SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

RoboCup 3D

Tímový projekt
Používateľská príručka

Tím: Androids (tím č. 5)
Vedúci tímu: Ing. Ivan Kapustík
Členovia tímu: Bc. Juraj Belanji
Bc. Miroslav Hruška
Bc. Roman Kováč
Bc. Andrej Minárik
Bc. Veronika Wolfová
Študijný odbor: Softvérové inžinierstvo

Akademický rok: 2010/2011

Obsah

1 Úvod

Tento dokument slúži ako používateľská príručka k častiam projektu tímu Androids, ktorá bola vytvorená v predmete Tímový projekt na tému RoboCup 3D v akademickom roku 2010/2011. V tejto príručke sa nachádzajú návody na použitie nástrojov na úspešné spustenie simulácie agenta – robota – pre tému simulovaného futbalu RoboCup 3D.

V prvej časti dokumentu sa nachádza krátky návod na inštaláciu a spustenie servera. Ide o verziu servera 0.6.3 (resp. 0.6.4).

V druhej časti je návod na spustenie agenta JIM. Tu sa nachádza aj návod na spustenie simulácie.

Tretia časť sa zaoberá tvorbou pohybov.

V štvrtej, poslednej časti tejto príručky sa nachádza návod na inštaláciu a spustenie testovacieho frameworku, ktorý slúži na testovanie agentov. Nachádza sa tu aj návod na písanie jednotlivých testov pre agenta JIM.

2 Inštalácia a spustenie servera

V tejto časti dokumentu sa nachádza návod na inštaláciu simulačného prostredia (servera) pre OS Windows, ktoré je potrebné pre následné spustenie agenta a simulácie. Ide o verziu servera 0.6.3 (resp. 0.6.4). Inštalácia simulačného prostredia pozostáva z nasledovných krokov:

1. Inštalácia najnovšej verzie Microsoft Visual C++ 2008 Redistribution Package, ktorá je dostupná na: http://www.microsoft.com/downloads/en/details.aspx?FamilyID=9b2da534-3e03-4391-8a4d-074b9f2bc1bf&displaylang=en

2. Inštalácia SimSpark dostupného na: http://sourceforge.net/projects/simspark/files/simspark/0.2.1/simspark-0.2.1-win32.exe/download

3. Inštalácia servera:

    a. Verzia 0.6.3 servera je dostupná na: http://sourceforge.net/projects/simspark/files/simspark/0.2.1/simspark-0.2.1-win32.exe/download

    b. Verzia 0.6.4 servera je dostupná na: http://sourceforge.net/projects/simspark/files/rcssserver3d/0.6.4/rcssserver3d-0.6.4-win32.exe/download

Po inštalácii sa dá server a monitor spustiť (pri zachovaní default súborov inštalácie) nasledovne:

4. `C:\Program Files\rcssserver3d 0.6.3\bin\simspark.cmd`

5. `C:\Program Files\rcssserver3d 0.6.3\bin\rcssmonitor3d.cmd`

Oficiálny návod na inštaláciu je možné prezrieť na stránke:
http://simspark.sourceforge.net/wiki/index.php/Main_Page

3 Inštalácia a spustenie agenta JIM

Táto kapitola dokumentu sa zaoberá inštaláciou a spustením prostredia, v ktorom bol vyvíjaný agent, ako aj inštaláciou balíkov potrebných pre spustenie agenta.

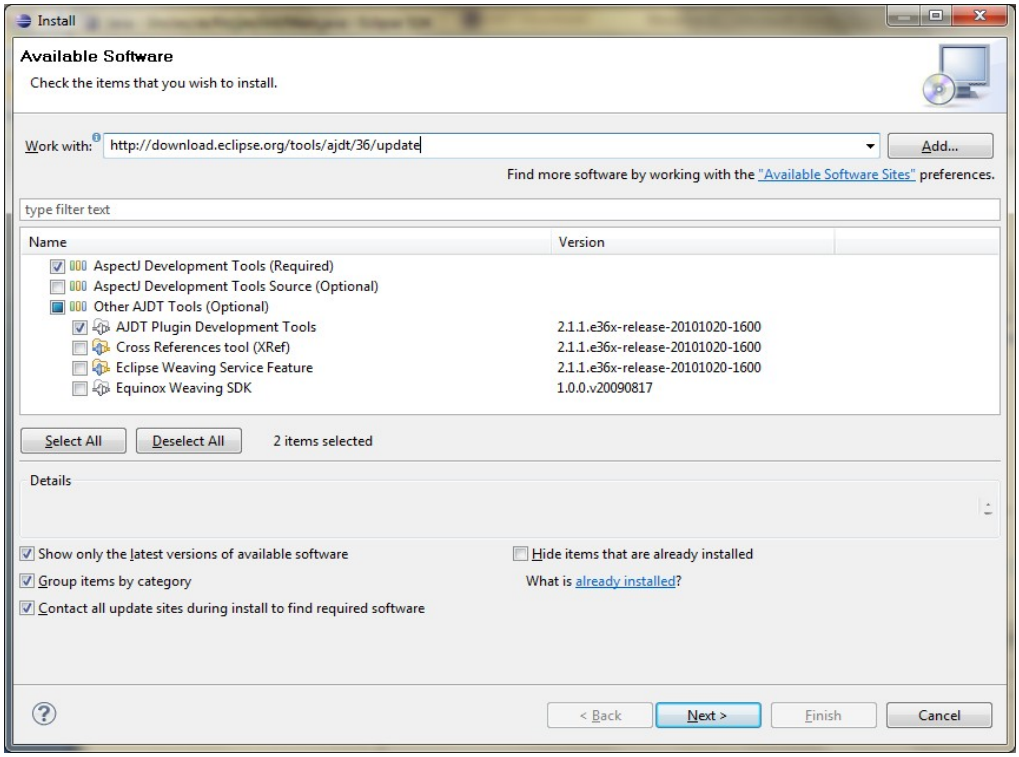
3.1 Nastavenie SVN

Pre spustenie agenta najprv potrebujeme vo vývojovom prostredí Eclipse nastaviť nasledovné:

AspectJ

Potrebné kvôli GUI v JIM-ovi.

1. Compiler
    a. Stiahnuť zo stránky http://www.eclipse.org/aspectj/downloads.php aspectj-1.6.10.jar
    b. Nainštalovať príkazom java -jar <umiestnenie stiahnutého jar> (v prípade OS Windows treba spustiť cmd ako admin)
    c. Postupovať podľa inštrukcií
2. Plugin do eclipsu
    a. Spustiť Eclipse
    b. Ísť do Help -> Install new software, tam zadať http://download.eclipse.org/tools/ajdt/36/update
    c. Zaškrtnúť časti zobrazené na tomto obrázku



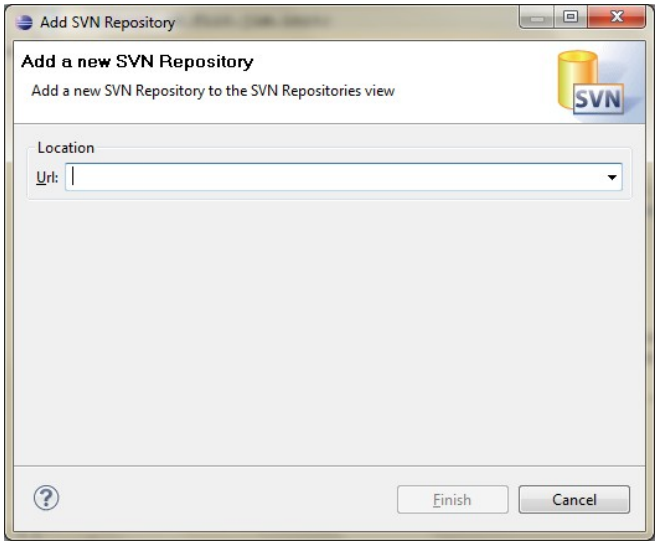
    d. Preklikať, potvrdiť všetko, reštartovať Eclipse

Subclipse

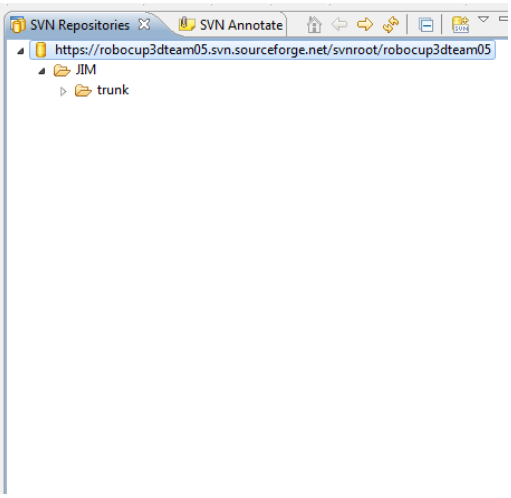
1. Spustiť Eclipse
2. Ísť do Help -> Install new software, tam zadať http://subclipse.tigris.org/update_1.6.x
3. Zaškrtnúť Subclipse
4. Preklikať, potvrdiť všetko, reštartovať Eclipse

SVN repozitár

1. Spustiť Eclipse
2. Vpravo hore kliknút na Open perspective -> Other -> SVN Repository
3. Keď sa otvorí perspektíva, vľavo kliknúť na Add SVN Repository
4. Zobrazí sa

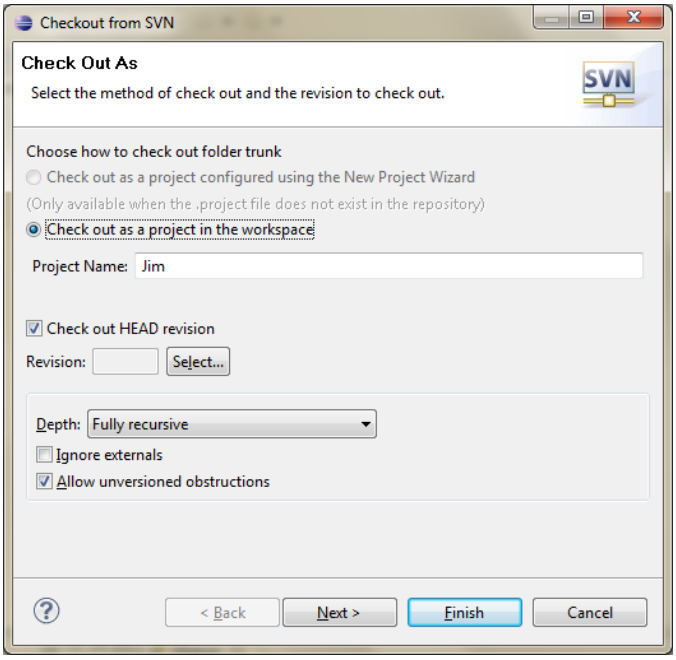


5. Zadať https://robocup3dteam05.svn.sourceforge.net/svnroot/robocup3dteam05
6. Čakať
7. Dostať sa do takéhoto stavu



8. Pravým kliknúť na adresár *trunk* a dať Checkout

9. Zobrazí sa



10. Nič nemeniť (jedine názov projektu je možné zmeniť)
11. Finish
12. Dlho čakať
13. Môže sa stať, že to zamrzne. V tom prípade vypnúť Eclipse a opakovať Checkout.
14. Ak to nezamrzlo, prepnúť späť perspektívu na Java a mal by tam byť funkčný JIM aj s GUI, ktoré sa zobrazí pri spustení JIM-a.

3.2 Nastavenie a spustenie agenta

Po ukončení inštalácie prostredia je agenta možné spustiť zo samotného vývojového prostredia alebo pomocou binárneho súboru, ktorý sa vygeneruje (pre OS Windows). Pred spustením agenta je samozrejme potrebné spustiť simulačný server (monitor nemusí byť spustený, avšak v tom prípade nebudeme vidieť, čo sa deje počas simulácie).

Pred samotným spustením agenta musíme najprv nakonfigurovať nasledovné:

1. V súbore scripts/config/settings.rb v riadku `Communication.instance.server_ip` potrebujeme zmeniť IP adresu podľa toho, kde beží simulačný server, na ktorý sa ideme pripojiť (127.0.0.1 ak ide o local host)
2. Pre nastavenie verzie servera je potrebné zmeniť `VERSION_0_6_X` (kde X treba nahradiť za 3 alebo 4 v závislosti od toho, na ktorom serveri sa vykonávajú simulácie)
3. Agent nerobí nič, kým sa nachádza v „BeforeKickOff“ móde. Je nutné spustiť hru (stlačením „b“ v monitore).

4 Tvorba pohybov

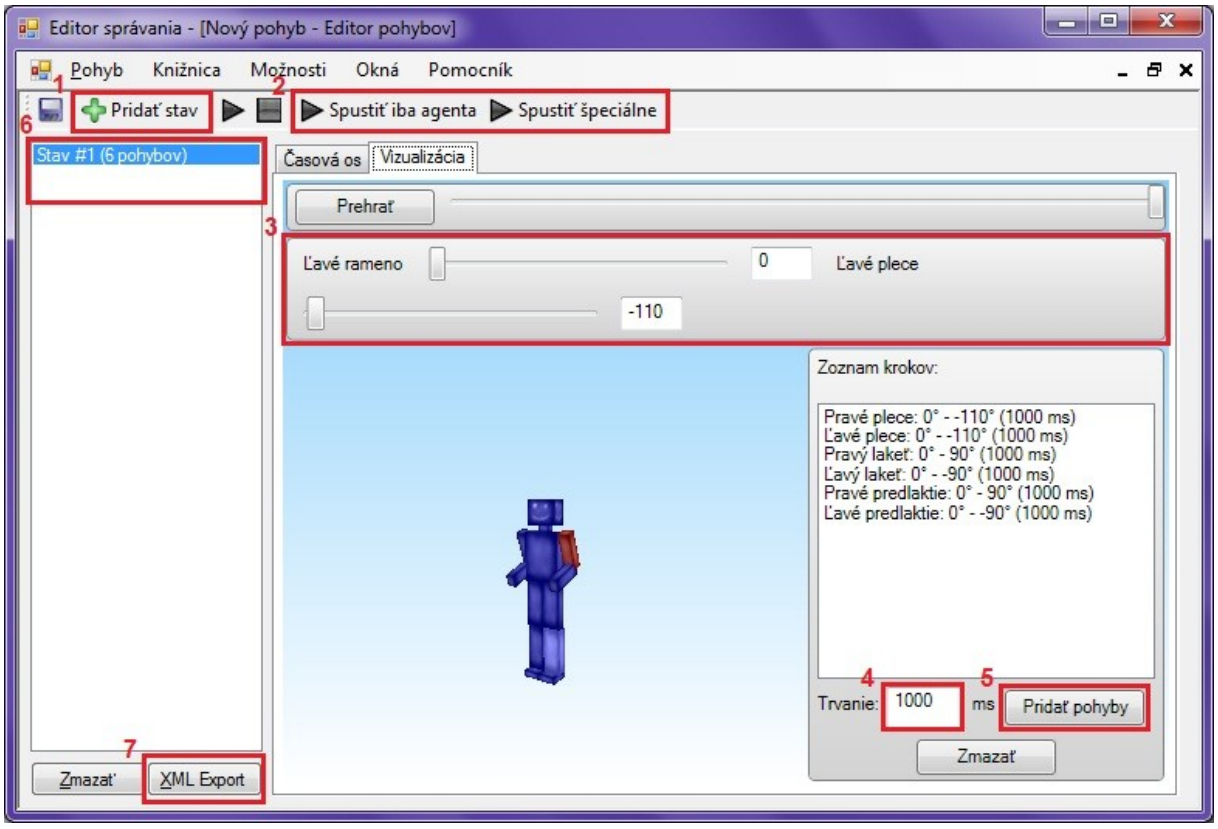
Pohyby sú dôležitým prvkom hráča, preto sme im aj my venovali veľkú pozornosť. K ich tvorbe možno pristupovať rôzne podľa toho, komu čo viac vyhovuje a kto je ako skúsený a akú má predstavivosť. V nasledovnom texte sú zhrnuté dostupné pomôcky, ktoré sme pri tvorbe pohybov používali, možné prístupy a know-how, ktoré sme počas našej práce na pohyboch získali.

4.1 Pomôcky

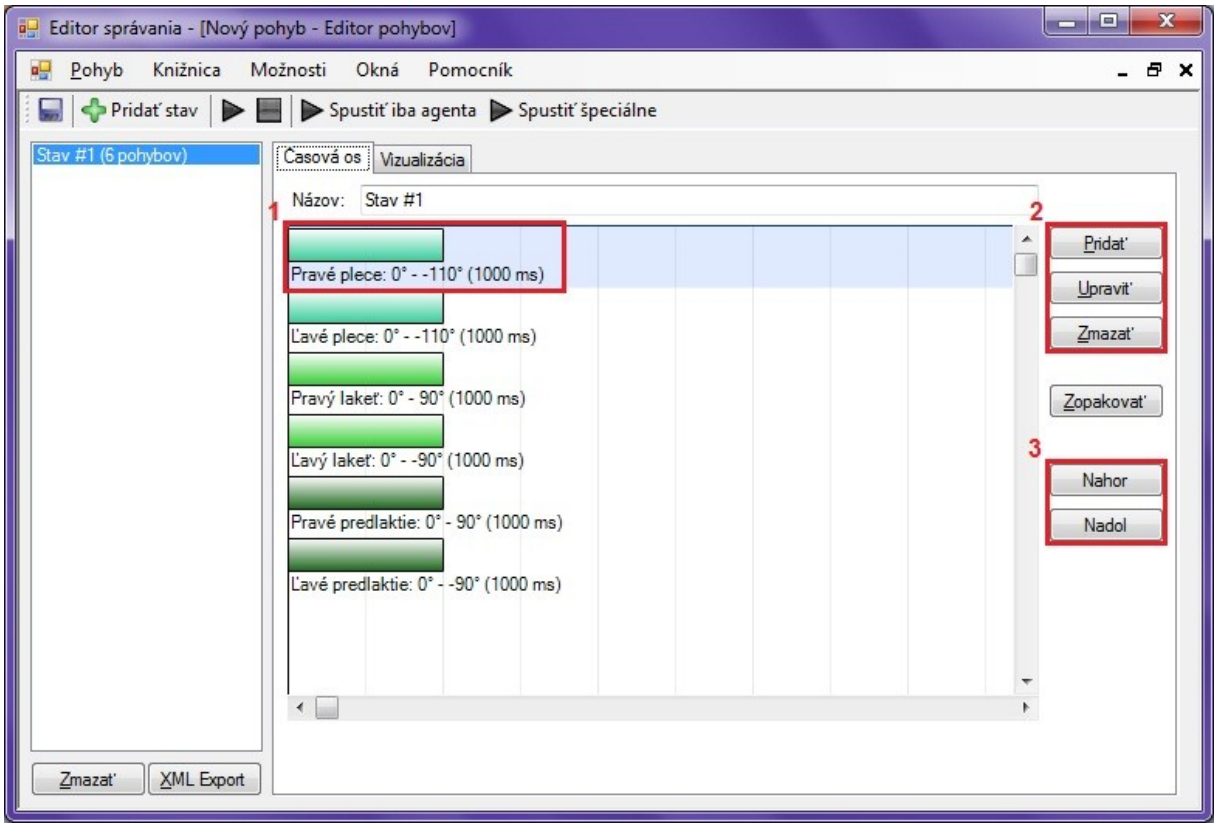
Pri tvorbe pohybov sme v zásade používali dve pomôcky. Prvou je grafický editor pohybov, ktorý vytvoril tím Agenty 007, druhou obrázok anatómie robota NAO, ktorý vytvoril člen nášho tímu.

4.1.1 Grafický editor pohybov

Je to užitočný nástroj na tvorbu pohybov hlavne pre začiatočníkov a v prípade, že si nevieme úplne dobre predstaviť, ako sa do zložitejšieho pohybu zapájajú jednotlivé kĺby. Taktiež sa dá použiť na rýchle prototypovanie pohybov, ktoré sa následne ručne dolaďujú priamo v príslušných XML súboroch. Na Obr. 1 a Obr. 2 sú zobrazené dôležité obrazovky grafického editora pohybov spolu so zvýraznením hlavných prvkov používaných pri tvorbe pohybov, na ktoré sa v časti *4.2.2.1 Použitie editora pohybov* budeme odvolávať.



Obr. 1: *Editor pohybov – vizualizácia, model robota*



Obr. 2: *Editor pohybov – časová os*

Nevyužívali sme všetky funkcie, ktoré tento nástroj poskytuje, keďže sme ich nepotrebovali a ani neboli navrhnuté pre agenta JIM. Treba zdôrazniť, že aj keď tento nástroj je veľmi užitočný, nie je podľa nás postačujúci pre plnohodnotnú tvorbu pohybov, hlavne kvôli množstvu nedostatkov, ktoré by bolo treba odstrániť, a odlišnej logike pohybov. Vzhľadom na priority a smerovanie nášho tímu jediným zásahom do editora pohybov z našej strany bol export XML súboru pohybu v štruktúre, akú požaduje hráč JIM, aby bola táto pomôcka výhodne použiteľná. Otvárajú sa široké možnosti ďalšieho vylepšovania a ladenia použitého editora pohybov.

Poznámka: Inštalácia je jednoduchá. Treba si stiahnuť RAR súbor tímu Agenty 007, ktorý obsahuje adresáre *Build* a *Server*. Tento súbor je nutné rozbaliť a v adresári *Build* nahradiť pôvodný súbor *RobotBehaviourEditor.exe* naším upraveným *RobotBehaviourEditor.exe* súborom, ktorý obsahuje XML export pohybov pre agenta JIM.

4.1.2 Obrázok anatómie robota NAO

Na tomto obrázku sú znázornené jednotlivé kĺby robota spolu s ich rozsahmi a osami otáčania. Dostupný je v dokumentácii k produktu na str. 66. Je výraznou podporou hlavne pri tvorbe pohybov priamym písaním XML súborov a pri dolaďovaní pohybov vytvorených v editore pohybov.

4.2 Možné spôsoby

V zásade existujú dva prístupy k tvorbe pohybov:

- bez použitia grafického editora pohybov
- s použitím grafického editora pohybov

Hlavne pri prvom spôsobe je nutné dbať na to, aby mali fázy jedinečné mená nie len v rámci pohybu, ale aj v kontexte ostatných pohybov. Pri druhom spôsobe sa nám o jedinečnosť názvov fáz v rámci pohybu stará export do XML a v kontexte pohybov tiež, ak nevyexportujeme dva rôzne pohyby pod rovnakým menom.

4.2.1 Tvorba pohybov bez použitia grafického editora pohybov

Tento spôsob je vhodný pre skúsenejšieho vývojára s dobrou predstavivosťou. V zásade ide o priame písanie XML súboru podľa šablóny nižšie a najväčšou výzvou tohto prístupu je zvoliť správne kĺby, ich koncové uhly a rozdelenie pohybov kĺbov do fáz.

```
<?xml version="1.0" encoding="utf-8"?>
<robot xsi:noNamespaceSchemaLocation="moves.xsd">
  <constants>
  </constants>
  <low_skills>
    <low_skill name="meno_pohybu" firstPhase="meno_prvej_fazy_pohybu" />
  </low_skills>
  <phases>
    <phase name="meno_fazy_pohybu" next="meno_nasledujucej_fazy_pohybu">
      <effectors>
      <oznacenie_klbu end="koncova_hodnota" />
      ...
      </effectors>
      <duration>trvanie_fazy_v_ms</duration>
    </phase>
    ...
      <phase name="meno_fazy_pohybu" next="meno_nasledujucej_fazy_pohybu"
isFinal="true">
      <effectors>
      ...
      </effectors>
      <duration>trvanie_fazy_v_ms</duration>
      <finalize>meno_prvej_ukoncovacej_fazy</finalize>
    </phase>
  </phases>
</robot>
```

Dôležitou fázou hlavne pri cyklických pohyboch, akým je napríklad chôdza, je fáza, v ktorej je možné pohyb pri požiadavke preplánovania ukončiť. Táto ukončovacia fáza sa vyznačuje príznakom isFinal="true" a v prípade požiadavky na preplánovanie sa pohyb vykonáva, až kým nedosiahne práve takúto fázu. Následne sa vykonajú fázy ukončenia, z ktorých prvá je uvedená v časti <finalize></finalize>. Cyklické pohyby majú aj v ukončovacej fáze uvedenú nasledujúcu fázu, ktorou sa pokračuje, kým nepríde požiadavka preplánovať pohyby, jednorázové pohyby atribút *next* v tejto fáze uvedený nemajú. Fáza s príznakom *isFinal* je dôležitá pre všetky pohyby.

4.2.2 Tvorba pohybov s použitím editora pohybov

Tento prístup je vhodný na rýchle vytvorenie aj zložitejšieho pohybu, ktorý si nevieme úplne presne predstaviť a rozdeliť v hlave do fáz. Najskôr si „vyklikáme“ pohyb v editore pohybov a vyexportujeme ho vo formáte XML pre agenta JIM. Je vysoko pravdepodobné, že tento pohyb bude nutné aspoň v minimálnej miere ďalej manuálne upraviť a doladiť priamo v XML, hlavne ak je to pohyb cyklický, keďže editor nekonečné cyklenie nepodporuje.

4.2.2.1 Použitie editora pohybov

Spustíme si aplikáciu a v hlavnom okne zvolíme možnosť *Nový pohyb*. Zobrazí sa nám hlavná obrazovka na tvorbu pohybu, zatiaľ bez záložiek *Časová os* a *Vizualizácia*. Najskôr si potrebujeme vytvoriť nový stav, preto klikneme na *Pridať stav* (Obr. 1, 1). Názov stavu nie je podstatný. Po kliknutí na novovytvorený stav sa nám zobrazia spomínané záložky a môžeme začať tvoriť nový pohyb. V tomto momente máme dve možnosti:

1. tvoriť pohyb v záložke *Časová os*
2. tvoriť pohyb v záložke *Vizualizácia*

Keďže prvý postup je len nešikovnejšou obdobou priameho písania pohybu v XML, opíšeme druhý postup, ktorý je hlavnou výhodou editora pohybov.

Prejdeme do záložky *Vizualizácia*. Kliknutím na časť tela robota sa nám sprístupnia posuvníkové nastavenia k nej prislúchajúcich kĺbov (Obr. 1, 3). Postupne nastavíme koncové polohy kĺbov, do ktorých sa chceme dostať v prvej fáze. Následne nastavíme trvanie fázy v milisekundách (Obr. 1, 4) a pridáme pohyby kĺbov (fázu) do stavu (celkového pohybu) tlačidlom *Pridať pohyby* (Obr. 1, 5). Rovnako nastavujeme a pridávame ďalšie fázy celkového pohybu.

Keď už máme celý pohyb „vyklikaný“ alebo len chceme vidieť priebežný výsledok, môžeme si pohyb prehrať. Tu máme znovu viac možností:

1. prehrať pohyb priamo v editore ako simuláciu modelu robota
2. použiť obyčajné tlačidlo „play“, ktoré spustí server, monitor a hráča, ktorého nechá vykonať náš pohyb (toto však nie vždy úspešne zafunguje, záleží od výkonu stroja)
3. spustiť externe server s monitorom a následne použiť možnosť *Spustiť iba agenta* (Obr. 1, 2)

4. spustiť externe server s monitorom a následne použiť možnosť *Spustiť špeciálne* (Obr. 1, 2) – toto je vhodné, ak chceme pohyb spomaliť/zrýchliť bez zásahu do časovania fáz

Vo všetkých prípadoch je nutné skontrolovať, či v zátvorke za stavom (Obr. 1, 6) je správny počet pohybov, teda či sa pohyby pridali do stavu, a ak číslo v zátvorke nie je aktuálne, treba kliknúť do časti okna, v ktorej je stav zobrazený (ale pozor, nie na stav samotný – toto zmaže neaplikované zmeny, pridané pohyby kĺbov).

Záložka *Vizualizácia* je vhodná len na sekvenčné pridávanie fáz do pohybu. Ak potrebujeme pohyb nejakého kĺbu upraviť, odstrániť alebo pridať do inej ako novej poslednej fázy, musíme tak spraviť v záložke *Časová os*. Tu sú jednotlivé kĺby farebne aj menne odlíšené a príslušnosť k fáze je vyjadrená vzdialenosťou od začiatku – ľavej strany obrazovky (Obr. 2, 1).

Nový pohyb kĺbu pridáme stlačením *Pridať* (Obr. 2, 2). Otvorí sa nám okno s nastaveniami pohybu. Najskôr zvolíme kĺb a čas začiatku jeho pohybu. Kliknutím do *Začiatočný uhol natočenia kĺbu* sa nám aktualizuje uhol, v ktorom sa kĺb nachádza na začiatku pridávaného pohybu. Ostáva už len nastaviť želaný koncový uhol, trvanie pohybu (fázy, do ktorej bude patriť), kliknúť do poľa *Uhlová rýchlosť*, aby sa nastavila podľa zadaných parametrov a potvrdiť pridanie tlačidlom *Pridať*. Poradie klikania je dôležité, keďže aplikácia nie je úplne doladená a pri zmene poradia sa hodnoty neočakávaným spôsobom prestavujú.

Existujúce pohyby upravíme tak, že klikneme na pohyb na časovej osi a stlačíme *Upraviť* (Obr. 2, 2) alebo na pohyb na časovej osi dvojklikneme. Zobrazí sa rovnaké okno ako pri pridávaní nového pohybu.

Tlačidlo *Zmazať* slúži na odstránenie pohybu kĺbu. Pohyb kĺbu nie je možné odstrániť tlačidlom *Zmazať* v záložke *Vizualizácia*. Pri jeho použití síce zmizne zo zoznamu, ale na časovej osi ostane a v stave tiež (pri pokuse o aktualizáciu počtu pohybov v stave sa vráti späť do zoznamu). Tlačidlá *Nahor* a *Nadol* (Obr. 2, 3) slúžia na presunutie nových alebo upravených pohybov tak, aby boli zoskupené podľa príslušnosti k fáze.

Keď už sme s pohybom spokojní alebo sa rozhodneme pokračovať v jeho ladení ručne v XML, stlačíme *XML export* (Obr. 1, 7), zadáme názov súboru, ktorý bude zároveň aj názvom pohybu a predponou názvov fáz, zvolíme miesto uloženia a operáciu potvrdíme. Predpokladom pre úspešné využitie grafického editora pohybov ako pomôcky pri tvorbe pohybov pre agenta JIM je pozornosť a disciplína vývojára a uvedomenie si možností a obmedzení, keďže nástroj nebol vytvorený na vývoj pohybov pre agenta JIM.

5 Inštalácia a spustenie testovacieho frameworku

Centrálnou funkcionalitou testovacieho frameworku je už zo samotného názvu testovanie tak samotných základných pohybov, ako aj vyššej logiky. Na to, aby bolo možné testovať, je potrebné implementovať abstraktnú triedu

`sk.fiit.testframework.trainer.testsuite.TestCase`

Po implementácii uvedenej triedy je možné vykonávať testovanie, o ktorého vykonávanie sa stará trieda

`sk.fiit.testframework.trainer.testsuite.tester.TestCaseTester`

5.1 Implementácia abstraktnej triedy TestCase

Pre úspešné použitie triedy `TestCase` pomocou `TestCaseTester` je potrebné implementovať abstraktné metódy. Najlepším príkladom ukážky ako vytvoriť vlastný `TestCase` je nasledovný príklad, ktorý testuje úspešnosť chôdze.

```
public class WalkTestCase extends TestCase {
    private AgentJim agent;
    private Vector3 initPos;

    public WalkTestCase(RobocupServerAddress address, AgentJim agent) throws
    IOException {
        super(address);
        this.agent = agent;
        initPos = new Vector3(0, 0, 0.4);
    }
    @Override
    public void init() {
        try {
            trainer.setPlayMode(PlayMode.PlayOn);
            trainer.setAgentPosition(agent, initPos.asPoint3D());
        } catch (IOException ex) {
            throw new RuntimeException(ex);
        }
    }
    @Override
    public TestCaseResult evaluate(SimulationState ss) {
        Player p = ss.getScene().getPlayers().get(0);
        if (p.isOnGround()) {
            return new TestCaseResult(-1);
        } else
            return new TestCaseResult(initPos.getXYDistanceFrom(initPos));
    }
    @Override
    public boolean isStopCriterionMet(SimulationState ss) {
        Player p = ss.getScene().getPlayers().get(0);
        if (p.isOnGround() || getElapsedTime() > 5)
            return true;

        return false;
```

```
} }
```

Konštruktor

V konštruktore priradíme testu fakty, ktoré sú počas celej existencie testu nemenné. V našom prípade je to konkrétne:

- agent
- a objekt špecifikujúci počiatočnú pozíciu (východisková pozícia v rámci každého opakovania testu)

init()

V metóde `init()` sa nachádzajú udalosti, ktoré sa majú udiať pri každom jednotlivom spustení testu. V našom prípade je to:

- nastavenie módu hry (rôzne testy môžu vyžadovať rôzne módy)
- nastavenie pozície agenta na počiatočnú pozíciu

isStopCriterionMet(SimulationState ss)

Metóda slúži na vyhodnotenie kritéria zastavenia testu. V našom prípade metóda vráti `true` ak:

- hráč je na zemi
- ubehla doba trvania testu

evaluate(SimulationState ss)

Táto metóda je zavolaná vtedy, keď sa vyhodnotí `isStopCriterionMet` na `true`. V prípade testovania chôdze to znamená, že buď agent spadol na zem (vtedy hodnota *fitness*, ktorá je zapuzdrená v objekte typu `TestCaseResult` je rovná -1.0), alebo nespadol a vtedy sa *fitness* chôdze rovná vzdialenosti, ktorú prešiel od počiatočnej polohy.

5.2 Volanie testu

Definícia testu v predchádzajúcej časti je prvým nevyhnutným krokom k použitiu testovania. Ďalším krokom je následné zavolanie testu priamo na serveri. Podobne ako v predchádzajúcom prípade bude najlepším spôsobom opisu fungovania testovania príklad, ktorý spustí vytvorený test. Uvedieme jadro príkazov, ktoré umožnia vykonanie testu.

```
public static void main(String[] args) throws UnknownHostException,
IOException {
    RobocupServerAddress address =
        new RobocupServerAddress(InetAddress.getLocalHost(), 3100, 3200);
```

```
    AgentJim agentJim = new AgentJim(address,
        new AgentJim.Data(10, Team.Left, 3070));

    TestCase testCase = new WalkTestCase(address, agentJim);

    TestCaseTester tester = new TestCaseTester(address, testCase);

    System.out.println(tester.runOnce());
}
```

Uvedený kód na volanie testu spočíva v nasledovnom:

1. `RobocupServerAddress` slúži na definovanie pripojenia na server, zahŕňa hostname, port servera a port monitora.
2. `AgentJim` špecifikuje agenta, konkrétne v tomto prípade už bežiaceho agenta na pozadí. Pomocou tejto definície budeme v nasledovných krokoch k agentovi pristupovať.
3. `TestCase` nám slúži na definíciu testu, ktorý budeme vykonávať.
4. `TestCaseTester` je objekt, ktorý nám umožňuje test vykonať.
5. V poslednom kroku vypíšeme výsledky testu po jednonásobnom vykonaní testu.

Uvedené príkazy sú jadrom vykonania testu. Na základe výsledkov z testu môžeme následne aplikovať strojové učenie, poslať agentovi zmenu jeho správania a podobne. Tieto zmeny, ktoré má agent vykonať, sa uskutočňujú prostredníctvom posielania súborov a skriptov. Tieto metódy sú implementované v triede `Trainer` a sú to nasledovné metódy:

```
public void executeRubyScript(AgentJim agent, String script);
public void invokeReplan(AgentJim agent);
```