navázat diplomovou prací nebo realizací soutěže přímo v praxi, kde by se daly tyto možnosti dále rozvést.

Větší rozsah testování

Testování bylo prováděno pouze sledováním očekávaného chování ve vizualizační utilitě. K lepšímu ověření správnosti implementace pravidel hry ve vyhodnocovacím serveru by bylo vhodné vytvořit testy z obslužných rutin pro hráčskou strategii, které by umožnily zkontrolovat chování vyhodnocovacího serveru v mnohých nestandardních situacích.

Animace ve vizualizační utilitě

Bylo by možné docílit, aby vizualizační utilita nezobrazovala pouze statické vstupní soubory do jednotlivých kol, ale přehrála průběh vyhodnocování kola. Nejdříve pohyb robotů, poté jejich střelbu a nakonec stavbu zdí tak, jak probíhá vyhodnocování jednotlivého kola. Při tomto způsobu zobrazování zápasu by se zužitkoval potenciál jazyka qml pro tvorbu animací.

Vytvoření více obslužných rutin

Byly vytvořeny obslužné rutiny pouze pro nejpoužívanější programovací jazyky v minulých ročnících soutěže RedBot. Vždy se najde několik soutěžících, kteří obslužné rutiny nevyužijí a implementují si je sami ve svém oblíbeném programovacím jazyce.

Podpora OS Windows

Všechny návody pro spouštění a ovládání vyhodnocovacího serveru a obslužných rutin byly napsány z pohledu uživatele pracujícího s Linuxem. Portace vytvořených programů na OS Windows by sebou přinesla i drobné změny ve zdrojových kódech, které na tomto systému nebyly testovány. Na multiplatformnost aplikace nebyl kladen důraz z důvodu, že soutěžní strategie musí být schopné běhu na shodném referenčním systému, kterým bude pravděpodobně podporovaná verze Fedory v době vyhlášení soutěže.

9 Závěr

V této bakalářské práci jsem se věnoval vytvoření zadání a vyhodnocovacího serveru pro příští ročník soutěže RedBot.

Nejprve jsem se seznámil se s programátorskou soutěží pro studenty středních a vysokých škol RedBot. V této soutěži studenti mají za úkol vytvořit počítačový program – strategii, která má hrát hru dle zadaných pravidel a porazit ve hře strategie vytvořené jinými studenty.

Cílem této práce bylo vymyslet zadání soutěže, vytvoření vyhodnocovacího serveru, vizualizační utility a základních rutin soutěžních strategií. Vytvořené zadání soutěže je jednoduché a jednoznačné, umožňuje rychle vytvořit primitivní strategii, kterou lze dále zdokonalovat. Vyhodnocovací server umožňuje hru soutěžních strategií proti sobě. Je nezávislý na programovacím jazyku, který soutěžící zvolí pro tvorbu strategie. Vizualizační utilita umožňuje graficky zobrazit průběh hry. Slouží soutěžícím pro ulehčení ladění strategií a také k vytvoření videozáznamu zápasu, který bude dostupný spolu s výsledky na stránce soutěže. Základní rutiny soutěžních strategií slouží soutěžícím ke zpracování vstupu programu a vytvoření výstupu v požadovaném formátu. Jejich použití není nutné, ale je pro ně výhodné, protože se při tvorbě strategie mohou soustředit pouze na její logiku.

Nejprve jsem se věnoval analýze zadání předchozích ročníků a tvorbě vlastního zadání, při čemž jsem se inspiroval pravidly deskových her Quoridor a RoboRally. Zadání jsem vytvořil tak, aby bylo originální a vhodné pro programátorskou soutěž. Vytvořené zadání jsem důkladně zanalyzoval a upravil tak, aby neexistovala jediná výherní strategie.

Podle požadavků zadavatele jsem navrhl vyhodnocovací server, grafickou vizualizační utilitu a komunikační protokol mezi serverem a soutěžními strategiemi. Vytvořil jsem kompletní zadání soutěže, ze kterého se studenti dozví pravidla hry. Jedná se o hru pro čtyři hráče, která se hraje na šachovnici o rozměrech 25x25 polí. Každý hráč má k dipozici 2 roboty a 20 zdí, které může v průběhu hry postavit. Cílem hry je přivést do cíle jednoho ze svých robotů dříve než soupeři. Dalším cílem hry je střílet roboty soupeřů, za jejichž zásahy hráč získává bonusové body. Vyhodnocovací server byl navržen tak, že bude ve smyčce provádět generování souboru s popisem kola, spuštění soutěžních strategií, převzetí jejich odpovědí, vyhodnocení pohybových akcí, vyhodnocení střelby a vyhodnocení stavby zdí. Vizualizační utilita byla navržena tak, aby umožnila výběr složky se záznamem hry a procházení informačních souborů popisujících jednotlivá kola hry. Umožní zobrazit pozice všech zdí, rozlišit zda zeď stojí nebo je zbořená, zobrazit pozice živých robotů a jejich pohledů. Bylo také navrženo, které údaje musí informační soubor obsahovat, aby hráči měli všechny potřebné informace ke tvorbě strategií.

Nastudoval jsem si vhodné programovací jazyky pro tvorbu implementace vyhodnocovacího serveru, vizualizační utility a obslužných rutin. Pro tvorbu vyhodnocovacího serveru byl zvolen jazyk C++. Tento jazyk je vhodný pro možnost objektově orientovaného návrhu, a také proto, že se jedná o kompilovaný jazyk a umožňuje znovuvyužití opakujících se bloků kódu, které vznikly při implementaci vyhodnocovacích serverů předchozích ročníků soutěže v jazyce C. Pro tvorbu vizualizační utility byl zvolen jazyk C++ s využitím knihoven Qt a jazyka QML pro tvorbu grafického rozhraní. Jazyk C++ byl zvolen z důvodu možnosti použití kódu vytvořeného pro vyhodnocovací serveru. Jazyk QML byl zvolen z důvodu snadné tvorby aplikací s možností animací. Programovací jazyky Python a C zvolené pro tvorbu obslužných rutin, byly vybrány pro jejich časté používání soutěžícími v předchozích ročnících soutěže.

Z vytvořených návrhů jsem vycházel při tvorbě komunikačního protokolu a při implementaci vyhodnocovacího serveru, vizualizační utility a obslužných rutin. Nejzajímavější problémy, se kterými jsem se setkal při implementaci, jsem vyřešil a podrobně popsal.

Tuto bakalářskou práci by bylo možné rozšířit o další úkony, na které zde již nebyl prostor. Jedná se o rozsáhlejší testování, animace ve vizualizační utilitě, tvorbu obslužných rutin pro další programovací jazyky nebo o podporu vývoje strategií na OS Windows.

Vytvořené zadání soutěže a k němu připravené aplikace doporučuji využít, protože při tvorbě strategií studenti získají nebo rozšíří své znalosti z oblasti umělé inteligence, teorie her a algoritmů pro hledání nejkratší cesty. Studenti se také zdokonalí a procvičí v programování. Studenti mohou využít připravených obslužných rutin pro jazyky C/C++ a Python, s jejichž pomocí rychle vytvoří primitivní strategii. Tuto strategii mohou dále vylepšovat s využitím vizualizační utility, s jejíž pomocí snadno odhalí slabé stránky své strategie. Vytvořené zadání je hravé a soutěživé, proto pevně věřím, že bude studenty zaujme.

Literatura

[1] RedBot. RedBot [online]. 2013 [cit. 2013-04-30]. Dostupné z: <https://red-bot.rhcloud.com/>

[2] VIRIUS, Miroslav. Jazyky C a C: kompletní průvodce. 2., aktualiz. vyd. Praha: Grada, 2011, 367 s. Knihovna programátora (Grada). ISBN 978-80-247-3917-5.

[3] WESLEY, Addison. Python 3: Výukový kurz. 2009. ISBN 978-0321680563.

[4] Qt [online]. 2013 [cit. 2013-04-29]. Dostupné z: <http://qt-project.org/>

[5] KNOLL, Byron. Rock Paper Scissors Programming Competition [online]. 2013 [cit. 2013-04-30]. Dostupné z: <www.rpscontest.com/>

[6] MASSACHUSETTS INSTITUTE OF TECHNOLOGY. Battlecode [online]. 2013 [cit. 2013-04-30]. Dostupné z: <www.battlecode.org>

[7] AI Challenge [online]. 2013 [cit. 2013-04-30]. Dostupné z: <aichallenge.org>

[8] RoboRally [online]. 2009 [cit. 2013-04-30]. Dostupné z: <www.deskovehry.cz/index.php/RoboRally>

[9] QUIT CODING. Qt Quick Game Programming. 2010. Dostupné z: <http://quitcoding.com/download/Qt_Quick_Game_Programming_1_0.pdf>

[10] C++ Resource Network [online]. 2000-2013 [cit. 2013-05-02]. Dostupné z: <www.cplusplus.com>

Seznam příloh

Příloha 1

Obsah CD

- RedBotPaintBall – Zdrojové soubory vyhodnocovacího serveru.
  - game_view – Zdrojové soubory vizualizační utility.
  - strategieC – Zdrojové soubory obslužné rutiny pro jazyk C/C++.
  - strategiePy – Zdrojové soubory obslužné rutiny pro jazyk Python.
- Technicka_zprava – Technická zpráva ve formátu pdf a odt.
  - Fonty – Font Palatino Linotype použitý v technické zprávě .
  - Obrazky - Obrázky použité v technické zprávě.