Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava

RoboCup S - Nové stratégie

Implementácia navrhovaného riešenia

verzia 0.2

História zmien

V tab. 1 je história zmien, ktoré sa udiali v dokumente.

Tab. 1: História zmien dokumentu

Dátum zmeny	**Verzia dokumentu**	**Popis zmeny**	**Autor**
10.5.2008	0.1	Vytvorenie parciálnych častí	Všetci
14.5.2008	0.2	Formátovanie dokumentu, štandardizácia	Marek Koperdák
18.5.2008	0.3	Prehľad a úprava drobných nezrovnalostí dokumentu	Martin Petráš, Jan Kohút

Tím

Bc. Ladislav Borženský	*lborzensky zavináč g_m_ a_ i_l bodka com*
Bc. Peter Brtáň	*pbrtan zavináč g_m_ a_ i_l bodka com*
Bc. Ján Kohut	*jan.kohut84 zavináč g_m_ a_ i_l bodka com*
Bc. Marek Koperdák	*koperdak zavináč g_m_ a_ i_l bodka com*
Bc. Vladimír Janov	*vjanov zavináč g_m_ a_ i_l bodka com*
Bc. Martin Petráš	*petras.martin zavináč g_m_ a_ i_l bodka com*

Webová stránka tímu: *http://www2.dcs.elf.stuba.sk/TeamProject/2007/team16is-si/*

Tímový mailinglist: *jahodovi-princovia zavináč googlegroups bodka com*

Obsah dokumentu

1 Úvod

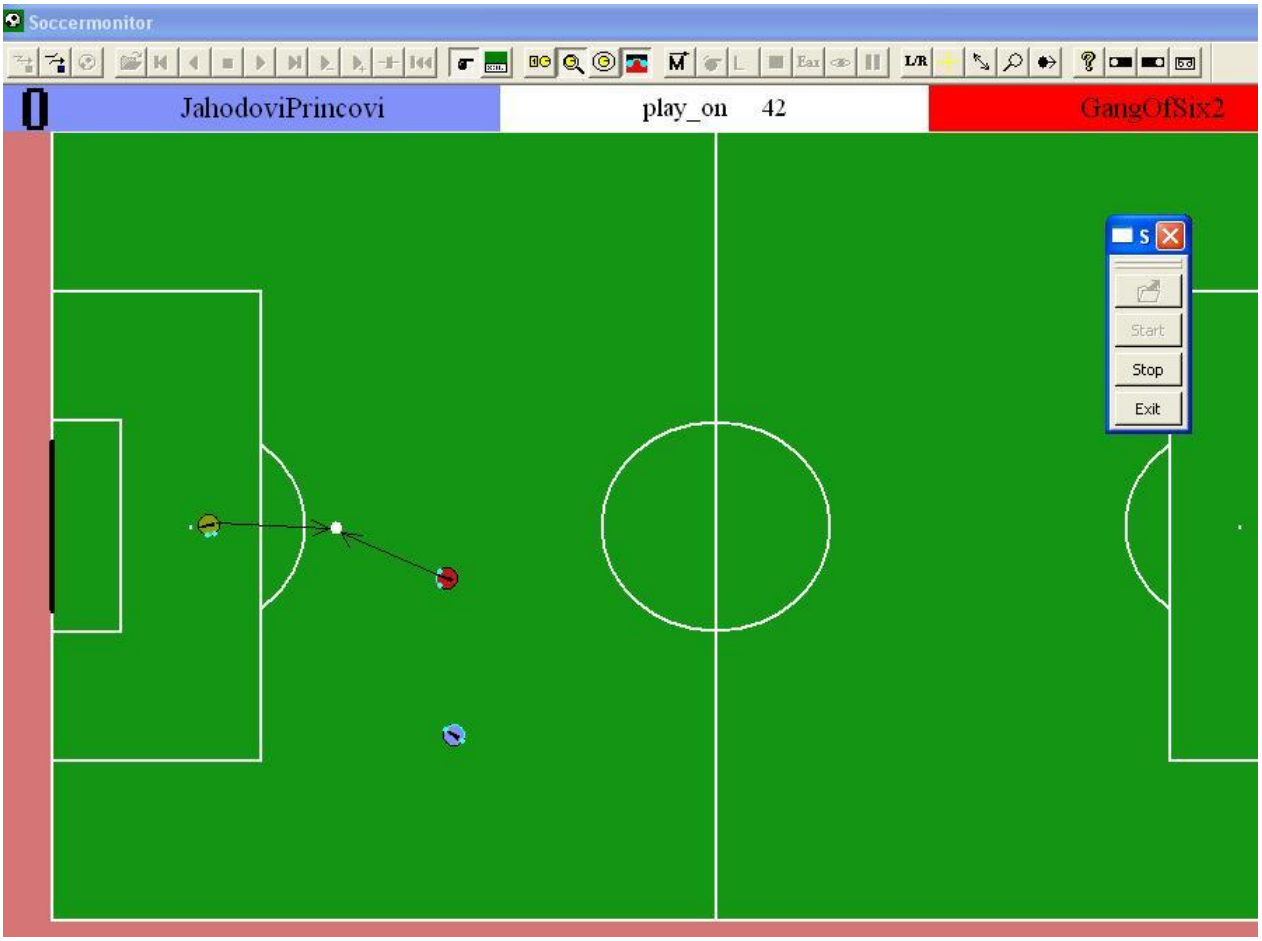
Tento dokument je výsledkom činnosti tímu č. 16 na predmete Tvorba softvérového systému v tíme. Dokument slúži ako správa o implementovaných riešeniach, ktoré boli navrhnuté v návrhu riešenia, resp. o zmenách, ku ktorých sme dospoli počas implemenotvania návrhu riešenia.

1.1 Prehľad dokumentu

Dokument je rozdelený do kapitol. Hneď v druhej kapitole dokumentu uvádzame zmenu oproti navrhovanému riešeniu vo výbehu brankára. Ďalšia kapitola sa tiež zaoberá brankárom a to jeho prihrávkami. Jedným z navrhnutých riešení bola zmena formácií hráčov počas zápasu, ktorá je rozpracovaná v kapitole č. 4. Veľkou výzvou, ktorú si tím č. 16 Jahodoví princovia dal, je Algoritmus eliminácie premenných. Kapitoly číslo 5, 6 a 7 sa venujú tejto problematike a spôsobu implementácie. Tím implementoval aplikáciu trénera, ktorá je určená k zdokonaľovaniu a sledovaniu zmien počas návrhu a implementácie hráča. Táto časť je opísaná v predposlednej časti dokumentu. Dokument uzatvára metodika pre debuggovanie hráča.

2 Zmena v navrhovanom riešení výbehu brankára

V prvej časti sme pojednávali o dvoch vylepšeniach pohybu brankára. Prvé z nich bolo zmenenie konštanty, ktoré malo mať za následok väčší výbeh brankára. Doposiaľ však nebolo možné overiť, či navrhovaná zmena konštanty bola tak efektívna, ako sme jej zmenou chceli dosiahnuť. Po použití trénera a jeho možností, ktoré ponúka na vytvorenie modelových hracích situácii, sme pripravili päť rôznych rozostavení hráčov a sledovali sme správanie sa brankára pri rôznych hodnotách konštanty. Po dlhom ladení sme zistili, že samotná zmena nie je prínosná, ale je pravým opakom. Po zväčšení hodnoty konštanty sa brankár začal správať z nevysvetliteľných príčin veľmi čudne. Po vybehnutí brankára za futbalovú šestnástku sa už vôbec nevracal a ako keby prestal sledovať loptu (viď Obr. 1). Preto sme dospeli k záveru, že táto zmena je neopodstatnená a je potrebné sa vrátiť k pôvodnému riešeniu problému.



Obr. 1 Ukážka prípadu, keď sa brankár rozhodne vybehnúť z bránkoviska

Druhé navrhované riešenie, ktoré sme v časti Prototyp nestihli implementovať je spojené s výsledkom prvého navrhovaného riešenia. Jeho výsledkom mal byť výber spôsobu vracania sa do futbalovej šestnástky po výbehu za ňu. Keď sme testovali pôvodne riešenie na tých istých modelových situáciách ako pred tým, tak sme zistili, že brankár vôbec nevybehuje za

futbalovú šestnástku. Preto aj tento návrh v nádväznosti na predchádzajúce zistenie sme sa rozhodli neimplementovať.

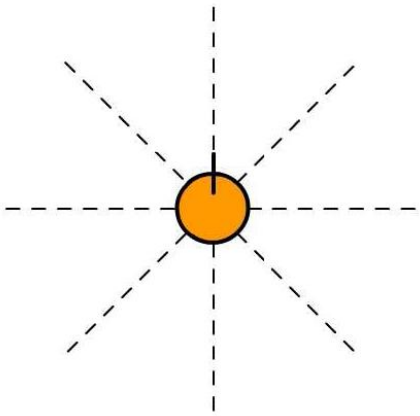
3 Príhrávky bránkára

3.1 Nenahrávať obsadeným hráčom

Po analýze algoritmu, ktorý tím Gang Of Six naimplementoval, sme si uvedomili, že má niekoľko nedostatkov. Často sa stávalo, že brankár pri rozohrávke nahral protihráčovi. Určili sme niekoľko dôvodov, prečo k takýmto situáciam dochádza. Jedným z problémov je, že pre každý lúč sa vyrátava fitness pre najlepšie postaveného spoluhráča a ďalší fitness pre najlepšie postaveného protihráča. Následne tieto dva hodnoty odčíta a výsledná hodnota predstavuje fitness celého lúča. V prípade, že všetci spoluhráči boli krytí protihráčmi, vybral si brankár jedného spoluhráča, ktorý síce mal najlepší fitness, ale v jeho blízkosti bol jeden alebo viac protihráčov. Treba si uvedomiť, že hráči nemajú kompletné informácie o dianí na ihrisku. Môže sa stať, že pri výpočte sú použité už staré informácie, a že sa poloha spoluhráča medzičasom zmenila. Takisto prihrávky (a strely vo všeobecnosti) nemusia byť presné tak, ako si to hráč zvolí, keďže server aj tu vkladá šum. Keď zoberieme do úvahy všetky tieto aspekty, je zrejmé, prečo v niektorých prípadoch brankár nahrá priamo súperovi. Riešenie tohto problému nie je jednoduché a zdá sa, že sa k ideálnym výsledkom môžme len priblížiť. V kóde sa často objavuje parameter *dConfThr*, čo je tzv. confidence threshold. Tento parameter sa používa pri iterácii nad množinou objektov (hráčov) vo modeli sveta (trieda WorldModel). Určuje pravdepodobnosť, s akou sa má hráč zahrnúť do množiny spoluhráčov alebo protihráčov. Ako som spomínal, informácie o hráčoch starnú v každom cykle, ak nie sú obnovované. Tento parameter určuje, ktoré informácie (a ktorých hráčov) má zahrnúť do výpočtu. Tím Gang Of Six nastavil pri prihrávkach tento parameter na 0,9. Zmenili sme túto hodnotu pri prihrávkach brankára na vyššiu. Týmto sme zabezpečili, že hráč nahráva len tým hráčom, o ktorých má najčerstvejšie infomácie. V ďalšom kroku sme zväčšili fitness protihráčov v pomere ku fitness spoluhráčov. Tým sme zabezpečili, že tí spoluhráči, pri ktorých bolo viac protihráčov, neboli vybraní pri prihrávaní. Tiež sme odhalili chybu v implementácii v rozdelení lúčov v priestore okolo hráča. Túto chybu sme opravili.

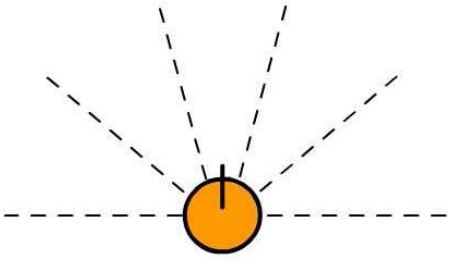
3.2 Prihrávky brankára

Tím Gang of Six používal jeden algoritmus prihrávania pre každého hráča tímu. Teda aj brankár aj hráči v poli používali rovnaký algoritmus vyrátavania najlepšej prihrávky. Hráč pri rozhodovaní sa, komu prihrať, si rozdelí priestor na ihrisku do niekoľkých lúčov. Všetky lúče spolu musia tvoriť samozrejme celý priestor okolo hráča. Situácia, keď si hráč rozdelí priestor okolo seba do lúčov je znázornená na Obr. 2.



Obr. 2 Lúče hráča pri prihrávaní

Hráč na obrázku si rozdelil hraciu plochu okolo seba na 8 úsekov. Následne podľa algoritmu opísaného v predchádzajúcej kapitole si hráč vyberie priestor kam kopne loptu. Rovnaký algoritmus používa aj brankár. Tento prístup však nie je vhodný, lebo brankár nemôže prihrávať smerom dozadu. Zbytočne si tak rozdelil priestor na úseky aj za sebou, keďže tento priestor vôbec neprichádza do úvahy. Túto časť sme vylepšili takým spôsobom, že brankár použije rovnaké množstvo lúčov ako ostatní hráči, ale len na rozdelenie priestoru pred sebou. Priestor brankára sa teda rozdelí na menšie úseky a brankár sa tak môže lepšie rozhodnúť kam prihrá. Na Obr. 3 je zobrazená rovnaká situácia ako na predchádzajúcom obrázku, ale teraz je zobrazeným hráčom brankár mužstva.



Obr. 3 Lúče brankára pri prihrávaní

Ako je možne vidieť, brankár už zbytočne nevyhodnocuje aj priestor za sebou. Naopak lúče, ktoré takto ušetril používa na detailnejšie rozdelenie priestoru pred sebou. Výhody tohto prístupu sú v tom, že brankár môže teraz kvalitnejšie vyhodnocovať situáciu pred sebou. Prihrávky brankára sú po vykonaní tejto zmeny skutočne úspešnejšie.

3.3 Výsledky našej implementácie

Podarilo sa nám viac vyladiť rozohrávanie brankára. V prípade, že všetci spoluhráči sú krytí protihráčmi, nahrával náš vylepšený brankár presnejšie a bezpečnejšie. Aj napriek tomu sa však stávalo, že brankár niekedy nahral priamo protihráčovi, pri ktorom nebol v blízkosti žiaden náš

hráč. Kvôli komplikovanosti výpočtu fitness, aký zvolil tím Gang Of Six, sa nám nepodarilo odchaliť príčinu týchto chýb. Gang Of Six totiž svoju prvotnú implementáciu refaktorizoval a prispôsobil na použitie v koordinačných grafoch. Zaviedol osekávanie fitness na špecifický interval <0, 12>. Je možné, že kvôli týmto zmenám sa nahrávanie zhoršilo resp. vniesli tým do kódu nejakú chybu.

4 Zmena formácií počas hry

4.1 Doterajšia logika

Doterajšia logika zmien formácii bola implementovaná prakticky v dvoch základných triedach – Player.cpp a PlayerTeams.cpp.

4.1.1 Trieda Player

V tejto triede, v hlavnej metóde *Loop*, ktorá sa vykonáva každý cyklus, sa na začiatku každého vykoná identifikácia typu hráča vo formácií a na základe tohto typu sa zavolá príslušná metóda v triede *PlayerTeams*. Je potrebné povedať, že okrem brankára má všetkých zvyšných 10 hráčov rovnakú metódu, v ktorej sú definované konkrétne akcie.

Abstrahovaním od konkrétneho kódu by sa v pseudojazyku dali opísať jednotlivé funkcie a postupnosť pri ich volaní asi nasledovne:

4.1.2 Trieda Player

```
aktualna_ formacia->zistiTypHraca();
      Ak hrac == brankar
            goalieManLoop();
      Ak hrac == stredopoliar
            midfielderMainLoop();
      ...
```

goalieMainLoop resp. *midfielderMainLoop* volajú metódu *deMeer5*, ktorá je implementovaná v triede *PlayerTeams*.

4.1.3 Trieda PlayerTeams

Metóda deMeer5

Ako už bolo spomenuté vyššie, metóda deMeer5 resp. deMeer5Goalie je kriticky dôležitá pre každého hráča, keďže obsahuje príkazy pre server pre vykonanie konkrétnych akcií. V samostnej metóde sa pred výkopom lopty, teda pred začiatkom zápasu nastaví počiatočná formácia pod názvom *FT_INITIAL*. Jej konfigurácia je spolu s ďalšími šiestimi typmi formáciami uložená v súbore *formations.conf*, ktorý sa nachádza v priečinku */build*. (viď Obr. 4). Zhora nadol je v každom riadku uložený údaj pre konkrétnu vlastnosť hráča (X-ová pozícia, Y-ová pozícia, typ hráča, X-ový atribút, Y-ový atribút atď.)

```
# Formation 1 = FT_initial = 111
1
-50.0 -16.0 -17.0 -17.0 -16.0  -8.0  -5.0  -5.0  -2.0  -1.0  -1.0  # X_pos
  0.0  16.0   5.0  -5.0 -16.0   0.0  10.0 -10.0   0.0  22.0 -22.0  # Y_pos
    1     4     2     2     4     5     6     6     8     7     7  # P_type
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0                # X_attr
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0                # Y_attr
    0     0     0     0     0     0     0     0     0                # Behind_ball
-49.0 -45.0   -45.0   -45.0   -45.0   -45.0   -45.0 -45.0 -45.0  # X_min
  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0                # X_max
```

Obr. 4 Príklad konfigurácie formácie s názvom *FT_initial*

Po výkope, teda po začiatku zápasu, sa formácia zmení na 433, čo znamená 4 obrancovia, 3 stredopoliari a 3 útočníci.

4.2 Implementácia zmeny formácii počas zápasu

Cieľom zmeny formácii počas zápasu je adekvátnejšie a pružnejšie reagovať na zmeny situácií. Na začiatku však bolo potrebné určiť, za akých situácii sa budú formácie meniť.

Ako prvú možnosť bola zmena na základe únavy hráča, teda pomocou sledovanie hodnoty *stamina*. Vďaka prostriedku *Logalyzer* sa nám však podarilo zistiť, že hodnota výdrže hráča (*stamina*) sa počas zápasu permanentne sama doplní, po vydaní určitého výkonu a vráti sa na pôvodnú hodnotu, čo v praxi znamená, že hráči sa takmer nikdy neunavia, resp. únavu pociťujú len v zanedbateľnom časovom intervale.

Ďalej sme takisto rozmýšlali nad možnosťou zmeny formácie vzhľadom na pozíciu lopty na ihrisku. Ak by bola lopta na našej strane, teda súper by bol v útoku, posilnila by sa obrana, a naopak, ak budeme na súperovej polovici, posilní sa útok. Táto stratégia by sa realizovala prebehovaním stredového hráča podľa konkrétnej sistuácie na ihrisku. Po odsimulovaní dostatočne veľkého počtu zápasov s rôznymi tímami sme usúdili, že takáto možnosť je najlepšou voľbou.

Konkrétna implementácia prebiehala v triede *PlayerTeams.cpp.* Na začiatku sme zisťovali pozíciu lopty a podľa nej sa nastavila požadovaná formácia – buď útočná alebo obranná.

Pre takmer neexistujúcu možnosť debuggovania bolo potrebné najskôr vytvoriť debuggovacie prostredie, čo sa aj neskôr podarilo a jeho popis je uvedený v inej kapitole. Aj napriek vytvoreniu tohto prostriedku nám vzhľadom na pomerne veľkú štruktúrovanosť a komplexnosť kódu trvalo hľadanie miesta, kde je možné daný kód implementovať, dlhšie ako sme čakali.

Samotná implementácia bola vykonaná v triede *PlayerTeams* v spomínanej metóde *deMeer5().*

```
ak je x-ová súradnica lopty > 0//lopta je na súperovej polovici
     nastav_formaciu (334_ofenzivna)
inak nastav_formaciu (433_defenzivna)
```

5 Algoritmus eliminácie premenných (APE)

Snahou nášho tímu bolo dopracovať myšlienku, ale hlavne samotnú implementáciu, náročného algoritmu eliminácie premenných, ktorý sa používa na riešenie lokálnych koordinačných problémov. Cieľom algoritmu je dospieť do stavu, kedy si agenti v danom cykle zvolia také akcie, ktoré maximalizujú globálnu výnosovú funkciu pre daný graf a daný cyklus.

V prvotnej fáze sa analyzoval aktuálny stav s popisovaným stavom v priloženej dokumentácií. Po veľmi krátkej dobe bolo zistené, že samotný APE nebol naimplementovaný do takej miery, ako bolo uvádzané a popisované. Následne sa analyzovala samotná myšlienka APE a vyvodili sa maličké zmeny v samotnom návrhu algoritmu a následne nasledovala jeho implementácia.

5.1 Cieľ APE

Samotným cieľ algoritmu eliminácie premenných nebol zmenený. Naďalej hlavnou myšlienkou ostáva vrátiť spoločnú akciu pre tím, ktorá bude maximalizovať spoločnú výnosovú funkciu. Zároveň privedie tím do požadovaného stavu - víťazstva.

5.2 Princíp a práca APE

1. Na začiatku sa prechádza pole všetkých agentov a prideľujú sa im role na základe, ktorých budú následne vystupovať v samotnom APE.
2. Následne sa vyberie prvý náhodný agent určený na elimináciu z množiny všetkých našich agentov. Na základe roly, ktorá mu bola v prvom kroku algoritmu pridelená, sa mu priradia akcie, ktoré je agent v danom cykle schopný vykonať.
3. V nasledujúcom kroku sa prechádzajú ostatný agenti za účelom vytvorenia množiny „atraktívnych“ agentov. Ide o vytvorenie skupiny agentov, s ktorými môže nami eliminovaný agent vytvoriť nejakú akciu.
4. Jednotlivým „atraktívnym“ agentom na základe ich roly sú priradené akcie, ktoré môžu vykonať.
5. Nastáva optimalizácia rozhodnutia vykonania akcie eliminovaného agenta. Vytvárajú sa všetky možné kombinácie akcií eliminovaného agenta s akciami „atraktívnych“ agentov. Všetky tieto kombinácie akcií sú ohodnocované výnosovými funkciami. Vracajú hodnotu (výnos) danej kombinácie. Vyberie sa akcia s najvyšším výnosom.
6. Skontroluje sa hodnota výnosu s aktuálnym výnosom. Ak je vyšší, čo znamená, že ho maximalizuje, tak je daný výnos prepísaný novou hodnotou a daná akcia sa priebežne uloží, ako výsledná akcia, ktorá sa má vykonať.
7. Agent je eliminovaný z grafu.

8. Proces algoritmu eliminácie náhodnej premennej pokračuje bodom 1 dovtedy, pokiaľ nie sú všetci agenti z grafu eliminovaní. Ak sa dostaneme do stavu, že v grafe sa nebudú nachádzať ďalší agenti na elimináciu, APE bude pokračovať bodom 9.
9. Po eliminácií posledného agenta z grafu dostávame najvýhodnejšiu spoločnú akciu pre aktuálny cyklus, ktorý maximalizuje celkový výnos.

5.3 Pseudokód algoritmu eliminácie premenných

V tejto časti sa nachádza pseudokód nami upraveného a následne implemetovaného APE. Zároveň v sebe zachytáva myšlienku samotného princípu a funkčnosti, ktorý bol vyššie popísaný a pomerne dosť dôsledne vysvetlený. Je veľmi dôležité pochopiť samotnú myšlienku algoritmu, aby bolo možne rozumieť nižšie uvádzanému ukážkovému príkladu.

Algoritmus eliminácie premenných APE:

```
for each agent
     giveRoleForAgents()
for each agent
     giveAllPossibleActionsForEliminateAgents()
                 for other agents
                      if otherAgents is accepted
               giveAllPossibleActionForOtherAgent()
                      createActionsCombinations()
               giveRatingForCombinations()
                    if is new Rating better
                         changeRating()
                         saveNewFinalAction()
                    endif
          endif
        endfor
endfor
```

Vysvetlenie základných pojmov zo pseudokódu APE:

- *giveRoleForAgents()* – priradenie všetkým agentom roly, ktoré sú pre APE veľmi dôležité
- *giveAllPossibleActionForEliminateAgents()* – eliminovanému agentovi je vytvorená množina akcií, ktoré vďaka svojej roly môže vykonať
- *if otherAgents is accepte* – vytvorenie množiny agentov, s ktorými môže eliminovaný agent vytvárať akcie
- *giveAllPossibleActionForOtherAgent()* – agentovi, s ktorým náš eliminovaný agent môže spolupracovať, je vytvorená množina akcií, ktoré vďaka svojej roly môže vykonať
- *createActionsCombinations()* – vytvárajú sa kombinácie medzi akciami eliminovaného agenta a spolupracujúceho agenta

- *giveRatingForCombinations()* – následné sa tieto kombinácie spoločných akcií ohodnotia výnosovými funkciami
- *changeRating()* – ak je nová hodnota výnosovej funkcie pre danú akciu vyššia, uloží sa
- *saveNewFinalAction()* – zároveň sa uloží aj priebežne výsledná akcia

6 Pomocné funkcie algoritmu eliminácie premenných

Tím GoS čiastočne rozpracovali APE a zanechali v zdrojových súboroch aj dokumentácií skeletóny pomocných funkcií k APE. My sme sa snažili implementovať telá týchto funkcií a doplnili sme im funkcionalitu.

6.1 Funkcia isPassBlockedCG

Player::isPassBlockedCG(int playerIndex, int teammateIndex, DirCG passDirection)

Táto funkcia vykonáva tieto činnosti:

- Získa si objekty nášho hráča a spoluhráča, a taktiež ich súradnice.
- Postupne v cykle prechádza všetkých protihráčov .
- Pre nich si vyrátame ich vzdialenosť od nášho hráča.
- Ak je vzdialenosť protihráča menšia ako vzdialenosť nášho spoluhráča, vypočítame, či je protihráč schopný zasiahnuť loptu a tým prerušiť našu prihrávku spoluhráčovi.

Na výpočet toho, či je protihráč schopný prerušiť našu prihrávku, sledujeme vzdialenosť protihráča od spojnice nášho hráča so spoluhráčom.

6.2 Funkcia isEmptySpace

Player::isEmptySpace(int playerIndex, DirCG direction)

Táto funkcia testuje, či je voľné miesto pred hráčom, t.j. či hráč môže driblovať s loptou, alebo môže viesť loptu až k bráne. Ako geometrický útvar pre vymedzenie voľného priestoru pred hráčom sme zvolili kvázy kužeľ (*cone*)

V tele tejto funkcie sme preťažili existujúcu funkciu *isFreeCone (double dAngle, double dLength)*, ktorá vyrátavala voľný priestor pred loptou, nie pred hráčom. Funkciu nevoláme, ale sme ju modifikovali a implementovali priamo v našej metóde. Modifikácia spočívala v zmene výpočtu voľného miesta pre hráča, nie pre loptu, takže algoritmus bol nasledovný:

- Na základe čísla hráča, ktorý je parametrom funkcie, vráť objekt hráča
- Zisti pozíciu hráča a podľa toho mu nastav dĺžku kužeľa (ak je na súperovej polovici tak nech je valec menší ako keď je na svojej)
- Zisti počet protihráčov v danom kuželi
- Ak je počet rovný 0, čo znamená, že hráč má pred sebou voľné miesto, vráť *true*, inak vráť *false*.

6.3 Funkcia uCG

Player::uCG(int teammateIndex, DirCG passDirection)

Úlohou tejto funkcie je vrátiť reálne číslo v rozmedzí 5 až 7, ktoré bude hodnotiť výhodnosť prihrávky pre jednotlivých spoluhráčov. Hodnoty blížiace sa k číslu 7 budú znamenať

prihrávku, ktorá má najvyššiu hodnotu pre daný tím, hodnota blížiaca sa k číslu 5 znamená najmenej výhodnú prihrávku spoluhráčovi pre celý tím.

Pri výpočte tejto fit funkcie, t.j. ak prihráme loptu spoluhráčovi, berieme z jeho pohľadu do úvahy tri veci:

- Vzdialenosti a rozmiestnenie protihráčov.
- Vzdialenosti a rozmiestnenie spoluhráčov.
- Umiestnenie hráča na ihrisku. Hráč, ktorý je bližšie pri súperovej bráne má vyššie hodnotenie.

Vzdialenosti a rozmiestnenie protihráčov rátame nasledujúcim spôsobom. Hráč, ktorý je najbližšie, má najväčšie ohodnotenie, s rastúcou vzdialenosťou klesá hodnotenie. Hodnotenie je nepriamo úmerné vzdialenosti súperov od nášho spoluhráča. Výsledná hodnota pre protihráčov sa získa sčítaním všetkých takýchto čiastkových výsledkov.

Podobné rátame vzdialenosti našich spoluhráčov od hráča ktorému ideme prihrať loptu. Tu taktiež sčítame čiastkové výsledky.

Výslednú hodnotu tejto fitnes funkcie vypočítame ako súčet:

- Záporná hodnota fitnes protihráčov.
- Fitnes spoluhráčov.
- Umiestnenie spoluhráčov na ihrisku.

7 Výnosové funkcie

V rámci algoritmu eliminácie premenných, bolo použitých 9 výnosových funkcií, ktorými je reprezentovaná celková stratégia tímu. Tieto výnosové funkcie reprezentuje herné situácie, ktoré uvažuje algoritmus eliminácie premenných. V tejto kapitole je postupne opísaných všetkých deväť výnosových funkcií. Ku každej funkcií je určené akú hernú situáciu reprezentuje, jej zápis vo forme logického výrazu, pridelený výnos a obrázok znázorňujúci danú funkciu.

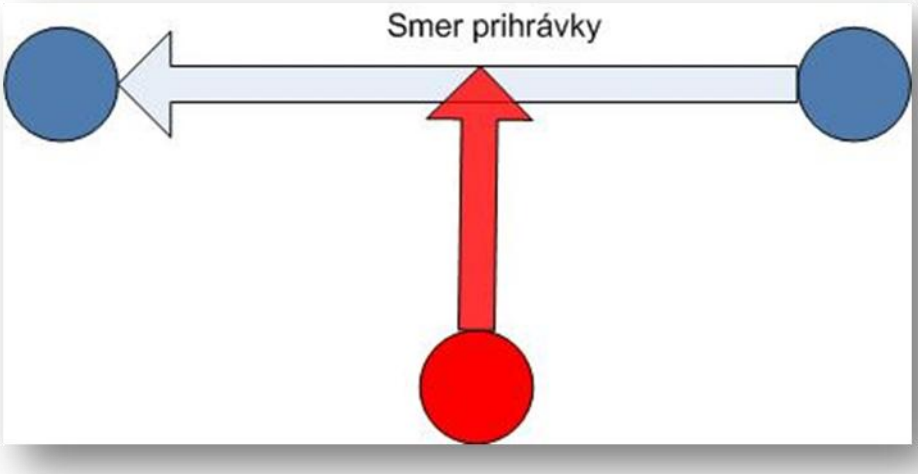
7.1 Výnosová funkcia č. 1 : Zastavenie súperovej akcie

Táto výnosová funkcia je v svojej podstate veľmi jednoduchá a reprezentuje hernú situáciu, kedy jeden z hráčov tímu dokáže zastaviť súperovu akciu. Spomínaný hráč teda dokáže úspešne vykonať príkaz – intercept.

Tejto hernej situácii je pridelený výnos 10 bodov. Táto herná situácia ma spolu so situáciou streľba na bránu priradený najvyšší výnos. Logický výraz pre túto výnosovú funkciu vyzerá nasledovne:

$$\langle p_1^{interc.} \quad ; \quad \text{intercept} : 10 \rangle$$

Na Obr. 5 je znázornený príklad situácie, v ktorej hráč nášho tímu (červený hráč), dokáže zastaviť súperovu akciu. V uvedenom prípade môže náš hráč zastaviť prihrávku medzi súperovými hráčmi.



Obr. 5 Výnosová funkcia č. 1

7.2 Výnosová funkcia č. 2 : Prihrávka spoluhráčovi

Ako už z názvu vyplýva táto výnosová funkcia opisuje hernú situáciu, v ktorej sa náš hráč snaží prihrať spoluhráčovi. V tomto prípade je potrebne overiť niekoľko predpokladov. Prvým je skutočnosť, čí hráč, ktorému chceme prihrať ma pridelenú rolu receiver. V opačnom

prípade by totiž tento hráč nedokázal spracovať prihrávku. Ďalší predpoklad, ktorý sa testuje je skutočnosť, čí je priestor medzi danými dvomi hráčmi voľný. Ak je v priestore medzi hráčmi člen súperovho tímu prihrávka sa nevykoná. Ak obidva predpoklady boli splnene prvý hráč prihrá druhému hráčovi v určitom smere a ten sa presunie na zamýšľané miesto prihrávky.

Výnos, ktorý sa pridelí takejto hernej situácii určuje pomocná funkcia uCG. Tá určí pre každú prihrávku jej výnos v rozmedzí 5 až 7 bodov. Logickým výrazom je možne túto hernú situáciu opísať nasledovne:

$$\langle p_2^{passer} \quad ; \quad \text{has-role-receiver}(j) \quad \wedge \\ \neg\text{isPassBlocked}(i, j, dir) \quad \wedge \\ a_i = \text{passTo}(j, dir) \quad \wedge \\ a_j = \text{moveTo}(dir) : u(j, dir) \in [5, 7] \rangle \ \forall j \neq i$$

Grafický je výnosová funkcia č. 2 znázornená na Obr. 6. Na tomto obrázku sú znázornený dvaja hráči, ktorý sú uvažovaný v tejto výnosovej funkcií. Na obrázku je taktiež vidieť aj zónu, ktorá sa využíva na testovanie, čí je priestor medzi dvojicou hráčov voľný.

Obr. 6 Výnosová funkcia č. 2

7.3 Výnosová funkcia č. 3 : Driblovanie hráča

Výnosová funkcia č.3 reprezentuje hernú situáciu, v ktorej sa hráč rozhodne driblovať s loptou určitým smerom. Pri tejto výnosovej funkcií sa testuje len jeden predpoklad a to čí je priestor v ktorom sa hráč rozhodol driblovať voľný. Ak je tento predpoklad splnený bude takejto hernej situácii pridelený výnos 2 body. Tento výnos je pomerne nízky, ale odráža fakt, že driblovanie jedného hráča je menej výhodne pre tím ako prihrávka, alebo streľba na bránu. Logický výraz pre túto hernú situáciu je uvedený na tomto mieste :

$$\langle p_3^{passer} \quad ; \quad \text{is-empty-space}(i, \text{n}) \quad \wedge \\ a_i = \text{dribble}(\text{n}) : 2 \rangle$$

Na obrázku číslo 3 je znázornená situácia, v ktorej ma hráč záujem driblovať určitým smerom. Na obrázku je taktiež vidieť zónu, ktorá sa používa na testovanie, čí je priestor pre driblovanie voľný.

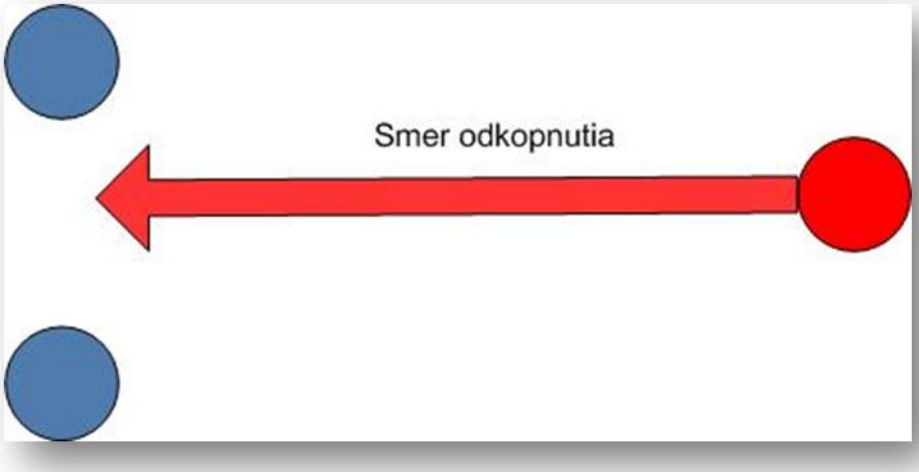
Obr. 7 Výnosová funkcia č. 3

7.4 Výnosová funkcia č. 4 : Odkopnutie lopty

Táto výnosová funkcia podobne ako výnosová funkcia č.1 je pomerne jednoduchá a nekontroluje žiadne predpoklady na jej vykonanie. Funkcia reprezentuje situáciu, pri ktorej sa hráč rozhodne odkopnúť loptu smerom na súperovu bránu k súperovým obrancom. Táto situácia predstavuje východisko z núdze ak nieje možne vykonať žiadnu z ostatných herných situácií. Tomuto faktu zodpovedá aj výnos funkcie, ktorého hodnota je 0.1 bodu a je najmenším zo všetkých výnosov. Logický zápis tejto hernej situácie je taktiež jednoduchý :

$$\langle p_4^{passer} \quad ; \quad a_i = \text{clearBall} : 0.1 \rangle$$

Situáciu, ktorú reprezentuje táto funkcia je možne vidieť aj na Obr. 8. Na tomto obrázku hráč odkopne loptu smerom medzi súperových obrancov, keďže nemôže vykonať žiadnu inú akciu.



Obr. 8 Výnosová funkcia č. 4

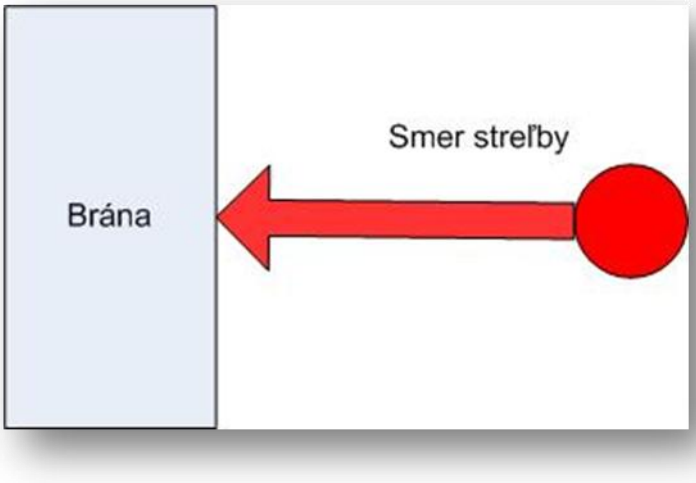
7.5 Výnosová funkcia č. 5 : Streľba na bránu

Výnosová funkcia č.5 predstavuje hernú situáciu, ktorá ma spolu s prvou hernou situáciou najväčší prínos pre tím. Reprezentuje situáciu, v ktorej hráč môže strieľať na bránu. Funkcia

kontroluje dva predpoklady: je hráč pred bránou? a môže hráč kopnúť do lopty?. Tieto predpoklady sú pomerne jednoznačne a zrozumiteľne, takže ich nieje potrebne bližšie opisovať. Keďže táto herná situácia je jednou z najužitočnejších aj jej výnos je vysoký – 10 bodov. Logický opis tejto výnosovej funkcie vyzerá nasledovne:

$$\langle p_5^{passer} \quad ; \quad \text{is-in-front-of-goal}(i) \quad \wedge \\ \text{is-ball-kickable}(i) \quad \wedge \\ a_i = \text{score} : 10 \rangle$$

Opisovaná herná situácia je znázornená aj na Obr. 9. Na tomto obrázku je vidieť hráča, ktorý je pred bránou a zároveň môže kopnúť do lopty. Splnene sú teda obidva predpoklady a hráč môže vystreliť na bránu.



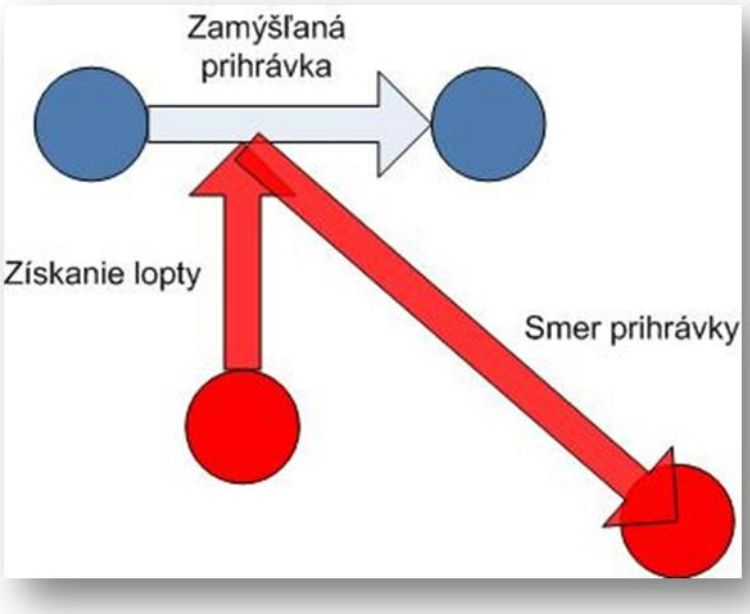
Obr. 9 Výnosová funkcia č. 5

7.6 Výnosová funkcia č. 6 : Prihrávka po získaní lopty

Výnosová funkcia č. 6 je jednou z komplikovanejších funkcií opisujúca vnímanie situácie z pohľadu hráča, ktorý je schopný prijať prihrávku. Funkcia reprezentuje situáciu, v ktorej spoluhráč zastavením akcie súpera získa loptu a po získaní lopty ju prihrá danému hráčovi. Táto funkcia ma dva predpoklady, ktoré kontrolujú, či spoluhráč ma pridelenú rolu interceptor (môže narušiť súperovu akciu) a či je priestor medzi týmito dvoma hráčmi voľný. Ak sú tieto predpoklady splnene, spoluhráč sa pokúsi prerušiť akciu súpera. Následne získanú loptu prihrá spoluhráčovi. Podobe ako pri každej hernej situácii súvisiacej s prihrávkami aj pri tejto je možný výnos v rozmedzí 5 až 7 bodov. O konečnej hodnote výnosu rozhoduje funkcia uCG. Logický je možne túto situáciu zapísať nasledovne:

$$\langle p_6^{receiver} \quad ; \quad \text{has-role-interceptor}(j) \quad \wedge$$
$$\neg\text{isPassBlocked}(j, i, dir) \quad \wedge$$
$$a_j = \text{intercept} \quad \wedge$$
$$a_i = \text{moveTo}(dir) : u(i, dir) \in [5, 7]\rangle \quad \forall j \neq i$$

Opisovaná herná situácia je zobrazená aj na Obr. 10. Na tomto obrázku sa červený hráč najprv snaží získať loptu zastavením súperovej akcie. Po prípadnom získaní lopty ju prihrá spoluhráčovi.



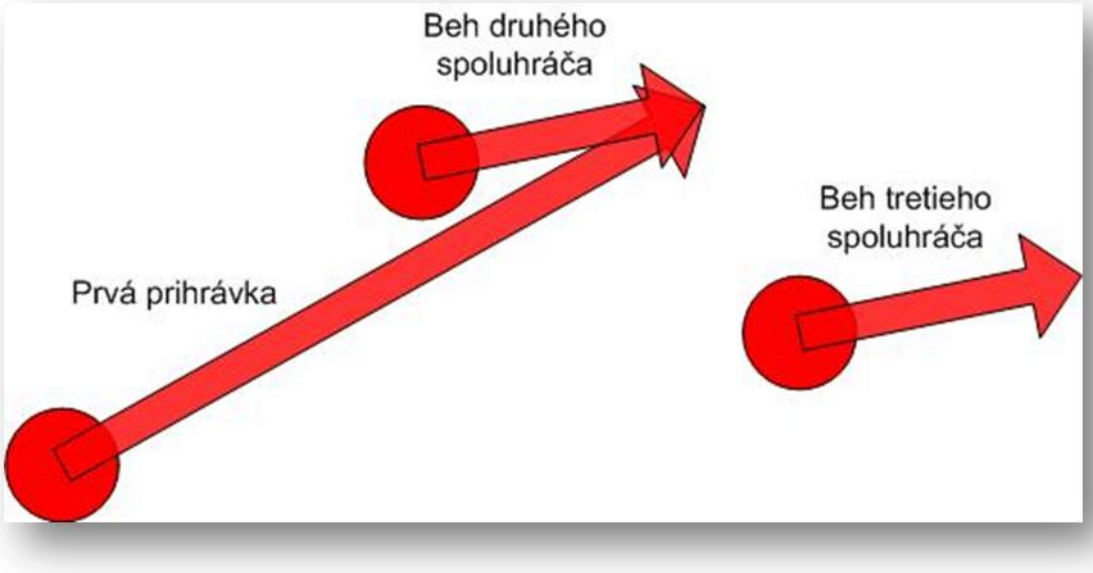
Obr. 10 Výnosová funkcia č. 6

7.7 Výnosová funkcia č. 7 : Dvojnásobná prihrávka

Táto funkcia opisuje hernú situáciu, v ktorej hráč prihrá spoluhráčovi a ten následne prihrá ďalšiemu spoluhráčovi. Dvaja hráči, medzi ktorými prebehne druha prihrávka samozrejme musia bežať rovnakým smerom. Táto funkcia kontroluje 2 predpoklady. Prvý predpoklad hovorí, či ma druhý spoluhráč pridelenú rolu receiver a či je priestor medzi prvými dvomi hráčmi voľný. Ak sú tieto podmienky splnene prvý hráč prihrá druhému spoluhráčovi a ten spolu s tretím spoluhráčom bežia rovnakým smerom. Túto situáciu je možne opísať logickým výrazom nasledovne :

$$\langle p_7^{receiver} \quad ; \quad \text{has-role-receiver}(k) \quad \wedge$$
$$\neg\text{isPassBlocked}(k, i, dir) \quad \wedge$$
$$a_j = \text{passTo}(k, dir2) \quad \wedge$$
$$a_k = \text{moveTo}(dir2) \quad \wedge$$
$$a_i = \text{moveTo}(dir) : u(i, dir) \in [5, 7] \rangle \ \forall j, k \neq i$$

Hernú situáciu č.7 je možne vidieť znázornenú aj na Obr. 11.



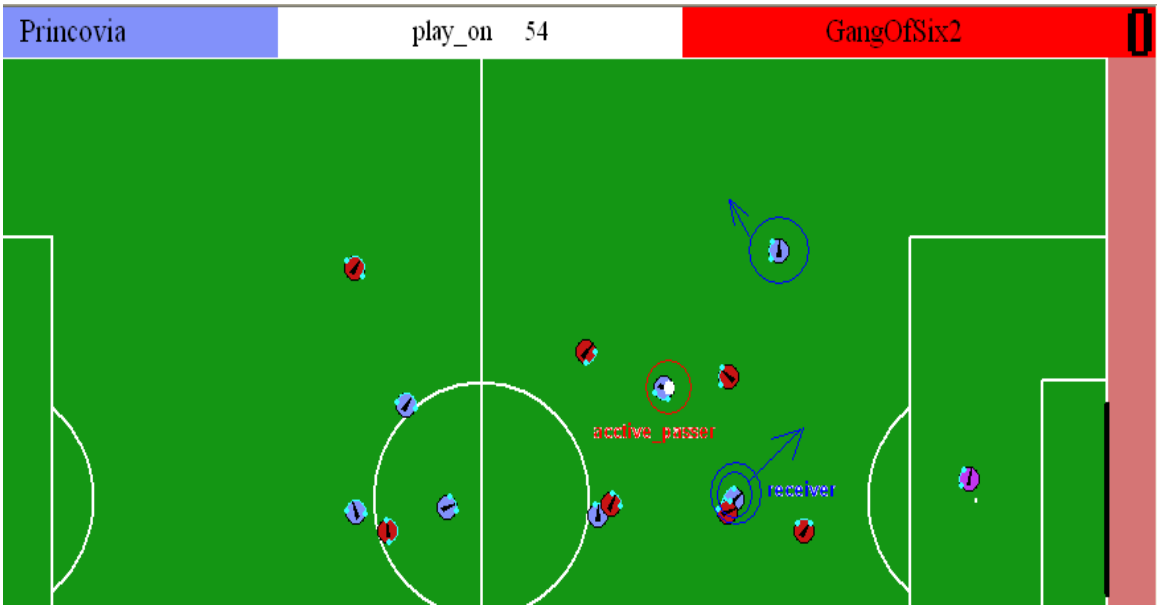
Obr. 11 Výnosová funkcia č. 7

7.8 Výnosová funkcia č. 8 a č. 9 : Presun na strategickú pozíciu

Tieto výnosové funkcie opisuje situáciu, keď sa hráč presúva na strategickú pozíciu, ktorá mu je priradená. Rozdiel medzi týmito funkciami je len v tom, že funkcia č. 8 sa aplikuje na rolu receiver a funkcia č. 9 sa aplikuje na rolu passive.

7.9 Príklad algoritmu eliminácie premenných (APE)

Nasledujúce riadky sú venované ukážke funkčnosti APE algoritmu na konkrétnom príklade. Obr. 12 zobrazuje situáciu potreby koordinácie agentov. Na príklade si nasimulujeme logiku navrhnutého algoritmu.



Obr. 12 Príklad APE

APE v okamihu zachytenom na obrázku:

1. Pridelia sa všetkým agentom roly
2. Vyberie sa agent na elimináciu v danom okamihu. Na lepšie nasimulovanie našej situácie sa vybral agent, ktorý sa nachádza pri lopte a je označený ◯ .
3. Tomuto agentovi bola pridelená rola *acttive_passer* (znamená to, že hráč môže vykonať akciu s loptou, môže do nej kopnúť) s príslušnými operáciami, ktoré môže v danom okamihu vykonať a to sú:
    a. leadding_pass_to – nebežnú prihrávku
    b. dribble – utekať s loptou smerom na bránu
    c. clear_ball – odkopnúť loptu
    d. score – strieľať na bránu
4. Následne sa začnú prechádzať všetci „spolu“ agenti. Vyberajú sa len agenti, s ktorými eliminovaný agent susedí a ma možnosť vytvoriť spoločnú akciu. Z nášho obrázku boli vybratí dvaja agenti označení ◯
5. Postupne sa prechádzajú títo agenti (určení na spoluprácu) a sú im priradené akcie, ktoré môžu vykonať na základe prislúchajúcej roly. V našom príklade si zvolíme hráča s označením ◎, ktorý je v roly *receiver*. Z toho vyplýva, že môže vykonať tieto akcie:
    a. move_to – presuň sa na určenú pozíciu
    b. move_to_strategy_posititon – presuň sa na domovskú pozíciu

6. V tomto bode sa začne s vyhodnocovaním akcií, ktoré môže vykonať eliminovaný hráč a akcií, ktoré môže vykonať v kombinácií s akciami poskytujúcimi s pomocným agent. Tieto akcie sú **ohodnocované výnosovými funkciami.** Sú predefinované ohodnocovanie výnosové funkcie, ktoré vznikli na základe rôznych možných kombinácií akcií agentov. Spomínané vyhodnocovanie funkcie dôsledne kontrolujú možne stavy pre dané akcie. V našom prípade budú ohodnotených 9 možných akcií a z nich bude vybratá tá, ktorá bude mať najvyšší výnos. Zoznam ohodnotených akcií výnosovými hodnotami pre daný príklad je:

    - Dribble
    - Score
    - Clear_ball
    - Leading_pass_to — Move_to

   V poslednom prípade ide o kombináciu možnej spolupráce eliminovaného agenta s vybraným agentom. V tomto prípade si treba uvedomiť, že treba brať do úvahy 8 možných smerov. Preto je možných akcií na ohodnotenie 11.

7. Následne bude vybraná najvýhodnejšia a najvýnosnejšia akcia eliminovaného agenta v danom okamihu s nami vybraným agentom. Následne sa vyberá ďalší možný agent. Postupne sa prejdú všetci agenti, s ktorými môže eliminovaný agent spolupracovať. Vyberie sa pre eliminovaného agenta najužitočnejšia a najvýnosnejšia akcia. Následne sa agent eliminuje z grafu.

Postupne by sme eliminovali všetkých agentov z grafu a dostali sa k spoločnej najvýnosnejšej akcii pre celý tím. To je hlavná myšlienka navrhnutého algoritmu.

8 Spúšťanie, skript a práca s trénerom

Tím implementoval aplikáciu trénera, ktorá je určená k zdokonaľovaniu a sledovaniu zmien počas návrhu a implementácie hráča. Hlavným dôvodom je používanie trénera v procese testovania hráča. Potrebou bola možnosť nastavovania polohy hráčov, lopty a opätovne opakovať trénovanú situáciu.

8.1 Umiestnenie a spúšťanie trénera

Aplikácia trénera sa nachádza v samotnom adresári projektu hráča. Je označená názvom adresára *trener-build*, kde sa nachádza sprievodný súbor s návodom na použitie. Zároveň na uľahčenie práce bol vytvorený v *build* adresári súbor *start-all.bat a* súbor *start-all-with-trainer.bat*, ktoré automatický spustia žiadaný počet hráčov, server, monitor a aj samotného trénera. Hlavným cieľom bolo zautomatizovanie spúšťania projektu ako celku.

Spúšťací súbor *start-all-with-trainer.bat* je možné upraviť tak, aby sa dal zápas spustiť proti rôznym súperom, aby sa soccer server, soccer monitor dali spustiť z ľubovoľného umiestnenia na disku, aby sa dal nastavovať počet hráčov a aby sa spúšťal aj tréner.

Úprava spustiteľného BAT súboru:

Pre zmenu súpera je potrebné upraviť súbor(i) nasledovne:

```
set team2=[lubovoľný názov timu]
set prog2=[existujúca cesta k exe súboru hráča]
```

Pre zmenu trénera je potrebné upraviť súbor(i) nasledovne:

```
set trener_path existujúca cesta k exe súboru trénera]
```

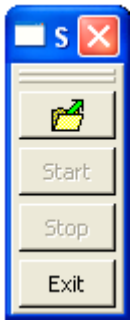
Ďalej je už iba potrebné spustiť jednotlivých hráčov a trénera(prípadne spolu s brankárom) nasledovne:

```
start %PROG2% -n 8 -t %TEAM2% -h %HOST%
start %trener_path%trainer.exe
```

Ak potrebujete zmeniť umiestnenie soccer servera a monitora, stačí v hornej časti bat súboru zmeniť počiatočné údaje:

```
set server_path=[platná cesta k exe súboru soccer servera]
set monitor_path=[platná cesta k exe súboru soccer monitora
```

Po spustení nami pripraveného skriptu sa nám zobrazí okno aplikácie trénera, v ktorej si následne zvolíme trénovanú akciu, rozostavenie, ktoré má byť spustené pomocou trénera (viď Obr. 13.).



Obr. 13 Okno aplikácie trénera

Samotný skript má formát XML (viď Obr. 14), v ktorom sú zachytené vlastnosti trénovanej akcie. Súbor má pomenovanie *situation.xml*. Medzi základné možnosti nastavenia akcie patria:

- *Možnosť nastavenia počtu hráčov* - v samotnom súbore sa nachádza časť s názvom players, v ktorom je popis jednotlivých vlastnosti hráčov. Počet hráčov určuje počet tágov s názvom <*player*>.
- *Nastavenie pozície hráčom* – pozíciu hráčovi určuje tag s názvom <*position*>
- *Nastavenie pozície lopty* – pozíciu lopty nastavíme pomocou tagu s názvom <*position*>, ktorý je podtagom tagu <ball>

Následný obrázok je ukážkou XML nastavenia trénovanej akcie.

```
<?xml version="1.0" encoding="UTF-8"?>
<situation xmlns="http://www2.dcs.elf.stuba.sk/TeamProject/2002/team01/Situation">
  <ball>
    <position x="-40" y="0"/>
    <velocity x="0" y="0"/>
  </ball>
  <players>
    <leftSide>
      <player number="1">
        <position x="-40" y="0"/>
        <velocity x="0" y="0"/>
        <neckAngle value="0"/>
      </player>
      .
      .
      .
      .
```

Obr. 14 Ukážka zápisu trénovanej akcie

8.2 Analýza logovania

Pre potreby testovania a debugovania sme sa pokúšali pracovať s logovacím systémom, ktorý bol už bol implementovaný v hráčovi. Funkcia logovania je implementovaná v súboroch:

- *Logger.cpp*
- *SituationsLog.cpp*

Podľa týchto dvoch súborov sa delí logovanie na:

- Štandardné logovanie, kde úroveň zobrazenia logov je daná ich nastavenou prioritou.
- Logovanie situácie na ihrisku. Tento logovací nástroj zapisuje pozície hráčov na ihrisku do xml súboru. Tento logovací súbor môže byť nápomocný pre neskoršie testovanie.

Logovanie situácie na ihrisku do súboru prebieha bez problémov, nakoľko každý hráč ma definovaný súbor, do ktoré sa tieto logované informácie zaznamenávajú.

Pre štandardné logovanie je situácia zložitejšia. Hráčovi je potrebné nastaviť úroveň logovania, t.j. rozsah, v ktorom sa budú log informácie zaznamenávať. Nastavuje sa minimálna a maximálna hodnota. Buď ako hodnota prepínača spúšťanému hráčovi (prepínač *–d*), alebo priamo v zdrojovom súbore *main.cpp*. Nastavením správneho rozsahu logovania, by sa mal vytvoriť testovací súbor. Podľa zdrojových súborov je prepínač *–o*, ktorý by mal umožniť vytvorenie logovacieho súboru. Nám sa nepodarilo spustiť toto logovanie.

Druhým spôsobom, ktorým sme sa pokúšali logovať, bolo logovanie prostredníctvom štandardného výstupu (stdout) cez funkciu jazyka C *printf()*. Tu sme narazili na problém, že nakoľko spúšťame simulácie prostredníctvom *bat* súborov, nie je možné v nich presmerovať výstup do súborov. Bez presmerovania výstupu bol výpis do konzoly prirýchly a nepoužiteľný.

9 Metodika pre debugovanie hráča

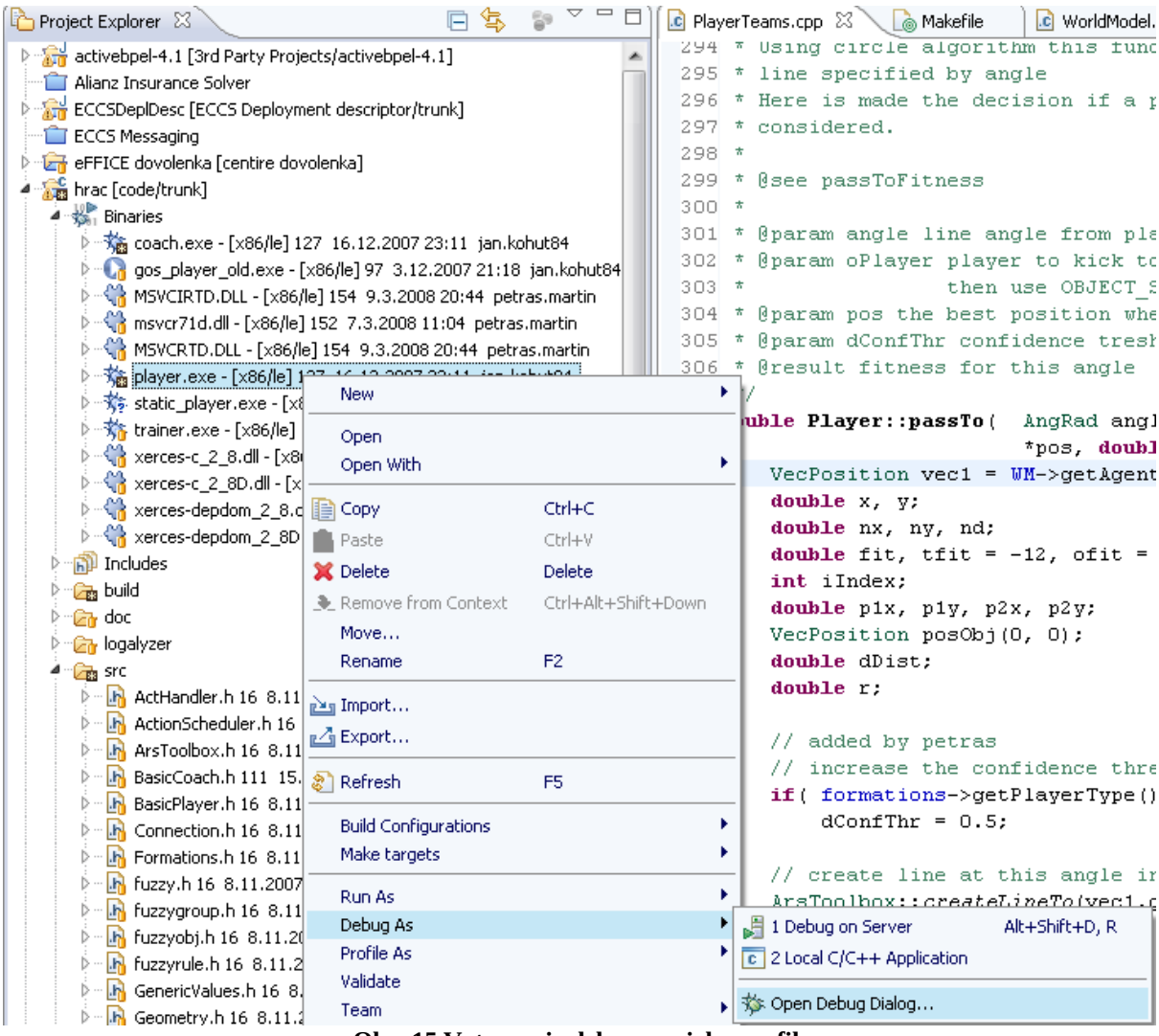
9.1 Úvod

Tím vychádzal z dokumentu *Eclipse Project CDT (C/C++) Plugin Tutorial* http://www.cs.umanitoba.ca/~eclipse/7-EclipseCDT.pdf

Je nutné doinštalovať GDB. Je to unixovský debugger, ktorý bude spúšťaný pod MinGW prostredím. Stiahnuť si ho môžete z webu MinGW. Pri kompilácií hráča musí byť definovaný špeciálny prepínač *–ggdb* viď makefile. Tento prepínač spôsobí, že kompilátor vygeneruje do exe súboru extra informácie potrebné pre debugger. Spustite *make clean* – to zmaže staré object subory (*.o). A opäť buildnite hráča. Spustite celú hru (start-all.bat alebo start-all-with-trainer.bat).

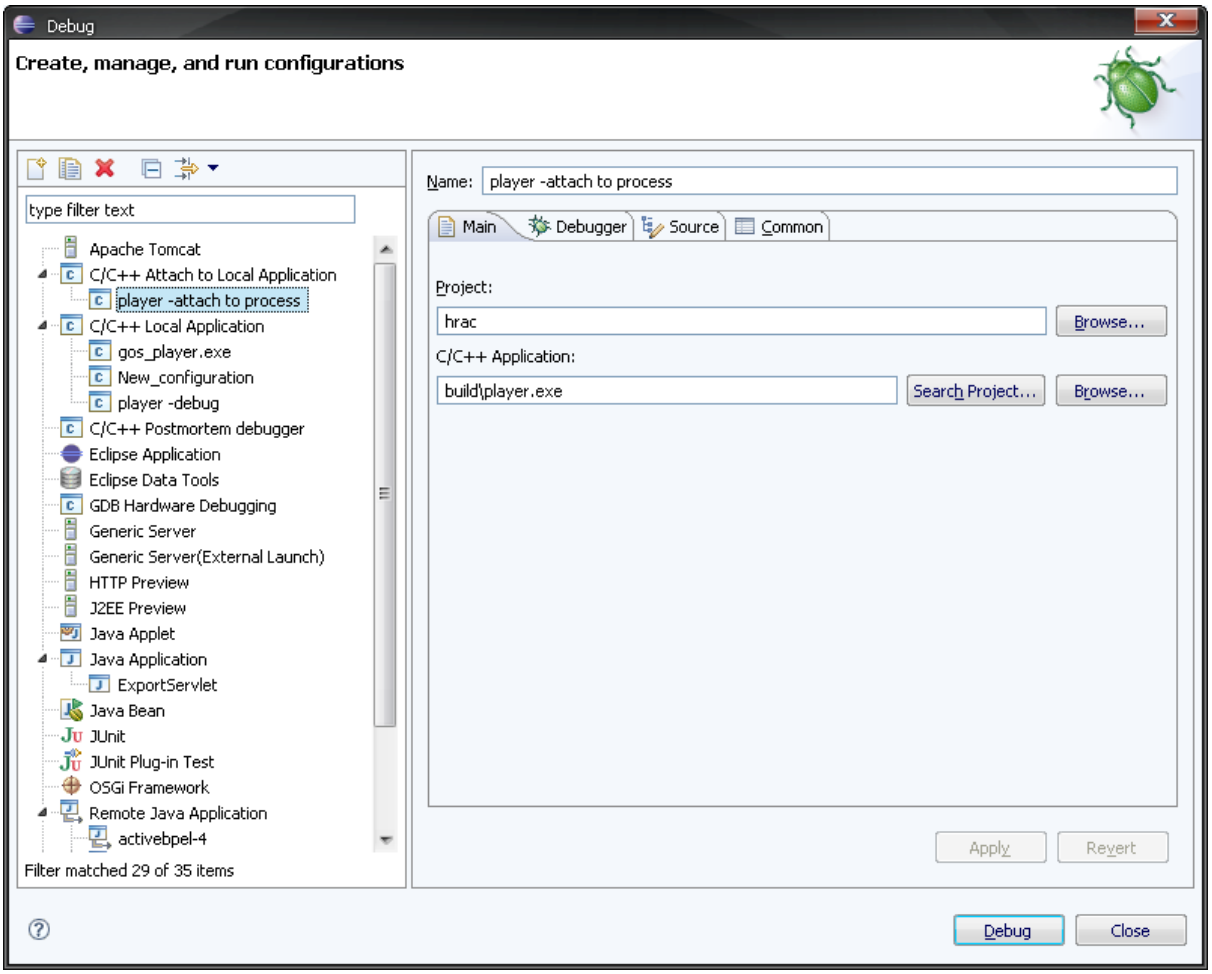
9.2 Vytvorenie profilu pre debugovanie hráča

1. Vytvorte nový debug profil (viď Obr. 15).



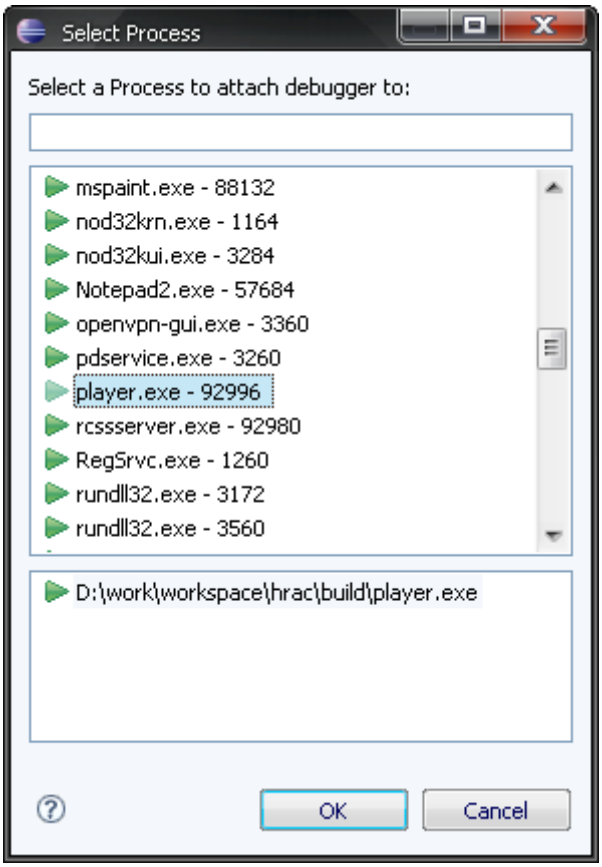
Obr. 15 Vytvorenie debugovacieho profilu

2. Ďalej Obr. 16.



Obr. 16 Vytvorenie debugovacieho profilu (2)

3. Pri spúšťaní debugovania je nutné vybrať už bežiaci proces,na ktorý sa napojíme.



Obr. 17 Výber procesu

9.3 Návody

1. Pri debugovaní spravidla stačí sledovať činnosť jedného agenta, preto odporúčam si spustit len jednu inštanciu nášho hráča, na ktorú sa napojíme pri debugovaní. Teda upravte si .bat súbor podľa potreby nejako takto:

```
set team=JahodoviPrincovia
set prog=player

set team2=GangOfSix2
set prog2=static_player

:: team 1
start %PROG% -n 1 -t %TEAM% -h %HOST%
sleep 1
start %PROG2% -n 2 -t %TEAM% -h %HOST%
sleep 1
start %PROG2% -n 3 -t %TEAM% -h %HOST%
sleep 1

:: team2
start %PROG2% -n 1 -t %TEAM2% -h %HOST%
sleep 1
```

```
start %PROG2% -n 2 -t %TEAM2% -h %HOST%
sleep 1
```

Teda nevadí, že s našim hráčom hraje v jednom tíme nejaký iný hráč.

2. Vytvoril som špeciálneho hráča *static-player.exe*. Tento hráč sa po celý čas nepohne z miesta. Teda nášmu debugovanému hráčovi sa nemení model sveta. Pri tomto hráčovi je nutné použiť trénera, ktorý mu nastaví pozíciu na ihrisku.

3. Dávajte si pri debugovaní pozor na fakt, že hráčovi sa s časom mení vnútorný model sveta. Teda, že hoci ho máte držíte na breakpointe, jeho model sveta sa ďalej aktualizuje v ďalšom threade. Takisto si treba uvedomiť, že informácie starnú v každom cykle, pokiaľ nie sú obnovované. Preto sa môže stať, že náš hráč začne niektoré informácia z modelu sveta ignorovať, lebo ich pokladá za príliš staré.

Záver

Dokument Implementácia navrhovaného riešenia má za úlohu informovať o stave realizácie navrhovaného riešenia. Tím č. 16 sa počas letného semestra akademického roku 2007/2008 snažil zrealizovať to, čo je uvedené v návrhu riešenia. Dokument postupne opisuje implementované vylepšenia hráča tímu Gang of Six. Cieľom tímu je vylepšenie hráča minimálne natoľko, aby bol schopný zvíťaziť proti hráčovi tímu Gang of Six, ktorý bol vytvorený v minulom akademickom roku. Algoritmus elmininácie premenných, formácie, brankár, využitie trénera, analýza logovania i debugovanie sú hlavné oblasti, na ktoré sa tím zameral. Tieto časti sú podrobne rozpísane v kapitolách dokumentu.

Prílohy

V tejto časti sú uvedené prílohy týkajúce sa dokumentu Implementácia navrhovaného riešenia.

Príloha A: Bibliografia

Príloha B: Zoznam tabuliek

Príloha C: Zoznam obrázkov