Pôvodná architektúra a motivácia na zmenu architektúry

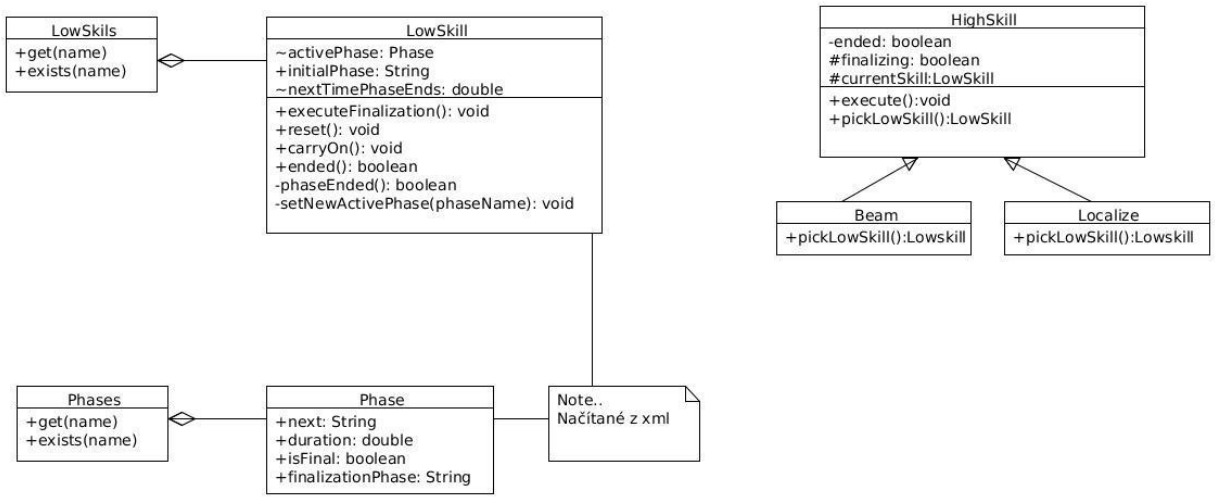
Na začiatku zimného semestra sme preberali projekt, v ktorom bolo použitých viacero programovacích jazykov - Java i Ruby - a na ktorom dovtedy pracovalo v priebehu niekoľkých rokoch viacero tímov, bakalárov a diplomantov. Zo začiatku bolo dôležitým problémom vysporiadať sa so stabilitou hráča, naučiť ho chodiť a orientovať sa na ihrisku.
Čím sa však projekt viac a viac rozvíjal, došlo k problému s komplexitou projektu, čo sa zanieslo hlavne použitím jazyka Ruby. Noví programátori vstupujúci do projektu neoboznámení s konvenciami v Ruby, kedy sa volali rovnaké metódy avšak s inými názvami spôsobovali dosť značné problémy, vďaka čomu si viacero ľudí vytváralo nové a duplicitné metódy. Každý vývojár si do projektu doniesol vlastné idei, avšak projektu chýbala komplexnejšia architektúra - akýsi framework - ktorým by sa programátori striktne riadili. Hlavná motivácia tohto zadania bola teda prepis architektúry, odstránenie Ruby, zuniformovanie zápisu highskillov a vytvorenie lepšej možnosti pre ľahšiu implementáciu pokročilých vecí do projektu v budúcnosti.

Diagramy pôvodnej architektúry

Diagramy pôvodnej časti Ruby projektu je možné nájsť na tímovej stránke na adrese http://team09-13.ucebne.fiit.stuba.sk/rdoc/ V pôvodnej časti projektu triedy v Ruby dedili od tried v Jave plus implementovali vlastný spôsob logovania.

Návrh architektúry

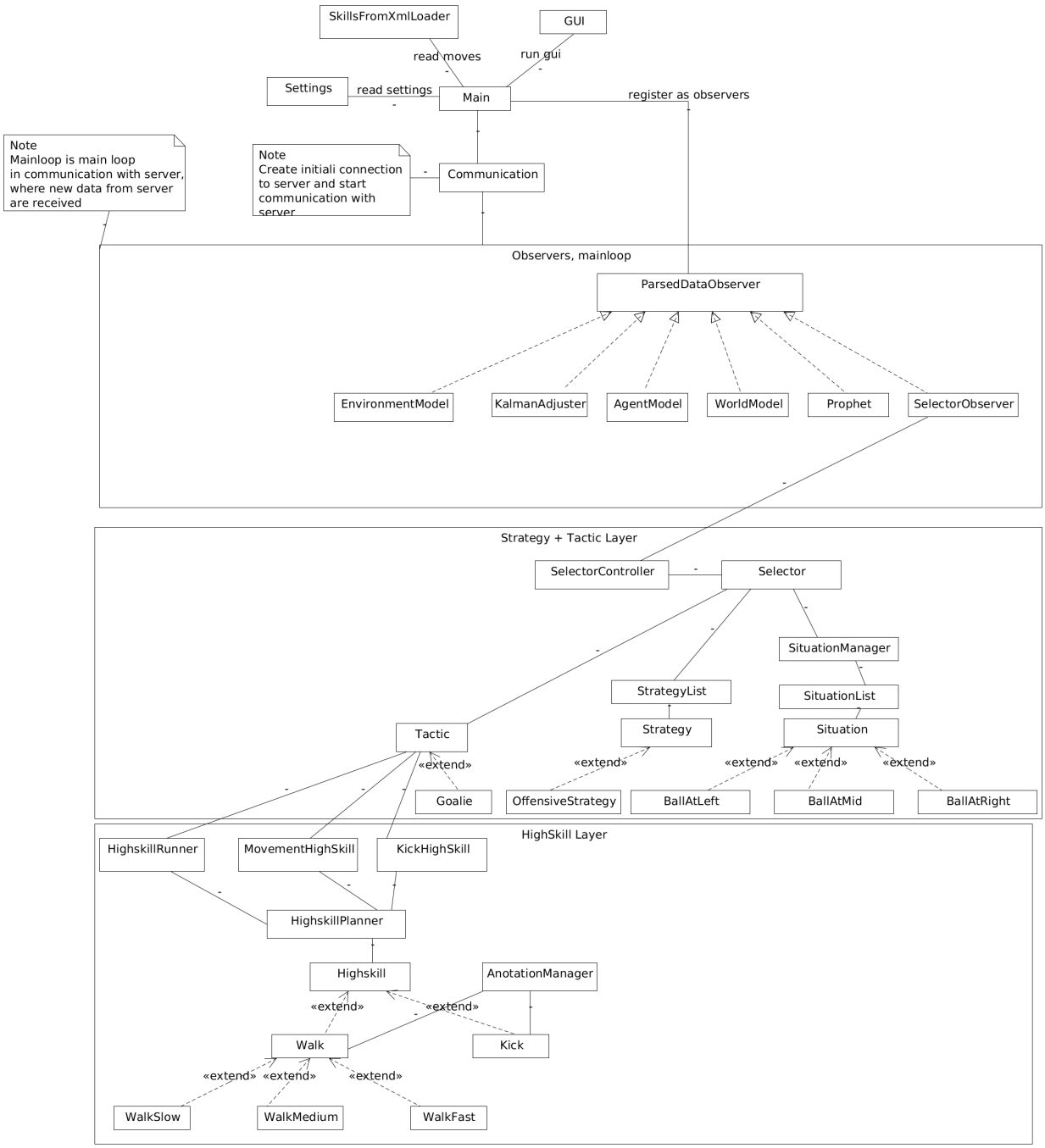
Spočiatku bolo potrebné prepísať časti v jazyku Ruby do jazyka Java. Zmenil sa tak diagram pohybov, nakoľko už Ruby highskilly nededia od Java highskillov. Aktuálny diagram je nasledovný:



Obr 1.: Diagram pohybov bez tried v jazyku Ruby

Na návrhu architektúry pracovali dva tímy, pričom implementáciu strategickej a taktickej vrstvy vrátane vyberania stratégii mal na starosti tím tp04. Náš tím tp09 mal na starosti vylepšovanie nižších vrstiev ako plánovač, highskilly, lowskilly a anotácie.

Návrh architektúry spočíval v navrhnutí strategickej a taktickej vrstvy za využitia situácií, podľa ktorých budeme vedieť lepšie riadiť hráčov na ihrisku a rozdeliť rozhodovaciu logiku do viacero úrovní spolu s odstránením množstva rozhodovacích stromov z kódu. Pôvodné plány, ktoré sa svojím zameraním blížili taktikám boli odstránené a nahradené vrstvou taktík.



Obr. 2 Implementácia návrhu architektúry

V diagrame je zobrazené zjednodušené UML jednotlivých vrstiev - od strategickej, cez taktickú až po vrstvu highskillov. V úvode je zobrazené i prepojenie tried vykonávajúce inicializáciu programu.

Jednotlivé taktiky, stratégie a highskilly a situácie je možné do projektu ľahko pridávať, avšak nie je to možné “za behu”, nakoľko sa pri spúšťaní programu vytvára list - zoznam stratégií/taktík.
Z projektu bol odstránený jazyk Ruby, zmenila sa i inicializácia, kedy už nie je potrebná inštalácia Ruby jazyka.

Celkový koncept pohybovania agenta

Agent na ihrisku je riadený kódom, ktorý sa skladá z viacerých modulov navzájom prepojených. Moduly sú v hierarchií a každý má inú funkcionalitu.

Selector

Selector má na starosti vyberanie vhodných stratégií a taktík na základe daných situácií na ihrisku. Trieda SelectorObserver implementuje rozhranie ParsedDataObserver. Triedy, ktoré implementujú toto rozhranie vždy po sparsovaní novej správy zo servera sú notifikované. Selector teda vyberá taktiky vždy s príchodom novej správy zo servera.

Metóda **controlTactic** v triede SelectorController preveruje, či aktuálna taktika stále spĺňa kritéria pre vykonávanie. Tj, či neexistuje taktika s oveľa väčšou fitness, alebo taktika nie je ukončená.

Metóda **controlStrategy** v triede SelectorController preveruje, či aktuálna stratégia stále spĺňa kritéria pre vykonávanie. (Neexistuje stratégie s väčšou fitness)

Stratégie

Stratégie predstavujú niečo, čo je dohodnuté pre celý zápas, alebo polčas. V reálnom futbale sa zvyčajne stratégie menia cez pauzu alebo koncom zápasu - ak je nepriaznivé skóre, alebo naopak, ak je skóre veľmi priaznivé. Stratégie teda môžu byť napríklad stratégie udržania skóre (obranná), alebo stratégia útočná, v ktorej budeme viac riskovať aby sme dokázali streliť gól.

Nad vrstvou stratégii je implementovaný vyberač vhodných stratégií - trieda Selector v balíku sk.fiit.jim.decision. Stratégie sa vyberajú na základe vhodnosti - tzv fitness. Výpočet fitness spočíva v porovnávaní aktuálnej situácie s predpripravenými situáciami tzv šablónami. Keďže sa nemusí podariť mať ošetrené všetky možné kombinácie situácií, v prípade nezhody berieme najlepšie vyhovujúcu situáciu.

Implementácia strojového učenia

Do tejto architektúry je v budúcnosti možné implementovať strojové učenie.

Príklad ako :

- Ak vybraná stratégia dala gól, zvýšiť jej fitness, popr. prioritizovať danú stratégiu

- Fitness sa môže logovať, čím bude možné vyhodnocovať dosiahnuté výsledky pri zvolenej fitness i po skončení programu

Kedy sa mení stratégia?

- Nenaplnili sa ciele stratégie

- zmena času

- zmena skóre hry

Taktiky

Vrstva taktík predstavuje správanie, ktoré sa mení počas hry. Taktika môže byť napríklad útok po ľavom krídle, kedy si agent naplánuje, že chce útočiť po ľavom krídle, prejsť s loptou pred bránou a tam sa taktika na základe danej situácie ďalej preplánuje. Taktika sa opäť vyberá v triede Selector, na základe porovnávania s danými situáciami.

Kedy sa (ne)mení taktika?

- Pokiaľ nemám oveľa lepšiu taktiku, tak ju nemením

- Pokiaľ prejdem do iného kvadrantu

Trieda Taktika obsahuje metódy:

getInitCondition(List<String> currentSituations) - metóda kontroluje, či táto taktika spĺňa predpodmienky a či teda môže byť aplikovateľná na základe aktuálnych situácií

getProgressCondition - kontroluje, či taktika ešte stále môže byť vykonávateľná, či sa situácia na ihrisku výrazne nezmenila.

Taktika vie kam sa má agent dostať a akým spôsobom - či použíť rýchlu alebo pomalú chôdzu, alebo radšej stabilnú. Taktická vrstva by však nemala vedieť o interných záležitostiach vrstvy s plánovačom a highskillmi, preto bola vytvorená medzivrstva, akési rozhranie, pomocou ktorého vie taktická vrstva plánovať vykonávanie highskillov.

Plánovanie vykonávania highskillov z taktickej vrstvy

Taktická vrstva môže naplánovať highskilly na vykonávanie pomocou volania naslednovných metód:
- KickHighSkill.kick(Vector3D kicktarget, KickTypeEnum kickType)
- MovementHighSkill.move(Vector3D position, MovementSpeedEnum prefferedSpeed)
- BeamHighSkill.beamAgent(Vector3D)

Tieto metódy na základe dodaných atribútov a na základe anotácii hľadajú najvhodnejší highskill - počíta sa tzv. fitness. Fitness sa počíta v metóde getWalkSuitability. Fitness sa počíta z troch vlastností - minimálnej vzdialenosti, na ktorú je highskill možné použiť, rýchlosti a stability. Tieto tri vlastnosti sú vo vzorci navzájom násobené. Vychádza sa z predpokladu, že rýchla chôdza nemôže byť použitá na veľmi krátku vzdialenosť a podobne pomalá chôdza na veľmi dlhú. Implementované boli highskilly WalkFast, WalkMediu a WalkSlow, ktoré dedia od highskillu Walk, čím je ich funkcionalita rovnaká, až na výber lowskillu, ktorý vyberajú na základe rýchlosti, ktorou majú kráčať.

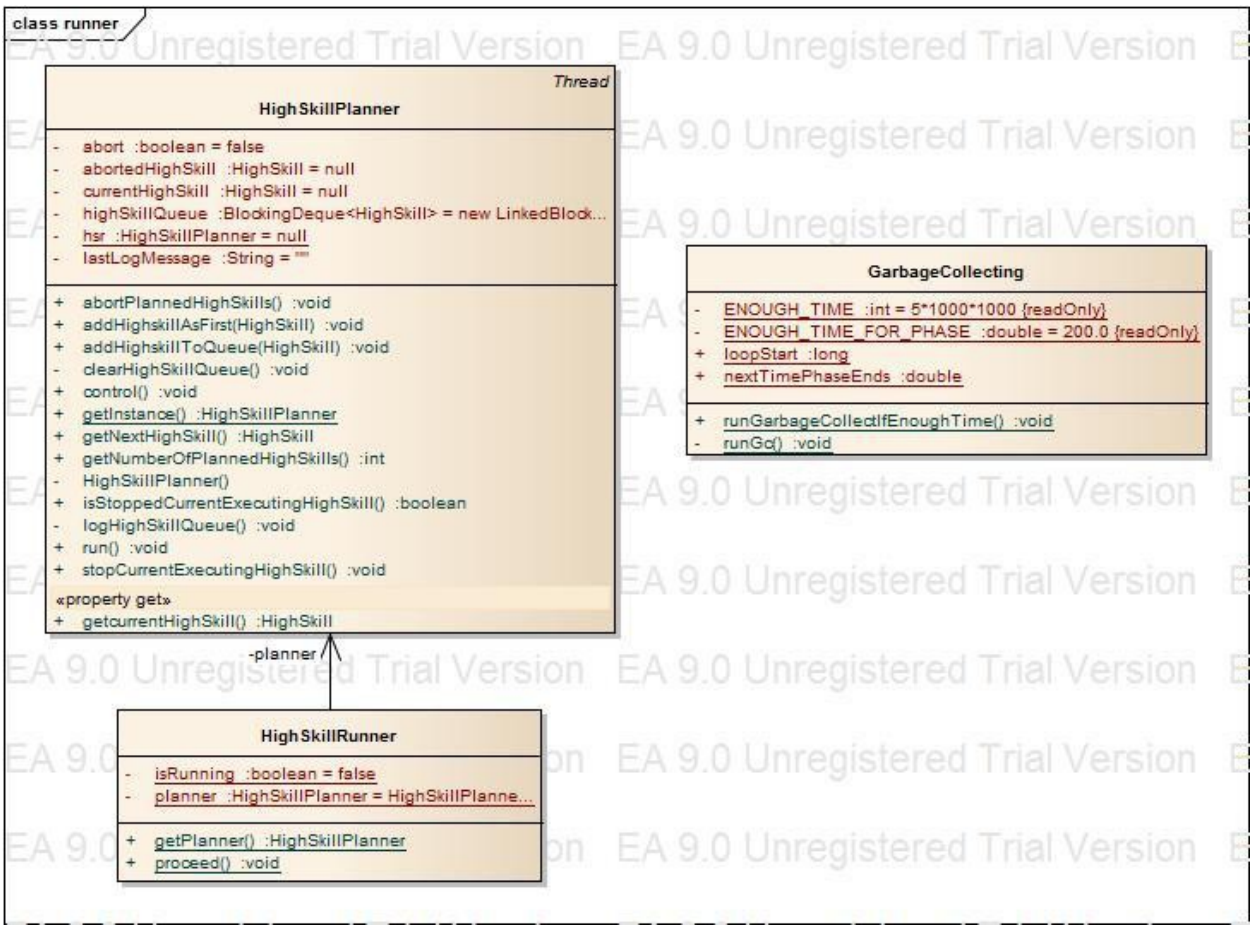
Ak bude treba pridať nový highskill a bude typu walk, bude musieť byť pridaný do enumu MovementEnum a musí mať v enume implementovanú metódu computeSuitability. Výhoda computeSuitability v enumoch spočíva v tom, že každý highskill, ktorý bude zaradený do enumu a bude môcť byť volaný taktickou vrstvou bude musieť mať implementovanú metódu na výpočet vhodnosti, ale navyše každý highskill v enume ju môže mať modifikovaný - tu sa počítalo s možnosťou zakomponovania strojového učenia, kedy by sa mohla fitness počas vykonávania hry dynamicky meniť podľa úspešnosti daného pohybu.

V budúcnosti by bolo vhodné, aby sa metódy rozšírili o voliteľný atribút - hashmapu - na preposielanie viac vlastností pre presnejší výber. V prípade, že je potrebné pridať ďalšiu medzivrstvu, je potrebné vytvoriť dané triedy v balíku highskill a vytvoriť inštanciu danej triedy v abstraktnej triede Tactic.

Nie je to žiadúce, ale je možné vkladať(plánovať) highskilly do plánovača i priamo volaním metódy HighSkillPlanner.addHighskillToQueue(HighSkill highSkill), kde ako argument bude predaná nová inštancia daného highskillu. Je to vhodné pre highskilly, pre ktoré sa nedá vytvoriť štruktúra ako pre pohyby walk, či kick, avšak takýto prístup veľmi rýchlo spôsobí, že taktika už nebude spĺňať súčasný návrh jej abstrakcie od highskillov.

Plánovač

Plánovač oproti pôvodnému riešeniu používa inú štruktúru na rad highskillov pre vykonávania - ArrayList sme vymenili za obojsmerný rad - LinkedBlockingDeque. Umožnilo nám to pridávať požiadavky ako na začiatok radu, tak i na koniec radu. Obrovskou výhodou je, že highskilly ako GetUp vieme zaradiť na začiatok radu - aby sa vykonávali prioritne. V plánovači pribudli metódy na ukončenie aktuálne vykonávaného highskillu a logovanie informácii o aktuálne vykonávaných highskilloch v plánovači. Plánovač bol presunutý do balíka sk.fiit.jim.agent.highskill.runner



Obr 3.: Plánovač

Pri inicializácii projektu sa vytvorí inštancia triedy Communication, v ktorej beží tzv mainloop - cyklus, ktorý číta správy zo socketu. Ak sú na vstupe dáta zo servera, v každej iterácii cyklu sa volá plánovač - HighSkillPlanner. Inštancia triedy HighSkillPlanner sa tvorí v triede HighSkillRunner, ktorá je i zodpovedná za jeho spustenie. Plánovač beží v samostatnom vlákne.

V plánovači sú metódy:
control() sa spúšťa vždy, keď príde správa zo servera. Metóda vždy kontroluje, či vyššia vrstva (taktika) nastavila príznak abort (zavolanie metódy abortPlannedHighSkills()) a či má v danej iterácii plánovač ukončiť aktuálne vykonávaný highskill. Inak, ak nie je prázdny rad highskillov čakajúcich na vykonanie, vyberie z radu ďalší highskill na vykonávanie a začne ho vykonávať.

addHighskillToQueue(HighSkill highSkill) je metóda určená pre taktickú vrstvu, ktorá vkladá highskilly to radu highskillov na vykonanie v plánovači

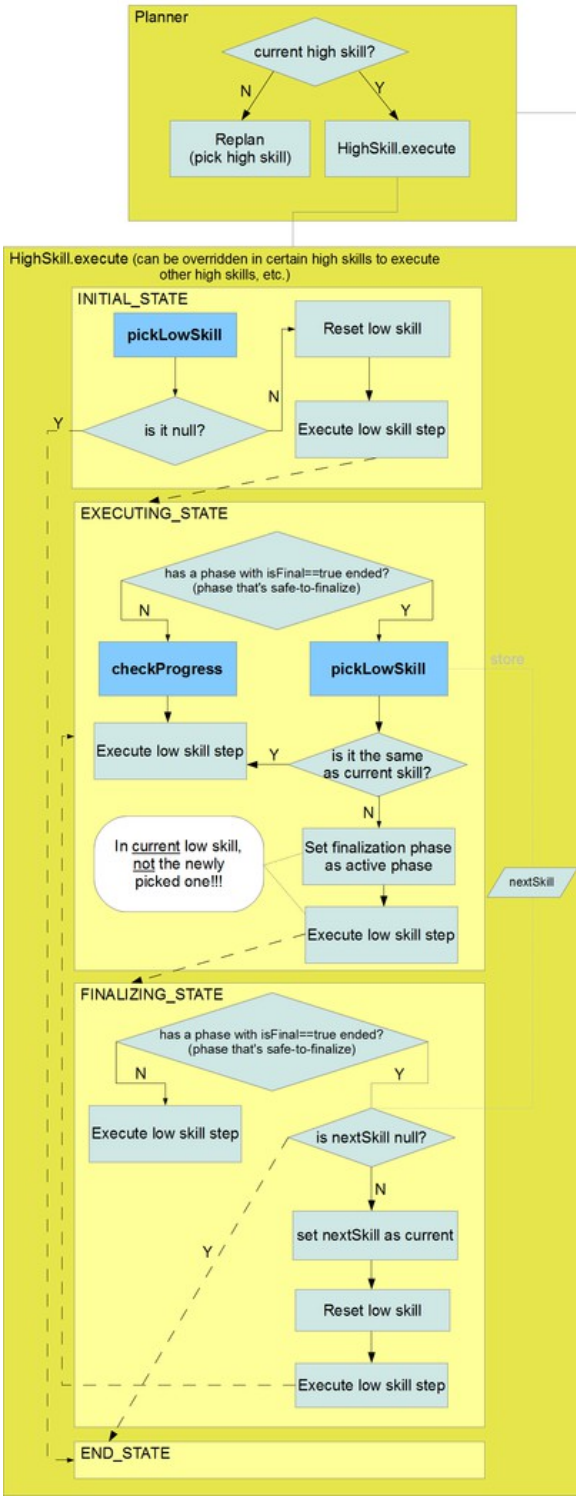
addHighskillAsFirst(HighSkill highSkill) metóda umožňuje taktickej vrstve prioritizovanie vykonávania niektorých highskillov tým, že highskill vloží na začiatok radu v plánovači.

getcurrentHighSkill() vracia aktuálne vykonávaný highskill

getNextHighSkill() vráti nasledujúci naplánovaný highskill na vykonanie

abortPlannedHighSkills() ukončí aktuálne vykonávaný highskill - ak je to možné, inak musí vykonávanie highskillu skončiť. Potom vymaže rad naplánovaných pohybov

Priebeh vykonávania highskillu v plánovači



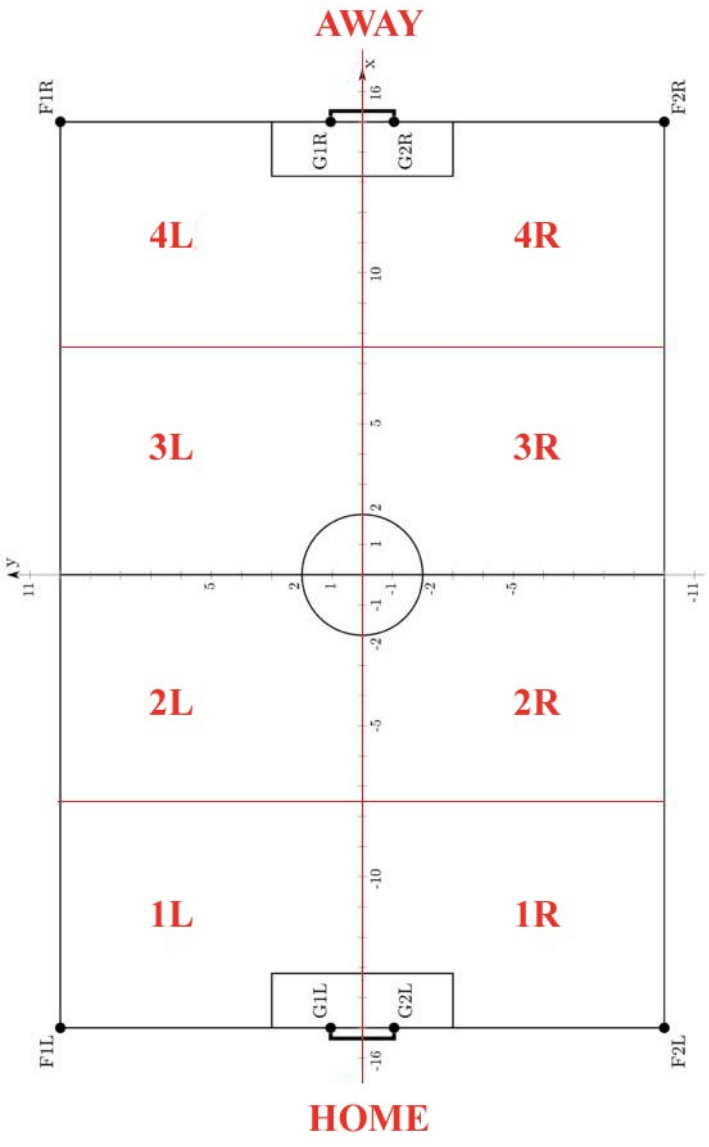
Situácie

Trieda situácie reprezentuje jednotlivé situácie, ktoré môžu počas hry nastať. Sú to situácie napríklad:

- loptu má protihráč a som v prvom kvadrante a lopta je v treťom kvadrante

- lopta je v druhom kvadrante, som v prvom kvadrante a nikto nemá loptu

V projekte je vytvorených veľa situácií, nachádzajú sa v balíku sk.fiit.jim.decision.situation. Ihrisko pre situácie bolo rozdelené do viacerých oktánov - viď obrázok.

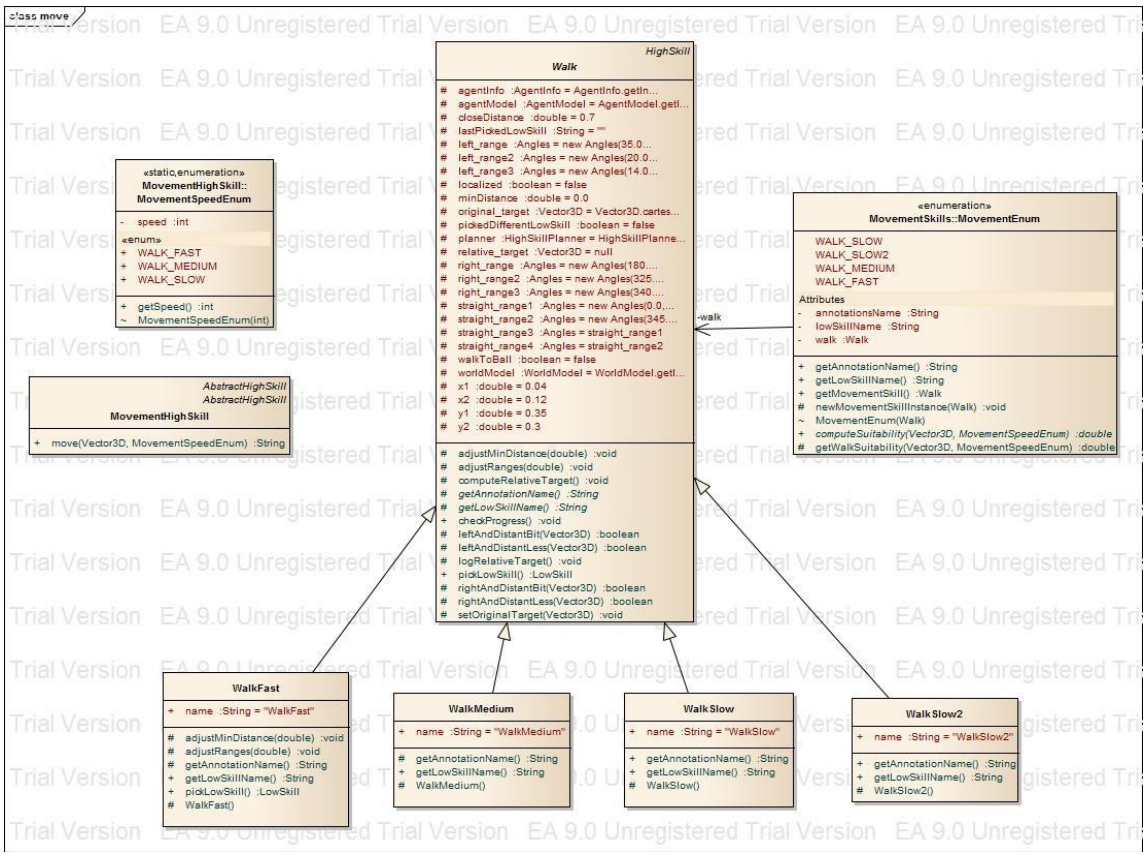


Highskilly - chôdza, kop

V pôvodnom riešení z ktorého vychádzame bolo nutné vyberať ručne najvhodnejší pohyb a ten implementovať v plánovači. Vďaka tomuto sa zabúdalo na vylaďovanie zvyšných pohybov, ktoré sa nepoužívali. Ďalším problémom bol aj výber nevhodného typu pohybu pri určitých situáciách.

V našom novom riešení má agent zadefinovaný konečný zoznam pohybov, ktoré vie vykonávať. Z tohto preddefinovaného zoznamu pohybov si vyberá najvhodnejší pohyb na základe atribútov poskytnutých z taktiky a na základe anotácií (štatistík) jednotlivých pohybov. Pri výbere pohybu sa vyhodnocuje vhodnosť všetkých dostupných pohybov a na základe výsledkov sa vyberá pohyb, ktorý dosiahol najvyšší výsledok.

Všetky z preddefinovaných pohybov zo zoznamu majú spoločnú rodičovskú triedu, v ktorej sú zadefinované spoločné atribúty a metódy. Táto rodičovská trieda implementuje rozhranie, ktoré definuje správanie každého vyššieho pohybu. V tejto triede je taktiež implementovaná všeobecná metóda na výber nižšej schopnosti, ktorá je použiteľná pre všetkých potomkov. Následne v jednotlivých triedach pohybov je dôležité zadefinovať názov pohybu, slúžiaci pri vyhľadávaní anotácií a nižších pohybov. V každej triede je následné možné preťažiť metódu výberu pohybov a zadefinovať vlastné vhodnejšie správanie.



Obr 4.: Move highskill

Obr 5.: Kick highskill

Lowskilly

Do projektu boli importované niektoré nové lowskilly z práce Petra Paššáka. Konkrétne išlo o lowskill turbo walk.

Anotácie

Do anotácií boli pridané nasledujúce parametre:

- *variance* – rozptyl – určuje, aký je uhlový rozptyl smeru kopnutej lopty v stupňoch
- *deviation* – odchýlka – určuje, aký je uhol medzi natočením agenta a smerom kopnutej lopty v stupňoch
    - *min* – minimálna nameraná odchýlka
    - *max* – maximálna nameraná odchýlka
    - avg – priemerná odchýlka
- *kickTime* – čas potrebný na vykonanie pohybu kopnutia do lopty (samotné vykonávanie pohybu bez započítania prípravnej fázy, kedy sa agent nastavuje k lopte) v sekundách
- *kickSuccessfulness* – úspešnosť trafenia lopty agentom v percentách
- *kickDistance* – priemerná vzdialenosť, ktorú prejde lopta po odkopnutí
- *agentPosition* – pozícia agenta voči lopte, z ktorej je možné vykonať úspešne kop a očakávať výsledok podobný ako je uvedený v ostatných parametroch anotácie.
    - *x-min* – minimálna vzdialenosť od lopty

- *x-max* – maximálna vzdialenosť od lopty
- *y-min* – minimálne posunutie do strany
- *y-max* – maximálne posunutie do strany

Okrem uvedených parametrov je pre kopy dôležitý aj parameter *fall* opísaný už v predošlom šprinte.

Následkom pridania nových parametrov do anotácií bolo potrebné upraviť triedy *Annotation, XMLParser, XMLCreator,* pridať triedu *AgentPosition* všetko v rámci balíku *annotation.data*. Ďalej bolo nutné modifikovať definičný súbor pre XML anotácie.

Staré a neaktuálne anotácie boli presunuté do súboru annotationOld, pre prípad možného využitia v budúcnosti.

Namerané údaje pre jednotlivé kopy

Nasledujú namerané údaje pre jednotlivé kopy uvedené v tabuľkách 8.1-8.4. Na meranie údajov bol využitý testframework z diplomovej práce Ing. Petra Paššáka.

kick_left_normal

agentPosition
- x-min -0,15
- x-max -0,20
- y-min -0,02
- y-max -0,08

kickSuccessfulness 100%
fall 0%
variance 0,07264

	deviation	**kickTime [s]**	**kickDistance [m]**
1	-4,91	1,6	2,00
2	-4,35	1,6	2,03
3	-4,30	1,6	2,04
4	-4,91	1,6	2,04
5	-4,46	1,6	2,04
Priemer:	-4,586	1,6	2,03

Tabuľka 8.1 Hodnoty testovania kopu kick_left_normal

deviation

- min -4,30
- max -4,91

kick_right_normal

agentPosition

- x-min -0,15
- x-max -0,20
- y-min 0,02
- y-max 0,08

kickSuccessfulness 100%
fall 0%
variance 0,067856

	deviation	**kickTime [s]**	**kickDistance [m]**
1	4,30	1,6	2,03
2	4,89	1,6	2,02
3	5,03	1,6	2,03
4	4,91	1,6	2,00
5	4,93	1,6	2,04
Priemer:	4,812	1,6	2,024

Tabuľka 8.2 Hodnoty testovania kopu kick_right_normal

deviation

- min 4,30
- max 5,03

kick_left_faster

agentPosition

- x-min -0,15
- x-max -0,25

- y-min -0,02
- y-max -0,08

kickSuccessfulness 100%

fall 0%

variance 1,490865

	deviation	**kickTime [s]**	**kickDistance [m]**
1	-4,97	1,5	2,98
2	-5,06	1,5	3,77
3	-4,36	1,5	2,95
4	-4,87	1,5	3,68
5	-4,59	1,5	3,06
6	-5,96	1,5	3,72
7	-4,18	1,5	2,95
8	-8,19	1,5	3,72
9	-6,21	1,5	3,69
10	-3,76	1,5	2,92
Priemer:	-5,215	1,5	3,344

Tabuľka 8.3 Hodnoty testovania kopu kick_left_faster

deviation

- min -3,76
- max -8,19

kick_right_faster

agentPosition

- x-min -0,15
- x-max -0,25
- y-min 0,02
- y-max 0,08

kickSuccessfulness 100%
fall 0%
variance 1,490865
kickTime [s] 1,5
kickDistance [m] 3,334
deviation
- min 3,76
- max 8,19
- avg 5,215

kick_step_strong_left

agentPosition
- x-min -0,25
- x-max -0,30
- y-min -0,05
- y-max -0,05

kickSuccessfulness 100%
fall 13%
variance 2,879706

	deviation	**kickTime [s]**	**kickDistance [m]**
1	-7,93	2,8	5,01
2	-7,82	2,7	4,95
3	-7,06	2,7	4,89
4	-8,04	3,0	4,62
5	-6,44	2,7	5,0
6	-10,48	3,0	4,57
7	-10,72	2,9	5,15
8	-11,31	2,9	5,87
9	-10,32	2,7	5,71
Priemer:	-8,9022	2,8222	5,0855

Tabuľka 8.4 Hodnoty testovania kopu kick_step_strong_left

deviation

- min -6,44
- max -11,31

kick_step_strong_right

agentPosition

- x-min -0,25
- x-max -0,30
- y-min 0,05
- y-max 0,05

kickSuccessfulness 100%
fall 13%
variance 2,879706
kickTime [s] 2,8222
kickDistance [m] 5,0855
deviation

- min 6,44
- max 11,31
- avg 8,9022

Known bugs

Vzhľadom na prepis architektúry a prepis časti kódu z Ruby do Javy, zostal nefunkčný testing server, ktorý momentálne nefunguje a nedá sa použiť na testovanie. Taktiež nefunguje i gui, ktoré vyskakuje pri štarte agenta a malo spôsobovať refresh načítavania nastavení a lowskillov. Nakoľko nastavenia sú už mimo časť ruby, toto gui už nebude fungovať.

Export backlog úloh

Pripájame export úloh z product backlogu, ktoré by boli vhodné riešiť v nasledovných rokoch.

Flag pre brankára
Brankár by mal byť odlíšený od ostatných hráčov pomocou flagu.

Vytvorenie defaultného lowskillu a highskillu
Vytvorenie defaultného low a high skillu, ktorý by sa vykonával ak žiaden low/highskill nevyhovuje - hrráč by stál na mieste a otáčal by hlavou za účelom získavania informácií o stave ihriska.

Logger - zmena spôsobu logovania
V projekte nie je konzistentne dodržiavané logovanie

Stabilne zastavenie lowskillu walk_turbo
Po pridaní lowskillu walk_turbo (turbo_walk(20130414_054044_1634) od Petera Paššáka) hráč síce chodí rýchlo, ale kedže tento lowskill nebol optimalizovaný aj pre stabilné zastavenie, často aj padá.

Yaml setting súbor
V časi ruby bolo možné nastavenia meniť cez súbor, po prepise do Javy to prestalo vyť možné. Bolo by vhodné vytvoriť súbor s nastaveniami vo formáte YAML, využiť knižnicu na parsovanie YAML a umožniť nastavenia pomocou súboru.

Nastavenia projektu

Nastavenia projektu sa dajú kustomizovať v triede Settings. Väčšina nastavení je nemenných, zaujímavé sú verzia servera, ktorá je momentálne nastavená a kód je prispôsobený pre verziu 0.6.7. Ďalej sa v triede menia názov tímu, počet hráčov, či testovacia taktika (debuggingTacticName), ktorá keď sa zadá, nemení sa taktika počas celého behu programu. Výhodné je to na testovanie nižších vrstiev - highskillov a lowskillov.

Inštalačná príručka

Rozsiahlu inštalačná príručka pre viaceré operačné systémy môžno nájsť na tímovej wiki na adrese http://team09-13.ucebne.fiit.stuba.sk/wiki/index.php/N%C3%A1vody_a_in%C5%A1tal%C3%A1cie

Pred načítaním projektu do Eclipse je potrebné zmeniť súbor .project.sample - vzorový project súbor pre Eclipse.

Pri iniciálnom pullnutí Git repozitáru si programátor musí skopírovať .project.sample súbor na súbor s názvom .project a až potom spustiť Eclipse. Tento postup sa volí preto, lebo niektorí členovia tímu cez Eclipse mali nainštalovanú inú verziu (iný zdroj) pre knižnicu aspektov aká je použitá v projekte a potrebovali používať tú vzhľadom na ich inštaláciu systému. Stane sa vám to ak si inštalujete Spring. Takto si ktokoľvek môže prispôsobiť závislosti projektu bez zmeny, ktorá by ovplyvnila nastavenia PC ostatných členov tímu.

Používateľká príručka

Pohyb
Balík sk.fiit.jim.agent.highskill.move.

1. Pridanie xml s pohybom do adresára moves
2. Pridanie xml s anotáciou pre pohyb do moves/annotations
3. Pridanie ďalšieho typu rýchlosti pohybu do MovementHighSkill

pr. pridanie WALK_NEW do zoznamu.

WALK_SLOW(1), WALK_MEDIUM(2), WALK_FAST(3), WALK_NEW(4);

3. V balíku sk.fiit.jim.agent.highskill.move treba vytvoriť odvodenú triedu od Walk s protected konštruktorom a preťažiť funkcie
String getAnnotationName() a
String getLowSkillName().
Je potrebné vytvoriť statickú funkciu s názvom getInstance() vracajúcu inštanciu danej triedy. Ak pohyb má svoje špecifiká ako napr. stabilizáiu alebo rozbiehanie, je potrebné preťažiť aj funkciu LowSkill pickLowSkill().

Príklad novej triedy

```
public class WalkNew extends Walk {
	private static WalkNew instance = null;
}

protected WalkNew(){
}

@Override
public String getLowSkillName() {
	return "walk_new";
}

@Override
public String getAnnotationName() {
		return "walk_new";
	}

public static WalkNew getInstance() {
	if( instance == null){
```

```
                instance = new WalkNew();
        }
        return instance;
}
```

4. V triede MovementSkills vo funkcií getWalkSuitability je potrebné pridať nastavenie koeficientov pre MovementSpeedEnum nového pohybu, ktorý bol vytvorený v kroku 3. Koeficienty treba nastaviť experimentálne podľa toho, čomu sa dáva priorita pri danom pohybe.

pr.

```
else if (speedEnum == MovementSpeedEnum.WALK_NEW){
        speed_koef = 1.9;
        range_koef = 0.2;
        uccess_koef = 0.01;
}
```

5. V triede MovementSkills v MovementEnum treba pridať daný enum s vytvorením inštancie triedy pohybu z kroku 3. a preťažiť computeSuitability a zavolať funkciu pre výpočet suitability.

pr.

```
WALK_NEW(WalkNew.getInstance()) {
        @Override
        public double computeSuitability(double relativeDistanceToTarget,
MovementSpeedEnum speedEnum) {
                return getWalkSuitability(relativeDistanceToTarget, speedEnum);
                }
};
```

6. V taktike stačí zavolať nový pohyb pri vhodnej situacií na základe enum parametra
napr.
this.moveExec.move(Vector3D.cartesian(ballX, ballY, 0), MovementSpeedEnum.WALK_NEW);
O taktikách, situáciach a volaní highskillov viac v danej príručke.

Kop
Balík sk.fiit.jim.agent.highskill.kick.

1. Pridanie xml s kopom do adresára moves
2. Pridanie xml s anotáciou pre pohyb do moves/annotations
3. V triede KickHighSkill treba rozšíriť KickTypeEnum o nový typ kopu.
Pr.
KICK_FAST(1), KICK_NORMAL(2), KICK_SLOW(3), KICK_NEW(4);

4. V triede Kick vo funkcii employBestLowSkill() treba pridať anotáciu pre jednu alebo / a druhú nohu s ktorou má hráč kopať podľa podmienky if (relativizedTarget.getX() < 0.0), ktorá určuje či je lopta napravo alebo naľavo od hráča.
Následne v tej istej funkcii treba nastaviť koeficienty pre suitability.
Napr.
else if (type == KickTypeEnum.KICK_NEW) {
speedKoef = 1.5;
distanceKoef = 1.5;
successKoef = 1.0;
}

5. V taktike stačí zavolať nový pohyb pri vhodnej situacií na základe enum parametra.
O taktikách, situáciach a volaní highskillov viac v danej príručke.

napr.
this.kickExec.kick(Vector3D.cartesian(ballX, ballY, 0),
KickHighSkill.KickTypeEnum.KICK_NEW);

Poznámka.
Kop používa rovnaké približovanie k lopte pre všetky kopy na základe anotácie vybraného kopu.