Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 812 19 Bratislava

Tímový projekt

Simulácia demonštrácie v meste

Dokumentácia k inžinierskemu dielu

Tím č. 9 - SimTeam

Bc. Jana Branišová
Bc. Adrián Kollár
Bc. Michal Kyžňanský
Bc. Miroslav Ort
Bc. Michal Ošvát
Bc. Filip Pakan

Vedúci tímového projektu: Ing. Peter Lacko, PhD.
Mailový kontakt na tím: tim2012@googlegroups.com
Akademický rok: 2012/2013

Obsah

1 Úvod

Dokument k inžinierskemu dielu sa zaoberá problematikou simulovania demonštrácií a poskytuje prehľad o problematike a činnostiach, ktoré sa uskutočnili s cieľom navrhnúť a realizovať riešenie. Pritom tím postupoval podľa metodiky SCRUM.

1.1 Účel a rozsah dokumentu

Dokument sa zaoberá analýzou, návrhom a implementáciou možného riešenia simulácie demonštrácie v meste.
Dokument je výsledkom práce členov tímu SimTEAM v rámci predmetu Tímový projekt.

1.2 Prehľad dokumentu

Účel a prehľad dokumentu je v kapitole 1. Ďalej je dokument rozdelený na kapitoly podľa šprintov. Každá kapitola zodpovedá jednému šprintu. V kapitole sú vždy opísané všetky činnosti, ktoré tím vykonal v prvom šprinte. Na začiatku kapitoly sa nachádza prehľadná tabuľka s mierou zodpovednosti každého člena tímu za príslušnú úlohu. Následne sú jednotlivé úlohy podrobne opísané.
Prvý šprint je opísaný v kapitole 2.
Druhý šprint je opísaný v kapitole 3.
Tretí šprint je opísaný v kapitole 4.
Štvrtý šprint je opísaný v kapitole 5.
Piaty šprint je opísaný v kapitole 6.
Šiety šprint je opísaný v kapitole 7.
Siedmy šprint je opísaný v kapitole 8.
Ôsmy šprint je opísaný v kapitole 9.
Deviaty šprint je opísaný v kapitole 10.
Architektúra produktu je opísaná v kapitole 11.
Zhrnutie projektu a pohľad na jeho budúce smerovanie je sumarizovaný v kapitole 12.
Zdroje informácií a použitá bibliografia je uvedená v kapitole 13.

Používateľská príručka je uvedená v prílohe A.
Inštalačná príručka je uvedená v prílohe B.
Obsah elektronického média je uvedený v prílohe C.

2 Prvý šprint

V nižšie uvedenej tabuľke (Tabuľka 2.1) je uvedený zoznam úloh, ktoré boli počas prvého šprintu vyriešené a osoby, ktoré sa danými úlohami primárne zaoberali.

Tabuľka 2.1 Zoznam úloh a k nim prislúchajúcich riešiteľov.

Názov úlohy	Zodpovedná osoba
Analýza práce s SVN a integrácia do NetBeans	Adrián Kollár
Integrácia MASON a GeoMASON	Filip Pakan
Import mapových podkladov do knižnice MASON	Filip Pakan
Analýza nástrojov na podporu riadenia projektov	Jana Branišová
Návod na používanie Jiri	Jana Branišová
Implementácia Web-stránky	Michal Kyžňanský
Integrácia GeoMASON, MASON a RVO	Michal Kyžňanský
Vytvoriť prvý používateľský scenár	Michal Kyžňanský
Analýza poriadkových zložiek	Michal Ošvát
Analýza benchmarku simulácie demonštrácie	Jana Branišová, Miroslav Ort
Analýza modelov správania	Adrián Kollár, Michal Kyžňanský, Miroslav Ort, Filip Pakan
Analýza šírenia emócií	Jana Branišová, Miroslav Ort

2.1 Analýza práce s SVN a integrácia do NetBeans

2.1.1 Úloha

Analýza práce s SVN a integrácia do NetBeans.

2.1.2 Analýza

2.1.2.1 Checkout

Checkout je potrebné vykonať ak ešte nemáte stiahnutú lokálnu kópiu projektu z SVN repozitára. Stačí ho teda vykonať iba jedenkrát a po stiahnutý projektu z repozitára budete používať príkazy *update* na stiahnutie aktuálnej verzie a *commit* na nahratie súborov do repozitára. *Checkout* sa v prostredí Netbeans vykonáva nasledovne:

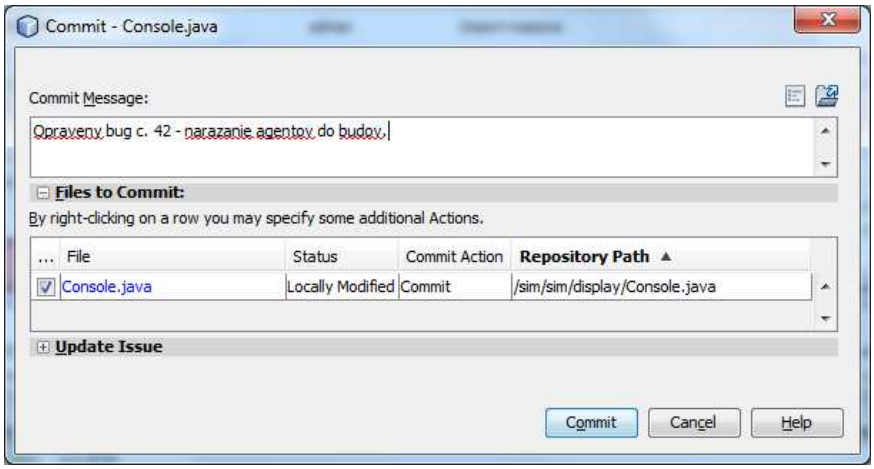
1. Kliknete na Team > Subversion > Checkout
2. Ako repository URL zadáte: svn://team09-12.ucebne.fiit.stuba.sk:443/svn/sim
3. Do políčok User a Password napíšete Vaše prihlasovacie meno a heslo pre SVN server
4. Kliknete na tlačidlo Next
5. Repository folder necháte pôvodný a upravíte si Local folder - adresár do ktorého sa stiahne lokálna kópia

6. Kliknete na tlačidlo Finish a počkáte kým sa stiahne lokálna kópia
7. Po stiahnutí sa Netbeans opýta či chcete zo stiahnutého projektu vytvoriť projekt v Netbeanse – zvolíte áno

2.1.2.2 Commit

Príkaz *commit* uloží premietne zmeny vykonané v lokálnej kópii do SVN repozitára. Je ho teda vhodné použiť po dokončení nejakej ucelenej funkcionality a následnom otestovaní ich funkčnosti. *Commit* sa vykoná nasledovným postupom:

1. Zvolíte Team > Show changes
2. V dolnej časti prostredia sa zobrazia všetky modifikované súbory
3. Zvolíte Commit All
4. V otvorenom okne do časti Commit Message napíšete správu ktorá bude obsahovať popis vykonaných úprav
5. V časti Files to Commit zvolíte požadované súbory na commit

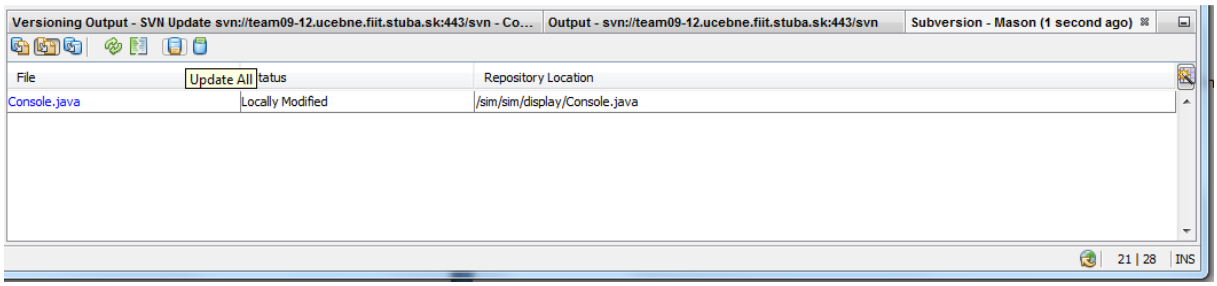


Obr. 2:1 Príkaz Commit.

2.1.2.3 Update

Update súborov aktualizuje lokálnu kópiu projektu podľa najnovšej verzie z repozitára. Je vhodné ho vykonať pred začatím práce na projekte a takisto pred vykonaním príkazu *commit*. *Update* vykonané zmeny v lokálnej kópii zachováva a aktualizuje len časti kódu, ktoré neboli modifikované. V prípade, že do SVN repozitára bola uložená novšia verzia presne tej istej časti kódu s ktorou ste pracovali, je následne nutné vykonať príkaz *resolve conflicts*. *Update* je možné vykonať nasledovne:

1. Označíte projekt ľavým tlačidlom myši
2. Zvolíte Team > Show changes
3. V dolnej časti prostredia sa zobrazí okno v ktorom zvolíte Update all.

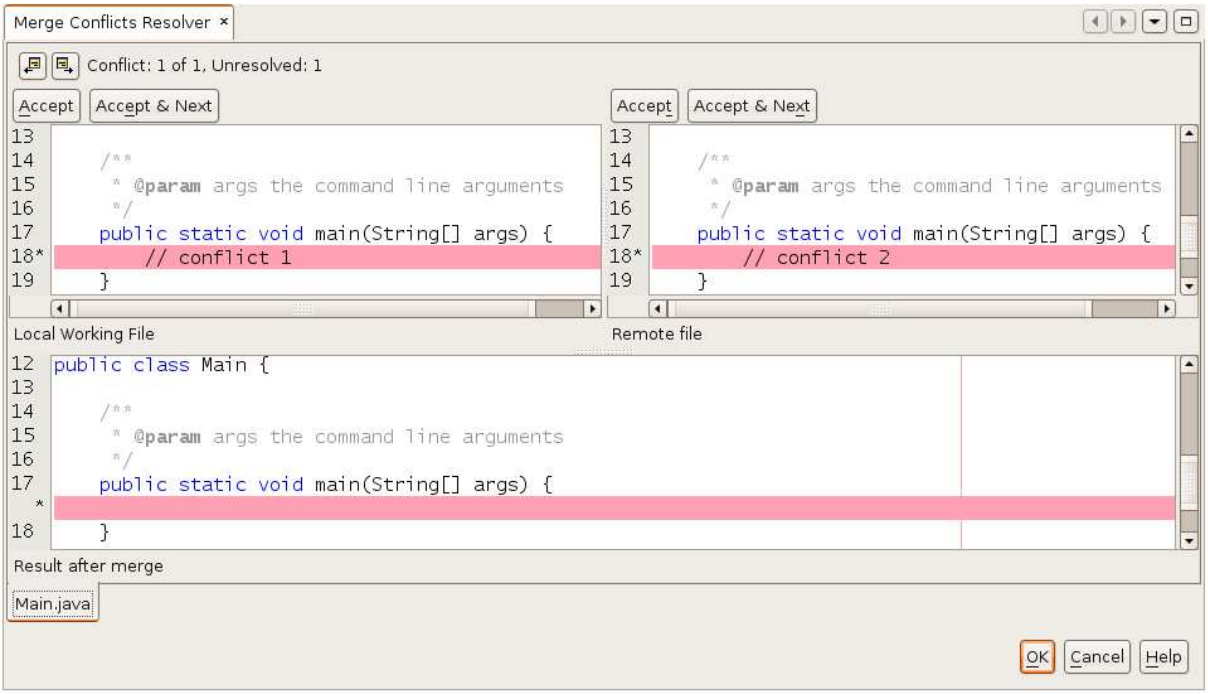


Obr. 2:2 Príkaz update.

2.1.2.4 Resolve conflicts

Resolve conflicts je potrebné vykonať ak sa počas vykonania príkazov *update* alebo *commit* vyskytnú konflikty, ktoré nie je možné automaticky vyriešiť. *Resolve conflict* je možné vyvolať nasledovne:

1. Kliknete pravým tlačidlom myši na príslušný súbor
2. Zvolíte Subversion > Resolve conflicts
3. Otvorí sa okno na vyriešenie konfliktov



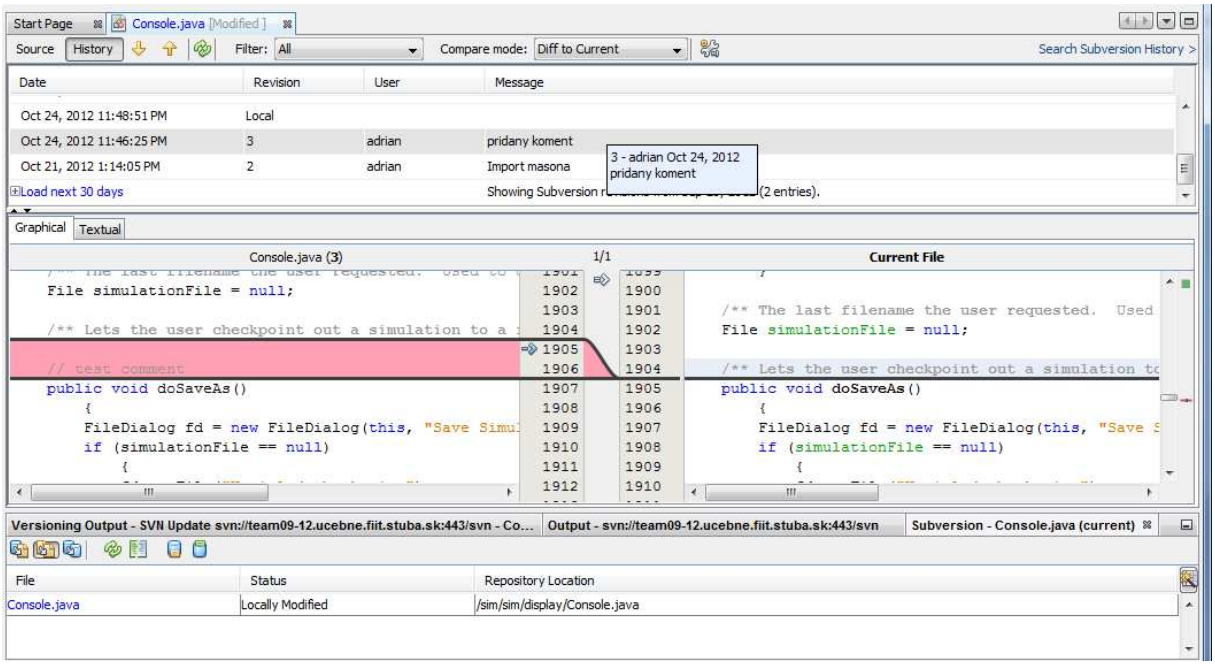
Obr. 2:3 Riešenie konfliktov.

2.1.2.5 History

Ak si chcete pozrieť kompletnú históriu zmien súboru, je to možné vykonať nasledovným postupom:

1. Kliknite pravým tlačidlom myši na požadovaný súbor
2. Zvolíte History > Show history

História zobrazí kompletný prehľad zmien daného súboru či už v repozitári alebo aj v lokálnej kópii projektu. Takisto zobrazí graficky zmeny danej verzie súboru oproti aktuálnej verzii.



Obr. 2:4 História súboru.

Pre kompletný prehľad zmien v repozitári je potrebné zvoliť Team > Search history.

2.2 Zhrnutie analýzy

Kompletný návod používania SVN repozitára v prostredí Netbeans je možné nájsť aj na internetovej stránke, z ktorej sme čerpali aj vyššie uvedené informácie [1].

2.3 Integrácia MASON a GeoMASON

2.3.1 Úloha

Analýza možností importovania mapových dát z OpenStreetMap do knižnice MASON.

2.3.2 Analýza

GeoMason:

- MASON štandardne nepodporuje dáta z geografických informačných systémov (GIS).
- Rozšírenie knižnice MASON, ktoré pridáva podporu pre vektorové geografické dáta.
- Umožňuje načítať prostredie pre simuláciu agentov z vektorových GIS dát (body, čiary, polygóny).
- Natívne podporuje čítanie a zápis z/do ESRI shapefile (štandard pre reprezentáciu geopriestorových vektorových dát) pomocou tried ShapeFileImporter a ShapeFileExporter.
- Podporuje aj iné GIS dátové formáty cez knižnice tretích strán (GeoTools, GDAL, OGR).
- Dokáže prinútiť agentov dodržiavať geografické ohraničenia (napr. pohyb len po ceste).

Shapefile:

- Dátový formát na ukladanie vektorových priestorových dát pre GIS.
- Priestorovo popisuje body, čiary, polygóny.
- Je otvorený pre dátovú interoperabilitu medzi rôznymi geografickými systémami.
- Má binárny formát.
- Vyvíjaný spoločnosťou ESRI a často označovaný ako ESRI shapefile.
- ESRI shapefiles sa dajú importovať do GeoMason.

OpenStreetMap:

- Otvorený, kolaboratívny projekt, ktorého cieľom je vytvoriť voľnú editovateľnú mapu sveta.
- Princíp podobný Wikipédii – mapu môžu upravovať ľubovoľní dobrovoľníci a všetky zmeny sa uchovávajú.
- Umožňuje stiahnuť mapové podklady zadarmo ľubovoľnej časti planéty.
- Mapové dáta reprezentuje v OSM súboroch, ktoré sú textové a založené na XML.
- OSM XML súbory sa nedajú priamo importovať do GeoMason.

2.3.3 Zhrnutie analýzy

Identifikovali sa tri možnosti:

- Použijeme GIS, ktorý dokáže stiahnuté OSM XML súbory s mapovými podkladmi prekonvertovať do ESRI shapefile súborov. Napríklad QGIS s pomocou QGIS OSM pluginu na načítanie OSM máp. Prípadne použijeme GeoConverter.
- Alebo použijeme už skonvertované OSM mapové dáta do ESRI shapefiles, ktoré dobrí ľudia za nás pripravili. Stiahnuť ich môžeme napríklad na Cloudmade alebo z Geofabric alebo z OpenStreetMap Data.
- Alebo rozšírenie GeoMason obohatíme o knižnicu OGR, ktorá podľa manuálu podporuje OSM XML formát. GeoMason obsahuje podporu pre knižnicu OGR prostredníctvom triedy OGRImporter.

2.4 Importovanie OpenStreetMaps do GeoMASON

2.4.1 Úloha

Importovanie OpenStreetMaps do rozšírenia GeoMASON.

2.4.2 Analýza

GeoMason [2] je voliteľné rozšírenie knižnice MASON, ktoré pridáva podporu pre vektorové a rastrové geografické dáta. Rovnako ako knižnica MASON je vyvíjané na George Mason University. Rozšírenie GeoMason sa dodáva so zdrojovými súbormi a množstvom demo príkladov. Je závislé na open source knižnici Java Topology Suite (JTS), ktorá poskytuje objektový model pre planárnu geometriu spolu s množinou fundamentálnych geometrických funkcií. Inštalácia GeoMason prebiehala v nasledujúcich krokoch:

1. Prekopírovanie adresára *sim* zo stiahnutého archívu do adresára s knižnicou MASON
2. Stiahnutie Java archívu rozšírenia GeoMason
3. Pridanie Java archívu rozšírenia GeoMason k projektu vo vývojovom prostredí.

Pre úspešné zostavenie a spustenie všetkých demo príkladov rozšírenia GeoMason je nevyhnutné nainštalovať ďalšie knižnice, na ktorých je GeoMason závislý:

- Java Topology Suite
- GeoTools
- GDAL / OGR

2.4.3 Implementácia

Pôvodný zámer bol načítať mapové údaje z OpenStreetMaps do knižnice MASON prostredníctvom rozšírenia GeoMason obohateného o knižnicu OGR. Knižnica OGR je však vyvíjaná v C++, a tak poskytuje svoje API pre jazyky C/C++. Existujú predkompilované rozhrania aj pre iné jazyky generované pomocou SWIG. Po pripojení k projektu sa ukázalo, že daná verzia knižnice OGR nie je kompatibilná s verziou metódy *OGRImporter* v rozšírení GeoMason.
Mapové dáta som preto nahral do programu nasledovným spôsobom. Na mapách OpenStreetMaps zvolíme možnosť Export. Vyznačíme oblasť na mape, ktorej mapové údaje chceme exportovať. Ako výstupný formát exportovaných údajov vyberieme OSM XML súbor.
Tento súbor otvoríme v geografickom informačnom systéme Quantum GIS pomocou pluginu QGIS OSM Plugin. V geografickom informačnom systéme Qunatum GIS máme možnosť načítané mapy editovať. Môžeme napríklad niektoré cesty alebo budovy odstrániť. Mapové dáta následne v QGIS uložíme do shapefile súborov. Do každého shapefile súboru uložíme jednu vrstvu z mapových dát, napríklad cesty, chodníky alebo budovy. Mapové dáta uložené v shapefile súboroch dokážeme načítať do programu prostredníctvom metódy *ShapeFileImporter.read*, ktorú nám poskytuje GeoMason.

2.4.4 Zhrnutie analýzy

Je odporúčané inštalovať knižnicu MASON a teda aj rozšírenie GeoMason do adresára, ktorého absolútna cesta neobsahuje medzeru. Niektoré metódy v rozšírení GeoMason volajú funkciu *Class.getResource*, ktorá nájde zdroj na základe mena a vracia referenciu na objekt, ktorý okrem iného obsahuje aj absolútnu cestu k zadanému zdroju. Vrátená

cesta však namiesto každej medzery obsahuje *%20*, čo znemožňuje nájdenie požadovaného zdroja v súborovom systéme, čo spôsobí vyhodenie výnimky.

2.5 Analýza softvéru pre podporu riadenia projektov

2.5.1 Úloha

Analýza softvéru pre podporu riadenia projektov, ktoré podporujú metodiku SCRUM.

2.5.2 Analýza

Analyzované nástroje:

- Redmine,
- Trac,
- Jira,
- Trello.

2.5.2.1 Redmine

Platforma – Ruby on Rails

Výhody:

- Fórum
- Web based
- Podporuje Svn,git,..
- Real time
- Gantt graf, Štatistiky, progress bar projektu a úloh
- Kalendár
- Dokumentácia – Kategórie : User documentation, Tech documentation
- Nastavenie priority – Imediate, Urgent, High, Normal,Low
- Nastavenie typu úlohy : Bug, Feature, Support
- Zaradenie úlohy do nami vytvorenej kategórie
- Priradenie úlohy k verzii projektu
- Od kedy do kedy, predpokladaný čas na vykonanie
- Email notification
- Väčšia funkcionalita v základnej verzii (Oproti Trac)

Nevýhody:

- Role – manager, developer, reporter
- Nie je priamo určený pre SCRUM
- Ruby on rails môže byť zložite na nastavenie
- Ťažšia inštalácia
- Menej pluginov (Oproti Trac) [3]

2.5.2.2 Trac

Platforma – Python

Výhody:

- Web based
- Real time
- Milestones
- Gantt graf, roud map (progress bar Milestone-ov)
- Priradenie úlohy k name vytvorenému Milestone-u
- Priloženie súboru k úlohe
- Podporuje Svn,git
- Nastavenie typu úlohy : Defect, Enhancement, Task
- Viacero pluginov (oproti Redmine)
- Nastavenie priority – Blocker, Critical, Major, Minor, Trivial

Nevýhody:

- Ťažšia inštalácia
- User account management je bez pluginou ťažkopádny
- Menšia funkcionalita v základnej verzii – rozšírenie plugin-mi (oproti Redmine) [4]

2.5.2.3 Trello

Výhody:

- Veľmi jednoduché
- Web based
- Drag and drop ovládanie – presúvanie kartičiek, prideľovanie riešiteľov ku kartičkám
- Zoznami s kartičkami – vizuálna priorita kartičiek podľa poradia
- Real-time
- Ovládanie aj so smart-phonu
- Google drive integrácia

Nevýhody:

- Ak by sme chceli scrum, tak by sme si museli adekvátne navrhnúť jednotlivé zoznamy kartičkami
- Nie sú tam rozšírené nastavenia pre úlohy – odkedy do kedy, predpokladaný počet hodín na riešenie, priorita vykonania úlohy [5]

2.5.2.4 Jira

Výhody:

- Web based

- Podporuje Svn,git,..
- Stavy úloh – Open, In progress, Resolved, Reopened, Closed
- Drag and drop ovládanie – presúvanie kartičiek
- Real-time
- Podpora agilného vývoja – User story, Product BackLog,
- Verzie – podkategórie projektu z hľadiska času
- Componenty – podkategórie projektu z hľadiska významu
- Nastavenie typu úlohy – User story, Epic (product backlog), Task, Bug, Improvement, New feature,
- Nastavenie priority – Blocker, Critical, Major, Minor, Trivial
- Od kedy do kedy, predpokladaný čas na vykonanie
- Email notification

Nevýhody:

- Chýba stav, v ktorom úloha čaká na skontrolovanie, ako je to napríklad v Redmine [6].

2.5.3 Zhrnutie analýzy

Pridávanie pluginov do podporného nástroja Trac-u by bolo dosť časovo náročné len preto, aby sme doplnili funkcie, ktoré RedMine a Jira ponúkajú v základnej verzii a zrejme budeme vyžadovať väčšiu funkcionalitu ako nám je schopný poskytnúť Trello, i keď je práca sním jednoduchá a intuitívna. Jira bude najvhodnejším riešením, pretože bude schopná najlepšie podporovať agilný spôsob vývoja, pre ktorý sme sa rozhodli.

2.6 Návod na používanie Jiri

2.6.1 Úloha

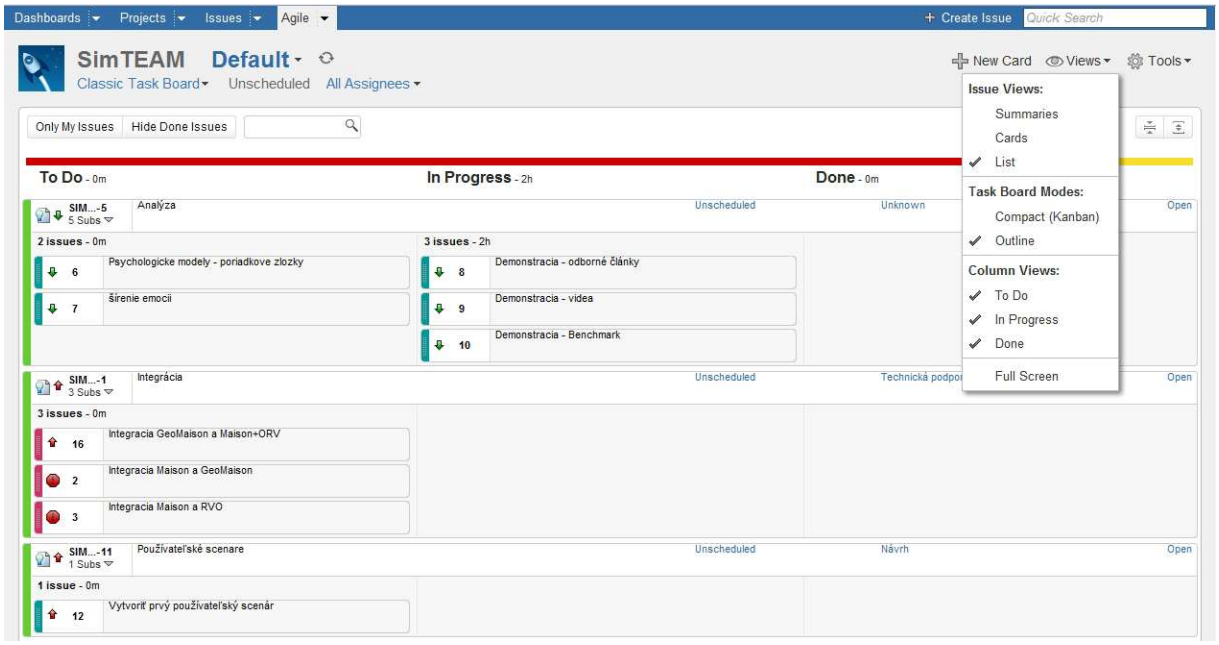
Napísanie manuálu k Jire nielen pre vývojárov ale aj pre používateľov.

2.6.2 Analýza

Jira je rozdelená na niekoľko záložiek:

- V prvej záložke **Dashboard** sú informácie o vašich úlohách.
- V záložke **Projects** sú informácie o projekte celkovo. V časti Summary sa nachádza ***Activity stream***, graf progresu projektu v priebehu mesiaca (dni - úlohy) atď.
- Záložka **Issues** slúži na prezeranie úloh. Je možné si vytvoriť vlastný filter na vyhľadávanie.
- V záložka **Agile** sa nachádza tabuľa s kartami. Na týchto kartách sú zobrazené všetky druhy vytvorených úloh. Kartičky sú farebne odlíšené. Farby predstavujú rozdielny typ úlohy. Červenou je označená technical task a modrou sub-task. Zelenou sú označené User story. Táto tabuľa sa dá rôzne filtrovať. Do pozornosti dávam filter s názvom ***Classic Task board***, ktorý si vyberiete

z rozbaľovacieho zoznamu s názvom ***Classic Planning Board*** (neskôr bude názov vybraného typu tabule) zobrazenom na obrázku (Obr. 2:5), kde je možné vidieť úlohy rozdelené podľa stupňa progresu (To do, In progress, Done) a podľa User story ku ktorej patria (zelene orámovaná karta, na ktorej sa nachádzajú sub-task-y). V tomto filtri sa dá ešte filtrovať podľa konkrétneho používateľa, ktorému bola úloha pridelená v rozbaľovacom zozname s názvom ***All Assignees*** (neskôr bude názov niekoho meno). Pre ešte prehľadnejšie prezeranie si môžete zvoliť v rozklikávacom zozname ***Views*** (oko) -> ***Issue Views*** -> ***List***.



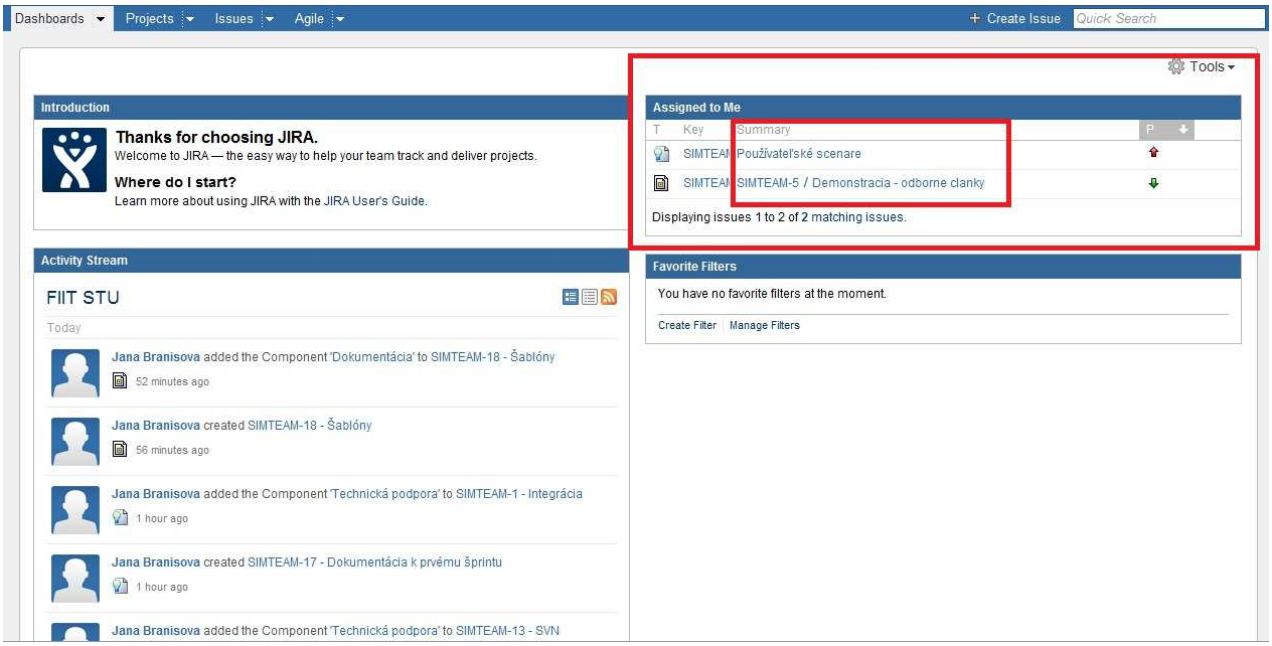
Obr. 2:5 Zobrazenie úloh v Jire podľa progresu

2.6.2.1 Kde nájdem úlohy, ktoré sú mi pridelené?

V záložke **Dashboards** vidíte tabuľky s tokom aktivít – **Activity Stream**, kde je zobrazená celá vaša činnosť, ako napríklad začatie úlohy, zalogovanie úlohy atď. Taktiež môžete vidieť ako napredujú všetci členovia tímu.

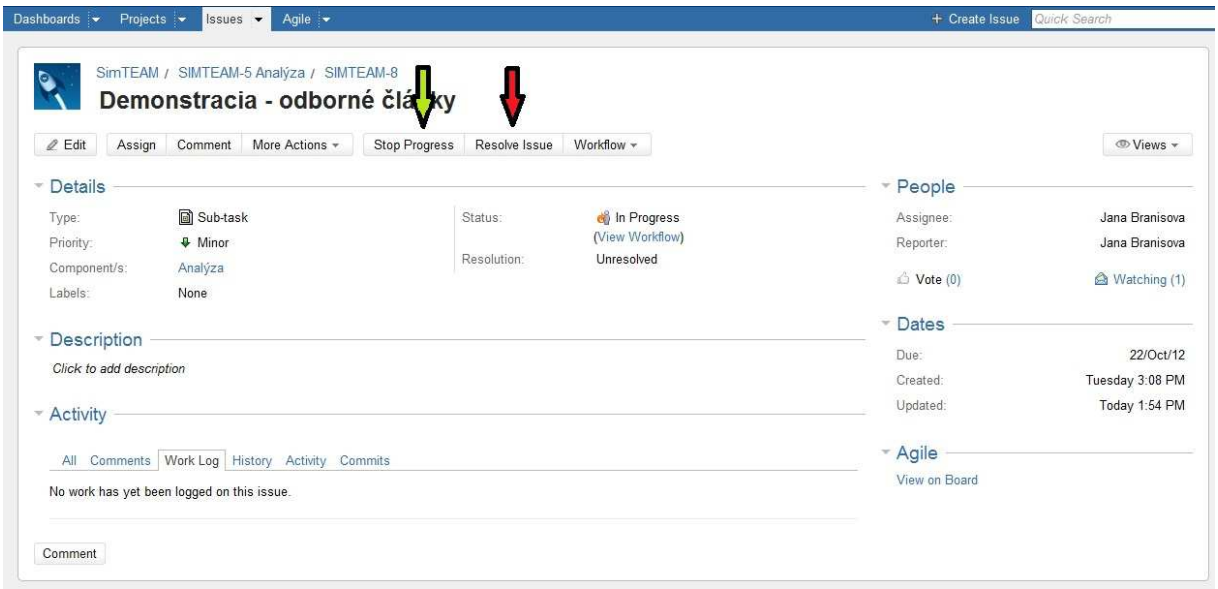
Na pravo je tabuľka s úlohami, ktoré sú pridelené vám, ako ukazuje obrázok ().

Riadok v tejto tabuľke označený obrázkom so žiarovkou, je **user story**, ktorá je vám pridelená. Ak sa jedná o analýzu, tak to znamená, že na nej máte pracovať najintenzívnejšie, ale zároveň sa na nej podieľajú aj ostatní používatelia. Obrázok s oranžovým štvorčekom označuje **sub-task** (link za lomkou), ktorá patrí do nejakej user story. Tento subtask je konkrétne pridelený vám a máte ho vykonať.



Obr. 2:6 Tabuľka s pridelenými úlohami

Keď kliknete na link sub-task-u v stĺpci **Summary**, otvorí sa vám okno s detailom úlohy, ktoré je zobrazené na obrázku (Obr. 2:7). User story bude označovať ako začaté ten člen tímu, ktorý má danú User-story pridelenú.



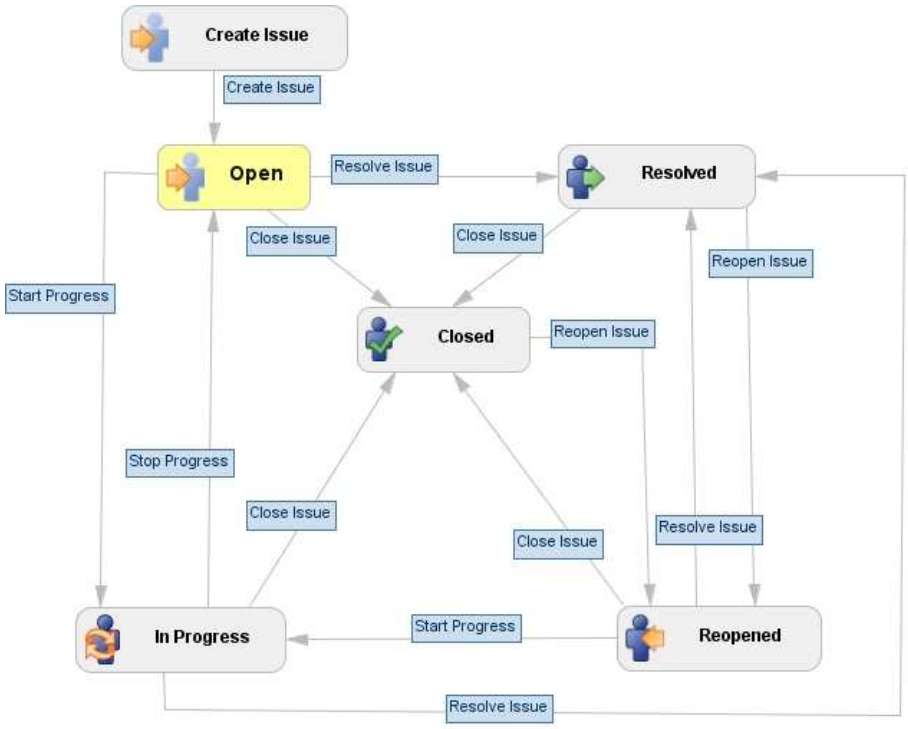
Obr. 2:7 Detail úlohy v Jire

2.6.2.2 Ako dám ostatným vedieť, že som už na sub-tasku začal pracovať?

V okne sub-task-u môžete zaznačiť do systému, že ste na úlohe začali pracovať stlačením tlačidla **Start Progress**. Po stlačení sa zmení *status* sub-stask-u na **In Progress**.

2.6.2.3 Čo robiť keď už som sub-task vyriešil ?

Stlačením tlačidla Resolve Issue oznámite ostatným, že ste úlohu vyriešili. Na obrázku (Obr. 2:8) sú zobrazené stavy, ktoré môže úloha nadobúdať.



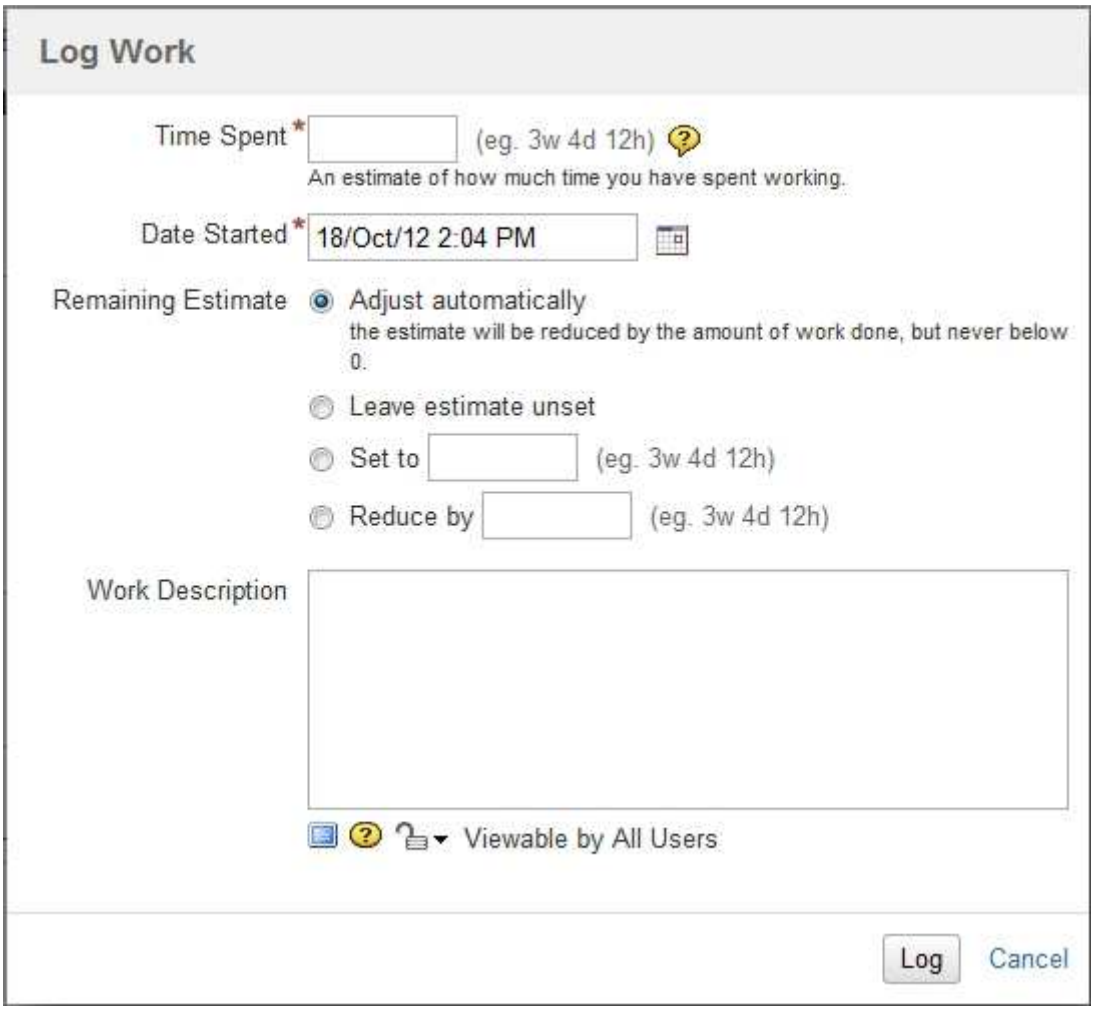
Obr. 2:8 Stavy úlohy v Jire (obrázok prevzatý zo systému Jira)

2.6.2.4 Schválenie a uzavretie sub-task-u

O znovuotvorení úlohy sa bude rozhodovať na tímovom stretnutí. Úloha bude znova pridelená vtedy keď je schválená aj pre ďalší šprint alebo nebola úspešne vyriešená. Znovu otvorenie úlohy (Reopen) bude mať na starosti manažér plánovania (v našom tíme Jana Branišová).

2.6.2.5 Ako zalogujem koľko hodín som na danej úlohe pracoval ?

V okne sub-task-u stlačte na klávesnici bodku, otvorí sa vám okno s príkazovým riadkom kde napíšete „log“ a stlačíte ENTER. Následne sa otvorí okno s formulárom pre zalogovanie úlohy, ktorý je zobrazený na obrázku (Obr. 2:9). Do okienka **Time Spent** napíšete počet hodín, ktorý ste na danom sub-tasku odpracovali. Políčko **Date started** obsahuje predvolené hodnoty (dátum a čas) kedy ste stlačili tlačidlo Start Progress. V zozname radio buttonov nechajte pre začiatok nastavenú prvú možnosť **Adjust automatically**. Je nutné hneď po ukončení práce na úlohe a zmene stavu úlohy na vyriešená (Resolved), opísať, ako ste vyriešili vám pridelenú úlohu. Vo formulári pre logovanie je na to vyhradené okienko **Work Description**. Po vyplnení údajov treba ukončiť logovanie potvrdením tlačidla **Log**.



Obr. 2:9 Formulár so zalogovaním času k úlohe

2.6.2.6 Hierarchia úloh

Najvyššiu úroveň bude mať User story. Pre **User story** sa budú vytvárať Sub-task-y, ktoré budú priradené konkrétnym riešiteľom úlohy. User story sa priradia prvému riešiteľovi, ktorému bude pridelený prvý Sub-task.

Na rovnakej úrovni User story sa nachádza aj typ úlohy s názvom **Epic**. Tento typ sa bude vytvárať pred každým šprintom. Členovia tímu majú za úlohu do Epic-u pridávať svoje nápady na nové User story pre ďalšie šprinty.

Ako už bolo vyššie uvedené, **Sub-task** má dva druhy a to, Technical Task a obyčajný sub-task. Pokiaľ sa bude jednať o také veci ako inštalovanie, správu servera a na ňom inštalovaných programov, nastavovanie IDE, implementácia, testovanie bude pridelená členovi tímu technická úloha. Ak sa bude jednať o analýzu, dokumentáciu alebo návrh bude mu pridelený obyčajný sub-task.

2.6.2.7 Componenty

Komponenty v Jire predstavujú podoblasti projektu. Vytvárať ich môže jedine administrátor. Vytvorila som tieto komponenty:

- Analýza
- Návrh
- Implementácia

- Testovanie
- Dokumentácia
- Technická podpora

Pri vyváraní úlohy sa budú úlohy zaraďovať do týchto kategórii. To by malo umožniť lepšiu filtráciu úloh.

2.6.2.8 Verzie

Verzie predstavujú iterácie v projekte. Budú sa členiť na jednotlivé šprinty a ich názov bude tiež od nich odvodený. Verzie má tiež právo vytvoriť iba administrátor. Úlohy sa budú priraďovať do jednotlivých šprintov pri vytváraní úlohy. Zatiaľ sú vytvorené tieto verzie:

- Prvý šprint
- Druhý šprint

Aby bolo rozdelenie úloh na verzie badateľné aj v záložke Agile, je nutné nastaviť pri editácii úlohy aj kolónky ***Fixed Versions*** na požadovanú verziu.

2.6.3 Zhrnutie analýzy

Tento krátky návod obsahuje to najnutnejšie, čo tím potrebuje vedieť pre úspešné rozbehnutie projektu a nemusí strácať čas s vyhľadávaním návodov na internete.

2.7 Integrácia MASON, RVO a GeoMASON

2.7.1 Úloha

Integrácia knižnice MASON s knižnicou RVO a rozšírením GeoMASON na účelom pohybu agentov v reálnom prostredí mapy.

2.7.2 Analýza

Knižnica MASON spolu s knižnicou RVO2 (Reciprocal Collision Avoidance) predstavujú základné kamene projektu demonštrácie, ktorý je riešený na predmete Tímový projekt. Knižnica MASON predstavuje samotnú platformu resp. framework na ktorom bude simulácia vystavaná, pričom RVO2 prinesie vedecký a dostatočne reálny prístup k vyhýbaniu sa agentov navzájom resp. voči budovám. Rozšírenie GEOMASON umožní zasadiť simuláciu do reálneho sveta, pretože budú použité skutočné mapové podklady (Bratislava - SNP).

2.7.3 Implementácia

2.7.3.1 Integrácia RVO2

Knižnica RVO2 bola originálne vytvorená pre C++ a neskôr pre C#. Aby ju bolo možné použiť pre jazyk Java, bolo nutné pomocou nástroja SWIG vytvoriť wrapper na existujúce metódy a následne vygenerovať wrapper pre Javu spolu s knižničným súborom .dll, ktoré bude wrapper v jazyku Java priamo externe volať. Vytvorený wrapper pre SWIG bol získaný z externých zdrojov (SAV - Štefan Dlugolinský) a

prekompilovanie knižnice .dll bolo vykonané cez Visual Studio 2012 Ultimate. Okrem samotnej knižnice je pre jej beh okrem Java wrappera potrebná aj inštalácia Visual C++ 2012 Redistributable, verzia x86 (viď. Návod na inštaláciu a spustenie aktuálnej verzie (pre WIN x64)).

Knižnica RVO2 obsahuje iba 3 triedy (***Vector2***, ***Line*** a ***RVOSimulator***), pričom iba jedna hutná trieda vykonáva všetku dôležitú prácu ohľadne výpočtov a obchádzania - ***RVOSimulator***. Na objasnenie funkčnosti a použitia RVO2 v našom projekte je nutné spomenúť dve dôležité metódy.

Metóda setupRVOScenario()

- Vytvára inštanciu triedy RVOSimulator.
- Nastavuje parametre simulácie, ktoré budú charakterizovať obchádzanie sa (neighborDist, maxNeighbors, timeHorizon, timeHorizonObst, radius, maxSpeed) [7].
- Po načítaní mapových podkladov metódou ***loadGISData()***, prejde jednotlivé polygóny mapy a vloží ich do "sveta RVO2" (metódami ***addPolygon*** a ***addMultiPolygon***) a následne metóda ***processObstacles()*** zabezpečí, že ich celá simulácia a agenti v nej budú brať do úvahy.

Metóda setPreferredVelocities()

- Aktualizuje vektor pohybu / rýchlosti agenta s ohľadom na iných agentov, prekážky a jeho cieľ.
- Ak sa agent nachádza v blízkosti cieľa, tak je vektor pohybu nulový ak opačne, tak sa nastavuje na smer a rýchlosť k danému cieľu.

Agentov v prostredí RVO2 jednoznačne identifikuje parameter tzv. id, ktorý je návratovou hodnotou metódy ***addAgent()*** vkladajúcej agenta do sveta RVO na zadanú pozíciu.

2.7.3.2 Integrácia GEOMASON

GEOMASON je rozšírenie samotného frameworku MASON, a umožňuje prácu s mapovými podkladmi, pretože obsahuje priamo integrované nástroje (trieda ***ShapeFileImporter***) na importovanie máp uložených vo formáte ESRI (tzv. shape file). Samotný GEOMASON využíva ešte ako externý komponent Java Topology Suite (jts-1.11.jar), ktorý má na starosti základné geometrické operácie.

Každá importovaná mapa musí pozostávať nie z jedného, ale minimálne 3 povinných súborov (.shp, .shx, .dbf), ktoré obsahujú informácie o mape a jej častiach. V projekte používame ešte aj ďalší typ súborov - .prf, ktorý obsahuje informácie o projekcii mapy v priestore a použitých jednotkách.

Načítanie máp sa vykonáva v metóde ***loadGISData()***, kde sú mapové podklady načítané do objektu triedy ***GeomVectorField***, ktorý tvorí model mapy (terminológia MVC). Vykreslenie mapy je v prostredí MASON totožné s vykreslením akéhokoľvek modelu, napr. objektu triedy ***Continuous2D***. Pri načítavaní viacerých mapových podkladov, prípadne agentov je nutné zosynchronizovať premietnutie, a to pomocou metódy ***setMBR()***, ktorá berie ako parameter hodnotu MBR (minimum bounding rectangles) pre prvú mapu a je aplikovaná na všetky načítané vrstvy.

Načítavané sú mapové podklady dodané z externého zdroja, a obsahujú časť Bratislavy, konkrétne časti Starého mesta vrátane námestia SNP. Sú umiestnené v adresári \simTeam\build\classes\simTeam\sav. Ďalšie mapy možno ľahko získať napr. z web stránky [8] cez možnosť export, pričom získate mapy vo formáte OpenStreetMap XML Data, ktoré je nutné prekonvertovať do vyššie spomínaného formátu ESRI pomocou externého nástroja, napr. Quantum GIS [9]. ***Avšak takto prekonvertovaná mapa dodaná tímovým kolegom nefungovala správne v prostredí RVO.***

2.7.3.3 Integrácia RVO2 + GEOMASON

Integrácia máp a RVO2 prestavovala problém, pretože po premietnutí máp do prostredia a ich vykreslení, bolo nutné použiť metódu ***shiftWorldToGroundZero()*** z externého zdroja (SAV - Štefan Dlugolinský), ktorá rieši konflikty, resp. nepresnosti pri float / double prepočtoch. Bez nej po vykreslení mapy a agentov, sa agenti navzájom obchádzali ale hranice budov prechádzali, pretože vizuálne premietnutie simulácie nebolo korektné.

2.8 Implementácia Web-stránky

2.8.1 Úloha

Implementácia webovej stránky tímu dostupnej aj cez stránku softvérového štúdia.

2.8.2 Implementácia

Je vytvorená web stránka z templatu, ktorý bol upravený pre naše potreby. Web stránka používa relatívne cesty ako bolo požadované. Každý týždeň bude aktualizovaná a pridaná zápisnica, dokumenty, týždňový program + postupne budú spracované aj úlohy (prepis plánov zo zápisníc).

2.9 Analýza poriadkových zložiek

2.9.1 Úloha

Analýza činností a koordinácie poriadkových zložiek pri demonštrácii.

2.9.2 Analýza

Poriadkové zložky budú zohrávať v simuláciu dôležitú úlohu. V reálnom svete majú poriadkové zložky pri demonštráciách a protestoch rôzne role.
Identifikované boli niektoré vlastnosti a schopnosti, ktoré by mohli poriadkové zložky (ako napr. ťažkoodenci, policajti, kukláči) v simulácii vykonávať:

- snažiť sa udržať dav ľudí pod kontrolou
    - Potrebné akcie: blokovať pohyb ľudí
- rozohnať príliš agresívny dav ľudí (vodné delo, slzný/paprikový plyn, fyzické pretlačenie pomocou štítov)
    - Potrebné akcie: po vyhodnotení stavu použiť na dav ľudí jeden z uvedených nástrojov
- zatknutie nebezpečných demonštrantov
    - Potrebné akcie: pri dostatočnej blízkosti dôjde k zadržaniu demonštranta
- schopnosť zablokovať určitú časť priestoru a zamedziť tak presunu demonštrantov
    - Potrebné akcie: presun na dané miesto
- pomoc raneným demonštrantom [10]
    - Potrebné akcie: presun k demonštrantom a následná pomoc
- schopnosť v prípade potreby požiadať o zvýšenie počtu poriadkových zložiek
    - Potrebné akcie: vedieť vyhodnotiť takýto stav a následne vydať požiadavku
- pri výbuchu bomby snaha o pomoc ľuďom (napr. odovzdanie informácie o najbližšom východe) [10]

Potrebné akcie: presun do danej oblasti, komunikácia s demonštrantami.

2.9.3 Zhrnutie analýzy

V simulácii demonštrácie sa budeme snažiť identifikovať a naimplementovať dva typy policajných agentov. Zároveň sa pokúsime nasimulovať ich vzájomnú koordináciu a komunikáciu a pri implementácii neobídeme ani akcie poriadkových zložiek a ich snahy o ukončenie demonštrácie primeraným spôsobom.

2.10 Analýza benchmarku simulácie demonštrácie

2.10.1 Úloha

Analýza možných validácií simulácie demonštrácie.

2.10.2 Analýza

Na overenie správnosti akejkoľvek simulácie je nutné vedieť, čo sa bude validovať. Preto sme analyzovali už existujúce spôsoby validácie simulácie demonštrácií.
Existujú dve kategórie požiadaviek, ktoré musí spĺňať model davu, aby bola simulácia realistická [11].
Prvá kategória:

- Flexibilita – schopnosť modelu prispôsobiť sa. V mnohých prípadoch sú rozhodnutia pevne naviazané na určité situácie, čo nie je dobre.
- Rozšíriteľnosť – schopnosť prijať nové vlastnosti bez veľkej námahy. Napríklad nové vlastnosti správania.

Druhá kategória:

- Výpočtová efektivita – považuje sa čas, ktorý treba na vykonanie simulácie za určitých podmienok, počas jedného scenára.
- Škálovateľnosť – dôležitá najmä pri mnohopočetnom dave. Škálovateľnosťou sa zaoberá paralelné počítanie a distribuované systémy.

Takéto validovanie simulácie je príliš všeobecné, a preto sa používajú aj ďalšie spôsoby, ako odhadnúť presnosť a správnosť simulácie.

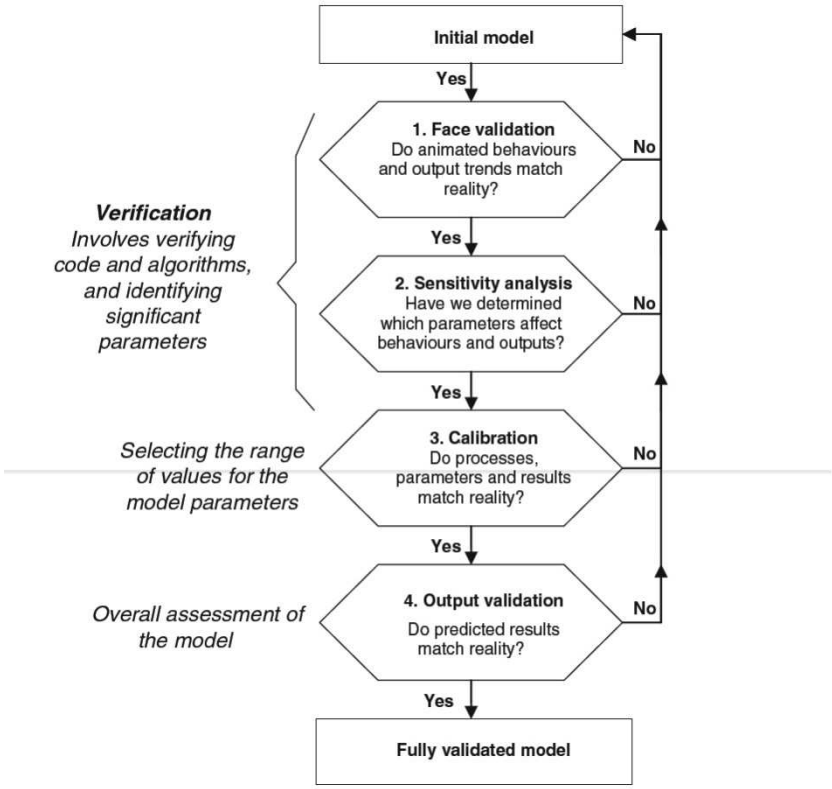
Podľa [12] sú simulácie davu, ktoré majú spoločné črty so simuláciami demonštrácií, evaluované nasledujúcimi metódami:

- manuálna kontrola,
- porovnanie so skutočným svetom (skutočnou udalosťou),
- štatistickou analýzou.

Validovanie simulácií demonštrácií nie je vo všeobecnosti triviálne. Na validáciu simulácie modelov založených na agentoch existujú nasledujúce spôsoby [13]:

- Replikačné validácie: kde sú modelové výstupy porovnávané s údajmi získanými z reálneho sveta.
- Prediktívne validácia: kde model je schopný predvídať správanie, ktoré mu nebolo pred tým známe, napr. také ktoré pochádza z teórií, alebo také ktoré by sa mohlo objaviť v budúcnosti.
- Štrukturálne validácia: ak model nie len reprodukuje pozorované správanie systému, ale naozaj reflektuje spôsob, akým systém funguje aby vyprodukoval toto správanie.
- Validácia zovňajšku: je často aplikovaný na začiatku fázy simulačnej štúdie v rámci zastrešenia koncepčnej validácie. Táto technika pozostáva z minimálne troch metodických prvkov:
    - Animácie hodnotenia: zahŕňa pozorovanie animácie celého simulovaného systému alebo jednotlivých agentov a sleduje ich konkrétne správanie.
    - Vnorené posúdenie: sleduje dynamiku konkrétneho agenta počas behu simulácie.
    - Výstupné hodnotenie: zistí, že výstupy spadajú do prijateľného rozsahu skutočných hodnôt a že výsledky sú konzistentné pre rôzne simulácie.
- Analýza citlivosti: hodnotí vplyv rôznych parametrov a ich hodnoty na konkrétne správanie alebo na celkový model výstupov.
- Kalibrácia: je proces identifikácie rozsahu hodnôt pre parametre a ladenie modelu aby pasoval na reálne dáta. To sa robí tak, že celkový model sa vníma ako čierna skrinka a použitím účinnej optimalizačnej metóde pre nájdenie optimálneho nastavenia parametrov.

- Výstup validácie: zahŕňa graficky a štatisticky zodpovedajúcu predpoveď modelu k množine reálnych dát (Obr. 2:10).



Obr. 2:10 Všeobecný proces validácie modelov založených na agentoch (prevzaté z **[13]**).

2.10.2.1 Validácia je problém

1. Dát je málo, živé modely sa robiť nedajú. Jediná validácia je na základe dát z minulosti, ale ak simulácia nasimuluje presne udalosť, ktorá sa stala, to neznamená, že vie určiť, ako bude prebiehať udalosť v budúcnosti.
2. Problém s pozorovaním dát v reálnom svete.
3. Ohodnotenie kvality dát. Ako sa dá určiť, čo je radosť, hnev, ...?
4. Správanie skupiny je určené prostredím, ľuďmi, udalosťami, ktoré sa odohrali. Kombinácia týchto parametrov tvorí príliš komplexný systém. Malá zmena nemusí mať predpokladaný účinok.
5. Malé množstvo dokumentácie k modelovaniu sociálnych systémov a ich simulácie.

Nehľadajme preto ako validovať, pozrime sa, či vieme určiť, že simulácia nie je vhodná pre daný účel. Ak nenájdeme, že by nebola, tak ju prehlásime za správnu. Ak nájdeme niečo, prečo nie je simulácia validná, tak to odstráňme, a tým ju vylepšíme [14].
Ďalší spôsob overenia validácie je referenčný scenár. Je to historická udalosť, ktorá sa nasimuluje čo najpresnejšie.

2.10.3 Zhrnutie analýzy

Rozhodli sme sa, že naša cesta validácie simulácie demonštrácie pôjde sledovaním a reprodukovaním reálnych udalostí. Budeme sledovať vopred stanovené parametre

demonštrácie, ktoré nastavíme ako vstupné parametre nami navrhnutého riešenia simulácie.

2.11 Analýza modelov správania

2.11.1 Úloha

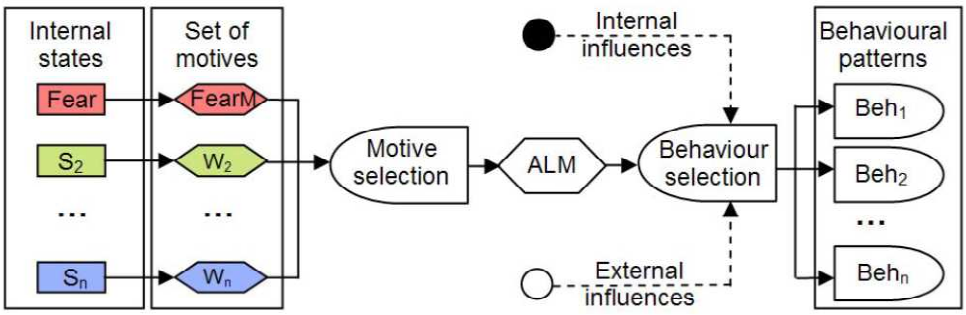
Analýza modelov správania.

2.11.2 Analýza

Modely správania, ktoré je možné použiť na simuláciu demonštrácie je niekoľko. V krátkosti ich predstavíme.

2.11.2.1 Model SimPan

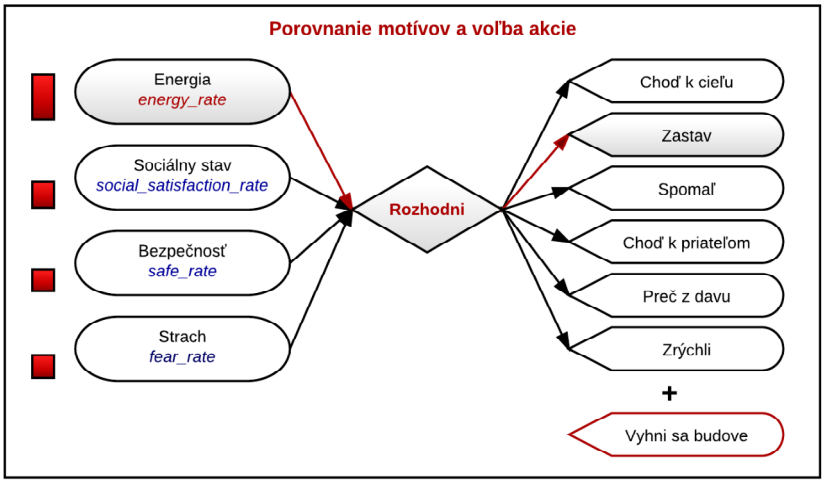
Model SimPan je založený na simulovaní strachu a paniky. Má byť ideálny pre masové zhromaždenia alebo manifestácie, pričom vychádza z modelu PECS a samotní autori odporúčajú pri jeho implementácii mapovať jeho komponenty na architektúru modelu PECS. Materiály opisujú všetky prístupy k modelovaniu ľudského správania a poskytujú veľké množstvo grafov a funkcií, ktoré opisujú charakteristiky správania ako strach, tzv. crowding, nálady agenta a mnoho iných. [15].



Obr. 2:11 Model SimPan - štruktúra komponentov modelu vrátane mechanizmu voľby akcie (prevzaté z **[15]**).

2.11.2.2 Model PECS

Autorom je Prof. Dr. Bernd Schmidt a hovorí o tom, že aktivita ľudí nie je spriemerovaná, ale správanie je výsledkom súperiacich motívov a v jednom momente sa vykonáva iba jedna resp. nízky počet akcií naraz. Tie slúžia na uspokojenie potreby z ktorej vychádza najsilnejší motív v danej chvíli. Model sa skladá z niekoľkých modulov, ktoré nie je nutné použiť všetky a popisujú oblasti konania agenta Physical Conditions, Emotional state, Cognitive Capabilities, Social Status. [16].

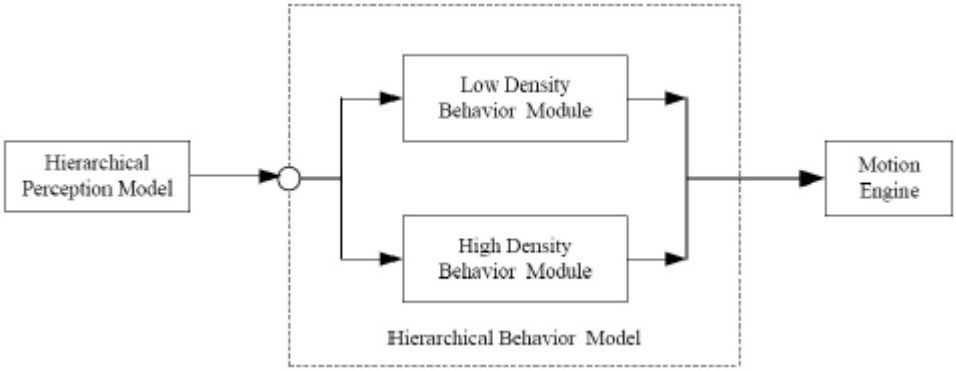


Obr. 2:12 Model PECS - porovnanie súperiacich motívov a následná voľba akcie (prevzaté z **[16]**).

2.11.2.3 Hierarchický model správania pre simuláciu davu

Existuje model správania agentov, ktorý je závislý od hustoty davu [17]. V prípade nízkej hustoty sa agent pohybuje voľne, keď dav zhustne, tak sa zvýši stres a prejavia sa prvky masového správania.

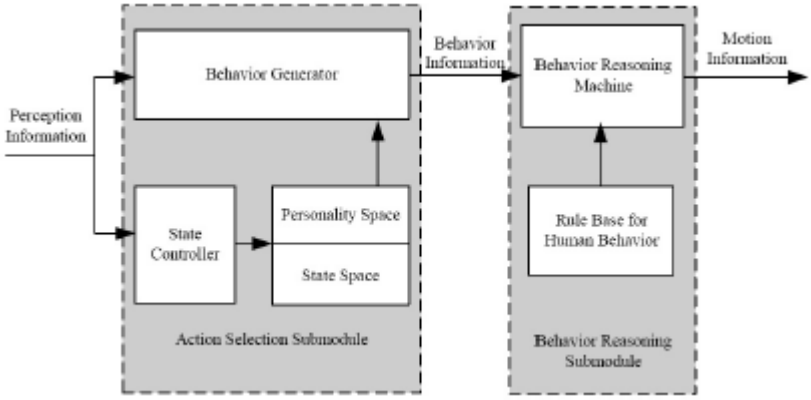
Modul pozostáva z dvoch vrstiev – modul správania pri nízkej hustote davu a modul správania pri vysokej hustote davu. Hustota davu sa chápe ako pomer medzi počtom osôb a veľkosťou miesta, kde sa dav nachádza. Hranica na určenie, či je hustota davu vysoká alebo nie považujú 2,8 človeka na meter štvorcový. Ak je hustota pod túto hranicu, tak sa prejavuje individuálne správanie. Celkovú štruktúru modelu vidno na obrázku nižšie (Obr. 2:13).



Obr. 2:13 Štruktúra hierarchického modelu správania **(**prevzaté z [17]).

Modul správania pri nízkej hustote

Človek sa správa podľa svojho uváženia a kráča maximálnou rýchlosťou (podľa schopností človeka). Modul správania a jeho štruktúra je na obrázku dole (Obr. 2:14).



Obr. 2:14 Štruktúra modulu správania pri nízkej hustote (prevzaté z **[17]**).

Výber akcie človeka závisí od vonkajších stimulov (vnímaná informácia o okolí) a vnútorných stimulov (fyziologické potreby, potreby bezpečia, potreba lásky a zaradenia, potreby sebarealizácie). Ostatné vnútorné informácie sa menia s časom (hlad, únava, ...).

Rozdeľme ľudské správanie do troch typov:

1. Fyziologické správanie – spúšťače sú fyziologické potreby a potreby bezpečia. Musia byť uspokojené.
2. Správanie podľa okolností – spúšťač sú vonkajšie informácie. Fyziologické správanie spustené vonkajšou informáciou.
3. Správanie v pohybe – spúšťače sú vonkajšie informácie a potreby lásky a zaradenia.

Priority sú podľa čísiel v zozname.

Modul správania pri vysokej hustote

Ak je osoba líder, tak sa správa podľa informácií z okolia, pamäti, svojej psychológie a emócií. Ak nie je líder, tak nasleduje ostatných, aj keď je šanca, že nebude nasledovať a bude sa správať podľa seba (akcie rovnaké, akoby bola hustota davu nízka). Rozdeľme ľudí na skupiny podľa nasledujúcich pravidiel:

- Ak sa človek nachádza blízko lídra, bude ho pravdepodobne nasledovať.
- Ak človek ide rovnakou rýchlosťou ako líder, bude ho pravdepodobne nasledovať.

Keď sa naplní cieľ, túžba lídra, upraví sa stav celej skupiny.

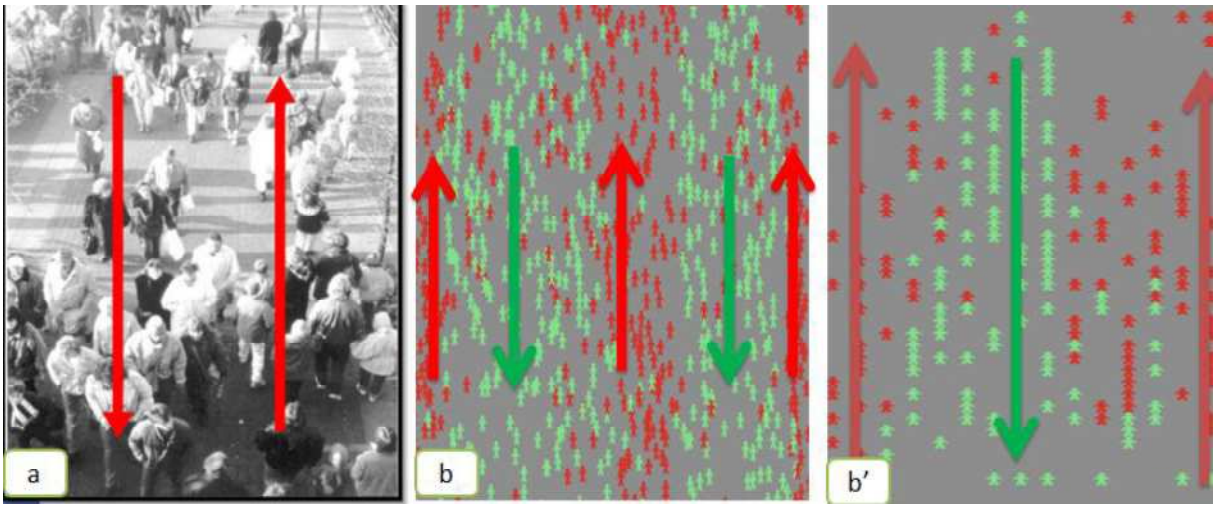
2.11.2.4 Model HuNAC

V modeli HuNAC [18] je každý model jednotlivec reprezentovaný parametrami:

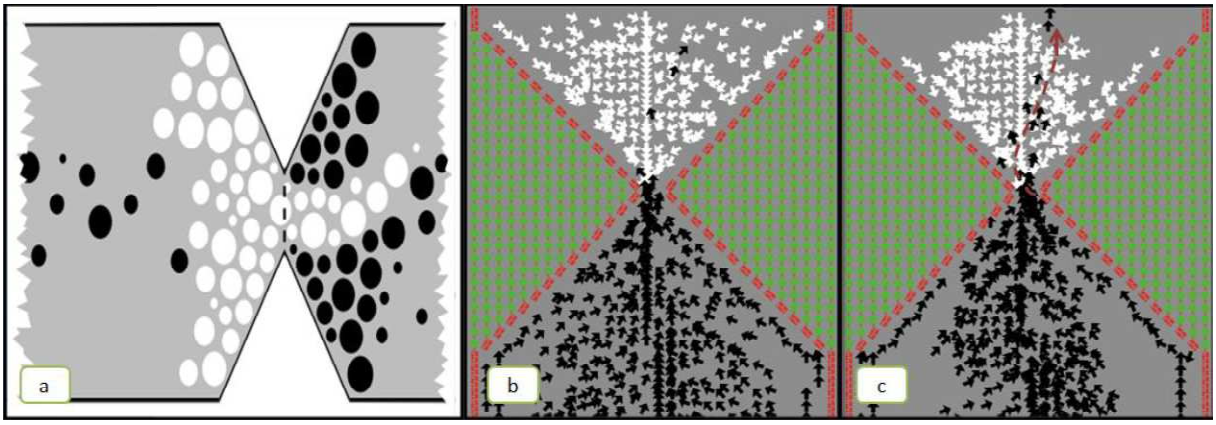
- rýchlosť – priemerná,
- rýchlosť, akou by chcel ísť – rýchlosť jednotlivca keby nebol v dave,
- smer – smer pohybu agenta,
- pozícia – pár súradníc na určenie polohy chodca v karteziánskej sústave.

Model vykazoval podobnosť s realitou. Ľudia sa pri vyhýbaní formujú do pruhov s jednosmernou premávkou. Aj agenti simulácie s použitím modelu mali takéto správanie (Obr. 2:15). Ľudia pri prechádzaní jedným východom z oboch strán oscilujú,

t.j. prechádzajú sprava doľava a naopak. Takéto správanie vykazovali aj agenti simulácie (Obr. 2:16).



Obr. 2:15 Formácia do pruhov (prevzaté z **[18]**).



Obr. 2:16 Oscilácia (prevzaté z **[18]**).

Simulácia dynamiky hustého davu

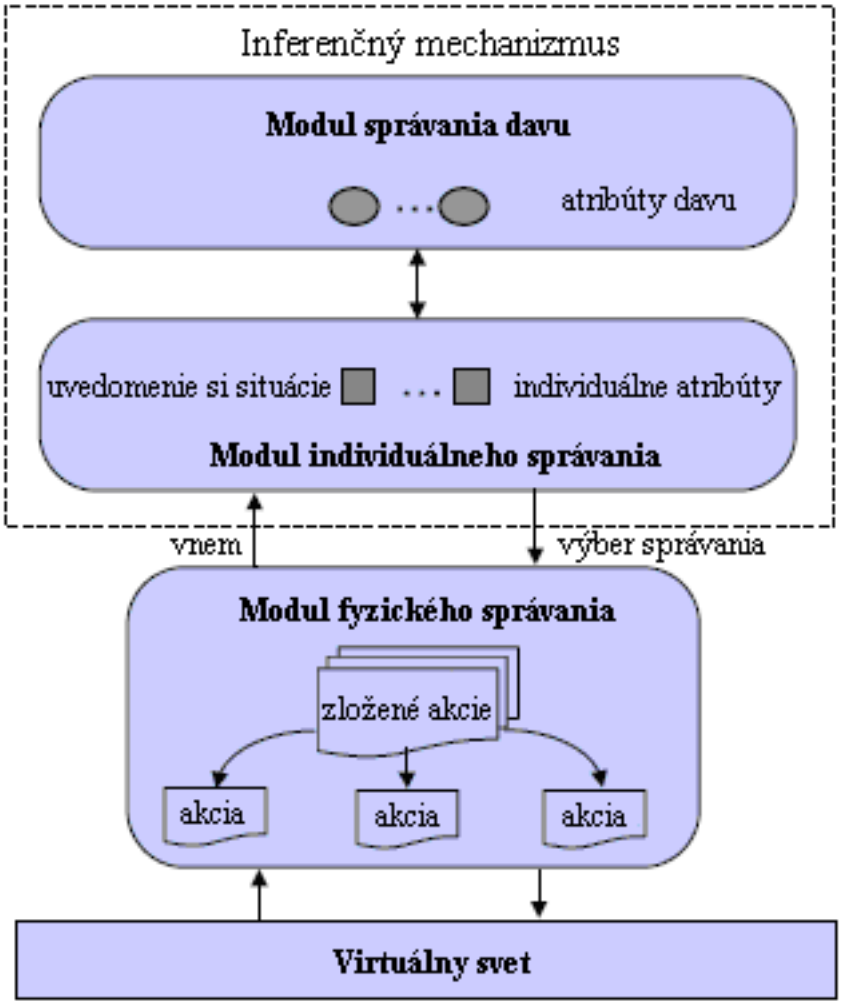
Článok [19] prezentuje návrh modelu správania agentov v simulácii davu pri normálnych a aj núdzových situáciách. Ich model využíva vrstvený framework, ktorý odráža prirodzené vzory procesov zodpovedných za rozhodovanie ľudí. Skupinovo založené správanie modeluje prostredníctvom zistení a teórií vypozorovaných zo sociálnej psychológie. Vďaka tomu sa ich agenti správajú realisticky a nezávisle od ostatných. Každý môže reagovať odlišne na podnety z prostredia a robiť rozhodnutia na základe rôznych psychologických, emocionálnych a sociálnych aspektov.

Dav ľudí je fascinujúci sociálny fenomén. Niekedy dav ľudí predstavuje vhodne organizovanú štruktúru demonštrujúcu svoju silu. V iných situáciách ľudia v dave často strácajú akékoľvek sociálne a morálne normy a z ľudí sa stávajú sebecké zvieratká.

Pre dosiahnutie čo najvernejšieho správania davu, treba pod davom rozumieť množinu heterogénnych indivíduí, ktorí sú vyzbrojení značnými schopnosťami robiť rozhodnutia ako reálni ľudia.

V tomto článku autori ponúkajú framework pre modelovanie ľudského správania, ktorý prirodzene reflektuje proces rozhodovania. Externé stimuly (udalosti, objekty, ľudia) priamo ovplyvňujú stav človeka. Ich framework je navrhnutý, aby modeloval

uvedomenie si situácie a následné zmeny na interné atribúty agentov. Ich model identifikuje rad psychologických, emočných a sociálnych atribútov, ktoré majú okamžitý vplyv na proces rozhodovania. Cieľom autorov je, aby ich model nevracal realistické výsledky iba v niekoľkých daných situáciách, ale aby fungoval ako ľudský mozog v zmysle robenia rozhodnutí.
Autori poukazujú na fakt, že psychologické a sociálne faktory podstatne ovplyvňujú proces rozhodovania u ľudí. Toto je obzvlášť pravda v dave, kde človek sa môže správať odlišne, ako by sa správal, keby bol sám.
Ich dvojvrstvový model určený pre modelovanie ľudského správanie je uvedený na nasledujúcom obrázku (Obr. 2:17).



Obr. 2:17 Dvojvrstvový model na modelovanie ľudského správania (prevzaté z **[19]**).

2.11.2.5 PMF

Pri simulácii ľudského správania vznikajú problémy:
Sústredenie na nízkoúrovňovú funkcionalitu – pri simuláciách sa často autori sústreďujú na realizáciu simulácie, ktorá je graficky realistická, geometricky presná a pohyb agentov je prirodzený v ich virtuálnom prostredí. Niektoré architektúry sa dokonca sústreďujú na mikropohyby agentov na základe malej škály emocionálnych vnemov alebo vnemov z prostredia. Tieto systému sú však reaktívneho typu a opomínajú strategické plánovanie, rozhodovanie a kognitívny proces.

Umelá inteligencia sa sústreďuje na kognitívnu funkcionalitu – umelá inteligencia sa snaží o vytvorenie rozumného agenta, ktorý je na základe formálnej logiky schopný rozhodovania, medzi agentovej komunikácie a autonómneho plánovania a učenia. Avšak realizácia daných schopností je výpočtovo veľmi náročná. Takisto je potrebné byť veľmi opatrný pri realizácii daných schopností, pretože sa ľahko môže stať, že agenti budú mať schopnosti, ktoré sú od reality veľmi vzdialené. Rozumní agenti vykonávajú rozhodnutia bez ohľadu na únavu, stres, teplotu, chorobu a iné faktory, ktoré ovplyvňujú výkonnosť a rozhodovanie reálnych ľudí.

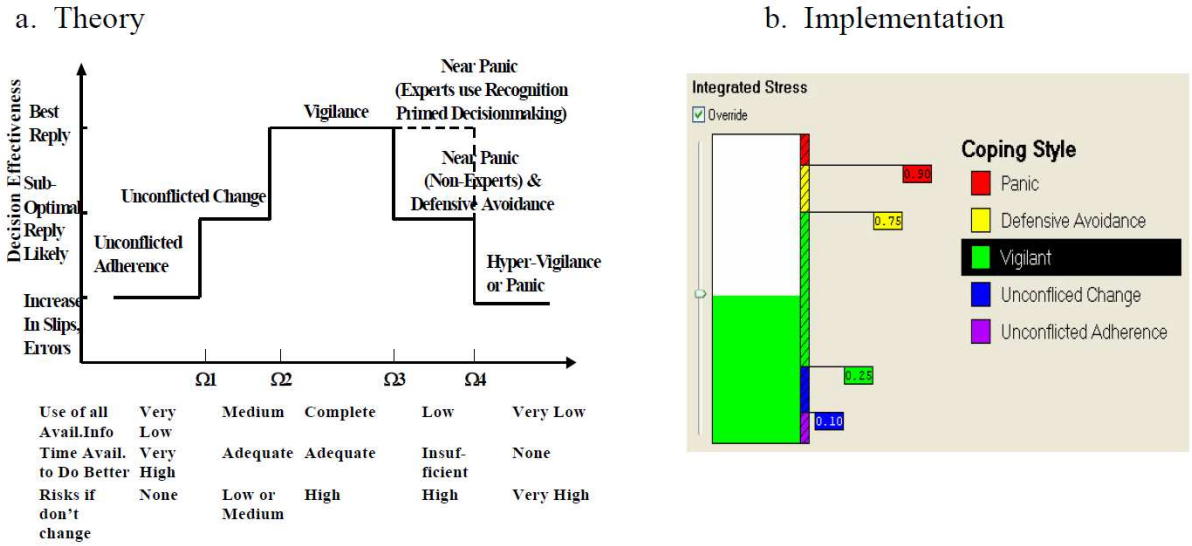
Pri skúmaní ľudského správania sa zabúda na integráciu a implementáciu – existujú stovky štúdií o ľudskom správaní a výkonnosti s ohľadom na demografické rozdiely, osobnostné rozdiely, emočné faktory, kognitívny proces, kultúru a iné. Tieto štúdie sú potenciálnym zdrojom informácií pre tvorbu agentov. Avšak takmer žiadna z týchto štúdii sa nevenuje interpretácii a pretransformovaniu nájdených poznatkov do formy, ktorá by bola vhodná na implementáciu. Často sú faktory ovplyvňujúce ľudské správanie len vágne kvantifikované.

Potreba validácie užitočných modelov správania – žiadny model nikdy nezachytí všetky detaily ľudských emócií a ich vplyvu na úsudok a rozhodovanie. Pri simulácii ľudského správania je však validita modelu aspoň z určitých aspektov dôležitá. Validácia modelu však nemá jednoduché riešenie. Je veľmi zložité spúšťať aktuálne simulácie a porovnávať ich s reálnymi situáciami z minulosti. Veľa modelov z odbornej literatúry o ľudskom správaní bolo odvodených z konkrétnej situácie, ale vývojári simulácií ich môžu chcieť použiť v inom kontexte. Takisto je problematické snažiť sa validovať modely umelej inteligencie, ktoré majú simulovať ľudské uvažovanie pri riešení úloh avšak s absenciou ľudského zdôvodnenia. Je vôbec možné validovať vplyv viacerých faktorov (napr. stres a únava), keď odborná literatúra je veľmi zameraná na štúdiu samotných vplyvov a nie vzájomnej interakcie?

Potreba validácie užitočných modelov správania – žiadny model nikdy nezachytí všetky detaily ľudských emócií a ich vplyvu na úsudok a rozhodovanie. Pri simulácii ľudského správania je však validita modelu aspoň z určitých aspektov dôležitá. Validácia modelu však nemá jednoduché riešenie. Je veľmi zložité spúšťať aktuálne simulácie a porovnávať ich s reálnymi situáciami z minulosti. Veľa modelov z odbornej literatúry o ľudskom správaní bolo odvodených z konkrétnej situácie, ale vývojári simulácií ich môžu chcieť použiť v inom kontexte. Takisto je problematické snažiť sa validovať modely umelej inteligencie, ktoré majú simulovať ľudské uvažovanie pri riešení úloh avšak s absenciou ľudského zdôvodnenia. Je vôbec možné validovať vplyv viacerých faktorov (napr. stres a únava), keď odborná literatúra je veľmi zameraná na štúdiu samotných vplyvov a nie vzájomnej interakcie?

Možným riešením na niektoré z uvedených problémov sú funkcie ***PMF(P****erformance* ***M****oderator* ***F****unction*) [20], ktoré boli odvodené na základe odbornej literatúry venujúcej sa ľudskému správaniu. Jedna z prvých štúdii v tomto odbore vytvorila Yerkes-Dodsonovu „obrátené u“ krivku, ktorá zobrazuje, že ak vplyv podnet alebo moderátora je zväčšený, výkonnosť je najskôr malá, potom sa zlepší a nakoniec znova upadá po prekročení istej hranice. To znamená, že výkonnosť môže byť lepšia v mierne

chaotickom a jemne stresujúcom prostredí, ako v prostredí v ktorom úplne absentuje stres. Model je zobrazený na obrázku (Obr. 2:18).



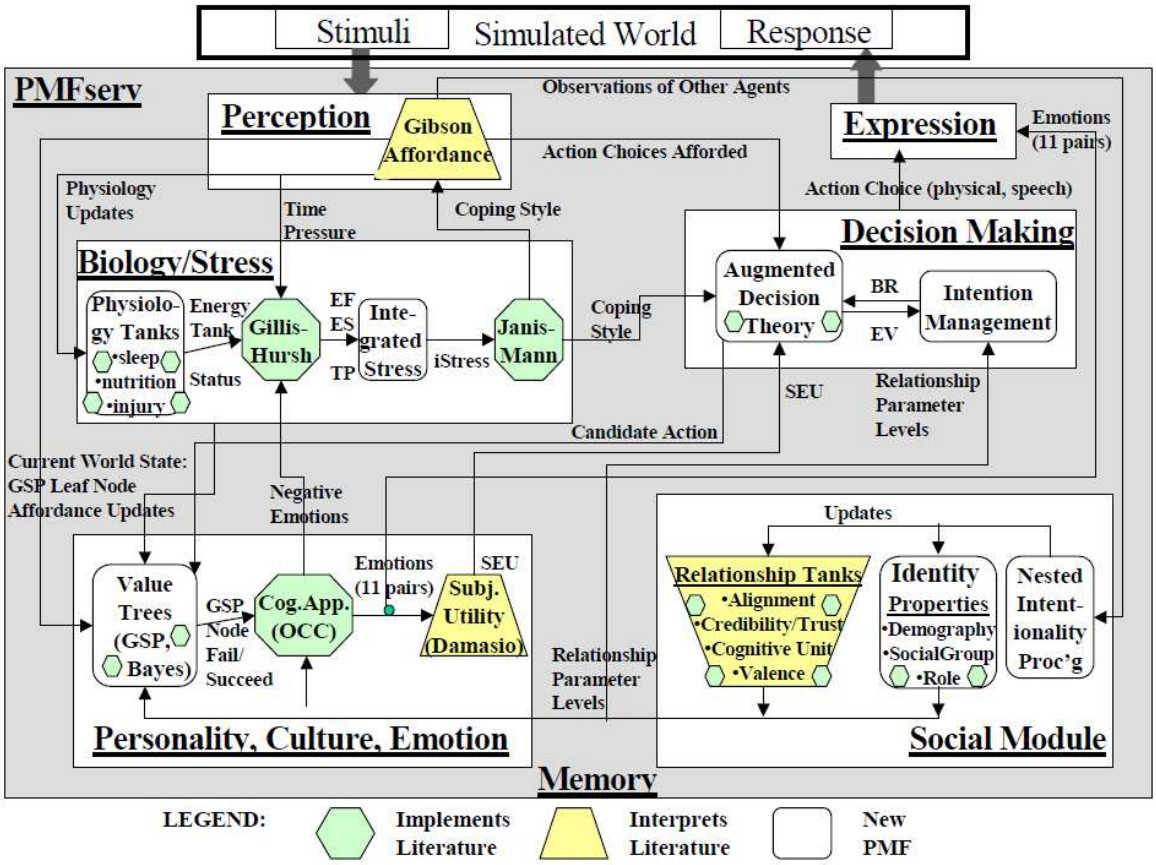
Obr. 2:18 PMF ako „obrátené u“ a jej implementácia (prevzaté z **[20]**).

Ako príklad PMF uveďme funkciu na výpočet integrovaného stresu agenta. Gillis a Hursh (1999) tvrdia, že hlavné zložky stresu sú: únava, stres z udalosti a časový stres. Pri výpočte integrovaného stresu sa sleduje *event stress (ES)*, ktorý reaguje na nepriaznivé a priaznivé udalosti, *time pressure (TP)*, čo je normalizovaný pomer dostupného ku požadovanému času na vykonanie úlohy a poslednou zložkou je *effective fatigue (EF)*. Funkcia na výpočet integrovaného stresu teda vyzerá nasledovne:

$$iSTRESS(t) = f\{ES(t), TP(t), EF(t)\}$$

Funkcia f{} je navrhnutá PMF funkcia, ktorá kvantifikuje úroveň integrovaného stresu agenta lineárnym sčítaním jednotlivých zložiek.

Navrhnutá bola aj celá unifikovaná architektúra (Obr. 2:19), ktorá využíva doteraz známe poznatky odbornej literatúry o ľudskom správaní s novo odvodenými funkciami *PMF*. V tejto architektúre agent vníma vnemy z prostredia, ktoré sú následne spracované a ovplyvňované faktormi ako sú *stres*, *osobnosť*, *emócie*, *pamäť* a následne sú vyhodnotené a agent vykoná náležitú reakciu.



Obr. 2:19 PMFserv implementácia unifikovanej architektúry (prevzaté z **[20]**).

2.12 Analýza šírenia emócií

2.12.1 Úloha

Analýza šírenia emócií a ľudského správania v dave.

2.12.2 Analýza

Pri analýze šírenia emócií v dave je nutné dav najskôr definovať. Dav je dočasné zhromaždenie väčšieho množstva ľudí v spoločnom priestore, v bezprostrednom kontakte, ktorí sú spojení určitou väzbou [21]. Viažu ich podobné alebo rovnaké intenzívne emócie a pudy vyvolané rovnakými dôvodmi. Dav sa správa odlišne ako jednotlivec:

- je veľmi dráždivý a ľahko popudlivý, stačí málo a môže dôjsť k výbuchu emócií,
- je ľahkovážny a veľmi jednoducho manipulovateľný,
- city a morálka sú veľmi zjednodušené, no o to silnejšie,
- dav je neschopný prijať argumenty alebo aspoň diskutovať. Inteligencia nie je silnou stránkou v dave, dav nie je schopný akcie vyžadujúcej vyššiu inteligenciu.
- pôsobí v ňom proces **sociálnej facilitácie** = prítomnosť druhých stimuluje jednotlivca k lepším výkonom.

Jedinec sa v dave správa odlišne:

- stráca racionalitu a kontrolu, čo môže viest k extrémnym prejavom,
- stráca pocit zodpovednosti, stáva sa anonymným a stráca zábrany = **deindividuácia**.

Dovoľujeme si popisovať emócie v dave, pretože demonštráciu považujeme za dočasné zhromaždenie osôb na určitom mieste, ktorí sú spojení istou väzbou, a teda dav.

2.12.2.1 Teórie o správaní osôb v dave

Na popísanie správania ľudí v dave vzniklo viacero teórií.

Le Bon – zákon mentálnej jednoty [22]

Dav je nebezpečný, lebo sa správa impulzívne a iracionálne. Neriadi sa vlastným svedomím, ale „zákonom mentálnej jednoty“. Kolektívna duša je však formovaná len dočasne, kým sleduje určité ciele.

Dôsledky deindividuácie [23]:

1. obmedzenie bežnej kontroly impulzívneho správania
2. zvýšenie citlivosti k emocionálnemu vzrušeniu a k situačným podnetom
3. neschopnosť sledovať alebo kontrolovať vlastné správanie
4. zníženie dôležitosti prikladanej spoločenskej prijateľnosti správania
5. zníženie schopnosti rozumného plánovania

Vyvrátenie teórie deindividuácie - štúdie futbalových fanúšikov ukázali, že sa vôbec nesprávajú impulzívne, ale podľa sociálnych konvencií. Futbalový dav má vlastnú sebareguláciu a normy správania a k násiliu dochádza častejšie vtedy, keď dôjde k narušeniu tejto sebaregulácie neprimeranou kontrolou autorít.

Existuje päť odporúčaní pre úspešnú kontrolu davu [23]:

- ponechať dav sebakontrole ak je to možné
- medzi políciou a organizátormi by mala byť efektívna komunikácia
- polícia by mala používať čo najmenšiu silu, aby nepôsobila ako provokujúci prvok
- tí, ktorí majú dav riadiť, by mali byť vyškolení v technikách účinnej interpersonálnej komunikácie
- polícia a úrady by mali byť za svoje konanie zodpovedné spoločnosti. Nemali by budiť dojem, že si môžu robiť čo chcú.

Teória vytvárajúcej sa normy

Ilúzia jednoty vzniká vytvorením normy. Situácia je často nejasná, ambivalentná a preto sa snažia ľudia nájsť to, čo je vhodné a akceptovateľné. Takto sa vytvoria v skupine normy. Účastníci davu ich odpozorujú od aktívnejších účastníkov davu. Ľudia konajú konformne s normou, lebo cítia určitý tlak v dave. Správanie v dave nie je iracionálne, ale je reakciou na normu [24].

2.12.2.2 Psychológia človeka

Psychologické faktory delíme na psychosociálne a psychofyzické. Pre realistickú simuláciu potrebujeme pochopiť oboje. Ako objekt pozorovania je vnímaný chodec.

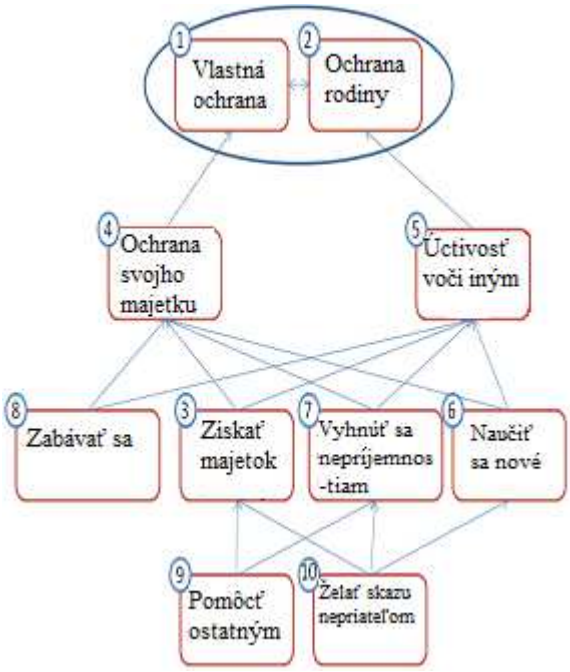
Vnímanie človeka:

- Horizontálne vnímanie delíme do troch kategórií:
  - vnímanie do 30° - človek vníma detaily,
  - vnímanie do 100° - človek vníma objekty,
  - vnímanie do 200° - človek vníma pohyb.
- Rýchlosť pohybu: závisí od hustoty davu, v ktorom sa pohybujeme. Kladek definoval:

$$V_{F,hi} = V_{F,hf} * [1 - e^{(-*\frac{1}{D} - \frac{1}{D_{max}})}]$$

- $V_{F,hi}$ je rýchlosť chodca v dave pri určitej hustote davu [m/s]
- $V_{F,hf}$ je rýchlosť chodca vo voľnom prostredí [m/s]
- μ je konštanta 1,913 p/m^2
- D je hustota davu [p/m^2]
- D_{max} je hustota davu aj so stojacimi chodcami

Ciele normálneho človeka sú zoradené podľa dôležitosti. Dajú sa reprezentovať pomocou stromu cieľov (Obr. 2:20).



Obr. 2:20 Strom štandardných cieľov (prevzaté z **[18]**).

Chodec si vyberá najrýchlejšiu cestu do cieľa. Je ale pravdepodobné, že si vyberie najpriamejšiu cestu, t.j. s minimom krížení a odbočiek a obchádzok.

2.12.2.3 Panika a jej šírenie

Dav zachvátený panikou nie je schopný triezvo hodnotiť situáciu. Vedomie je zúžené len na jednu emóciu – strach a pocit absolútneho ohrozenia.

Šírenie emócií je jav, pri ktorom stav emócií jednej osoby prenesie na iné osoby okolo explicitným aj implicitným spôsobom. Jav bol pozorovaní v menších aj väčších skupinách. Stáva sa, že aj okoloidúci, ktorý nevedia, čo sa udialo, môžu rozšíriť paniku.

Toto môžeme označiť za „zdedenú obavu“. Ľudia prejdú okolo seba a „zdedia“ paniku od susedov [25].
Šírenie paniky:

- niekoľko osôb v sociálnom styku zároveň predvedie prudký strach a všetci utekajú (alebo predvedú dezorganizáciu), alebo zostanú nehybní.
- Každý individuálny strach a jeho vyhodnotenie nebezpečenstva sú argumentmi alebo signálmi, ktoré prijíma od iných.
- Útek je indikovaný ako jediný mysliteľný spôsob konania, ktorý je odovzdávaný signálmi od ostatných.
- Snaha dosiahnuť bezpečie - ľudia šliapu jeden po druhom v zvláštnom úsilí dosiahnuť bezpečie.
- Vedia o prostredí – existuje len malý počet únikových ciest, majú dojem, že tieto cesty sa zatvárajú, útek je obmedzený.

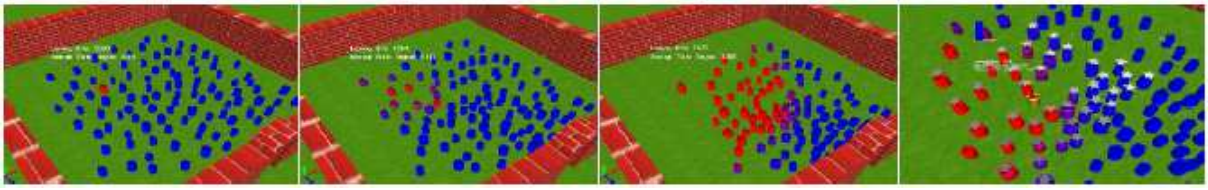
Pri davoch sa stretávame aj s pojmami masová a davová psychóza [26].
Masová hystéria:

- hystéria – túto chorobu možno popísať ako nervové alebo telesné „zrútenie sa“, ktorým postihnutý uniká zo situácie preňho neznesiteľnej.
- „choroba“ davu má rovnaké symptómy, dav uteká z neznesiteľnej situácie, utieka sa do konania bez rozmyslu, je vzrušený, dramatizuje…

Davová psychóza:

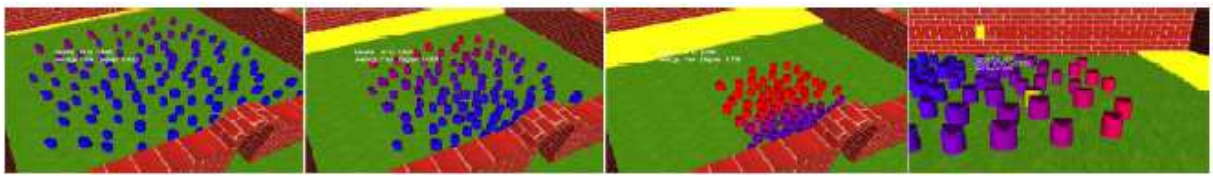
- všeobecný nepokojný duševný stav alebo poruchu duševnej rovnováhy až choromyseľnosť (masové samovraždy),
- zmanipulovaná masa, dav až hypnoticky ovládnutý vodcom.

Na obrázku nižšie (Obr. 2:21) je ukážka šírenia paniky v skupine osôb [27]. Skupina je v normálnom stave, červeného jedinca chytila panika. Panika sa šíri na ostatných naokolo a pozvoľne sa rozšíri do celého davu. Na obrázku vpravo vidieť jedincov s hviezdičkou, to sú tí, ktorí vedia prešíriť paniku na žltého jedinca.



Obr. 2:21 Šírenie paniky v dave, červení panikária, modrí sú v normálnom stave (prevzaté z **[27]**).

Na obrázku je (Obr. 2:22) vidieť šírenie paniky v dave pred blížiacim sa ohňom.



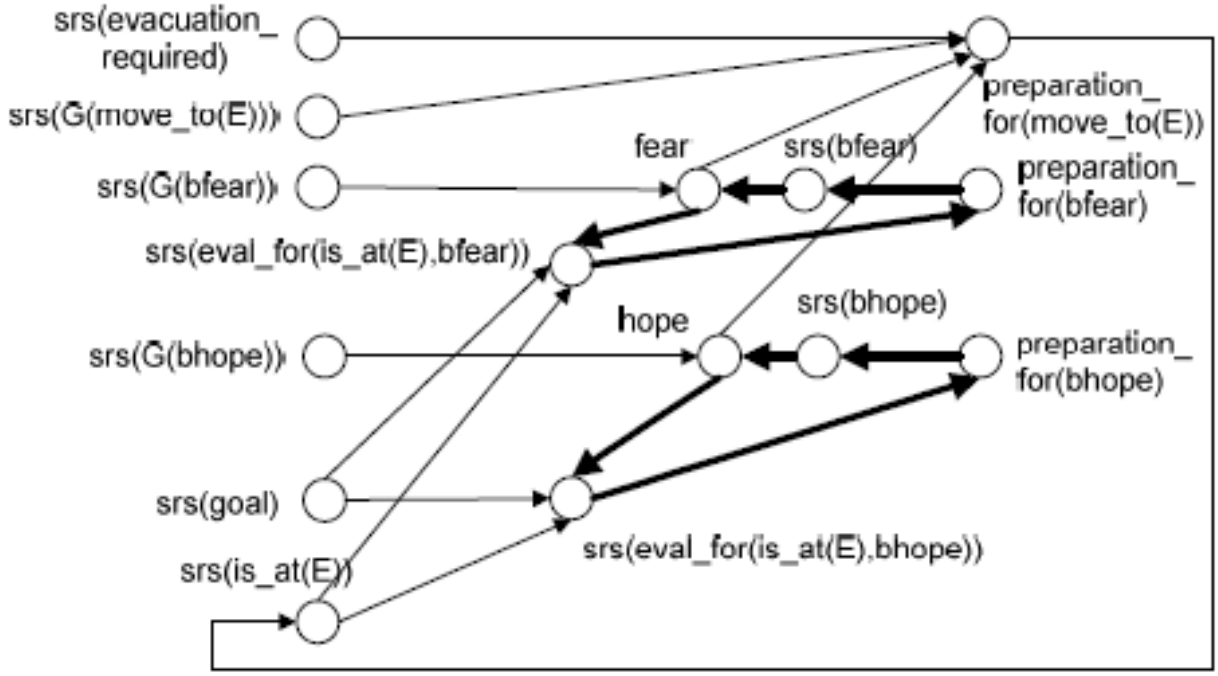
Obr. 2:22 Šírenie paniky pred ohňom (prevzaté z **[27]**).

Autori článku [28] stavajú na fakte, že emócie ovplyvňujú rozhodovanie ľudí. V článku skúmajú rolu emócií na rozhodovanie v rozsiahlom dave. Za týmto účelom navrhujú výpočtový model, ktorý integruje existujúce neurologické a kognitívne teórie

emocionálneho rozhodovania. Na základe tohto modelu simulujú niekoľko scenárov evakuácie veľkého davu.
Uvažujú dve hypotézy:

1. Emócie zvyšujú kontinuitu rozhodnutí v skupine a robustnosť skupiny voči externým narušiteľom
2. Emócie vznikajúce pri sociálnom rozhodovaní zvyšujú súdržnosť skupiny

Autori v článku simulujú evakuáciu davu agentov z horiacej stanice. Postupnosť akcií modelujú prostredníctvom reťazcov správania. Ako je vidno na obrázku (Obr. 2:23), situácia horiacej stanice vyvolá aktiváciu stavu agenta *evacuation_required*, ktorá vedie k príprave na akciu *preparation_for(move_to(E))*, kde E je jeden z východov z horiacej stanice.



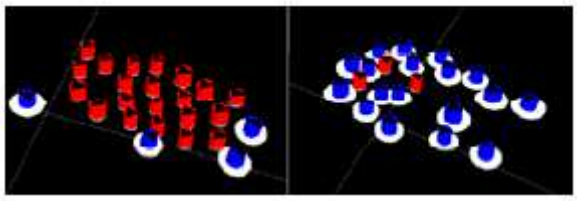
Obr. 2:23 Reťazce správania agentov (prevzaté z **[28]**).

Autori skúmali evakuáciu z rakúskej vlakovej stanice, ktorú verne vytvorili pomocou CAD nástrojov. Vo svojej simulácii uvažovali 1000 agentov reprezentujúcich ľudí. Vybraných 50 agentov bolo vybavených asistenčnými systémami, ktoré im poskytovali informáciu o zahltení každého východu. Vďaka tomu každý agent s asistenčným systémom má všetky potrebné informácie k úspešnej evakuácii a môže ich prešíriť ostatným agentom.
Následne na základe strachu a nádeji agentov autori skúmali ako sa emócie šíria v dave a koľko agentov sa rozhodne zmeniť únikový východ a do akej miery vplýva na ich rozhodnutie skupina, v ktorej sa nachádzajú a do akej miery agenti s asistenčnými systémami.
Analýzou výsledkov simulácie dospeli k záveru, že šírenie emócií v dave zvyšuje odolnosť skupín agentov meniť názor a podporuje kontinuitu rozhodnutí v skupine, čím sa potvrdili obidve hypotézy z úvodu.

2.12.2.4 Tlačenie v dave

V dave dochádza aj k tlačeniu ľudí medzi sebou [27]. Obrázok (Obr. 2:24) v hornej polovici ukazuje, ako sa obchádzajú dvaja normálni jedinci. Jedinci sa snažia obísť tak, aby do seba nenarazili. Druhá, spodná, polovica obrázku zobrazuje, ako sa obchádzajú jedinci, z ktorých jeden je v normálnom stave a druhý pociťuje paniku. Panikáriaci jedinec tlačí toho druhého. Na ďalšom obrázku (Obr. 2:25) vidieť, ako sa obchádzajú dve skupiny. V ľavej polovici obrázku sa panikáriaci jedinci tlačia cez dav osôb v normálnom stave, ktorí im urobia priestor. V pravej časti obrázku sa niekoľkí panikáriaci jedinci tlačia cez dav s osobami v normálnom stave a rozháňajú dav. Skupina panikáriacich má snahu byť pri sebe bližšie ako skupina v normálnom stave.

Obr. 2:24 Tlačenie sa dvoch osôb, biely krúžok zobrazuje osobný priestor (prevzaté z **[27]**).



Obr. 2:25 Tlačenie sa skupín (prevzaté z **[27]**).

2.12.2.5 Panický útek ako forma kolektívneho správania [29]

V dvadsiatych rokoch 19. storočia sa sociológovia zaoberali štúdiom kolektívneho správania. Štúdium zahŕňalo aj netradičné a emergentné správanie. Toto štúdium začalo diskusiu o rôznych druhoch paniky v riskantných situáciách, ako napríklad v prípade požiaru v preplnených budovách. V tej dobe prevládal názor, že panika je sprevádzaná pocitom extrémneho strachu, ktorý sa šíri medzi ľuďmi nákazlivým spôsobom. Výsledkom tohto šírenia strachu bol neracionálny panický útek jednotlivých ľudí, v ktorom sa jednotlivci často pri úteku ušliapali.

Avšak štúdie vykonané v polovici 20. storočia preformulovali niektoré závery ohľadne povahy paniky a podmienkach potrebných na jej šírenie. Výskum ukázal, že panický útek je veľmi vzácnym javom pri katastrofách. Dovtedy prevládajúci názor, že panika sa šíri nákazlivým spôsobom, bol spochybnený. Uvedené správanie sa nejavilo ako iracionálne z pohľadu zúčastnených osôb. Neexistovali žiadne dôkazy, že niektoré typy ľudí by boli náchylnejšie na panický útek.

Panický útek sa objavuje iba pri špecifických podmienkach. Zúčastnení ľudia musia veriť, že existuje bezprostredné riziko ohrozenia ich života. O útek sa nepokúšajú ľudia, ktorí si myslia že sú úplne obkľúčení. Inak povedané, musí existovať nádej, že útek je uskutočniteľný. Osoby, ktoré utekajú v panike, poznajú čo je hroziace nebezpečenstvo a konkrétne miesto, od ktorého utekajú. V tomto prípade prevláda strach, a nie úzkosť. Rozhodujúce je aj správanie ostatných zúčastnených ľudí.

2.12.2.6 Preventívne opatrenia

Podľa John J. Fruina môže byť väčšine davových nešťastí zabránené, jednoduchými stratégiami na manažovanie davu. Pojednáva o situáciách zahŕňajúcich športové podujatia, hudobné festivaly, nepokoje alebo protesty. Na vysvetlenie jednotlivých elementov týkajúcich sa davových nešťastí, John J. Fruin vytvoril model FIST(**F**orce, **I**nformation, **S**pace, **T**ime) [30]. Model je znázornený v tabuľke nižšie (Tabuľka 2.2).

Tabuľka 2.2 Model FIST.

Model FIST (Sila, Informácie, Miesto, Čas)	
Sila	Objavuje sa pri panike kvôli strkaniu a dominovému efektu tlačiacich sa ľudí
Informácie	Zahŕňa všetky spôsoby komunikácie zúčastnených ľudí, autorít a prostredia. Dôležitý faktor na šírenie paniky ako aj jej prevenciu alebo kontrolu
Miesto	Konfigurácia, kapacita a priepustnosť zariadení určujú stupeň hustoty davu, ktorý zohráva svoju úlohu pri davových nešťastiach
Čas	Element zrýchľujúci alebo spomaľujúci pohyb davu, teda ovplyvňujúci davovú hustotu

Sily, ktoré sa v dave môžu vytvoriť na základe dominového efektu, môžu dosahovať takých veľkostí, ktorým nie je možné odporovať alebo ich kontrolovať. Podľa Fruina tieto sily v dave môžu dosiahnuť viac ako 4500N.

Ako príklad časového faktoru Fruin uvádza postupne gradujúci príchod ľudí pred udalosťou, oproti masívnemu počtu a veľkej hustote ľudí počas odchodu po udalosti.

V realite je potrebné rozlišovať systematické plánovanie pred udalosťou a kontrolovanie davu, teda obmedzenia kolektívneho správania, v prípade núdze. Pajonk a Dombrowsky (2006) tvrdia: Intervencie po prepuknutí paniky nemajú takmer žiadne šance na úspech, preventívne opatrenia pred extrémnymi udalosťami naopak majú veľkú šancu na úspech. Nasledujúca tabuľka (Tabuľka 2.3) ukazuje opatrenia podľa Fruina, Pajonka a Dombrowskeho na prevenciu paniky a opatrenia počas prepuknutia nepokojov:

Tabuľka 2.3 Opatrenia na prevenciu paniky (prevzaté z **[30]**).

Manažment paniky a riadenie hromadných akcií	
Preventívne opatrenia: Systematické plánovanie, dozor, riadený pohyb a rozmiestnenie ľudí vrátane • Zbieranie množstva informácií o mieste a ľuďoch vopred, vrátane povahy skupiny a skúseností s podobnými skupinami • Neustále monitorovanie správania davu • Zvolenie proaktívneho manažmentu • Trénovanie osôb zodpovedných za manažment davu • Vyhodnotenie očakávaných typov aktivít a konania skupiny • Preskúmanie miesta s dôrazom na trasy príchodu, odchodu a únikové cesty a zabezpečenie ich priechodnosti • Vyčlenenie sektorov s priechodnými prístupovými cestami medzi sektormi • Kontrolovanie transportných ciest a zabezpečenie ich priechodnosti • Zabezpečenie jednoduchých a priamych rozmiestnení ciest • Meranie a obmedzenie intenzity pohybu • Zabezpečenie komunikácie	Pri nehode: Rýchle autoritatívne zásahy a obmedzenia limitujúce skupinové správanie, vrátane • Okamžité privolanie poriadkových zložiek • Dostatočne hlasné a jasné oznámenia • Poskytnutie jasných a jednoznačných informácií • Zameranie sa na podstatné veci • Informovať pravdivo • Prezentovať oznámenia vecným a rozvážnym spôsobom • Dávať konkrétne inštrukcie • Jednoduché úlohy • Obnoviť autonómnosť rozhodovania • Postaviť bariéry ktoré upravujú skupinový pohyb • Aplikovanie extrémnych opatrení ako použitie sily, zatknutí alebo hrozby prípadného zranenia

Zoznam poukazuje na niektoré kľúčové požiadavky, ktoré je treba dodržiavať vo všeobecnosti:

Informácie – zahŕňa monitorovanie a komunikáciu počas všetkých štádií, rôzne stupne plánovania, rozhodovania, organizácie a vykonania

Tréning – zabezpečenie dostupnosti osôb oboznámených so situáciou a zodpovedných za manažment davu

Jasnosť – vyjadrení, informácií, signálov a poskytovaných inštrukcií, umiestnenia východov

Úprimnosť – je potrebné informovať pravdivo a vybudovať dôveru zúčastnených za všetkých okolností

3 Druhý šprint

V nižšie uvedenej tabuľke (Tabuľka 3.1) je uvedený zoznam úloh, ktoré sa počas druhého šprintu vyriešili a osoby, ktoré sa danými úlohami primárne zaoberali.

Tabuľka 3.1 Zoznam úloh a k nim prislúchajúcich riešiteľov.

Názov úlohy	Zodpovedná osoba
Analýza fyzikálnych veličín a tlaku pôsobiaceho na človeka	Jana Branišová
Návod na rozbehanie SVN	Adrián Kollár
Realtime zmena parametrov RVO	Michal Kyžňanský
Návrh diagramu tried	Michal Kyžňanský
Analýza tagov a branchov v SVN	Michal Ošvát
Výber metrík na validovanie simulácie	Miroslav Ort

3.1 Analýza fyzikálnych veličín a tlaku pôsobiaceho na človeka

3.1.1 Úloha

Zanalyzuj fyzikálne veličiny a tlak pôsobiaceho na človeka.

3.1.2 Analýza

Tolerancia biomechanických zranení
AIS - Abbreviated Injury Scale, je na anatómii založený bodovací systém ohodnocujúci zranenie každej časti tela v závislosti od vážnosti. Tento bodovací systém rozdeľuje zranenia podľa vážnosti na šesť stupňov vážnosti [31]:

1. Minimálne
2. Mierne
3. Vážne
4. Ťažké
5. Maximálne (neliečiteľné – currently untreatable)

AIS rozdeľuje telo na 9 regiónov [31]:

1. Hlava
2. Tvár
3. Krk
4. Hrudník
5. Brucho
6. Chrbtica
7. Horné končatiny
8. Dolné končatiny
9. Povrch a iné

Tabuľka obsahujúca stručnú informáciu o silách, ktoré jednotlivé časti ľudského tela sú schopné vydržať sa nachádza nižšie (Tabuľka 3.2).

Tabuľka 3.2 Znášanlivosť ľudského hrudníka a brucšnej dutiny na tlak (prevzaté z **[31]**).

	Hrudník		Brucho	
Sila	Predný(frontal)	Bočný (lateral)	Predný(frontal)	Bočný (lateral)
Hrudná kosť(sternum)	3.3kN			
Hruď a ramená	8.8kN	10.2kN		
AIS 3+			2.9kN	3.1kN
AIS 4+		5.5kN	3.9kN	6.7kN

Model pôsobenia sily v dave

Faktor pôsobenia sily v dave je vhodné nezanedbať z týchto troch dôvodov:

1. Sila má priamy vplyv na pohyb
2. Sila dokážeme merať
3. Je vďaka nej možné simulovať zranenia v dave.

Zranenia v dave môžu byť od minimálnych až po smrteľné. Pri niektorých fatálnych demonštráciách došlo aj k prehnutiu kovových zábradlí. Na ohnutie takéhoto kovového zábradlia bolo potrebné vynaložiť silu viac ako 4500N. Sily pôsobiace v dave môžu byť skutočne obrovské a ich simulovaním by sme mohli pomôcť zistiť ako predísť obetiam na životoch.

Kirchnerov model

Tento model je založený na náhodnom riešení dynamiky davu a upriamuje sa na jednotlivca. Prostredie v tomto modeli je vnímané ako mriežka, teda je rozdelené na časti. Reprezentuje dva faktory individuálneho správania:

1. Silná túžba dostať sa k východu (uniknúť)
2. Nasledovanie ostatných

Využíva techniky inteligencie roja, ako napríklad zdieľanie šírenej informácie. Jeho veľkou nevýhodou je, že absentuje silový faktor.

Swarm force model

Tento model tvorí nadstavbu Kirchnerovho modelu. Snaží sa ho rozšíriť o pôsobenie častíc vyvíjajúcich sa v čase reprezentujúcich silu, ktoré by sa prenášali agentmi a na agentov.

Základné aspekty sily

- Má smer
- Má dôsledky – uvažovali nad dôsledkom zapríčiňujúcim zranenie a stratu individuálnej kontroly
- Rastie s počtom agentov, ktorý touto silou na niečo pôsobia

- Sila je generovaná zámerne - agenti pôsobia, silou keď im niečo stojí v ceste a oni sa tam túžia dostať

Oproti Kirchnerovmu modelu obsahuje tento aj polia s dynamickým pôsobením síl. Sila nie je totižto častica, ale je vektorom
Sila je v ňom reprezentovaná ako častice, ktoré tvoria jednotky silového vektora. Silový vektor má okolo seba silové pole, ktoré určuje smer a veľkosť(počet častíc) týchto častíc na agenta na políčku mriežky.
Sila začína byť generovaná, keď sa agent nemôže dostať na políčko, na ktoré chce, pretože je osadené iným agentom. Kapacita sily, ktorú môžu agenti vynaložiť sa zadáva ako parameter. Silové pôsobenie sa prenáša aj na susedné bunky. Sila v bunke a sila, ktorá na ňu pôsobí sa skalárne sčítavajú. Ak silové pole zasahuje do prázdnej bunky alebo bunky, v ktorej sa nachádza neživý objekt, sila sa stráca.
Agenti nemôžu vstúpiť na políčko kde už stojí iný agent.
Zranení agenti prestávajú byť aktívny, prestávajú sa hýbať a ostatní agenti sa k nim správajú ako k prekážkam. Agenti pôsobiaci opačnými silami môžu zraniť agenta, ktorý stojí medzi nimi [32].

3.1.3 Zhrnutie analýzy

Pôsobenie síl v dave nám umožní simulovať zranenia, a tak pomôcť prechádzať takýmto situáciám. Je dôležité poznať aké pôsobenie sily človek ešte znesie a aké už nie. Rozhodli sme sa uvažovať nad silou pôsobiacou na hrudník. Analyzovaním Swarm force modelu sme dostali základný obrázok o tom, ako by mohlo pôsobenie síl v našom projekte prebiehať a aký by mohli mať efekt.

3.2 Návod na rozbehanie SVN

3.2.1 Úloha

Návod na správne napojenie na SVN repozitár a spustenie projektu v NetBeans.

3.2.2 Implementácia

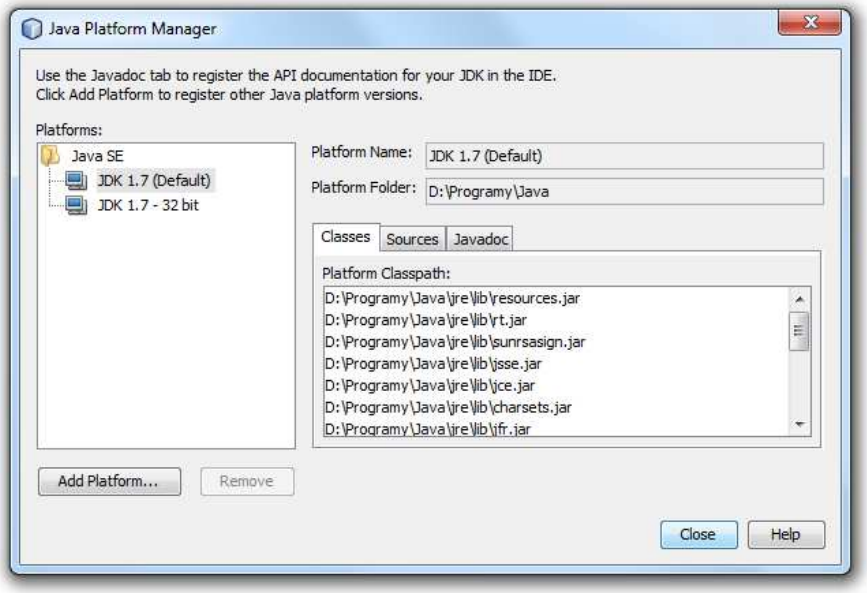
Na inštaláciu a spustenie aktuálnej verzie projektu pre 64-bitový operačný systém Windows je nutné mať nainštalované:

- Inštalácia JDK verzia x86 [33],
- Inštalácia Java3D [34],
- Inštalácia Visual C++ 2012 Redistributable, verzia x86 [35].

Po inštalácii je potrebné vykonať nasledovné kroky:

1. Vykonať checkout projektu v Netbeanse z SVN repozitára podľa návodu na SVN s nasledujúcimi zmenami:
    a. Repository URL zadať: svn://team09-12.ucebne.fiit.stuba.sk:443/svn/netbeans
    b. Na základe dohody umiestniť projekt priamo na disk C:/ t.j. local folder pri operácii checkout nastaviť na C:/

2. Po stiahnutí projektu z SVN repozitára zvoliť open project. Ak sa objaví chyba referencie na JDK, zvolíte resolve. Následne by sa malo zobraziť nasledujúce okno, v ktorom treba zvoliť 32 bitové JDK (Obr. 3:1):

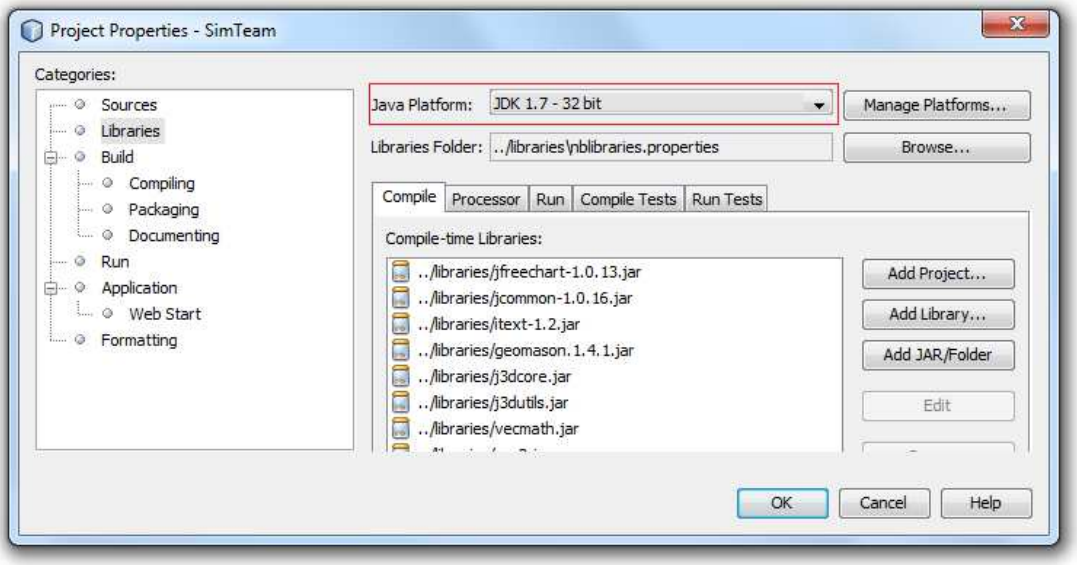


Obr. 3:1 Pridávanie novej platformy (32-bitovej verzie Java) do prostredia NetBeans.

V prípade, že v zozname platforiem sa 32 bitové JDK nenachádza, zvolíte Add Platform a vyhľadáte 32 bitové JDK v súborovom systéme. Typicky sa na 64 bitových systémoch nachádza 32 bitová verzia JDK v adresári C:/Program Files (x86)/Java/jdk1.7.0_09.

Ak sa chyba referencie na JDK neobjaví, je možné nastaviť 32 bitové JDK aj nasledujúcim spôsobom (Obr. 3:2):

a. Pridanie 32 bitového JDK do zoznamu platforiem je možné vykonať zvolením Tools > Java Platforms > Add Platform
b. Nastavenie 32 bitového JDK sa vykoná kliknutím pravého tlačidla myši na projekt a zvolením Properties > Libraries > Java Platform

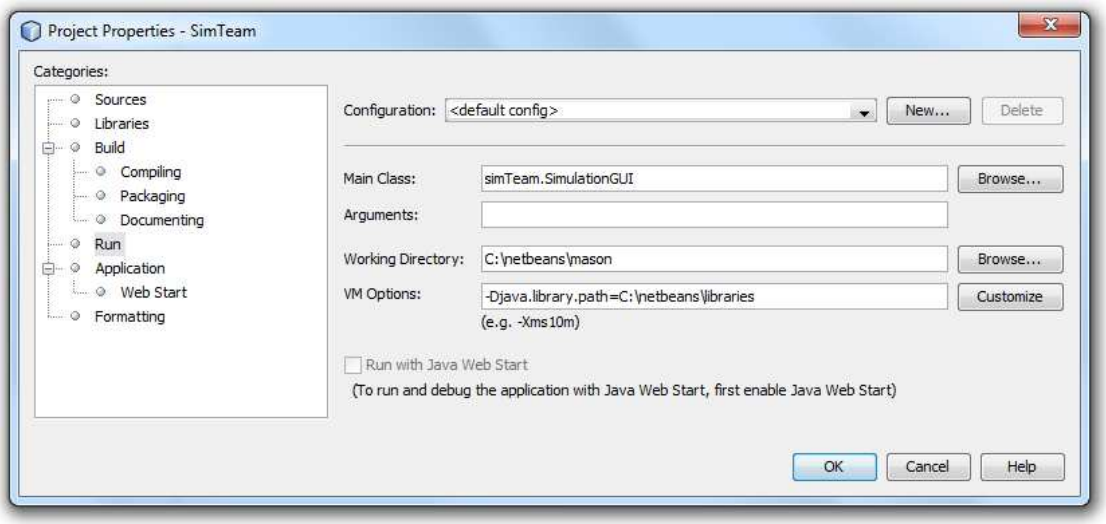


Obr. 3:2 NetBeans obrazovka so zoznamom knižníc a voľbou platformy Java.

3. Zvoliť Run project v Netbeans a projekt by sa mal spustiť. Po spustení projektu by sa mali objaviť 2 okná a pre spustenie simulácie je potrebné v nasledujúcom okne v dolnej lište kliknúť na tlačidlo play (Obr. 3:3):

Obr. 3:3 Hlavné okno bežiacej aplikácie vytvorenej v prostredí frameworku MASON.

4. V prípade problémov kliknúť pravým tlačidlom na projekt a zvoliť Properties. Skontrolovať a prípadne upraviť záložku Categories > Run nasledovne (Obr. 3:4):
   - Main Class: simTeam.SimulationGUI
   - Working Directory: C:\netbeans\mason
   - VM Options: -Djava.library.path=C:\netbeans\libraries



Obr. 3:4 Obrazovka s nastaveniami behu aplikácie v NetBeans.

3.3 Realtime zmena paramatrov v RVO

3.3.1 Úloha

Analýza zmeny vstupných parametrov v knižnici RVO v reálnom čase.

3.3.2 Analýza

Knižnica RVO poskytuje nasledujúce parametre [7]:

- maxNeighbors – maximálny počet iných látok agenta berie do úvahy navigáciu,
- maxSpeed – maximálna rýchlosť agenta,
- neighborDist – maximálna vzdialenosť od iných agentov, ktorá sa berie do úvahy pri navigácii,
- position – aktuálna pozícia agenta,
- prefVelocity – aktuálna preferovaná rýchlosť agenta, tj. taká, ktorú by agent mal, keby neexistovali ostatní agenti,
- radius – sféra pôsobenia agenta,
- timeHorizon – minimálny čas, pri ktorom sú rýchlosti agentov, počítané v simulácii, bezpečné s ohľadom na ostatných agentov,
- timeHorizonObst – minimálny čas, pri ktorom sú rýchlosti agentov, počítané v simulácii, bezpečné s ohľadom prekážky,
- velocity – aktuálna rýchlosť agenta.

Počas simulácie vieme meniť týchto 6 parametrov:

- neighborDist,
- maxNeighbors,
- timeHorizon,
- timeHorizonObst,
- radius,
- maxSpeed.

3.3.3 Implementácia

Agentom je možné okrem spomenutých 6 parametrov meniť tieto parametre pre každého zvlášť, a to dokonca aj pri akomkoľvek kroku simulácie. Toto je využité pre vytvorenie agentov, ktorí stoja na mieste - policajti. Ďalej metóda ***queryVisibility()*** dokáže povedať, či agent daný bod vidí v závislosti na rozmiestnení prekážok. Taktiež je možné zistiť aktuálnu rýchlosť agenta.

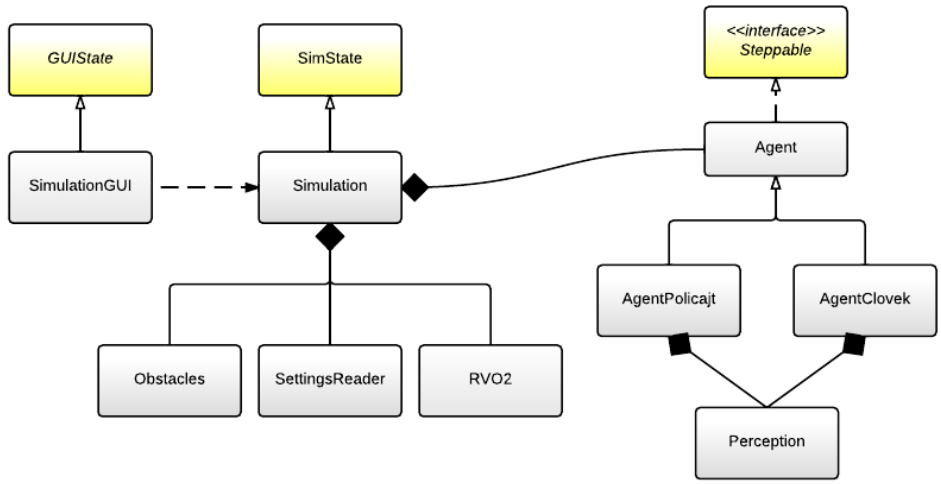
3.4 Návrh diagramu tried

3.4.1 Úloha

Návrh diagramu tried v pre navrhované riešenie simulácie.

3.4.2 Návrh

Diagram tried je vyobrazený na nasledujúcom obrázku (Obr. 3:5).



Obr. 3:5 Návrh diagramu tried.

3.5 Analýza tagov a branchov v SVN

3.5.1 Úloha

Analýza použitia tagov a branchov v SVN

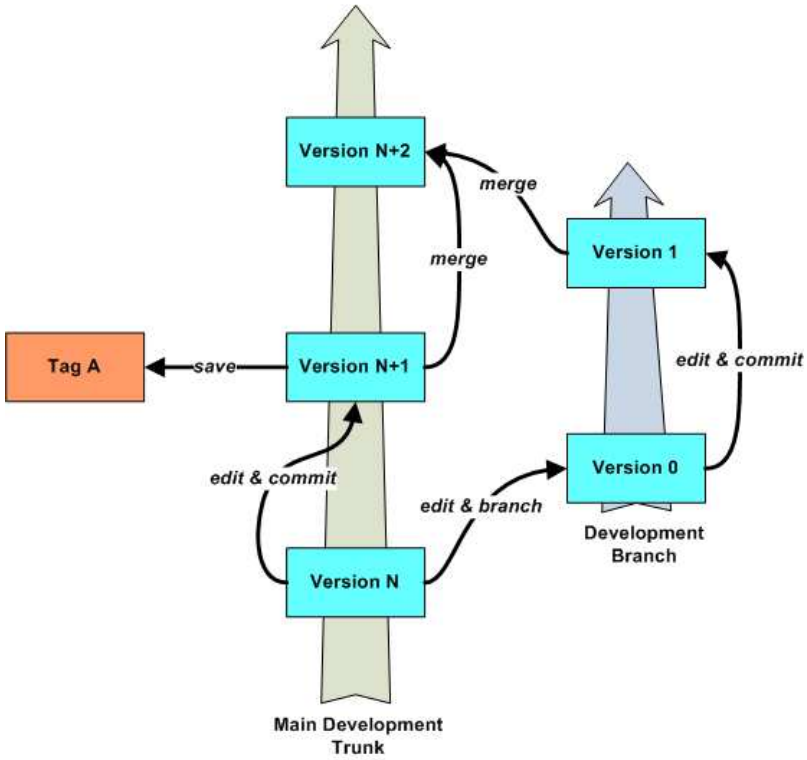
3.5.2 Analýza

Verziovací systém (Subversion-SVN) je nástroj, ktorý používa tím primárne na verziovanie zdrojového kódu [36]. Jeden z najlepších aspektov SVN je možnosť pracovať nad zdieľaným kódom. Celý tento zdieľaný kód sa nazýva **TRUNK**. Jedná sa o kód nad ktorým budeme denne pracovať a rozvíjať ho.

Trunk si môžeme predstaviť ako kmeň stromu pri vývoji. Je to určitý základ nášho projektu. Od kmeňa sa neskôr odvíjajú vetvy (**BRANCH**) [37]. Tie majú všetky určitú časť zdrojového kódu spoločnú (konkrétne kmeň - trunk).

TAG predstavuje v podstate vetvu/brench, ktorú budeme považovať za ucelenú časť pri vývoji. **TAG** sa po vytvorení ďalej **nemení**. Z dátového hľadiska je branch a tag to isté.

Na stiahnutie aktuálneho repozitára zo servera na lokálny disk sa používa príkaz **UPDATE**. Proces, ktorý naše lokálne vykonané zmeny odovzdá na server sa nazýva **COMMIT**.



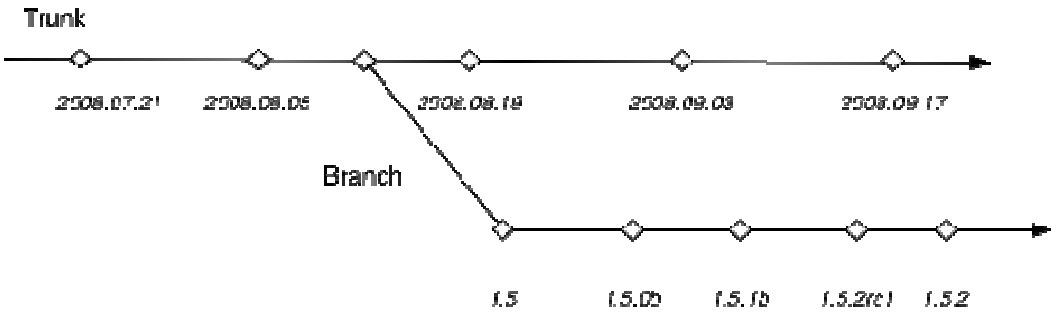
Obr. 3:6 Príklad na verziovanie softvéru (prevzaté z **[38]**).

Ako vyplýva z obrázku (Obr. 3:6), trunk predstavuje na začiatku určitý základ, od ktorého sa ďalej odvíja nová vetva (Version 0), pričom vývoj prvotnej verzie N pokračuje. Neskôr je možné spojiť vytvorenú verziu 1 inej vety, s verziou N+2 z hlavnej vetvy. Tag vytvorený pri verzii N+1 predstavuje určitú časť zdrojového kódu, ktorú interne považujeme za ucelenú (a nazveme ju napr. v.1.0).

Pri práci sa môže vyskytnúť problém, ktorý sa rieši príkazom **MERGE**. Jedná sa o konflikt verzií, ak napr. dvaja používatelia naraz pracujú s jedným súborom a rôzne ho doplnia/upravia. Potom je možné konflikt riešiť nasledovnými 3 spôsobmi:

- Prepísať a aktualizovať / Override and update
- Prepísať a použiť commit / Override and commit
- Spojiť zmeny / Merge changes

Nová vetva pri vývoji sa nazýva **Branch**. Ako vyplýva z obrázku nižšie (Obr. 3:7), Trunk predstavuje základ, a neskôr je možné pri vývoji vytvoriť nové vetvy za rôznym účelom (iná verzia softvéru, potreba inej funkcionality a pod.)



Obr. 3:7 Príklad na proces vývoja (prevzaté z **[38]**).

3.5.3 Zhrnutie analýzy

Pri vývoji začíname z jednotného bodu (trunk). Neskôr môžeme (ak to bude potrebné) po dohode celého tímu vytvoriť inú vetvu (branch). Ak vytvoríme určitú ucelenú časť, môžeme ju označiť určitým číslom verzie (1.0 , 2.0 atď.) a označiť ako TAG.

3.6 Analýza koordinácie poriadkových zložiek

3.6.1 Úloha

Analýza poriadkových zložiek

3.6.2 Analýza

Poriadkové zložky predstavujú jednu špecifickú skupinu ľudí v simulácii. Ich správanie, resp. pohyb je dôležité koordinovať, a táto koordinácia by mala slúžiť na zabezpečenie požadovaných cieľov.
Analyzovala sa práca [39]. Jedná sa o simuláciu pohybu ľudí v meste. Policajné/poriadkové zložky majú zabezpečovať asistenciu a pomoc, prípadne usmerňovať dav ľudí. Model policajných zložiek neobsahuje osobnostné charakteristiky alebo emócie, ale vie plánovať cestu (Path planning). Jedná sa konkrétne o dva typy policajných agentov- „Beat Patrol“ a „Dispatcher“. Dispatcher zbiera informácie od Beat Patrol agentov, a môže im zadať rôzne úlohy. Beat Patrol môže taktiež požiadať Dispatcher-a o ďalších Beat Patrol agentov (ak to vyžaduje situácia). V našej simulácii by sme mohli *zvážiť* použitie tejto metódy, t.j. požiadanie o posily, pokiaľ to situácia bude vyžadovať.
Podľa [40] je úspešná práca policajných zložiek vtedy, ak splnia svoj účel, a tým je rozdelenie davu na dve polovice, a konkrétne vytvorenie cesty pre prechod automobilu. Koordinácia (ako aj komunikácia) ľudí v simulácii môže byť komplikovaná, pokiaľ berieme do úvahy aj správanie a emócie.
Koordinácia poriadkových zložiek by mala byť založená na komunikácii medzi agentmi a na snahe dodržať vopred definované správanie pre dosiahnutie cieľa, ktorý môže byť rôzny. V analýze poriadkových zložiek sú definované rôzne ciele, ako napríklad:

- snaha udržať dav ľudí pod kontrolou,
- rozohnanie davu,
- zablokovanie určitej časti priestoru.

Pri všetkých cieľoch je koordinácia rozdielna a správanie členov poriadkových zložiek je potrebné zadefinovať špecificky pre konkrétny cieľ. Pri snahe udržať dav ľudí pod kontrolou by sa malo jednať hlavne o vytvorenie súvislej línie členov poriadkových zložiek a následné neprepustenie člena demonštrácie za svoju líniu.

Rozohnanie davu je už zložitejšie, ale koordinácia môže spočívať v náhodnom prechádzaní poriadkových zložiek medzi ľuďmi a tak by sa malo zabrániť vytváraniu skupín a zvyšovaniu napätie v dave.

3.7 Výber metrík na validovanie simulácie

3.7.1 Úloha

Výber vhodných metrík, ktoré použijeme pri validácii simulácie demonštrácie.

3.7.2 Analýza

Validovať simuláciu znamená porovnať, či sa simulovaný objekt správa počas simulácie rovnako alebo veľmi podobne ako v reálnom svete. Simulácia demonštrácie má svoje charakteristiky. Pri validácii sa vyskytujú nasledovné problémy [14]:

- Dát je málo, živé modely sa robiť nedajú. Jediná validácia je na základe dát z minulosti, ale ak simulácia nasimuluje presne udalosť, ktorá sa stala, to neznamená, že vie určiť, ako bude prebiehať udalosť v budúcnosti.
- Problém s pozorovaním dát v reálnom svete.
- Ohodnotenie kvality dát. Ako sa dá určiť, čo je radosť, hnev, ...?
- Správanie skupiny je určené prostredím, ľuďmi, udalosťami, ktoré sa odohrali. Kombinácia týchto parametrov tvorí príliš komplexný systém. Malá zmena nemusí mať predpokladaný účinok.
- Malé množstvo dokumentácie k modelovaniu sociálnych systémov a ich simulácie.

Napriek všetkým prekážkam sa pokúsime simuláciu validovať. Demonštrácia má vždy dva základné typy osôb, demonštrantov a poriadkové zložky. Nadefinujeme cieľ demonštrácie a príslušnú psychológiu demonštrantov. Zároveň implementujeme reálne techniky práce poriadkových zložiek. Takúto simuláciu ešte overíme tým, že nájdeme dobrý referenčný zdroj. V našom prípade to bude ukážka reálnej demonštrácie. Referenčný zdroj nemusí byť len jeden. V každej reálnej demonštrácii boli použité rôzne postupy poriadkových zložiek ako aj prostriedky použité demonštrantami. Preto si simuláciu overíme najlepšie pri použití viacerých zdrojov. Parametre simulácie nastavíme podľa referenčného zdroja. Toto kladie na simuláciu aj ďalšie nároky ako flexibilitu a rozšíriteľnosť a to preto, lebo parametre simulácie bude musieť byť možné meniť. Ak dostaneme výsledky simulácie rovnaké alebo podobné ako v prípade referenčného zdroja, budeme simuláciu pokladať za validnú.

Parametre, ktoré nastavíme podľa referenčného zdroja [41]:

- počet demonštrantov,
- počet príslušníkov poriadkových zložiek,
- veľkosť územia demonštrácie a prekážky na území,

- rýchlosť postupu jedincov,
- priemerná hustota jedincov,
- použité prostriedky demonštrantov a poriadkových zložiek.

3.7.3 Testovanie

Výsledky simulácie musia byť rovnaké alebo podobné ako v referenčnom zdroji, aby sme prehlásili simuláciu za validnú. Rovnaké alebo podobné výsledky znamená:

- demonštrácia bude ukončená rovnakým spôsobom,
- demonštrácia skončí za rovnaký čas,
- počet zranených a zatknutých jedincov bude rovnaký.

Môže sa stať, že sledované parametre simulácie nebudú plne zhodné s referenčným modelom. Pri simulácii pripustíme istú odchýlku. Táto odchýlka bude empiricky stanovená po prvých testoch simulácie. Budeme sledovať aj parametre týkajúce sa pohybu jednotlivcov a ich vzájomného vyhýbania sa ako aj vyhýbania sa prekážok. Na tieto účely používame overenú knižnicu RVO. Budeme sledovať, či vyhýbanie sa jednotlivcov, ktorú knižnica zabezpečuje, zodpovedá realite a najmä zákonu najmenšieho vynaloženého úsilia. Pri nastavovaní všetkých parametrov budeme postupovať podľa pozorovania tak, aby sme simuláciu čo najviac priblížili referenčnému modelu pomocou EM-algoritmu.

4 Tretí šprint

V nižšie uvedenej tabuľke (Tabuľka 4.1) je uvedený zoznam úloh, ktoré sa počas tretieho šprintu, ktorý sme interne pomenovali Noc Ohňov, vyriešili a osoby, ktoré sa danými úlohami primárne zaoberali.

Tabuľka 4.1 Zoznam úloh a k nim prislúchajúcich riešiteľov.

Názov úlohy	Zodpovedná osoba
Plánovanie trasy agentov	Adrián Kollár
Plánovanie trasy agentov	Filip Pakan
Code Review	Michal Kyžňanský
Návrh PECS a jeho realizácia	Michal Kyžňanský
Návrh PECS a jeho realizácia	Miroslav Ort
Analýza pôsobenia tlaku	Jana Branišová
Analýza pôsobenia tlaku	Michal Ošvát
Analýza pôsobenia tlaku	Miroslav Ort

4.1 Code Review

4.1.1 Úloha

Kolektívna úloha, ktorej cieľom bolo oboznámiť ostatných členov tímu so získanými poznatkami, ktoré vyplynuli z integrácie jednotlivých komponentov projektu (GEOMASON, RVO2) a implementácie prvého scenára simulácie. Úloha pozostávala z dvoch častí:

a) Prezentácia kľúčových častí (metód a konštrukcií) implementovaného projektu

b) *Diskusia a odpovedanie na nejasnosti ostatným členov tímu.*

4.1.2 Zhrnutie analýzy

Témy objasňovania kódu:

1. **Predstavenie štruktúry projektu (UML diagram) a demonštrácia na zdrojovom kóde.**

   Prezentovaná implementácia pozostávala z troch tried (Agent, Simulation, SimulationGUI), pričom bola prezentovaná rozšírená a odobrená verzia návrhu v podobe UML diagramu, kde sú jednotlivé zložky správania v podobe PECS modulov rozbité do samostatných tried (viď. obrázok č. 1). Trieda Agent je abstraktná a jej inštancia ani nikdy nutná nebude. Obsahuje však základne mechanizmy opisujúce triviálne správanie agenta a taktiež agreguje vyššie spomínané moduly obsahujúce stavové premenné a metódy modulov PECS. Z triedy Agent vychádzajú dvaja priami potomkovia - trieda AgentClovek a AgentPolicajt, ktoré predstavujú konkrétne implementácie resp. stratégie určujúce skupinu agentov so všetkými špecifikami danej skupiny. Trieda Simulation agreguje všetkých agentov, pričom niekoľko ďalších tried je rovnako

vo vzťahu agregácie k tejto triede kvôli oddeleniu záujmom (separatin of concerns) a prehľadnosti.

2. **Objasnenie hodnôt RVO parametrov.**
   Modul RVO, ktorého úlohou je riešiť vzájomné kolízie a obchádzanie agentov pri pohybe za svojím cieľom disponuje šesticou parametrov, ktoré ovplyvňujú správanie simulácie resp. každého agenta. Parametre sa nastavujú pre celú simuláciu, avšak v ktoromkoľvek bode behu simulácie je možné pre ľubovoľného agenta parametre modifikovať. Popis parametrov:

   neighborDist - *max. vzdialenosť agentov, ktorých bude daný agent brať do úvahy pri plánovaní pohybu.*
   maxNeighbors - max. počet agentov, ktorých bude daný agent brať do úvahy pri výpočtoch. Čím vyššie číslo => tým viac stúpajú nároky na výpočtovú kapacitu. Ak príliš nízke, simulácia nie je korektná.
   timeHorizon - min. čas reakcie daného agenta na iných agentov. Čím vyššia hodnota => tým skôr reaguje na prítomnosť iných agentov, avšak menšia sloboda pri výbere pohybu. (hodnota > 0)
   timeHorizonObst - min. čas reakcie agenta na prekážky. Čím vyššia hodnota => tým skôr agent reaguje na prekážky, avšak menšia sloboda pri výbere smeru pohybu. (hodnota > 0).
   radius - rádius videnia agenta. (hodnota >= 0).
   maxSpeed - *max. rýchlosť agenta pri pohybe (*hodnota >= 0*).*
   Všetkých 6 parametrov možno nastaviť metódou ***setAgentDefaults***() triedy RVOSimulator, ideálne na začiatku simulácie. Pre každého agenta možno meniť tieto parametre metódami s prefixom setAgent + zvolený parameter v konvencii CamelCase, pričom táto metóda berie ako argument ID agenta vo svete RVO a hodnotu parametra, ktorý meníme.

   Príklad: ***setAgentMaxSpeed****(agentID, newMaxSpeed)*

3. **Vysvetlenie dôležitých metód jednotlivých tried a ich úloha v projekte.**
   Trieda Agent
   public void step(SimState state)
   Metóda predstavuje krok agenta v simulácii a je cyklicky volaná triedou Simulation na každého agenta, ktorého obsahuje (v náhodnom poradí). Je to miesto, kde sa budú vykonávať všetky procesy súvisiace so správaním agenta.

   public boolean reachedGoal(Simulation sim)
   Metóda zisťuje, či sa agent dostal dostatočne blízko aktuálnemu cieľu a vracia hodnotu true/false.

   public void setPreferredVelocity(Simulation sim)

Metóda vyhodnocuje podmienku, či sa agent dostal do žiadanej vzdialenosti od cieľa (jeho pohyb bude potom nulový vektor), alebo nie, čím aktivuje RVO vyhýbanie a následne sa vypočíta vektor pohybu. Metóda je iba prepísanou Java verziou pôvodnej, ktorá je dostupná v manuáli knižnice RVO.

Trieda Simulation

public void loadConfiguration()

V tejto metóde sa budú vykonávať všetky počiatočné načítavania parametrov z externých zdrojov (XML súbor).

public void start()

Metóda inicializujúca všetky aktivity simulácie, t.j. prečistenie dátových vrstiev, vytvorenie agentov, načítanie mapy a prenos do RVO sveta spolu s definovaním krokovania simulácie, ktorá bude rovnakým časovým krokom zviazaná a zosynchronizovaná s krokovaním simulátora, ktorý obsahuje trieda RVOSimulator.

private void addPolygon(RVOSimulator rvo_sim2, Polygon p)

private void addMultiPolygon(RVOSimulator rvo_sim2, MultiPolygon mp)

Vyššie uvedené metódy majú na starosti transformáciu načítanej mapy do polygónov, ktorým rozumie trieda RVOSimulator.

private void loadGISData()

Metóda načítava mapové podklady zo súboru pomocou triedy ShapeFileImporter a synchronizuje na základe MBR (minimal bunding box) veľkosti, aby bolo premietnutie vrstiev korektné.

private void shiftWorldToGroundZero()

Metóda z externého zdroja (SAV), ktorá rieši nepresnosti ohľadne premietnutia mapy (double/float). Avšak riešenie nie je dostatočné, pretože premietnutie spôsobuje napriek tomu problémy.

private void setupRVOScenario()

Metóda vykonáva všetky operácie súvisiace s inicializovaním a naplnením RVO. Vytvorí inštanciu triedy RVOSimulator a nastaví predvolené hodnoty parametrov pre všetkých agentov simulácie. Následne volá vyššie spomínané metódy loadGISData() a shiftWorldToGroundZero(). Po pridaní jednotlivých polygónov mapy do sveta RVO volaním metódy processObstacles() zabezpečí, aby ich agenti brali do úvahy ako prekážky.

Trieda SimulationGUI

public void setupPortrayals()

Metóda nastavuje zobrazovacie vrstvy, t.j. napája dátové zdroje na vizuálnu stránku simulácie, nastavuje spôsob zobrazenia objektov v týchto dátových vrstvách (farba, tvar, veľkosť / mierka).

4. **Identifikácia zbytočných častí kódu.**
   Na stretnutí boli identifikované aj časti kódu, ktoré v danom momente nemali žiaden význam a boli pozostatkom rôznych implementačných testov a vyradených konštrukcií. Príkladom je atribút position triedy Agent, ktorý kvôli využitiu geometrických dátových vrstiev nie je potrebný.

Otázky členov tímu:

1. **Ako modifikovať parametre RVO pre konkrétneho agenta?**
   Parametre konkrétneho agenta možno modifikovať podľa vyššie spomínaných pravidiel, pričom v tele agenta sa musíme odvolať na referenciu triedy Simulation, ktorá obsahuje inštanciu triedy RVOSimulator. Každému agentovi je pri jeho vytváraní resp. vkladaní do sveta RVO vygenerovaný unikátny identifikátor ID, podľa ktorého je ho možné jednoznačne identifikovať pri volaniach metód triedy RVOSimulator. Jedná sa o zápis ***this.id***.

   Príklad #1:
   this.simulation.getRVOSimulator().***setAgentMaxSpeed****(this.id,newSpeed)*;
   Príklad #2:
   this.simulation.getRVOSimulator().***setAgentPosition****(this.id, position)*;

2. **Ako vkladať ďalšiu vrstvu zobrazenia (pre potreby kreslenia bodov plánovania cesty)?**
   Proces pozostáva z niekoľkých krokov.
   a) Do atribútov triedy Simulation treba doplniť deklaráciu pre nový geometrický kontajner (nasledujúci riadok):
   ```
   public GeomVectorField pointEnviroment = new GeomVectorField(WIDTH, HEIGHT);
   ```

   b) Do atribútov triedy SimulationGUI treba doplniť deklaráciu pre novú vrstvu zodpovednú za vykreslenie vrstvy deklarovanej v predchádzajúcom kroku.
   ```
   private GeomVectorFieldPortrayal pointsPortrayal = new
   GeomVectorFieldPortrayal();
   ```

   c) V metóde setupPortrayals() triedy SimulationGUI napojíme zdroj údajov (vrstva pointEnviroment) na zobrazovaciu vrstvu a určíme akým zobrazením bude reprezentovaná (tvar, farba, mierka).
   ```
   pointsPortrayal.setField(simulation.pointEnviroment);
   pointsPortrayal.setPortrayalForAll(new OvalPortrayal2D(Color.GREEN, 2.0) );
   ```

   d) V metóde init() triedy SimulationGUI napojíme vrstvu zodpovednú za vykresľovanie na tzv. objekt display metódou attach() a pomenujeme ju.
   ```
   display.attach(this.pointsPortrayal, "Body");
   ```

e) Vložíme do triedy Simulation na vhodné miesto (napr. metóda start()) dva nasledovné riadky, ktoré vložia na zvolené súradnice vytvorenej vrstvy bod, ktorý predstavuje inštanciu triedy MasonGeometry.

```
GeometryFactory fact = new GeometryFactory();
this.pointEnviroment.addGeometry(new MasonGeometry(fact.createPoint(new
Coordinate(150, 200))));
```

3. **Ako zistiť či je daný bod v mape pri zobrazení budova?**
   Na stretnutí sme testovali použitie metódy isCovered(), ktorá je volaná na geometrickú vrstvu triedy GeomVectorField obsahujúcu prekážky resp. budovy, avšak jej návratové hodnoty (true/false) nezodpovedajú realite a ďalší postup zatiaľ nebol nájdený.

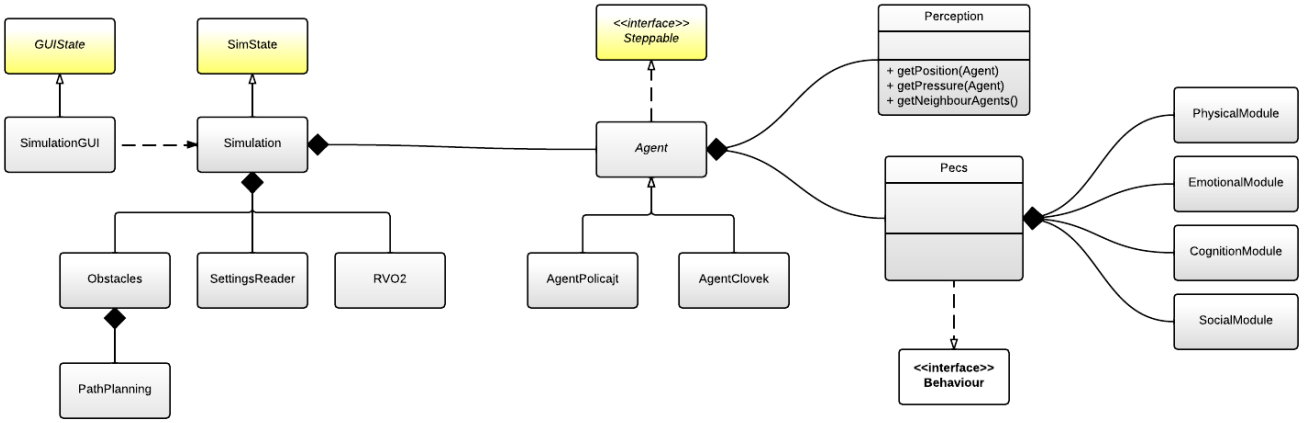
4.2 Návrh PECS a jeho realizácia

4.2.1 Úloha

Návrh architektúry softvérového riešenia simulácie demonštrácie s prihliadnutím a použitie modelu PECS.

4.2.2 Návrh

Navrhnutá architektúra systému je znázornená diagramom tried na obrázku nižšie (Obr. 4:1).



Obr. 4:1 Diagram tried softvérového riešenia simulácie.

4.2.3 Implementácia

Navrhnuté riešenie (Obr. 4:1) sme aj implementovali. Vytvorili sme rozhranie ***IBehaviour***. V súčasnosti neobsahuje žiadne metódy, tie sa budú dopĺňať postupne podľa potreby a požiadaviek na správanie agentov. Zároveň sme vytvorili 4 triedy ***PhysicalModule***, ***EmotionalModule***, ***CognitionModule*** a ***SocialModule***. Tieto moduly budú obsahovať stavové atribúty a metódy, ktoré určujú, počítajú fyzický, emocionálny, rozumový a sociálny stav jedincov.
Následne sme vytvorili triedu ***PECS***. Táto trieda je odvodená od rozhrania ***IBehaviour*** a agreguje všetky 4 moduly správania agenta.
Keďže agent musí vnímať aj okolie, tak musia existovať metódy, ktoré budú okolie agenta vyhodnocovať. Tieto sme umiestnili do triedy ***Perception***.
Abstraktná trieda ***Agent*** agreguje triedy ***PECS*** aj ***Perception***. ***Agent*** je abstraktná trieda, pretože v našej simulácii existujú dva typy agentov, poriadkové zložky a demonštranti. Preto sme odvodili od abstraktnej triedy ***Agent*** triedy ***AgentClovek*** a ***AgentPolicajt***.

4.3 Analýza pôsobenia tlaku

4.3.1 Úloha

Analýza pôsobenia tlaku v dave, návrh a implementácia jednoduchého scenára pre pôsobenia tlaku v dave.
Analyzovanie vzorcov pre výpočet sily pôsobiacej v dave a operácií s vektormi dôležitými pre výpočet sily v dave.
Návrhnutie a implementácia jednoduchého scenára pôsobenia síl v dave.

4.3.2 Analýza

Osoby v dave strácajú sebakontrolu a ich pohyb je ovplyvnený davom. Pri hustote davu 7 osôb/meter^2 sa z davu stáva jedna tečúca masa [42]. Tlak davu na človeka môže byť tak veľký, že zťažuje dýchanie. Slabší jedinci v dave môžu od teploty okolitých tiel a zťaženého dýchania odpadnúť. Prístup k jedincom, ktorý odpadli alebo spadli na zem je problematický. Ukázalo sa, že prakticky všetky úmrtia v dave boli zapríčinené nedostatkok kyslíka, udusením jedinca. To vzniká pri tlaku davu na človeka, ktorému je ťažko čeliť. Tlaky davu spôsobujú udusene jedincov ešte rýchlejšie, ak sa jedinci snažia pretlačiť davom na voľnejšie miesto. Experimenty ukazujú, že sila spôsobená nakláňaním a tlačením je zrovnateľná so silou, ktorou pôsobí 30% až 75% hmotnosti jedinca. Ak ľudia panikária, tak sa táto sila ešte zvyšuje. Podľa experimentov sa dokázalo iba 5 osôb vyvinúť silu až 3430N ak boli v panike [42]. Sila, ktorú dokáže vyvinúť dav je až 4500 $\frac{N}{m^2}$, preto je dôležité neopomínať tento faktor pri simulovaní demonštrácie.
Na agentov v dave pôsobia rôzne druhy síl. Keď agent prechádza pomedzi domy, tak tie na neho pôsobia odpudivou silou, čo znamená, že nemôže len tak prejsť cez stenu domu, a ani to nechce urobiť. Ďalšia sila, ktorá môže na agenta pôsobiť je sila, ktorou na neho pôsobí iný agent, napríklad pri tesnom obchádzaní alebo v tlačenici. Je nutné

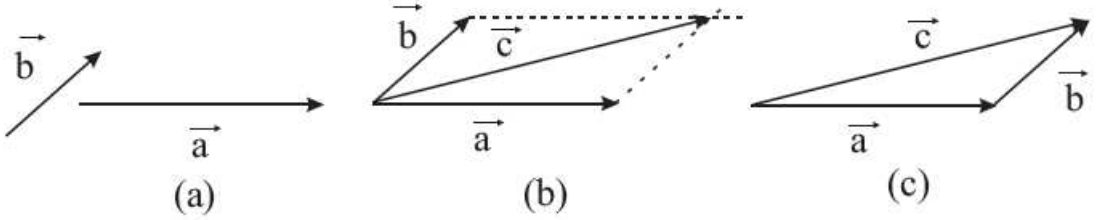
poznať tieto sily aj z výpočtového hľadiska aby sme ich mohli implementovať. Preto nižšie uvádzam najzaujímavejšie vzorce pôsobenia síl a pre úplnosť aj vektorové operácie.

4.3.2.1 Operácie s vektormi

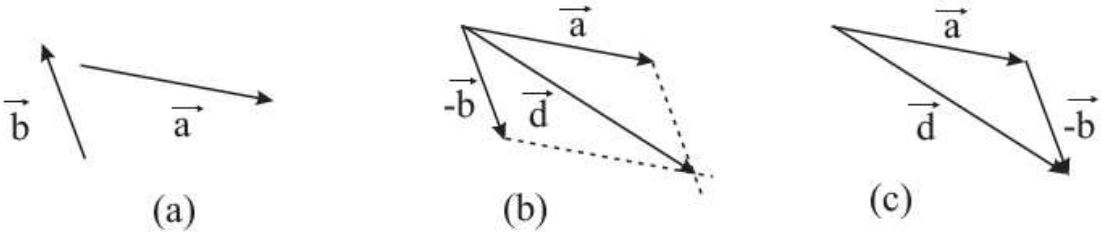
Sčítanie a odčítanie vektorov

Na obrázku (Obr. 4:2) je graficky zobrazený spôsob sčítavania dvoch vektorov **c = a + b** a na obrázku (Obr. 4:3) je odčítavanie dvoch vektorov **c** = **a** – **b**.

a,b,c – vektory



Obr. 4:2 Sčítanie dvoch vektorov (prevzaté z **[43]**).



Obr. 4:3 Odčítanie dvoch vektorov(prevzaté z **[43]**).

Násobenie a delenie vektorov

Násobenie vektora s kladným číslom – skalárny násobok vektora **c** = **a**n. Po vynásobení vektora skalárom dostaneme vektor, ktorý má rovnaký smer a jeho veľkosť je n– krát väčšia.

Veľkosť vektora

i,j – jednotkový vektor

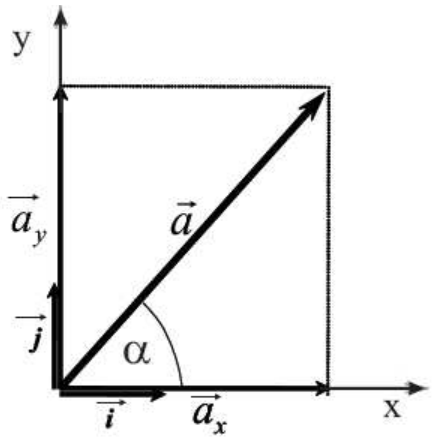
α – uhol medzi osou x a vektorom

$|\mathbf{a}| = \sqrt{a_x^2 + a_y^2}$

$\boldsymbol{a_x} = |\mathbf{a}| \cos \alpha \, \mathbf{i}$

$\boldsymbol{a_y} = |\mathbf{a}| \cos \alpha \, \mathbf{j}$

Na obrázku (Obr. 4:4) je grafická reprezentácia výpočtu veľkosti vektora.



Obr. 4:4 Výpočet veľkosti vektora (prevzaté z **[43]**).

Skalárny súčin dvoch vektorov

a . **b** = |**a**| |**b**| cos α

Vektorový súčin

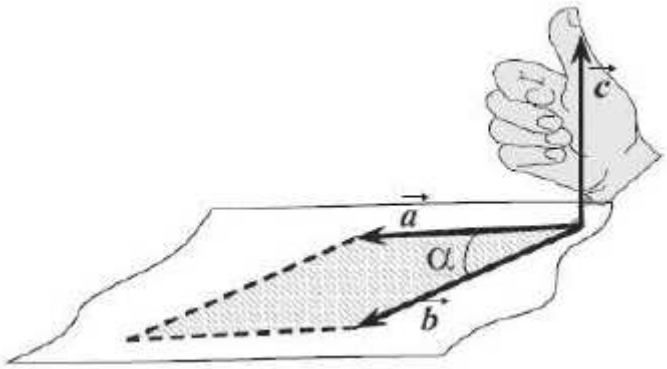
v – jednotkový vektor v smere vektora **c**

c = a x **b**

c = |**a**| |**b**| sin α **v**

c – je vektor kolmí na rovinu určnú **a** a **b.**

Obrázok (Obr. 4:5) ukazuje akým spôsobom sa dá zistiť smer vektora **c.**



Obr. 4:5 Pomôcka na určenie smeru vektora (prevzaté z **[43]**).

Skladanie síl pôsobiacich v jednom bode

m – hmotnosť telesa, na ktoré pôsobí sila

a – zrýchlenie, ktorým sa pohyb hmotného bodu za pôsobenia sily vyznačuje

$\boldsymbol{a}_0$ - zrýchlenie bez pôsobenia sily

F = m($\boldsymbol{a} - \boldsymbol{a}_0$)

Pri pôsobení viacerých síl:

$$\boldsymbol{a_n} = \boldsymbol{a_o} + \frac{\boldsymbol{F_1}}{m} + \frac{\boldsymbol{F_2}}{m} + \frac{\boldsymbol{F_3}}{m} + \cdots + \frac{\boldsymbol{F_n}}{m} = \boldsymbol{a_o} + \frac{\sum_{i=1}^{n} \boldsymbol{F_i}}{m}$$

4.3.2.2 Všeobecné vzorce pre výpočet sil pôsobiacich v dave

Vzájomné pôsobenie síl medzi agentmi a vzájomné pôsobenie síl medzi agentom a stenou

N – počet agentov

v_i^0 - rýchlosť, ktorú chce človek dosiahnuť
m_i – hmotnosť agenta
e_i^0 – smer pohybu
v_i - aktuálna rýchlosť agenta
τ_i – čas, za ktorý ide danou rýchlosťou
j – vzdialenosť od ostatných agentov ovplyvňujúca rýchlosť
W – vzdialenosť od stien ovplyvňujúca rýchlosť
f_{ij} - vzájomné pôsobenie síl dvoch agentov
f_{iW} - vzájomné pôsobenie síl agenta a steny
t – čas, za ktorý sa rýchlosť zmení
$r_i(t)$ - zmena pozície za čas
$v_i(t) = \frac{dr_i}{dt}$
Vzorec pre vzájomné pôsobenie síl medzi agentmi a vzájomné pôsobenie síl medzi agentom a stenou [44]:

$$m_i \frac{dv_i}{dt} = m_i \frac{v_i^0(t)e_i^0(t) - v_i(t)}{\tau_i} + \sum_{j(\neq i)} f_{ij} + \sum_W f_{iW}$$

Psychologická tendencia vzájomného vyhýbania sa dvoch agentov
A_i, B_i – konštanty
d_{ij} - vzdialenosť medzi agentami
$d_{ij} = |r_i - r_j|$
n_{ij} – normálový vektor, ukazujúci od jedného agenta k druhému
$n_{ij} = (n_{ij}^1, n_{ij}^2) = \frac{r_i - r_j}{d_{ij}}$
Ak $d_{ij} < r_{ij} = (r_i + r_j)$, tak sa agenti dotýkajú. Vzorec:

$$A_i \exp\left[\frac{(r_{ij} - d_{ij})}{B_i}\right] n_{ij}$$

K,k – konštanty

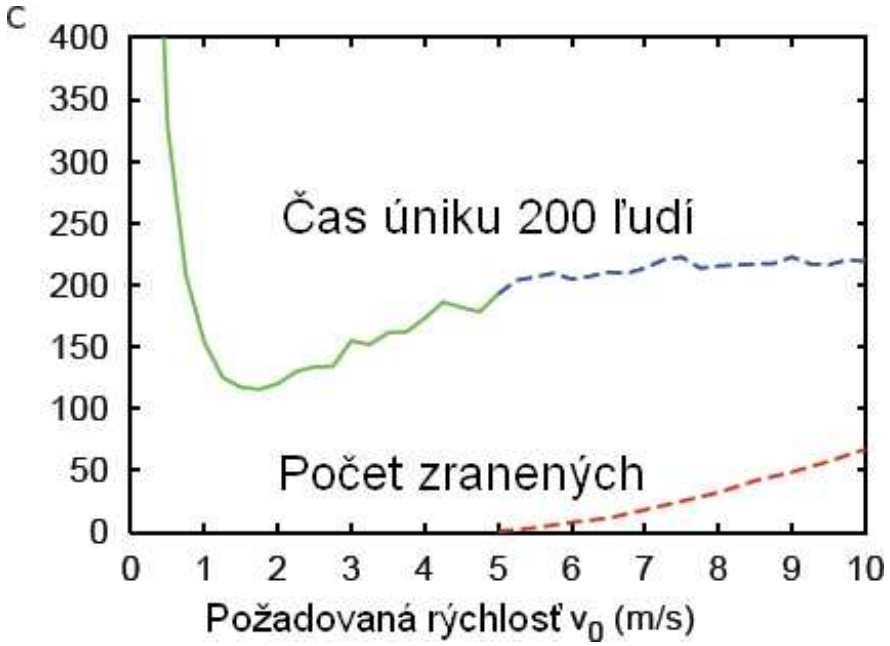
Silou tela sa myslí odbudivá sila tela pri jeho stlačení [44]: $k(r_{ij} - d_{ij})\, n_{ij}$
Sila pri šmykovom trení, keď agent i sa približuje k agentovi [44]: $K(r_{ij} - d_{ij})\Delta v_{ij}^t t_{ij}$
Po pridaní odpudivej sily tela a trecej sily sa vzorec pre vzájomné pôsobenie síl medzi agentmi zmení tak ako je ukazuje vzorec [44]:

$$f_{ij} = A_i \exp\left[\frac{(r_{ij} - d_{ij})}{B_i}\right] + kg(r_{ij} - d_{ij})n_{ij} + kg(r_{ij} - d_{ij})\Delta v_{ij}^t t_{ij}$$

a vzorec pre vzájomné pôsobenie síl človeka a steny zobrazuje je [44]:

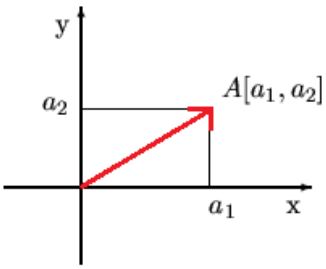
$$f_{iW} = A_i \exp\left[\frac{(r_{ij} - d_{iW})}{B_i}\right] + kg(r_{ij} - d_{iW})n_{ij} + kg(r_{ij} - d_{iW})(\mathrm{v_i}.\, t_{iW})t_{iW}$$

Ak sú ľudia v pokoji, tak ich požadovaná rýchlosť býva približne 0,6 m/s. Keď sú ich emócie pod normálnou hranicou, tak sa ich požadovaná rýchlosť zvýši na 1 m/s. Ak sú nervózni, je ich požadovaná rýchlosť približne 1,5 m/s.
Jednou simuláciou pri aplikácii týchto vzorcov sa dokázalo, že zranení ľudia v dave začínajú pribúdať od veľkosti požadovanej rýchlosti približne 5 m/s ako je vidno na nižšie uvedenom obrázku (Obr. 4:6).



Obr. 4:6 Počet zranených v závislosti od požadovanej rýchlosti davu (prevzaté z **[44]**).

Pohyb agenta v simulačnom *dvojrozmernom* prostredí je popísaný pomocou vektorov. Samotný vektor je definovaný pomocou dvoch súradníc: „x“ a „y“, ktoré predstavujú bod v dvojrozmernom priestore: v = [x,y]. Počiatok súradnicovej sústavy reprezentuje nulový vektor: 0 = [0,0].
Vektor má svoj smer a veľkosť (Obr. 4:7).



Obr. 4:7 Vektor A.

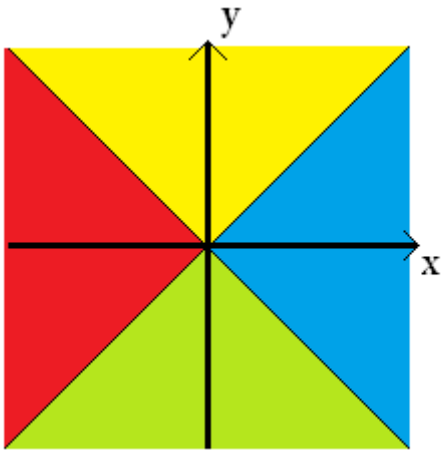
Problém vznikajúci pri simulácii je veľké množstvo agentov, a preto je netriviálne sčítať ich sily a dostať výsledný vektor, pretože ak by sa dav pohyboval v jednom smere, a my by sme začali sčítavať sily v smere opačnom, nevznikalo by zreťazenie a nahromadenie sily celého davu, a dostali by sme skreslené výsledky. Preto je potrebné začať sčítavať sily od tzv. vonkajších agentov (tí, ktorí sú na konci radu agentov, ktorí sa pohybujú určitým smerom).

4.3.3 Zhrnutie analýzy

Simulovanie šírenia tlaku v dave nie je triviálna úloha. Smer, odkiaľ je vhodné začať vektory sčítavať je počiatok centrálneho vektora, ktorý vypočítame súčtom všetkých vektorov v dave ľudí. Výsledný tlak, ktorý vznikne na konci davu, môže dosahovať zaujímavé hodnoty, ktoré pri prekročení stanovených hodnôt môžu spôsobiť zranenie agenta alebo roztlačenie policajných zložiek.

4.3.4 Návrh

Navrhujeme použiť pre odhad smeru Centrálny vektor „C“, ktorý bude aproximovať smer davu na jeden zo štyroch základných smerov (sever,západ,juh,východ), a na základe jeho smeru sa začnú sčítavať jednotlivé vektory postupne.



Obr. 4:8 Rozdelenie priestoru.

Na obrázku (Obr. 4:8) je uvedené a farebne rozlíšené rozdelenie priestoru na základe smeru výsledného vektora. Jeho veľkosť teraz nie je dôležitá, berieme do úvahy jeho smer.
Štyri farebné oblasti označíme rímskymi číslicami od žltej v protismere hodinových ručičiek, a uvedieme ich rozsahy:

I. [46°,135°]
II. [136°,225°]
III. [226°,315°]
IV. [316°,0°] u [0°,45°]

Algoritmus výpočtu je teda nasledovný:

1. Sčítame všetky vektory agentov, ktorí sú v zhustenej oblasti a je ich aspoň 15.
2. Po sčítaní dostaneme výsledný centrálny vektor, ktorý dosadíme do nasledovného vzorca:

$$\cos\alpha = \frac{y}{\sqrt{|x^2| + |y^2|}}$$

3. Výsledkom je smer centrálneho vektora v stupňoch, pričom na základe smeru zaradíme výsledný centrálny vektor do jednej zo štyroch oblastí.
4. Sčítavanie síl agentov v dave prebieha od začiatku centrálneho vektora.

Scenár sme navrhli tak, že policajti sa budú nachádzať v rade, nebudú nebudú nevidieť svojich susedov a teda zabránia ostatným priechod skrz nich. Hromada demonštrantov bude vyvíjať tlak na policajtov, pretože budú chcieť ísť tým smerom ako stoja policajti. Agenti v strede tak budú pociťovať účinky dvoch protichodných síl najviac.
Počas simulácie môže vzniknúť viacero skupín so susediacimi agentmi. Tieto skupiny je potrebné rozlišovať. Navrhujeme pridať agentovi premennú objektu, ktorá bude reprezentovať *ID* skupiny, do ktorej patrí. Každý krok sa bude prechádzať hashset so súradnicami všetkých agentov a na základe stanovenej vzdialenosti, do ktorej sa bude mať hľadať sa bude agentom nastavovať *ID* skupiny. Vždy sa vezme postupne jeden, zistí sa, či má do danej vzdialenosti viac ako *X* susedov. Ak áno, nastaví sa im *ID* skupiny na rovnaké číslo. Bude sa taktiež kontrolovať či daný agent nemá už nastavenú skupinu. Ak áno, preskočí sa. Takto sa nastaví skupina každému agentovi, ktorý má viac ako *X* susedov. Počet susedov *10* je experimentálne stanovená hodnota, ktorá sa môže meniť.
Scenár som navrhla tak, že policajti sa budú nachádzať v rade, nebudú nevidieť svojich susedov a teda zabránia ostatným priechod skrz nich. Hromada demonštrantov bude vyvíjať tlak na policajtov, pretože budú chcieť ísť tým smerom ako stoja policajti. Agenti v strede tak budú pociťovať účinky dvoch protichodných síl najviac.

4.3.5 Implementácia

Vytvorila sa metóda z názvom ***SetAgentGroupId(int MinimalNeighbourCount)***, ktorá berie ako parameter najmenší počet susedov, ktorých musí agent mať aby mohol vytvoriť jadro skupiny. Metóda nevracia nič, len nastavuje premennú *groupId*. Toto nastavovanie sa robí len pre inštancie agentov triedy ***AgentClovek***. Metóda je schopná zatiaľ hľadať agentov do vzdialenosti 1.
Pre každého agenta sa zisťuje koľko má susedov a na základe toho sú potom agenti usporadúvaný do prioritného frontu.
Prioritný front sa prechádza a pre každého vybratého agenta sa nastaví *groupId* a taká istá hodnota *groupId* sa nastaví aj všetkým jeho susedom. Pri prechádzaní prioritného frontu sa kontroluje, či už má agent *groupId* nastavené, teda či sa nerovná nule. Ak už má *groupId* nastavené, tak sa pre tohto agenta susedia neprehľadávajú.
Tento cyklus sa vykonáva dovtedy, do kým nenarazí na agenta s nižším počtom susedov ako je ***MinimalNeighbourCount***.

4.4 Plánovanie trasy agentov

4.4.1 Úloha

Počas simulácie môže nastať situácia, že agenti uviaznu v budove (Obr. 4:9). V takom prípade nedokážu dosiahnuť zadaný cieľ, čo neodpovedá reálnemu správaniu ľudí. Knižnica pre vyhýbanie sa prekážkam a agentov navzájom RVO, žiaľ nerieši plánovanie cesty agentov z počiatočného bodu do cieľového.

Obr. 4:9 Uviaznutie agentov v budove.

Úlohou je analyzovať možnosti plánovania trasy agentov do cieľa. Analyzované riešenie navrhnúť a vykonať jednoduchú implementáciu.

4.4.2 Analýza

Existuje viacero algoritmov plánovania trasy agentov. Cieľom je najmä nízka výpočtová zložitosť, aby plánovanie trasy nespomaľovalo príliš celú simuláciu. Dosiahnuť chceme správanie sa agentov, čo najpodobnejšie správaniu ľudí.
Všetky algoritmy hľadania cesty však hľadajú iba najkratšiu cestu. Reálne však ľudia v neznámom prostredí nevedia vždy zvoliť práve najkratšiu cestu. Ideálne by teda bolo nenavigovať všetkých agentov jednou najkratšou cestou, ale viacerými cestami.

4.4.2.1 Dijkstrov algoritmus

Najznámejší algoritmus hľadania najkratšej cesty je nepochybne Dijkstrov algoritmus. Ten využíva relaxáciu, čiže metódu na opakované znižovanie horného ohraničenia dĺžky najkratšej cesty pre každý vrchol.
Ak by sme v každom vrchole si nepamätali iba cenu najkratšej cesty, za ktorú sa dá do vrcholu dostať, ale aj cenu druhej, resp. tretej najkratšej cesty a vrcholy odkiaľ sme prišli, dokázali by sme pomocou takto modifikovaného Dijkstrovho algoritmu nájsť viacero najkratších ciest. Následne by si podľa určitej pravdepodobnosti agenti vybrali konkrétnu cestu.

*4.4.2.2 Hľadanie A**

Dijkstrov algoritmus pri efektívnej implementácii pomocou haldy síce zbehne v čase O (H * log V), avšak nevyužíva žiadnu dodatočnú informáciu, vďaka ktorej by sa rýchlejšie približoval k cieľu.
Najpopulárnejší algoritmus umelej inteligencie A* využíva heuristickú informáciu, ktorá odhaduje cenu cesty do cieľa. Na rozdiel od lačného hľadania, A* uvažuje aj už prejdenú cestu od začiatku, pretože v konečnom dôsledku nás bude zaujímať cena celej cesty. Vyhodnocovacia funkcia vyzerá nasledovne:

$$f(u) = g(u) + h(u)$$

Hľadanie A* rozvíja uzly v poradí zvyšujúcej sa hodnoty vyhodnocovacej funkcie. Požiadavka na heuristickú funkciu je, aby bola prípustná (nenahodnocovala cenu optimálnej cesty do cieľa). Hľadanie A* je úplné a optimálne. Vizualizácia algoritmu A* je uvedená na (Obr. 4:10).

8+3	7+4	6+3	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2		5+6	4+7	3+8						5
6+1			3	2+9	1+10	0+11	1	2		4
7+0	6+1									5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6

Obr. 4:10 Hľadanie A*.

Vždy, keď A* rozvinie uzol, tak už našiel optimálnu cestu do stavu reprezentovaného týmto uzlom. Pokiaľ však chceme od A* nie len najkratšiu cestu, ale viacero ciest, je nutné hľadanie modifikovať.

Jednoduchá myšlienka spočíva vo vygenerovaní niekoľkých checkpointov (Obr. 4:11). Hľadať cestu nebudeme zo štartu do cieľa, ale zo štartu do jedného z checkpointov a z checkpointu do cieľa. Tým zabezpečíme, že cesta bude viesť cez checkpoint a koľko checkpointov vygenerujeme, toľko rôznych ciest získame.

Obr. 4:11 Vygenerované checkpointy.

4.4.2.3 Floydov-Warshallov algoritmus

Predošlé algoritmy hľadali cestu z jedného počiatočného uzla. Floydov-Warshallov algoritmus hľadá najkratšiu cestu zo všetkých vrcholov do všetkých v čase $O(V^3)$. Myšlienka je predpočítať si cesty z každého bodu do každého pre danú mapu dopredu, aby nebola simulácia spomalená hľadaním cesty počas behu simulácie.

Pre každé políčko na mape by sme si museli zapamätať cesty do všetkých ostatných políčok. To, že by takýto výpočet trval dlhšie by nemuselo vadiť, keďže by sa vykonal iba jedenkrát pre danú mapu a simuláciu by nijakým spôsobom nespomalil.

Horšie by to bolo s pamäťovou zložitosťou. Nie len, že by si bolo treba pamätať cestu zo všetkých políčok do všetkých, ale keďže nechceme len jednu cestu, ale viac ciest, bolo by potrebné si pamätať všetky cesty zo všetkých políčok do všetkých. Jediným riešením v takomto prípade by bolo rozdeliť si mapu na mriežku agregovaných bodov ako ďalšiu vrstvu nad mapou.

4.4.3 Návrh

V tejto fáze riešenia projektu sme sa rozhodli navrhnúť a implementovať hľadanie A*. Algoritmus A* sa spustí iba v prípade, že agent uviazne v budove. Tým, že sa hľadá cesta iba pre uviaznutých agentov, je toto riešenie šetrné k procesorovému času.

Zatiaľ neuvažujeme nad agregáciou pixelov do väčších jednotiek. Hľadanie sa teda vykonáva nad najmenšími možnými bodmi mapy – pixelmi. Pokiaľ by sa pixely agregovali do väčších oblastí, dá sa očakávať výrazné urýchlenie hľadania.

4.4.4 Implementácia

Počas implementácie nastali určité problémy. Zistili sme, že nedokážeme zistiť z programu takú elementárnu informáciu, ako napríklad či daný bod na mape je voľný alebo je na ňom časť budovy. Bez takejto informácie sa plánovanie trasy robí veľmi ťažko, dokonca by som až povedal, že je to nemožné.

Za účelom implementovania plánovania cesty bol vytvorený *package* s názvom ***simTeam.Planning*** a trieda ***PathPlanning***.

Podarilo sa implementovať hľadanie A* s použitím funkcie ***isEmptyArea(int x, int y)***, ktorá by v ideálnom prípade mala vrátiť ***true***, ak bod [x][y] je prázdny a ***false***, ak je v ňom budova. Vzhľadom na to, že túto informáciu zatiaľ nevieme z programu zistiť, funkcia v tejto verzii vracia pre každý bod ***true***.

To má za následok, že implementovaný A* nájde vždy priamu cestu od pozície agenta do cieľa. Každopádne je však implementácia hľadania A* pripravená na jednoduchú modifikáciu funkcie ***isEmptyArea***, keď budeme vedieť zistiť, či bod je prázdny alebo nie. Korektným implementovaním funkcie ***isEmptyArea*** bude hľadanie A* fungovať správne. Na záver sa nájdená cesta vykreslí ako množina bodov do mapy. Vykreslená trasa do mapy je však posunutá kvôli pretrvávajúcim problémom so súradnou sústavou mapy.

Ak bude problém so zisťovaním, či bod je budova alebo prázdny pretrvávať, je možné pokúsiť sa o implementáciu stochastického algoritmu navigácie agentov do cieľa. Princíp by bol veľmi jednoduchý. Ak by agent uviazol, nechali by sme ho pohybovať sa

náhodnými smermi a po každom kroku by sme kontrolovali, či vzdialenosť do cieľa sa znížila, ako keď bol v uviaznutom stave. Pokiaľ áno, tak spustíme opäť RVO, ktoré sa ho pokúsi dostať do cieľa.
Takto by agenti nikdy neostali trčať v uviaznutom stave a stále by sa pohybovali. Vďaka náhodnosti sa dá predpokladať, že by sa skôr či neskôr dostali do cieľa. Aby to však netrvalo príliš dlho, dalo by sa s náhodou trochu štatisticky pohrať. Napríklad od stavu uviaznutia by bola väčšia pravdepodobnosť pohybu opačným smerom a postupom času by sa táto pravdepodobnosť znižovala a zvyšovala by sa pravdepodobnosť pohybu smerom do cieľa.

5 Štvrtý šprint

V nižšie uvedenej tabuľke (Tabuľka 5.1) je uvedený zoznam úloh, ktoré sa počas štvrtého šprintu, ktorý sme interne pomenovali Bostonské pitie čaju, vyriešili a osoby, ktoré sa danými úlohami primárne zaoberali. Zoznam úloh a k nim prislúchajúcich riešiteľov.

Tabuľka 5.1 Zoznam úloh a k nim prislúchajúcich riešiteľov.

Názov úlohy	Zodpovedná osoba
Logovanie stavu agentov	Filip Pakan
Plánovanie trasy agentov	Filip Pakan
Plánovanie trasy agentov	Adrián Kollár
Konfiguračný súbor na vstupné parametre	Miroslav Ort
Analýza pôsobenia tlaku + implementácia	Jana Branišová
Analýza pôsobenia tlaku + implementácia	Michal Ošvát
Analýza pôsobenia tlaku + implementácia	Miroslav Ort

5.1 Logovanie stavu agentov

5.1.1 Úloha

Zaznamenávať stav všetkých agentov počas krokov simulácie do súboru, ideálne v XML formáte. Ide najmä o stavové premenné ako napríklad aktuálna pozícia agenta, cieľ agenta, úroveň strachu a podobne. Cieľom je mať možnosť pri pohľade do XML súboru zistiť, ako sa stav agenta menil v čase, či už na ladiace účely alebo na overenie psychologického modelu.
Taktiež by sa mal dať nastaviť parameter, ktorý bude určovať každý koľký krok sa bude logovať. Napríklad pre hodnotu 5 by sa logoval stav agentov iba každý piaty krok simulácie. Očakávame, že zmena stavu agenta medzi dvoma nasledujúcimi krokmi nebude veľká, navyše pre veľké množstvo agentov a veľa krokov simulácie by výsledný XML súbor bol príliš objemný. Preto logovanie iba niektorých krokov simulácie je užitočná funkcia.

5.1.2 Analýza

Existuje viacero Java XML parserov. Analyzoval som *DOM*, *SAX*, *JDOM*, *JAXB* a *Properties*.

5.1.2.1 DOM XML Parser

DOM XML Parser je najjednoduchší Java XML Parser na použitie. Rozparsuje celý XML dokument a uloží ho do pamäti. Pre jednoduchý pohyb v XML dokumente ho modeluje pomocou Objektov.

5.1.2.2 SAX XML Parser

SAX XML Parser pracuje inak ako *DOM* Parser. Nenačíta celý dokument do pamäti a nevytvára objektovú reprezentáciu dokumentu. Namiesto toho využíva callback funkciu na informovanie klientov o štruktúre XML dokumentu.

5.1.2.3 JDOM XML Parser

JDOM poskytuje spôsob na reprezentáciu dokumentu pre jednoduché a efektívne čítanie, manipuláciu a zápis. Je to alternatíva k *DOM* a *SAX* parserom.

5.1.2.4 JAXB

JAXB umožňuje konvertovať Java objekt do XML súboru a naopak konvertovať XML súbor do Java objektu.

5.1.2.5 Properties

Trieda *java.util.Properties* je špeciálnym typom hashovacej tabuľky. Navyše má integrovanú funkcionalitu na konverziu položiek do XML formátu a naopak.

5.1.2.6 Zhrnutie analýzy

Všetky analyzované riešenia poskytujú to, čo aktuálne potrebujem. Pre jednoduchosť použitia a rozšírenosť som zvolil *DOM XML Parser*.

5.1.3 Návrh

Cieľom je navrhnúť flexibilnú štruktúru, ktorá umožní jednoduché pridávanie nových XML elementov na logovanie. Ideálne je vytvoriť novú triedu, ktorá bude obsahovať metódy potrebné pre logovanie stavu agentov.
Klientsky kód by si mal vedieť nastaviť, každý koľký krok simulácie sa bude stav agentov logovať. Toto sa dá dosiahnuť napríklad uvedením v konštruktore logovacej triedy.
Logovanie stavu agentov bude prebiehať počas celej simulácie do hlavnej pamäti a pri ukončení simulácie sa uloží do súboru. Toto riešenie som zvolil kvôli rýchlosti, pretože inak by bolo potrebné so zalogovaním každého agenta súbor otvoriť, zapísať do neho a zavrieť ho. Avšak vieme, že čítanie alebo zápis na pevný disk je veľmi pomalé a prístupová doba sa pohybuje rádovo v desiatkach milisekúnd.

5.1.3.1 Štruktúra XML dokumentu

5.1.3.1.1 Element simulation

Simulation je koreňový element, ktorý obsahuje jeden atribút a tým je dátum simulácie.

5.1.3.1.2 Element step

Element ***step*** predstavuje jeden krok simulácie. Ako atribút obsahuje číslo kroku. V rámci jedného kroku (jedného elementu step) budú uložené stavy všetkých agentov.

5.1.3.1.3 Element agent

Element ***agent*** reprezentuje jedného agenta simulácie. Ako atribút obsahuje identifikátor agenta. Vzhľadom na to, že simulácia sa vykonáva paralelne, nemusia byť agenti zapísaní v poradí, v akom boli vytváraní.

5.1.3.1.4 Element position

Position obsahuje aktuálnu pozíciu agenta. Má dva atribúty ***x*** a ***y***, ktoré vyjadrujú jeho pozíciu na danej osi.

5.1.3.1.5 Element target

Element ***target*** reprezentuje súradnice cieľa agenta. Má dva atribúty x a y, ktoré vyjadrujú pozíciu cieľa.

5.1.3.1.6 Element dummyFear

Element ***dummyFear*** vyjadruje úroveň strachu agenta. Na základe tejto hodnoty sa agenti prefarbujú vo vizualizačnej vrstve.

5.1.4 Implementácia

Za účelom implementovania tejto úlohy som vytvoril triedu ***Logger***, ktorá zabezpečuje logovanie agentov. Trieda agent agreguje statický objekt triedy ***Logger***, pretože všetci agenti sa budú logovať do jedného súboru, a teda musia zdieľať spoločný ***logger***.
Samotné logovanie sa vykonáva v metóde ***step()*** triedy Agent zavolaním funkcie: ***logger.logAgentState(this, totalSteps)***.
Agent pošle metóde ***logAgentState*** referenciu na samého seba, aby objekt ***logger*** mohol vytiahnuť z agenta všetky stavové premenné a zaznamenať ich do súboru. Premenná ***totalSteps*** obsahuje celkový počet krokov simulácie, ktoré od začiatku vykonal agent ***this***.
Pri ukončení simulácie sa zaznamenané dáta v pamäti uložia do súboru pomocou volania:
Logger.writeToFile().

5.1.5 Testovanie

Vytvorené riešenie som dôkladne otestoval. Aj pri dlhšej dobe simulovania (1000 krokov), neprekročila veľkosť logovacieho súboru 5 MB. Otestoval som aj logovanie každé 2 resp. každých 7 krokov simulácie. Všetko funguje korektne.

5.1.5.1 Ukážka časti XML logu

Ukážka XML výstupu je uvedená na obrázku (Obr. 5:1).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<simulation date="Sat Dec 01 21:27:21 CET 2012">
	<step number="0">
		<agent id="1">
			<position x="1015.0" y="817.0"/>
			<target x="200.0" y="0.0"/>
			<dummyFear>1.4547996812683706</dummyFear>
		</agent>
		<agent id="0">
			<position x="1008.0" y="813.0"/>
			<target x="200.0" y="0.0"/>
			<dummyFear>3.146898391459132</dummyFear>
		</agent>
	</step>
	<step number="1">
		<agent id="0">
			<position x="1011.2894013976029" y="803.296402369055"/>
			<target x="200.0" y="0.0"/>
			<dummyFear>8.430919710231535</dummyFear>
		</agent>
		<agent id="1">
			<position x="1007.1726389565137" y="812.7804703016902"/>
			<target x="200.0" y="0.0"/>
			<dummyFear>3.146898391459132</dummyFear>
		</agent>
	</step>
</simulation>
```

Obr. 5:1 Ukážka časti XML logu.

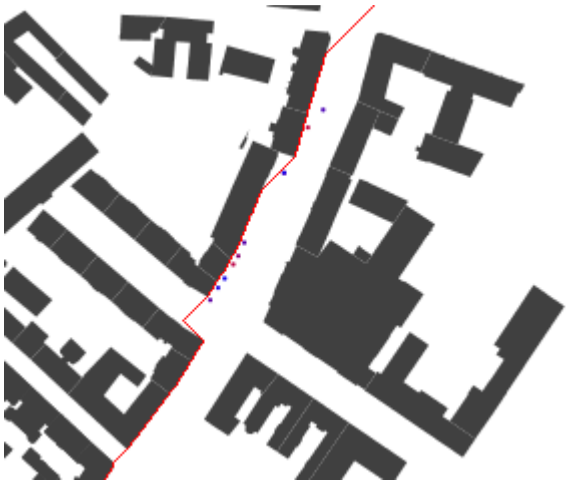
5.2 Plánovanie trasy agentov

5.2.1 Úloha

Implementovanie možnosti plánovania trasy agentov do cieľa. Vytvorenie časovo efektívneho algoritmu.

5.2.2 Implementácia

Pri implementácii sa vyskytol problém so zisťovaním, či bod je budova alebo je prázdny. Na vyriešenie problému zisťovania voľných a obsadených bodov mapy sme vyskúšali viacero postupov. Z vyskúšaných postupov bolo veľa postupov nefunkčných alebo nevhodných pre naše účely. Pre účely zisťovania voľných bodov mapy sme vytvorili novú geometrickú vrstvu ***pathPointEnviroment***. Do tejto vrstvy sú vkladané jednotlivé body, o ktorých potrebujeme zistiť, či sú voľné alebo tvoria prekážku na mape, teda sú časťou budovy. Po vložení bodu do tejto vrstvy sa v cykle prechádza kolekcia geometrických objektov budov, získaných z geometrickej vrstvy prekážok ***obstaclesEnviroment***. V jednom z vyskúšaných postupov sme v tomto cykle zisťovali, či daný bod je súčasťou budovy. Ak nebol súčasťou žiadnej budovy v kolekcii geometrických objektov, považovali sme tento bod za voľný. Pri danom postupe však trasy algoritmu A* takmer kopírovali pôdorysy budov, ako je znázornené na obrázku (Obr. 5:2). Trasa je vyznačená červenou farbou.

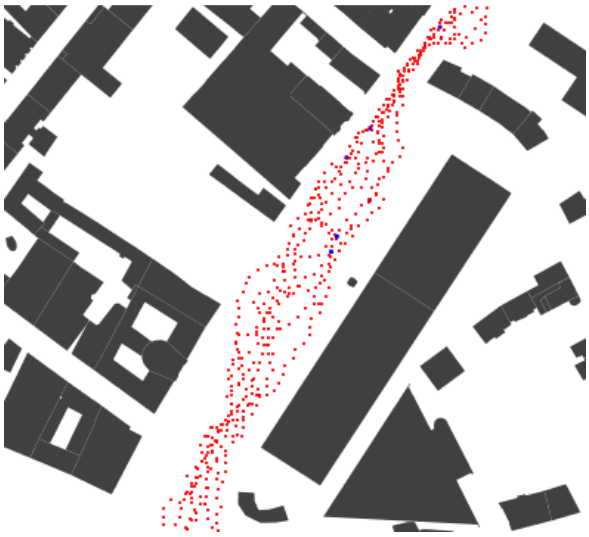


Obr. 5:2 Trasa A* verzia 1.

Toto riešenie sa však nepodobalo na realitu a bolo potrebné ho ďalej vylepšovať. Jedno z vylepšení bolo zisťovanie vzdialenosti bodu od budovy. Vzdialenosť od budovy je nastavená parametrom ***distanceFromBuildings***. V prípade, že pri nastavenej vzdialenosti od budovy nie je nájdená žiadna trasa do cieľa, je tento parameter nastavený na nižšiu hodnotu a hľadanie trasy je spustené opakovane. To umožní nájsť trasu aj v úzkych uličkách a nájdená trasa nebude kopírovať pôdorysy budov ako to bolo v prvom prípade. Výsledná trasa s použitím vzdialenosti od budov je zobrazená na obrázku (Obr. 5:3).

Obr. 5:3 Trasa A* verzia 2.

Viditeľným nedostatkom je veľká podobnosť, priam až identickosť nájdených trás. Tento nedostatok sme vyriešili pridaním istého prvku náhodnosti. Výsledné trasy agentov sú znázornené na obrázku (Obr. 5:4) ako červené body.



Obr. 5:4 Trasa A* verzia 3.

Uvedené riešenie však bolo veľmi časovo náročné, najmä kvôli výpočtovej zložitosti zisťovania prázdnych bodov mapy. Z toho dôvodu bolo do riešenia pridané predpočítavanie prázdnych bodov aj s možnosťou ukladania a načítavania prázdnych bodov máp do súboru.

5.3 Konfiguračný súbor na vstupné parametre

5.3.1 Úloha

Použitie konfiguračného súboru na načítavanie všetkých parametrov použitých v simulácii.

5.3.2 Implementácia

Simuláciu, na ktorej pracujeme, je parametrizovaná simulácia. To znamená, že simulácia potrebuje vstupné parametre, ktoré bude simulácia zohľadňovať a riadiť sa ich hodnotami. Parametre simulácie sme dosiaľ definovali v zdrojovom kóde simulácie tam, kde sme parameter potrebovali. Tento spôsob definície parametrov je nepraktický, najmä z dôvodu, že pri akejkoľvek zmene ktoréhokoľvek parametra sa musel zdrojový kód simulácie zostaviť odznova. Vytvorili sme preto súbor, v ktorom sú parametre simulácie uchovávané a dajú sa vidieť a meniť nezávisle od zdrojového kódu programu.
Na implementáciu som použil triedu *java.util.Properties*. Trieda reprezentuje množinu paramterov, ktoré sa môžu čítať a ukladať počas behu programu.
Parametre sú uchovávané v súbore ***Config.properties***. Odtiaľ ich novovytvorená trieda ***LoadParameters*** načíta pri vytvorení každej inštancie triedy. Trieda obsahuje jedinú metódu ***GetParameter(String parameter)***. Metóda má ako argument reťazec s názvom parametru, ktorý požadujeme načítať. Metóda vracia načítaný parameter ako objekt typu *String* alebo *null*. Ak požadovaný parameter nie je typu String, je potrebné ho pred použitím správne pretypovať.
Ukážka použitia:

```
LoadParameters params = new LoadParameters();
neighborDist = Double.parseDouble(params.GetParameter("neighborDist"));
```

Parametre simulácie nepotrebujem do súboru ukladať, preto má trieda ***LoadParameters*** iba metóda na získavanie parametrov. Získanie parametru je definované v samostatnej tried, aby sme boli schopní získať parameter z konfiguračného súboru v ktorejkoľvek triede programu importovaním triedy ***LoadParameters***.
Parametre, ktoré simulácia potrebuje k jej správnemu spusteniu, sa načítavajú v metóde ***loadConfiguration()*** v triede ***Simulation***. Tento prístup nám zároveň umožnil zmeniť parametre pri každom spustení simulácie bez nutnosti reštartovať celý program.

5.4 Analýza pôsobenia tlaku a implementácia

5.4.1 Úloha

Analyzovať, navrhnúť a implementovať mechanizmus pre pôsobenie tlaku v dave.

5.4.1 Analýza

Pôsobenie tlaku v dave vzniká na miestach, kde je vyššia hustota ľudí. Ak takáto situácia nastane, agentom v simulácií sa priradí atribút skupiny(spoločné identifikačné číslo, ďalej len „ID grupy“), na základe ktorého vieme, že agenti s rovnakým ID grupy sú zhromaždení pri sebe

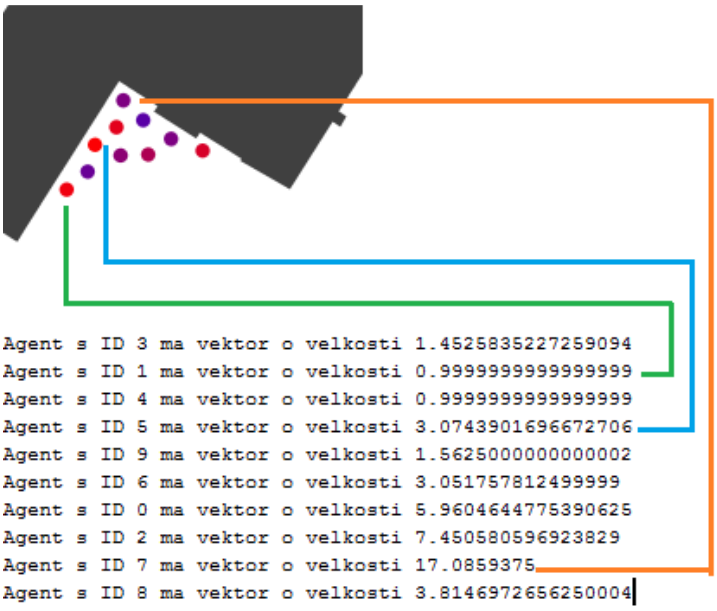
5.4.2 Návrh a implementácia

Návrh algoritmu pre pôsobenie tlaku možno zhrnúť do nasledovných krokov:

- Priradenie ID grupy všetkým agentom
- Vypočítanie centrálneho vektora pre každú skupinu
- Výpočet a prenos síl pomocou vektorov rýchlost v smere centrálneho vektora
- Odovzdanie energie (vektora rýchlosti) do nasledovného riadku/stĺpca

5.4.3 Testovanie

Testovanie prebiehalo na rozličných situáciách. Počiatočný prenos sily v jednom smere bol testovaný, pokiaľ agenti uviazli na jednom mieste ohraničení budovou. Pri testovaní bola deaktivovaná funkcia plánovania trasy pre umožnenie lepšieho otestovania daného algoritmu.
Výsledky testovania sú znázornené na obrázku (Obr. 5:5).



Obr. 5:5 Výsledok testovania.

6 Piaty šprint

V nižšie uvedenej tabuľke (Tabuľka 6.1) e uvedený zoznam úloh, ktoré sa počas piateho šprintu, ktorý sme interne pomenovali Dobytie Bastilly, vyriešili a osoby, ktoré sa danými úlohami primárne zaoberali. Zoznam úloh a k nim prislúchajúcich riešiteľov.

Tabuľka 6.1 Zoznam úloh a k nim prislúchajúcich riešiteľov.

Názov úlohy	Zodpovedná osoba
Plánovanie trasy agentov	Adrián Kollár
Nájdenie videa demonštrácie s vhodným scenárom	Miroslav Ort
Refaktoring zdrojového kódu	Michal Kyžňanský
Refaktoring zdrojového kódu	Miroslav Ort

6.1 Plánovanie trasy agentov

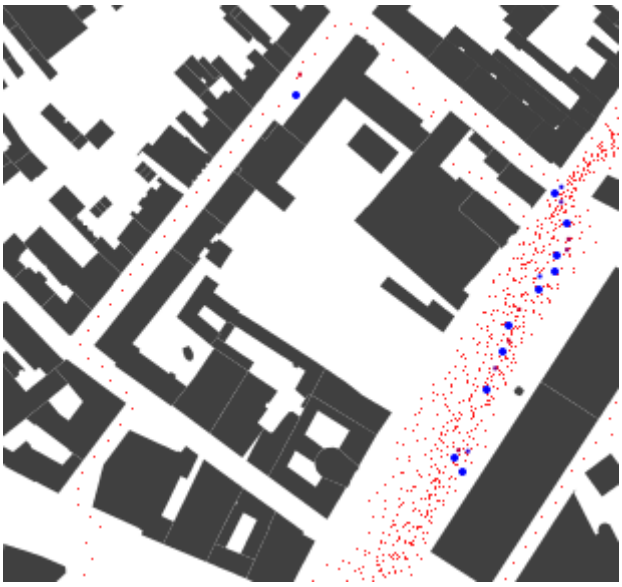
6.1.1 Úloha

Implementácia efektívneho riešenia plánovania trasy agentov. Odstránenie chyby pri spustení plánovania trasy.

6.1.2 Implementácia

Za účelom zefektívnenia výpočtu plánovanej trasy sme do riešenia plánovania trasy agenta implementovali predpočítavanie dvojice máp bodov. Jedna mapa reprezentuje voľné a obsadené body mapy. Po voľných bodoch mapy sa môžu agenti pohybovať, obsadené body mapy tvoria budovy. Druhá mapa obsahuje pre každý bod mapy vzdialenosť daného bodu od najbližšej budovy. Táto mapa je nevyhnutná pre využitie parametra ***distanceFromBuildings***, ktorý bol do riešenia implementovaný v predchádzajúcej verzii za účelom zreálnenia nájdených ciest. Obidve mapy sú súčasťou triedy ***ObstacleDistanceMap***. Trieda obsahuje metódu ***getObstacleDistanceMap()***, ktorá vracia predpočítanú mapu vzdialeností prázdnych bodov od najbližšej budovy.
Prepočítavanie daných máp je časovo náročné, preto po vypočítaní bodov pre zvolené mapu sú dané mapy prázdnych bodov aj vzdialeností ukladané do súboru. Pri následnom spustení simulácie sú následne dáta obidvoch máp načítané zo súboru. Iba v prípade, že súbor pre konkrétnu mapu nie je nájdený, je znova vykonaný proces predpočítavania.
Do riešenia bol pridaný aj parameter hustoty vytvárania bodov nájdených ciest, ***pathWaypointDensity***, ktorý sa nachádza v triede ***PathPlanning***. Implementovali sme aj vykresľovania najbližšieho cieľa agentov. Zapnutie tejto funkcie je možné nastavením parametra ***drawNearestTarget*** v triede ***PathPlanning*** na hodnotu ***true***. Na

obrázku (Obr. 6:1) je znázornené vykresľovanie najbližších cieľov agentov. Cieľ agenta je vykreslený modrou farbou.



Obr. 6:1 Najbližšie ciele agentov.

Pri prechádzaní agentov daným cieľom, vznikol problém pri blízkosti veľkého počtu agentov. V danej situácii mohli byť niektorí agenti ostatnými agentmi odtlačení od svojho plánovaného cieľa. Z toho dôvodu, na kontrolu prejdenie cieľa agentom, bola implementovaná metóda ***isNearGoal()*** v triede ***Agent***.

6.2 Nájdenie videa demonštrácie s vhodným scenárom

6.2.1 Úloha

Nájdenie a analyzovanie audio vizuálneho materálu zachytávajúceho reálnu demonštráciu.

6.2.2 Analýza

Dosiaľ sme nevedeli, či je možné simuláciu demonštrácie, ktorú vyvíjame, označiť za simuláciu uplatniteľnú v praxi. Zároveň sme si uvedomili, že vyvíjať simuláciu demonštrácie bez reálnych pokladov, ktoré by nám pomohli vyvíjanú simuláciu validovať, nie je vhodné práve z dôvodu validácie. Preto sme sa rozhodli, že validáciu simulácie urobíme porovnaním výsledkov simulácie s výsledkami reálnej demonštrácie. Písané informácie (počet demonštrantov, počet príslušníkov poriadkových zložiek, dĺžka trvania demonštrácie, ...)b y však nemuseli stačiť na zachytenie celého priebehu demonštrácie, a teda by sme stále neboli schopní plnohodnotne preukázať, že naša simulácia je korektná, validná. Na dosiahnutie a overenie správnost simulácie potrebujeme zachytiť priebeh reálnej demonštrácie. Demonštrácia bude slúžiť ako referenčná udalosť, voči ktorej budeme nami vyvíjanú simuláciu validovať a ktorej parametre vložíme ako vstupné parametre do našej simulácie.
Analýzu audiovizuálnych materiálov sme začali hľadaním videa demonštrácie, ktoré by zachytávalo priebeh demonštrácie. Z vhodného videa sme schopní získať informácie

o pohybe demonštrantov ako aj o pohybe a taktikách príslušníkov poriadkových zložiek. Video však nevypovedá o všetkých detailoch demonštrácie. Je nevyhnutné mať o demonštrácii aj informácie, suché fakty. Tie sa dajú získať z novinových článkov. Obe tieto podmienky, dosť dlhé a kvalitné video a množstvo informácií o demonštrácii, spĺňajú len videá publikované tlačovými agentúrami, ktoré k videu dopĺňajú aj článok s popisom demonštrácie, a teda aj videa. Uverejnenie informácií o demonštrácii tlačovou agentúrou však kladie isté podmienky aj na samotnú demonštráciu. Demonštrácia musí byť dostatočne početná, musí sa týkať pálčivých verejných otázok. Takéto videá sme aj našli [45] alebo [46].

6.2.3 Návrh

Uvedomujeme si, že nie sme jediní, ktorí vytvárajú simulácie demonštrácie v meste, a preto sme chceli vedieť, ako sa pozerajú na otázku validácie a snahy o reálnosť simulácie iní. Zistili sme, že nie sme jediní, ktorí validáciu a verifikáciu simulácie demonštrácie robia pomocou porovnávania simulácie s reálnou udalosťou [47]. Autori článku simulovali výtržnosti v meste Los Angeles v roku 1965. My sme potrebné informácie o udalosti nezískali, a preto ju ako referenčnú demonštráciu neuvažujeme. Na validáciu našej simulácie použijeme videá zu demonštrácií, ktoré prebiehali v Európe v ostatných rokoch. Je možné získať o nich dostatočné množstvo aj vizuálnych údajov a mentalita demonštrantov ako aj taktiky príslušníkov poriadkových zložiek je nám príbuzná.

6.3 Refaktoring zdrojového kódu

6.3.1 Úloha

Refaktorovanie zdrojového kódu projektu za účelom lepšej čitateľnosti kódu a zavedenia použitia objektovo orientovaných návrhových vzorov.

6.3.2 Návrh

Komplexnosť projektu a nutnosť vyššej paralelizácie práce v tíme si vyžiadala refaktorovanie zdrojového kódu, ktorý po mnohých zásahoch začal byť neprehľadný a niektoré časti neboli umiestnené na správnom mieste z hľadiska logiky a hierarchie. Na základe vyššie spomenutých kritérií sme navrhli rozdelenie zdrojového kódu do tried podľa obrázku nižšie (Obr. 6:2).

6.3.3 Implementácia

Ako vidieť na obrázku [Obr. 6:2], prostredie simulácie bolo vyňaté z hlavnej triedy ***Simulation***, aby bolo možné v prípade nutnosti nahradiť iným. Obsahuje prakticky celé prostredie, v ktorom agenti existujú, dátové vrstvy a tiež mapové vrstvy vrátane RVO modulu.

Taktiež samotný scenár, ktorý obsahuje pozície agentov a ich ciele bol vyňatý do samostatnej triedy ***Scenario***, aby bolo možné scenáre ľahko meniť bez zložitých zásahov. Bola vytvorená nová trieda Environment, ktorá obsahuje všetku logiku

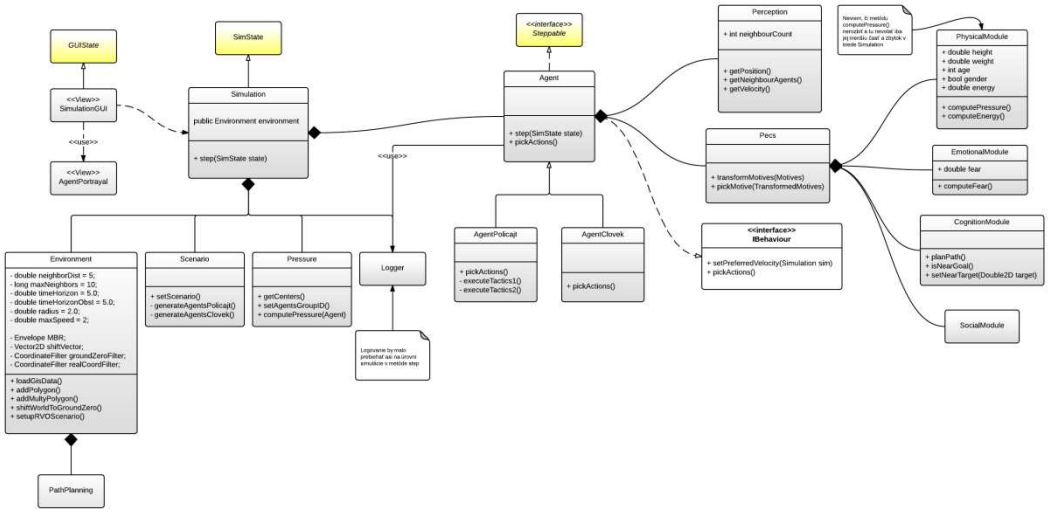
pracujúcu prostedím simulácie, napríklad metódy ***loadGisData()*** alebo ***addPolygon()***. Táto trieda zároveň agreguje triedu ***PathPlanning***, ktorá slúži na plánovanie trasy agentov.

V simulácii logujeme aj stav agenta pre štatistické a validačné účely. Keďže sa loguje stav agenta, tak v triede ***Agent*** je vytvorená inštancia triedy ***Logger***, ktorá logovanie stavu agenta zabezpečuje.

Psychologický model PECS bol rozdelený do jednotlivých častí a modulov v podobe tried a metódy spolu s atribútmi boli umiestnené do týchto tried ako napr. metóda, ktorá riadi výpočet strachu v každom kroku agenta bola umiestnená do modulu ***emotionalModule***, ktorého inštanciu obsahuje trieda ***PECS*** a inštanciu triedy PECS zase obsahuje samotný agent.

Do triedy ***PECS*** boli pridané metódy na vyhodnocovanie a normalizovanie motívov. Z agenta samotného bolo ďalej vyňaté vnímanie okolitého sveta do triedy ***Perception***, čo je v súlade z logickým členením psychologického modelu správania PECS.

Správanie agenta bolo vyprofilované tak, že implementujú rozhranie ***IBehaviour*** a každá skupina agentov bude mať určité špecifiká.



Obr. 6:2 UML diagram projektu.

7 Šiesty šprint

V nižšie uvedenej tabuľke (Tabuľka 7.1) je uvedený zoznam úloh, ktoré sa počas šiesteho šprintu, ktorý sme interne pomenovali Októbrová revolúcia, vyriešili a osoby, ktoré sa danými úlohami primárne zaoberali. Zoznam úloh a k nim prislúchajúcich riešiteľov.

Tabuľka 7.1 Zoznam úloh a k nim prislúchajúcich riešiteľov.

Názov úlohy	Zodpovedná osoba
Psychologický model	Michal Kyžňanský
Pridanie komponentu Božena	Adrián Kollár
Vyhľadávanie skupín agentov v dave	Filip Pakan
Dokončenie implementácie pôsobenia tlakov v dave	Michal Ošvát
Návrh a implementácia policajných línií a ich postupu	Miroslav Ort

7.1 Psychologický model

7.1.1 Úloha

Implementácia psychologického modelu agentov simulácie použitím psychologického modelu PECS

7.1.2 Analýza

Samotný model PECS pozostáva zo 4 modulov, ktoré pokrývajú 4 aspekty správania agenta. Tieto moduly tvoria telo agenta a jeho výstup je priamo zodpovedný za tvorbu finálnej akcie pomocou transformácie na tzv. motívy a z nich vyplývajúce aktivity. Daná aktivita smeruje k uspokojeniu aktuálne najakútnejšej potreby, ktorú samotný motív popisuje. Moduly obsahujú tzv. prechodové funkcie a stavové premenné, ktoré spoločne charakterizujú agenta po všetkých skúmaných stránkach.
Časť modelu PECS nazývaná Behaviour (správanie), ktorá je vlastne vykonávateľom samotného správania, je implementované v podtype agenta pomocou rozhrania ***IBehaviour*** viď. časť refaktorovanie kódu [kapitola 6.3].

7.1.2.1 Modul Emotions (Emócie)

Modul Emotions je zodpovedný za spracovanie podnetov, ktoré simulujú emocionálnu rovinu agenta. Obsahuje dôležitú premennú fear (strach), ktorá je komplexnou prechodovou funkciou transformovaná do ďalšieho stavu na základe viacerých atribútov agenta samotného a tiež jeho okolia.
Metódu ***computeFear()*** možno rozdeliť do troch častí, pričom každá zachytáva odlišnú dynamiku strachu v daných podmienkach.

a) Pozvoľné klesanie

Hodnota strachu pozvoľne klesá, čo predstavuje postupné upokojovanie sa. Graf v čase <0, 80>.

$$\boldsymbol{f_{fear}} = max_{fear} * e^{\left(-0.04*\ln\left(\frac{fear}{10}\right)*(-25)+\,0{,}5\right)}$$

b) Zľaknutie sa

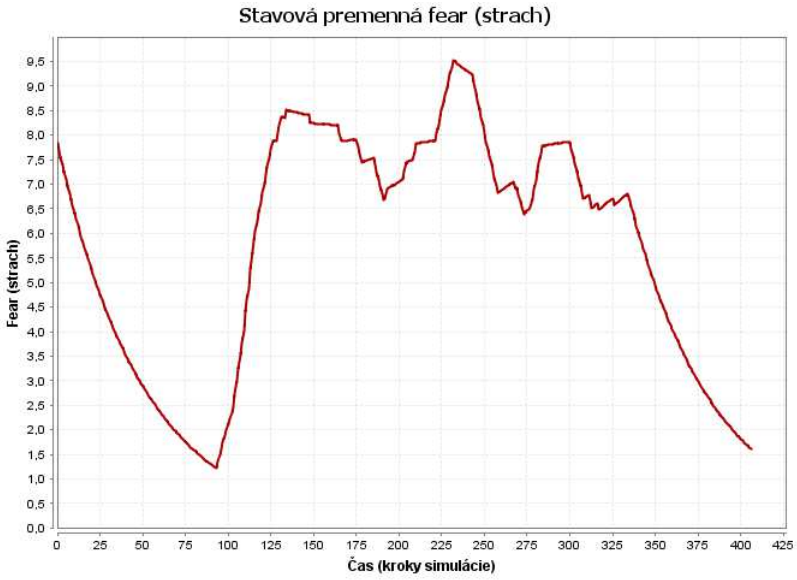
Realizuje diskrétne zvýšenie strachu, ktorého veľkosť je priamo úmerná počtu iných agentov nachádzajúcich sa v tzv. intímnej zóne agenta (INTIMATE_ZONE_DIAMETER). Táto časť opisu dynamiky strachu je význačná, pretože simuluje vznik paniky, keď sa dav začína prehusťovať. Graf v čase <225, 235>.

$$\boldsymbol{f_{fear}} = fear + \, tooClose * 0.010$$

c) Pozvoľné stúpanie

Posledná časť funkcie simuluje prenos strachu z okolia (okolitých agentov), tzv. šírenie strachu v dave. Tento spôsob zvyšovania strachu ovplyvňuje agenta výrazne menej ako diskrétne zľaknutie sa, avšak predstavuje kľúčový element pre dotvorenie komplexnej dynamiky strachu ako takej. Graf v čase <212, 218>.

$$\boldsymbol{f_{fear}} = fear + e^{\left(\frac{agentsFear\,-\,fear}{3}\right)} - 1$$



Obr. 7:1 Vývoj strachu.

7.1.2.2 MODUL Physical conditions (Fyzické vlastnosti)

Modul Physical conditions má na starosti spracovanie podnetov, ktoré simulujú všetko, čo súvisí s fyzickou interakciou agenta v simulácii. Obsahuje niekoľko dôležitých atribútov (premenných) ako napríklad age (vek agenta) a heart_rate (tep srdca). Množstvo energie, ktoré má agent pri pohybe ešte k dispozícii je vyjadrené premennou energy (energia).

Pri uvažovaní o tom ako modelovať tep človeka resp. agenta vychádzame z všeobecne známeho faktu, že človek má v pokoji tep okolo 60 úderov srdca za sekundu a tep ako taký ovplyvňuje aktuálny strach / stres. Taktiež tep srdca ovplyvňuje pohybová aktivita (v našom prípade chôdza / beh), t.j. rýchlosť pohybu. Maximálny bezpečný tep vyjadruje všeobecne akceptovaná aproximácia $\max_{\text{heartRate}} = (220 - \text{age})$.

Z danej úvahy vyplývajú tieto skutočnosti. Z vyšších frekvencií klesá tep pomaly (pomalé upokojovanie sa). Strach dokáže zvýšiť tep človeka prakticky v okamihu na vysoké hodnoty a tep taktiež postupne stúpa aj pri pohybe (chôdza / beh), kým nedosiahne vysoké stabilné hodnoty. Na základe týchto faktov sme zostavili podobne ako pri funkcií popisujúcej dynamiku strachu jednotlivé časti funkcie simulujúcej tep srdca so všetkými menovanými aspektmi.

- **Pozvoľné klesanie tepu (upokojovanie sa)**

  Realizuje záporná exponenciálna funkcia, ktorej hodnoty klesajú pozvoľne z maximálneho bezpečného tepu pre daného agenta.

  $$\boldsymbol{f_{heart\_rate}} = (max_{heartRate} - 60) * e^{-0.004*\left(-250*\ln(heart\_rate - 60)*\frac{1}{max_{heart\_rate}-60}+20\right)} + 60$$

- **Vplyv strachu na tep**

  Simuluje exponenciálna funkcia, a čím viac sa agent bojí, tým vyššie prírastky v podobe zvyšujúceho sa tepu funkcia prináša.

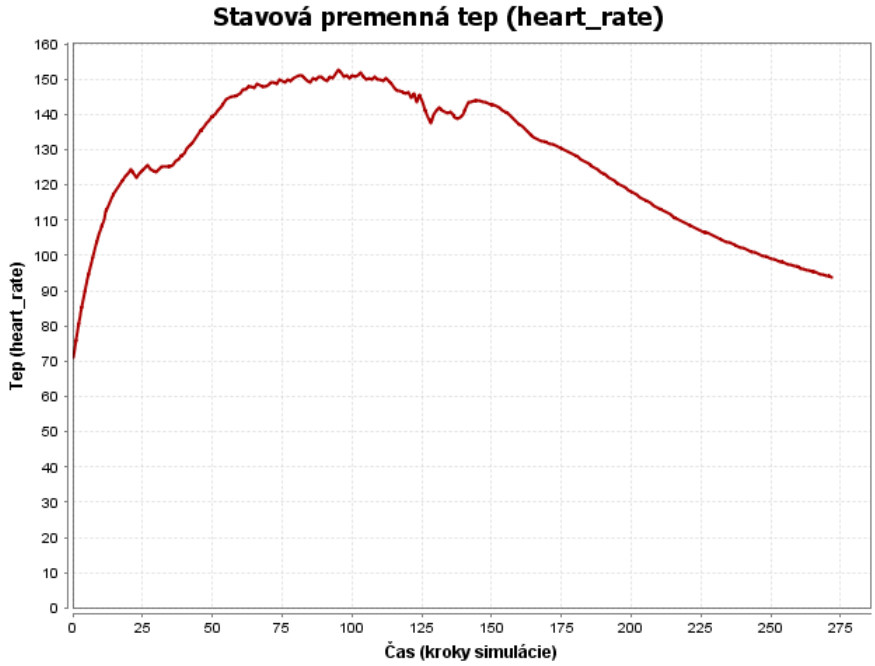
  $$\boldsymbol{f_{heart\_rate}} = heart\_rate + e^{0.1*fear}$$

- **Vplyv pohybu na tep**

  Rýchlosť chôdze/behu má na tep menší vplyv ako strach. Pohyb, resp. beh agenta spôsobuje postupné stúpanie tepu (logaritmické).

  $$\boldsymbol{f_{heart\_rate}} = heart\_rate + 5\ln(0.4 * speed + 1)$$

Hodnoty funkcie sú korigované, a je ohraničená zhora hodnotou bezpečného tepu pre daného agenta - $\max_{\text{heartRate}}$.



Obr. 7:2 Vývoj tepu človeka.

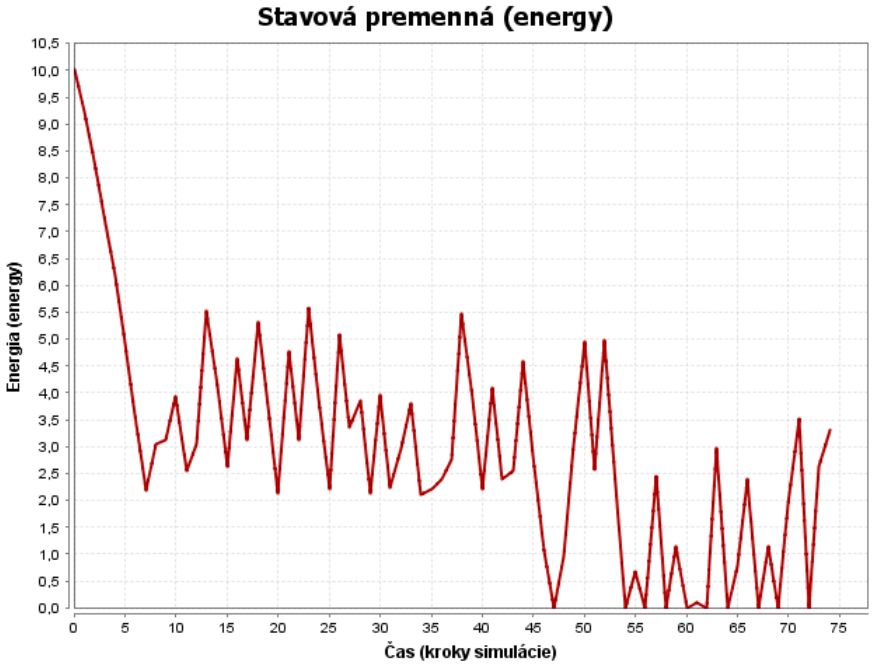
Energiu, ktorá je vydaná pri pohybe možno stotožniť s množstvom kalórií, ktoré sú pri pohybe spálené. To nám poskytne dobrú východiskovú pozíciu na modelovanie výdaju energie pomocou funkcie publikovanej v časopise Journal of Sports Sciences. Tá berie do úvahy vek, hmotnosť a aktuálny tep srdca na výpočet vydaných kalórií. Pre aproximáciu použijeme formulu určenú pre mužov a hmotnosť agenta budeme uvažovať na úrovni 80kg.

$$\boldsymbol{f}_{\boldsymbol{calories/min}} = \frac{(-55{,}0969 + 0{,}6309 * \ heart\_rate + 0{,}1988 \text{ x } 80 + 0{,}2017 * \ age)}{4{,}184}$$

Agent bude pri pohybe strácať energiu (spaľovať kalórie) ak zastane, energia sa mu bude dobíjať. Starší agent spotrebuje energiu rýchlejšie, čo simuluje znížený fyzickú výkonnosť vo vyššom veku. Taktiež pri zastavení si starší agent energiu dopĺňa pomalšie, t.j. je dlhšie unavený.

$$\boldsymbol{f}_{\boldsymbol{energy}} = energy - \ 0.04 * \ f_{calories/min} \quad (pre\ speed > 0)$$

$$\boldsymbol{f}_{\boldsymbol{energy}} = energy + \frac{10}{age} + 6rand \quad (pre\ speed = 0, rand \ \in < 0{,}1))$$



Obr. 7:3 Vývoj energie človeka v závislosti od času.

7.1.2.3 MODUL Cognitive capabilities (Myslenie)

Tento modul predstavuje v rámci tela agenta akúsi rozhodovaciu zložku, ktorá je zodpovedná za ťažšie kognitívne úkony agenta a taktiež sa v tejto časti vykonáva rozhodovanie ohľadne výberu trasy resp. plánovania trasy.

Záležitosti ohľadne plánovania trasy riešia metódy ***planPath()***, ***isNearGoal()*** a ***setNearTarget()***.

7.1.2.4 MODUL Social status (Sociálny stav)

Tento modul je zodpovedný za spracovanie podnetov v rovine sociálnych vzťahov v rámci agentov. Nateraz ostáva tento modul nenaplnený, v neskoršej fáze projektu bude pravdepodobne doplnený.

7.1.2.5 Výber a transformácia motívu

Trieda ***PECS*** po vykonaní všetkých potrebných prechodových funkcií musí transformovať motívy vyplývajúce z jednotlivých stavových premenných resp. ich kombinácií vytvoriť motívy. Tie musia byť transformované tak, aby sa dali numericky porovnávať, ktorý ma vyššiu prioritu. Túto časť zabezpečuje metóda ***transformMotives()*** vyššie menovanej triedy.

Na základe tejto transformácie možno zistiť, ktorý motív prevláda a zvoliť ho. Tento proces má na starosti metóda ***pickMotive()***.

Na základe motívu je zvolená aktivita agenta, ktorú rieši komponent Behaviour v podtype agenta na základe špecifických definícií správania typického pre demonštrantov resp. políciu.

7.2 Božena RIOT

7.2.1 Úloha

Cieľom tejto úlohy je implementovanie vozidla Božena RIOT, na zvládanie davu pri nepokojoch a výtržnostiach, do prostredia našej simulácie. Vozidlo Božena RIOT bude v simulácii použité pri rôznych scenároch demonštrácií na usmernenie a zvládnutie nepokojného davu ľudí.

7.2.2 Analýza

BOŽENA RIOT je diaľkovo ovládané obrnené vozidlo, určené na reguláciu nepokojov a kontrolu davu v uliciach a v zastavaných oblastiach. Systém ponúka ako ochranu poriadkových policajných jednotiek v akcii, tak aj riešenie pre udržiavanie verejného pokoja [48].

BOŽENA RIOT využiteľná ako vysokomobilný a efektívny nástroj schopný:

- usmerňovať alebo rozptyľovať zhromaždené skupiny do požadovaného smeru alebo zón
- regulovať prístup k oblastiam, bezpečne a efektívne
- umožniť priame pozorovanie a monitorovanie situácie v dave, s možnosťou identifikácie hlavných narušiteľov
- pomocou hydraulickej plošiny zabezpečiť prístup k budovám, mostom a iným vyvýšením pozíciám, ktoré môžu byť obsadené narušiteľmi/útočníkmi
- komunikovať s davom/výtržníkmi pomocou výkonných akustických zariadení a chráneným programovateľným displejom umiestneným v prednej časti systému
- vykonávať rôzne ďalšie činnosti, ako odstraňovanie barikád, vozidiel, nebezpečných objektov a materiálov za pomoci prídavných zariadení

7.2.3 Návrh

Pre potreby našej simulácie bude postačujúce implementovanie modelu Boženy RIOT, ktorý bude schopný zvládať funkcie usmerňovania zhromaždených skupín do požadovaného smeru a regulovanie prístupu k určitým oblastiam. V prostredí simulácie bude model Boženy RIOT vykreslený ako polygón, ktorý je znázornený na obrázku (Obr. 7:4).

Obr. 7:4 Model Boženy RIOT.

7.2.4 Implementácia

Model Boženy RIOT je v našej simulácii reprezentovaný triedou ***Bozena***, ktorá obsahuje všetky potrebné metódy na vytvorenie modelu Boženy, pohyb modelu, rotovanie modelu a vkladanie modelu do prostredia simulácie. Božena je vytváraná ako polygón pomocou triedy ***GeometryFactory*** z knižnice *Mason*. Aby bola zabezpečená správna reakcia agentov na objekt Boženy v simulácii, bolo potrebné ju vkladať aj do prostredia vrstvy prekážok. Avšak Božena netvorí klasickú prekážku simulácie, pretože oproti ostatným prekážkam sa Božena dokáže pohybovať. Za týmto účelom je potrebné Boženu vkladať do prostredia ako dynamickú prekážku, čo umožnila modifikovaná knižnica ***RVO2***, ktorú upravila Slovenská Akadémia Vied. Pohyb Boženy nie je riadený pomocou knižnice ***RVO2***, ale bude predprogramovaný podľa konkrétneho scenára, v ktorom bude objekt Boženy použitý.

7.2.5 Testovanie

Pri testovaní správnosti implementácie modelu Boženy RIOT bolo potrebné otestovať najmä správnu reakciu agentov na model Boženy, tj. registrovanie modelu Boženy, korektné vyhýbanie sa modelu Boženy a schopnosť Boženy odtláčať skupiny agentov. Všetky uvedené vlastnosti boli pri testovaní Boženy splnené. Testovanie vytláčania skupiny agentov je znázornené na obrázku (Obr. 7:5).

Obr. 7:5 Testovanie vytláčania skupiny agentov.

7.3 Vyhľadávanie skupín agentov v dave

7.3.1 Úloha

Cieľom tejto úlohy je analyzovať, navrhnúť, implementovať a otestovať funkčný algoritmus na vyhľadávanie skupín agentov. Súčasný implementovaný algoritmus pracuje do značnej miery podozrivo a neefektívne.
Algoritmus na zhlukovanie agentov je kľúčovou súčasťou programu. Je nevyhnutný pre korektné počítanie tlakov v dave a vykonávať sa bude každý krok simulácie. Je preto prirodzené, že nároky na nízku časovú zložitosť a správnosť výpočtu sú očividné.

7.3.2 Analýza

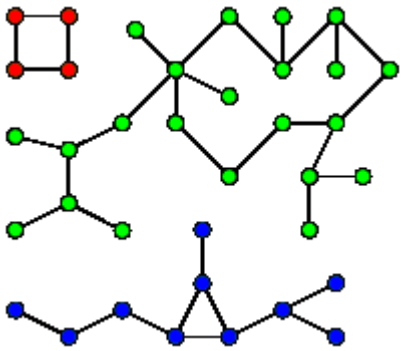
Existuje viacero možností ako zhlukovať agentov do skupín. Medzi štandardné algoritmy strojového učenia na zhlukovanie patria rôzne variácie algoritmu *k-means*.

7.3.2.1 K-means

Nevýhodou algoritmu k-means je, že je pre naše účely príliš všeobecný. Nám stačí vytvárať skupiny agentov, ktorý sú pri sebe do určitej vzdialenosti. Ďalšou nevýhodou je to, že by sme potrebovali dopredu stanoviť hodnotu *k*, t.j. počet zhlukov.

7.3.2.2 Connected components

Connected components (komponenty súvislosti) je pojem z teórie grafov. Komponent súvislosti grafu G je maximálny súvislý podgraf. To znamená, že v rámci jedného komponentu sa dá dostať po hranách medzi ľubovoľnými dvoma vrcholmi (Obr. 7:6).



Obr. 7:6 Graf s tromi komponentmi súvislosti.

Algoritmus funguje nasledovne. Každému ešte nezaradenému vrcholu sa priradí ID komponentu, do ktorého patrí. Následne sa všetkým jeho susedom priradí to isté ID komponentu. Rovnako tak pre všetkých susedov susedov vrcholu sa priradí to isté číslo komponentu. Prehľadávanie môže byť buď do šírky alebo do hĺbky. V každom prípade sa však každý vrchol navštívi iba raz a prehľadávajú sa iba tie vrcholy, ktoré ešte nemajú priradené číslo komponentu. Preto zložitosť algoritmu je lineárna.
Poznamenajme, že lepšiu zložitosť už nie je možné dosiahnuť, pretože je potrebné priradiť číslo komponentu každému vrcholu. A to bude vždy stáť minimálne O(n).
V oblasti spracovania obrazu sa rovnaký algoritmus nazýva *Flood Fill*. Len namiesto vrcholov uvažuje pixely a namiesto hrán spájajúcich susedné vrcholy uvažuje okolité pixely.

7.3.3 Návrh

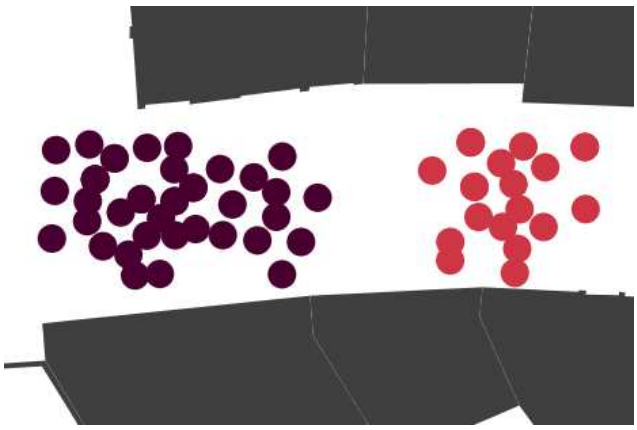
Agentov si môžeme predstaviť ako vrcholy grafu. Medzi dvoma agentmi bude hrana práve vtedy, ak spolu susedia, t.j. sú pri sebe do určitej vzdialenosti. Algoritmus bude mať na vstupe parameter *maxDistance*, ktorý bude určovať maximálnu vzdialenosť na to, aby dvaja agenti boli označení ako susední.
Rovnako tak algoritmus bude schopný zaznamenať všetky skupiny a roztriedi všetkých agentov do príslušnej skupiny. Takto bude v ďalšej fáze pri počítaní tlakov triviálne zistiť, ktorí agenti patria do ktorej skupiny.

7.3.4 Implementácia

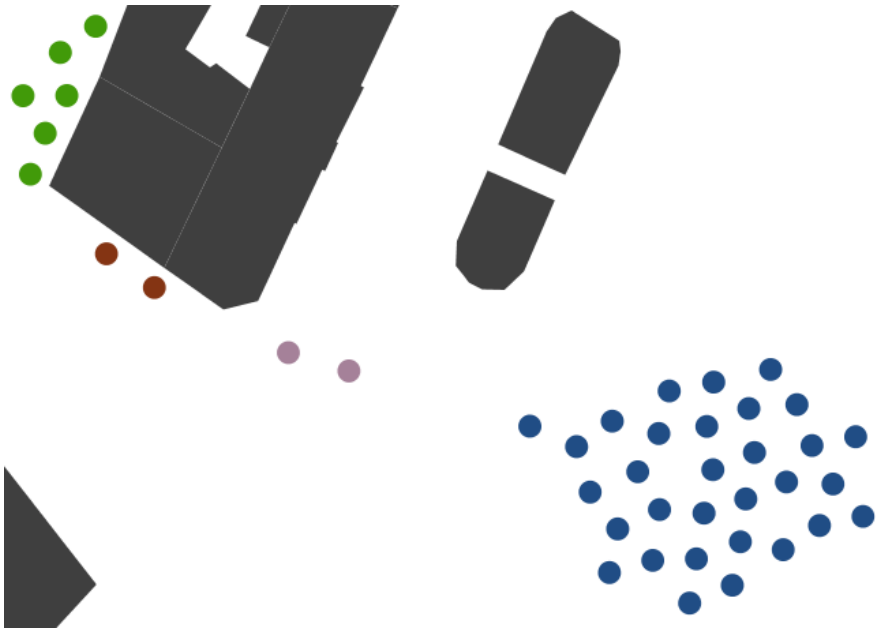
Implementovaný bol algoritmus na hľadanie komponentov súvislosti. Na zisťovanie susedov konkrétneho agenta používame funkciu *getObjectsWithinDistance(...)* z knižnice *GeoMason*.
Počas implementácie sme zistili, že pri pohybe objektov vo vrstve *GeomVectorField* je potrebné aktualizovať geometrický index (*spatialIndex*), ktorý využíva knižnica *GeoMason* a rovnako tak aj *Java Topology Suite* na vrátanie objektov do určitej vzdialenosti.

7.3.5 Testovanie

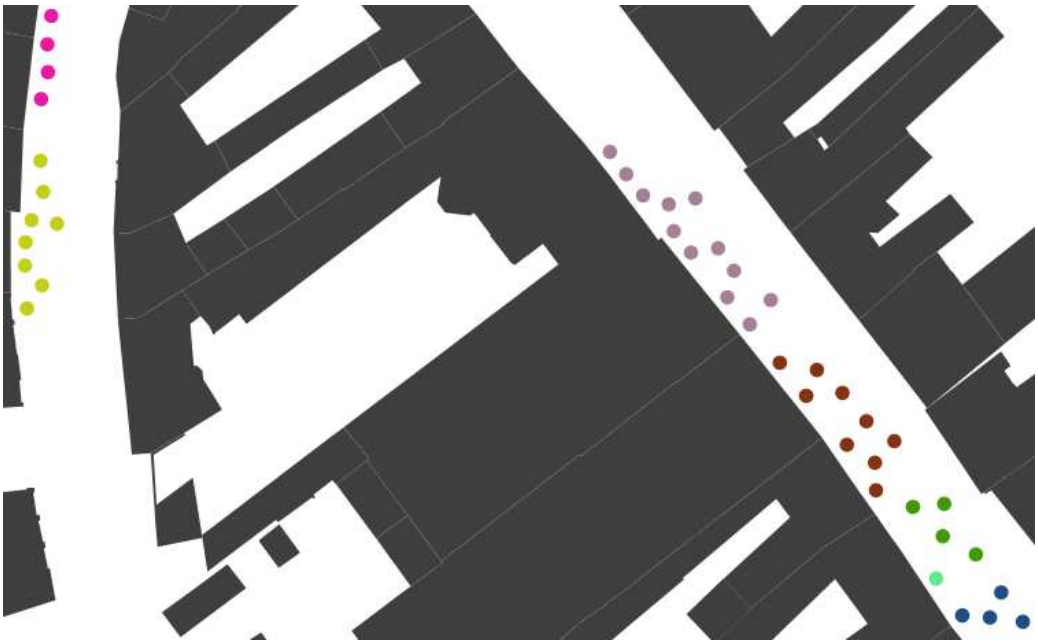
Implementovaný algoritmus pracuje správne a vracia očakávané výsledky. Podľa dokumentácie ku knižnici *Java Topology Suite*, však môže občas dôjsť k tomu, že funkcia nám vráti ako susedné objekty aj také, ktoré sú ďalej ako je stanovená vzdialenosť.
Tento problém sa dá dodatočne vyriešiť tým, že získané susedné objekty preveríme, či sa naozaj nachádzajú v rámci určenej vzdialenosti. Na to môžeme použiť funkciu *isObjectWithinDistance(...)*.
Výsledky zhlukovania agentov môžete vidieť na nasledujúcich obrázkoch (Obr. 7:7, Obr. 7:8, Obr. 7:9).



Obr. 7:7 Vytvorenie dvoch zhlukov agentov.



Obr. 7:8 Vytvorenie štyroch zhlukov agentov.



Obr. 7:9 Vytvorenie siedmych zhlukov agentov.

7.4 Dokončenie implementácie pôsobenia tlakov v dave

7.4.1 Úloha

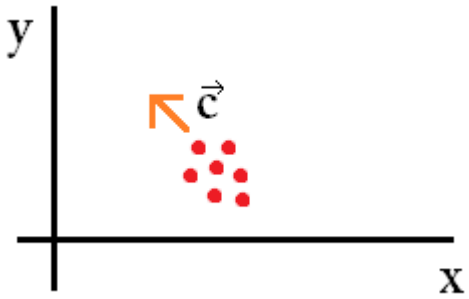
Dokončenie implemetácie pôsobenia tlaku v multiagentovom prostredí. Implementácia bude vylepšená použitím univerzálneho riešenia, pri ktorom sa bude tlak v dave šíriť ľubovoľným smerom (360°).

7.4.2 Analýza

Dvojrozmerný priestor je v simulácii reprezentovaný pomocou osí x,y. Pozícia každého agenta je preto reprezentovaná pomocou dvojice [x1,y1]. Skupina agentov sa môže pohybovať do všetkých smerov, a preto je potrebné zaviesť výpočet pôsobenia tlaku v dave (napr. pri stretnutí sa s radom policajtov) taktiež od všetkých smerov.

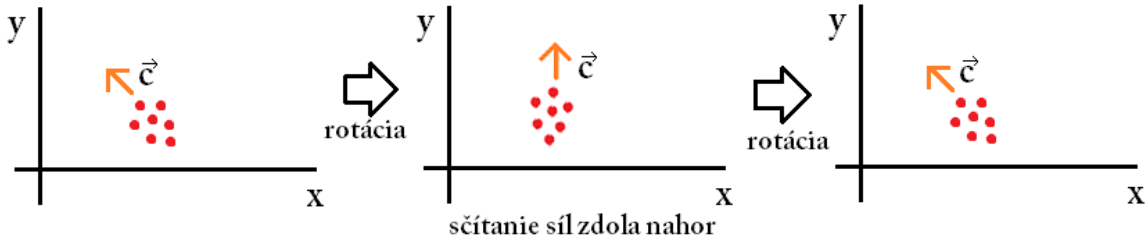
7.4.3 Návrh

Riešenie problému šírenia tlaku v dave do všetkých smerov je netriviálne riešenie.
Na obr.1 je znázornená skupina agentov, a vektor ich smeru. Centrálny vektor C dostaneme výpočtom: $\sum_{i=1}^{n} v_i$, kde n je počet agentov v grupe, v_i je vektor i-tého agenta (Obr. 7:10).



Obr. 7:10 Ilustračný príklad.

Problémom je spôsob sčítavania síl, keďže nie je jednoduché implementovať postup sčítavania v inom smere, ako sú 4 základné(nahor, nadol, doľava, doprava). Preto návrh spočíva v riešení pretočiť body v rovine tak, aby ich centrálny vektor smeroval nahor (Obr. 7:11), a potom už implementácia sčítavania síl v smere zdola nahor nie je náročná.



Obr. 7:11 Grafické znázornenie algoritmu.

Algoritmus prebieha v nasledovných krokoch:

1.) Vypočítaj centrálny vektor a stred skupiny
2.) Zrotuj body v priestore o uhol tak, aby výsledný vektor smeroval nahor
3.) Sčítaj pôsobenie tlaku v smere zdola nahor
4.) Zrotuj body v priestore naspäť.

7.4.4 Implementácia pre všetky smery centrálneho vektora

Implementácia pôsobenia tlaku bude vylepšená zrušením aproximácie len do štyroch smerov (nahor, nadol, doľava, doprava), a nahradením univerzálnym riešením, pri ktorom sa bude tlak v dave šíriť ľubovoľným smerom (360°).

7.4.5 Testovanie

Testovanie prebiehalo vo mnohých smerov, pričom sa overovalo výsledné pôsobenie síl, ktoré bolo vždy najvyššie u „hraničného agenta“ (napr. na Obr. 7:10 je to agent vľavo hore).

7.5 Návrh a implementácia policajných línií a ich postupu

7.5.1 Úloha

Rozmiestnenie policajtov, ich pohyb a rozširovanie rozostupov medzi nimi v prípade rozširujúcej sa ulice. Obchádzanie objektov líniou policajtov.

7.5.2 Návrh

V simulácii demonštrácie nie je vylúčené, že bude potrebné poslať radu príslušníkov poriadkových zložiek, aby strážili a prípadne vytláčali demonštrantov postupujúc rozširujúcou sa ulicou. Pri takomto postupe poriadkových zložiek je nevyhnutné, aby počas celej doby postupu pozdĺž ulice bránili demonštrantom prejsť skrz alebo popri poriadkových zložkách. Je preto nutné vytvoriť takú taktiku príslušníkov poriadkový zložiek, aby bránili celú šírku ulice.
V našej simulácii sme mali problém, že aj tesné usporiadanie agentov poriadkových zložiek umožňovalo prechod demonštrantov skrz nich. Za tesné usporiadanie považujeme situáciu, kedy boli agenti rozmiestňovaní vo vodorovnej alebo zvisle línii s rozostupom 1 pixel. Takéto správanie demonštrantov bolo v najväčšou pravdepodobnosťou zapríčinené metódami RVO2, ktoré nevieme ovplyvniť. Preto sme sa rozhodli, že poriadkové zložky podporíme tak, že na ich pôdoryse bude vytvorený objekt, ktorý bude farebne splývať s pozadím a spôsobí, že agenti predstavujúci demonštrantov nebudú prechádzať cez líniu vytvorenú poriadkovými zložkami.

7.5.3 Implementácia

Príslušníci poriadkových zložiek, ktorí tvoria jednu líniu, tvoria uzly orientovaného grafu s príslušnou váhou hrán. Graf je použitý na odovzdávanie informácií členom jednej línie poriadkových zložiek. Zároveň nám graf dáva možnosť určovať poradie členov línie poriadkových zložiek a určovať vzdialenosti medzi nimi. Graf je vytvorený pomocou dátovej štruktúry *Network,* ktorá je súčasťou framework-u MASON [49]. Pohyb poriadkových zložiek tak, aby bezpečne ochránili celú šírku ulice zabezpečíme tak, že krajní agenti sa budú pohybovať v malej ale nulovej vzdialenosti od okrajov budov na ulici. Táto funkcionalita bude implementovaná v ďalšom šprinte.

Objekt zaručujúci zadržiavanie demonštrantov pred poriadkovými zložkami je dynamický objekt pomocou triedy ***GeometryFactory*** z knižnice *Mason*. Vytvára sa po vytvorení línie agentov poriadkových zložiek. Objekt tvoriaci prekážku pre demonštrantov je v simulácii vytvorený a reprezentovaný triedou ***PoliceObstacle***, ktorá obsahuje všetky atribúty a metódy na vytvorenie objektu. V nasledujúcich šprintoch chceme doplniť triedu o metódy, ktoré budú zabezpečovať rotáciu a pohyb objektu, ktorý bude zhodný s pohybom poriadkových zložiek. Na zabezpečenie reakcie agentov reprezentujúcich demonštrantov na objekt je nevyhnutné objekt vložiť aj do prostredia vrstvy prekážok.

8 Siedmy šprint

V nižšie uvedenej tabuľke (Tabuľka 8.1) je uvedený zoznam úloh, ktoré sa počas siedmeho šprintu, ktorý sme interne pomenovali Soľný pochod, vyriešili a osoby, ktoré sa danými úlohami primárne zaoberali. Zoznam úloh a k nim prislúchajúcich riešiteľov.

Tabuľka 8.1 Zoznam úloh a k nim prislúchajúcich riešiteľov.

Názov úlohy	Zodpovedná osoba
Návrh a implementácia policajných línií a ich postupu	Miroslav Ort
Rotácia komponentu božena	Jana Branišová

8.1 Návrh a implementácia policajných línií a ich postupu

8.1.1 Úloha

Rozmiestnenie policajtov, ich pohyb a rozširovanie rozostupov medzi nimi v prípade rozširujúcej sa ulice. Obchádzanie objektov línou policajtov.

8.1.2 Implementácia

Pohyb poriadkových zložiek v rozširujúcej sa ulici je implementovaný z dôvodu zabránenia demonštrantom prechodu cez ulicu. Poriadkové zložky neustále udržiavajú pod kontrolou celú jej šírku. Ulica, ktorá sa rozširuje ale spôsobuje, že príslušníci poriadkových zložiek musia zväčšovať rozostupy medzi sebou, ale musia zostať v jednej línii. Implementovať takéto správanie predstavuje netriviálny problém.
Agenti poriadkových zložiek predstavujú body, uzly, v grafe, ktorý je vytvorený pomocou triedy *Network*. Trieda je integrálna súčasť simulačného prostredia MASON. Takto vieme medzi príslušníkmi poriadkových zložiek iterovať a nastavovať každému jednotlivo parametre. V línii policajtov majú dvaja krajní zvláštnu úlohu. Držia si primeraný konštantný odstup od budov, ktoré obopínajú ulicu. Konštantný odstup od budovy je počítaný pre agentov v metóde ***setAvoidance*** v triede ***PoliceScalling***. Princípom je prechádzanie predpočítanej mapy prekážok ***obstacleDistanceMap*** v okolí miesta, kde sa agent nachádza a nájdenie bodu, ktorý je v primeranej požadovanej vzdialenosti od budovy na ceste pozdĺž ulicou. Do tohto bodu následne agent smeruje svoj nasledujúci pohyb.
Ostatní agenti, policajti v strede línie, nevedia nič o tom, že sa ulica rozširuje. Ak by si každý počítal svoju polohu voči budove a voči ostatným agentom, tak by sa neúmerne zväčšila náročnosť výpočtu a tým celá simulácia zvýšila potrebu výpočtových zdrojov. Líniu policajtov teda vedú dvaja krajní agenti, ktorí sa držia popri budovách a ostatným len povedia, že sa ulica rozšírila. Implementácia šírenia informácie o rozšírení ulice je zabezpečená pomocou iterácie cez graf tvorený policajtmi a výpočtom rovnice priamky. Krajní policajti sa pri pohybe pozdĺž budovy od seba vzďaľujú. V každom kroku simulácie je vypočítaný vektor medzi dvomi krajnými policajtami. Jeho dĺžka sa podelí

počtom policajtov v línii. Následne sa od prvého krajného policajta (pri implementácii je jedno, ktorý z dvoch krajných je prvý, ale jeden by zvolený musí) vypočíta bod, ktorý leží na priamke tvorenej pozíciami dvoch krajných policajtov a je vo vzdialenosti

$$X = poziciaPrveho + \frac{\text{vzdialenostKrajnych}}{\text{pocetPolicajtov}} * \text{poradiePolicajta}$$

od prvého krajného policajta. Takto sa iteratívne vypočíta pozícia pre každého policajta, do ktorej sa nastaví jeho cieľ. Línia policajtov vytvorí úsečku, ktorá má hranice dvoch krajných policajtov a rozostupy sa vhodne upravujú postupom policajnej línie. Keďže línia policajtov je vlastne priamka, po ktorej sa príslušníci poriadkových zložiek pohybujú, tak tento prístup funguje. Ukážky zo simulácie je možné vidieť na obrázkoch nižšie (Obr. 8:1, Obr. 8:2).

Obr. 8:1 Postup policajnej línie rozširujúcou sa ulicou.

Obr. 8:2 Postup policajnej línie rozširujúcou sa ulicou.

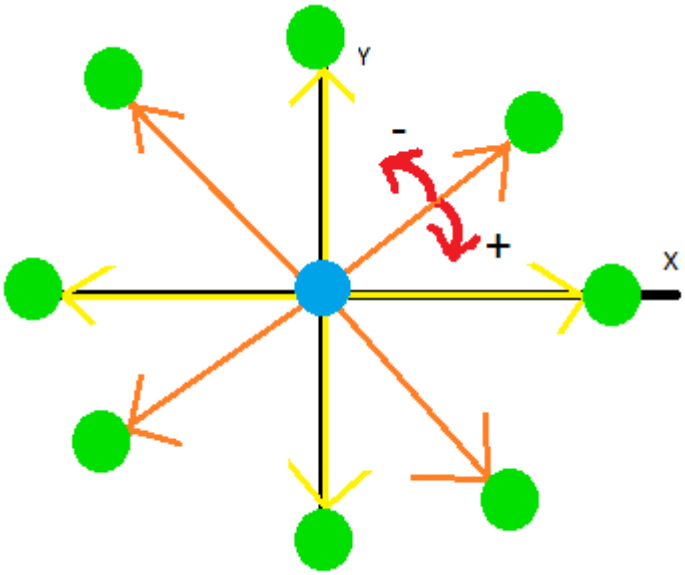
8.2 Rotácia komponentu Božena

8.2.1 Úloha

Návrh riešenia rotovania komponentu Božena a jeho implementácia.

8.2.2 Návrh

Rotácia komponentu Božena sa odvíja od troch bodov. Bod so súradnicami jej poslednej pozície, bod jej aktuálnej pozície a bod pozície kam smeruje. Bod poslednej pozície a bod aktuálnej pozície tvoria vektor a určujú jej aktuálne natočenie. Bod aktuálnej pozície a bod kam smeruje tvoria vektor, ktorý rozhoduje ktorým smerom pôjde. Na základe týchto troch bodov sa za pomoci goniometrických funkcií vypočíta uhol, o ktorý sa má Božena otočiť a smer, do ktorého sa bude otáčať. Tento smer môže byť pravotočivý alebo ľavotočivý.
Pre výpočet uhla, o ktorí sa má Božena otočiť uvažujem 8 smerov, z ktorých sa mohla dostať na aktuálnu pozíciu a 8 smerov, ktorými sa môže vydať.



Obr. 8:3 Grafické znázornenie možných smerov Boženy.

Na obrázku (Obr. 8:3) sú zobrazené všetky možné smery Boženy. Modrý bod znázorňuje východiskový bod odkiaľ ide (kde sa nachádza) a zelený kde sa nachádza (kam smeruje). Čiary znázorňujú smery, ktorými sa Božena môže vydať alebo, z ktorého smeru šla. Žltá čiara je pre smery, kde má aktuálna pozícia Boženy rovnakú x alebo y súradnicu. A oranžová čiara označuje smer na pozíciu, ktorá má rozličnú x a y súradnicu.

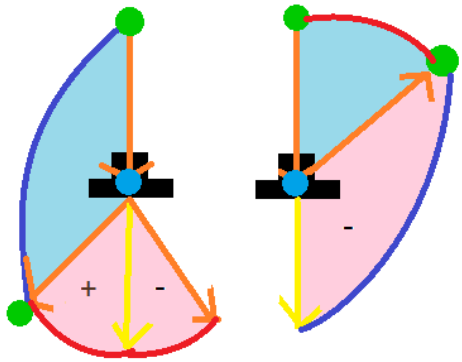
Výpočet uhla na základe týchto smerov podľa tejto goniometrickej rovnice:

vector1x = position.x - oldPosition.x;
vector1y = position.y - oldPosition.y;
vector2x = position.x - newPosition.x;
vector2y = position.y - newPosition.y;

$$vector1Magnitude = \sqrt{(vector1x * vector1x + vector1y * vector1y)}$$
$$vector2Magnitude = \sqrt{(vector2x * vector2x + vector2y * vector2y)}$$

$$angle = \cos^{-1}\left(\frac{vector1x * vector2x + vector1y * vector2y}{vector1Magnitude * vector2Magnitude}\right)$$

Najskôr sa vypočítajú súradnice vektorov (*vector1x*, *vector1y*), potom sa vypočítajú ich dĺžky (*vector1Magnitude*) a na základe toho sa vypočíta samotný uhol (*angle*). Tento uhol nie je však výsledný, pretože Božena sa môže pohybovať o ten uhol v smere hodinových ručičiek alebo v protismere ako je zobrazené na obrázku (Obr. 8:4). Ak v smere vtedy uhol nadobúda zápornú hodnotu a ak v opačnom tak zápornú.



Obr. 8:4 Grafické znázornenie kladného a záporného rotovania Boženy.

Na obrázku (Obr. 8:4) je modrou plochou vyznačený uhol, ktorý sa vypočítal v rovnici vyššie. Tento uhol sa musí odpočítať od 180° aby sme dostali uhol, o ktorý sa má Božena rotovať, (Obr. 8:4) znázornený ružovou plochou. Výsledný vzorec bude:

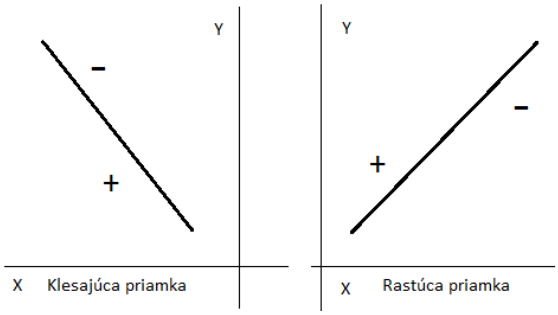
$$(180° - angle) * sign$$

kde *sign* predstavuje znamienko pre smer otočenia. Znamienko sa dalo pre niektoré smery jednoducho určiť a pri smere po diagonále sa počíta pomocou rovnice priamky:

$$sign = ax + by + c$$

kde a, b, c sú vypočítané z priamky určenej bodmi polohy odkiaľ ide a kde sa nachádza a x, y sú súradnice bodu, kam Božena smeruje. Priamka môže byť klesajúca alebo rastúca a od toho sa určí nakoniec výsledné znamienko pre uhol. Sklon priamky sa vypočíta:

$$k = \frac{-a}{b}$$



Obr. 8:5 Grafické znázornenie sklonu priamky.

Na obrázku (Obr. 8:5) vidíme ako sa mení znamienko nad priamkou a pod ňou v závislosti na sklone priamky. Výsledok s rovnice priamky nám udáva či sa bod nachádza pod alebo nad priamkou a jej sklon či sa znamienko uhla bude meniť alebo nie. Pri klesajúcej priamke sa znamienko uhla bude rovnať znamienku z rovnice priamky (*sign*) a pri rastúcej sa bude rovnať –*sign.*

8.2.3 Záver

Tento návrh bol optimalizovaný a v zdrojovom kóde sa nachádza práve táto optimalizovaná verzia.

9 Ôsmy šprint

V nižšie uvedenej tabuľke (Tabuľka 9.1) je uvedený zoznam úloh, ktoré sa počas ôsmeho šprintu, ktorý sme interne pomenovali Pochod na Washington, vyriešili a osoby, ktoré sa danými úlohami primárne zaoberali. Zoznam úloh a k nim prislúchajúcich riešiteľov.

Tabuľka 9.1 Zoznam úloh a k nim prislúchajúcich riešiteľov.

Názov úlohy	Zodpovedná osoba
Implementácia scenárov	Filip Pakan
Vytvorenie konfigurovateľného scenára	Filip Pakan
Návrh a implementácia policajných línií a ich postupu	Miroslav Ort

9.1 Implementácia scenárov

9.1.1 Úloha

Cieľom tejto úlohy je vytvoriť niekoľko jednotných scenárov, ktoré budú použiteľné aj na demonštračné účely.
Doposiaľ si počas vývoju každý člen tímu vytvoril vlastný scenár, na ktorom testoval svoj kód. S tým má byť po novom koniec. Na niekoľkých jednotných scenároch odprezentujeme všetky „features“ našej simulácie.

9.1.2 Analýza

V prvom rade bolo potrebné určiť, koľko scenárov chceme mať implementovaných. Pre začiatok som si vybral 2 scenáre – jeden základný a jeden komplexnejší.

9.1.2.1 Základný scenár

Základný scenár bude prezentovať základnú funkcionalitu simulácie. Bude pozostávať z dvoch skupín demonštrantov, jednej línie policajtov a jednej Boženy. Jedna skupina demonštrantov bude namierená proti policajtom a druhá proti Božene.
Cieľom tohto scenára je overiť si interakcie medzi jednotlivými typmi účastníkov.

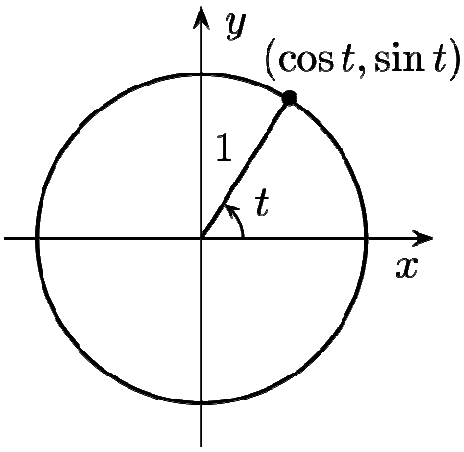
9.1.2.2 Komplexný scenár

Komplexný scenár bude predstavovať rozsiahlejšiu demonštráciu v centre mesta. Plánujem nasadenie piatich Božien, šiestich skupín demonštrantov a troch línií policajtov. Celkovo bude scenár obsahovať 145 agentov, čím sa zároveň overí aj efektivita našej simulácie.

9.1.3 Návrh

Základný scenár navrhnem tak, aby demonštranti po prekonaní policajnej zábrany a vyhnutí sa Božene, obkľúčili prezidentský palác. Okolo prezidentského paláca sa rozostúpia po kružnici, čo vytvorí dojem obkľúčenia.

Polohu agentov na kružnici jednoducho odvodím z jednotkovej kružnice (Obr. 9:1)



Obr. 9:1 Jednotková kružnica.

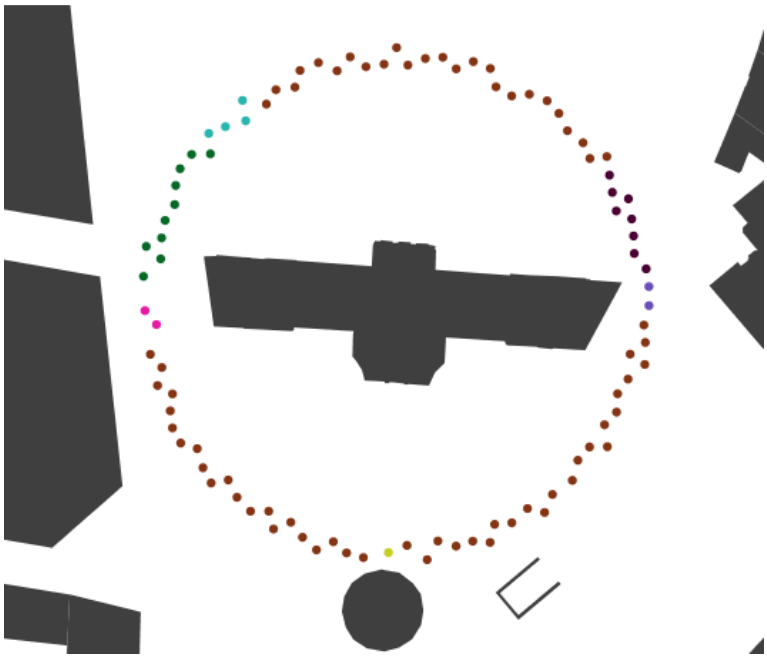
$$x = radius * \cos(\varphi) + centerX$$
$$y = radius * \sin(\varphi) + centerY$$

9.1.4 Implementácia

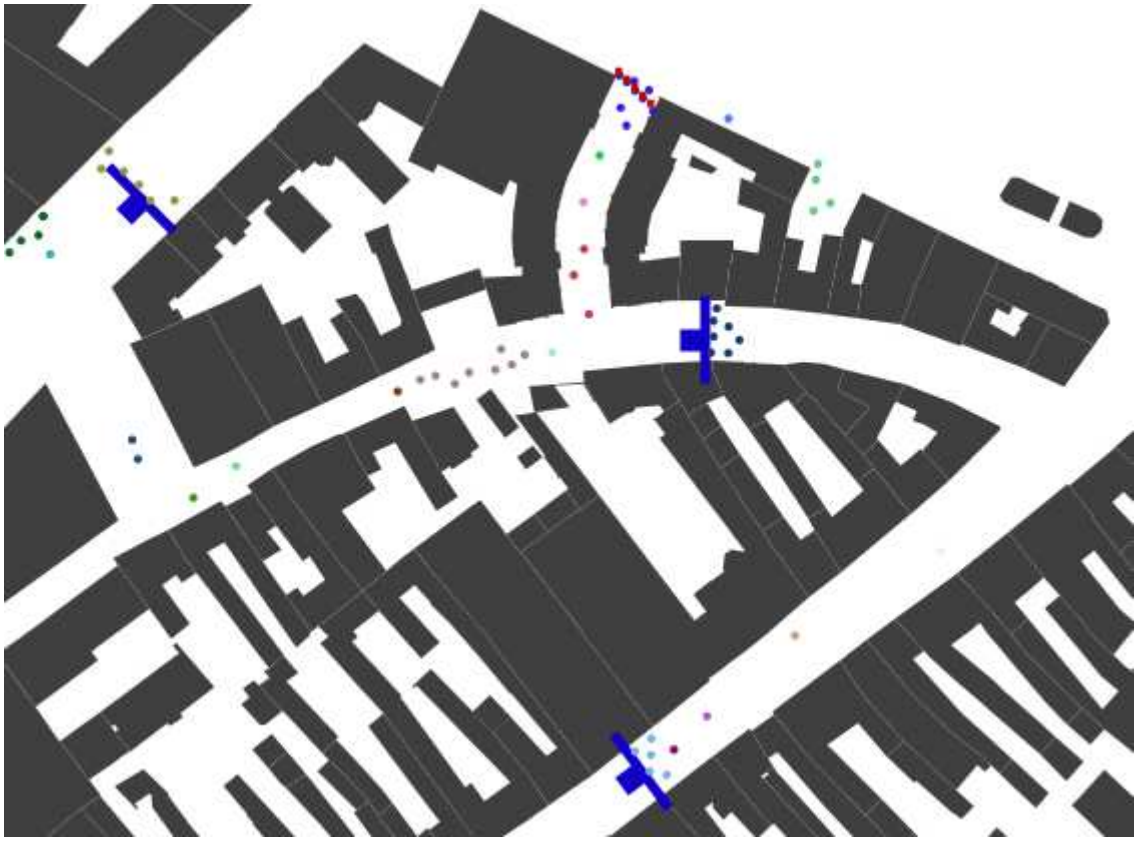
Implementácia prebiehala bezproblémovo. Trieda *Scenario.java* bola obohatená o nové scenáre, ktoré prezentujú funkčnosť našej simulácie.
Implementovanie scenára spočíva najmä v nastavení počiatočných a cieľových pozícií pre jednotlivých agentov. Pritom je potrebné si dávať pozor, aby nebola niektorá pozícia nastavená v rámci prekážky.

9.1.5 Testovanie

Počas testovania sa ukázali niektoré neduhy, ktoré sprevádzajú našu simuláciu. Prvým problémom bol prechod agentov cez Boženu. Druhým problémom bolo podliezanie policajnej zábrany. A tretím problémom bolo zasekávanie simulácie pri plánovaní trasy.
Prvý a tretí problém sa podarilo úspešne odstrániť, druhý zatiaľ pretrváva, hoci kolega, už má nápad, ako by to vedel opraviť.
Obrázky z testovania sú zobrazené nižšie (Obr. 9:2, Obr. 9:3).



Obr. 9:2 Obkľúčenie Prezidentského paláca.



Obr. 9:3 Nastavenie komplexného scenára.

9.2 Vytvorenie konfigurovateľného scenára

9.2.1 Úloha

Po vytvorení základného scenára sme došli na to, že by bolo dobré mať implementovanú akúsi všeobecnú šablónu scenára, ktorý by sa inicializoval podľa konfiguračného súboru.
To umožní vytváranie bohatých scenárov bez nutnosti napísať riadok zdrojového kódu a zároveň to umožní vymeniť scenáre v run-time.

9.2.2 Analýza

V prvom rade bolo potrebné analyzovať, aké všetky parametre sú nevyhnutné na konfiguráciu scenára. Cieľom je, aby konfigurácia pomocou textového súboru bola rovnako silná, ako konfigurácia scenára priamo v kóde. Takže nestačí zahrnúť iba základné parametre scenára, ale je potrebné uvažovať všetky dostupné možnosti nastavenia. A to pre každú skupinu agentov.

9.2.2.1 Parametre demonštrantov

- Počet skupín demonštrantov
- Pre každú skupinu:
  - Počet agentov v skupine
  - Počiatočná pozícia agentov v skupine
  - Cieľová pozícia agentov v skupine

9.2.2.2 Parametre policajtov

- Počet skupín policajtov
- Pre každú skupinu:
  - Počet agentov v skupine
  - Počiatočná pozícia prvého agenta v skupine
  - Posunutie ďalšieho agenta v počiatočnej línii
  - Cieľová pozícia prvého agenta v skupine
  - Posunutie ďalšieho agenta v cieľovej línii

9.2.2.3 Parametre Boženy

- Počet Božien
- Pre každú Boženu:
  - Počiatočná pozícia Boženy
  - Cieľová pozícia Boženy

9.2.3 Návrh

Parametre scenára sa budú načítavať z konfiguračného súboru. V konfiguračnom súbore budú uložené vo vzťahu kľúč-hodnota, čo nám umožní abstrahovať od striktných požiadaviek na formát súboru.

Vzhľadom na množstvo kódu, ktoré je nevyhnutné napísať, bude vhodné, ak sa vytvorí samostatná trieda pre konfigurovateľný scenár. Táto trieda bude rozširovať triedu všeobecného scenára.

9.2.4 Implementácia

Počas implementácie bola založená trieda *ScenarioConfigurable.java*. Obsahuje metódy pre čítane konfiguračného súboru a samotnú šablónu scenára.
Ukážka konfiguračného súboru je nižšie.

```
# Pocet skupin demonstrantov
numOfGroupsOfRioters=2

# Skupina demonstrantov 1
numOfRioters=50
startPosX=400
startPosY=300
destPosX=230
destPosY=620

# Skupina demonstrantov 2
numOfRioters=50
startPosX=200
startPosY=300
destPosX=230
destPosY=620

# Pocet skupin policajtov
numOfGroupsOfCops=1

# Skupina policajtov 1
numOfCops=20
startPosFirstX=280
startPosFirstY=507
startOffsetX=2
startOffsetY=0
destPosFirstX=280
destPosFirstY=507
destOffsetX=2
destOffsetY=0


# Pocet Bozien
numOfBozenas=1

# Bozena 1
startPosX=167
startPosY=450
destPosX=200
destPosY=300
```

9.2.5 Testovanie

Vo fáze testovania bolo zistené, že parsovanie konfiguračného súboru zlyhá, ak bude v komentári uvedený práve jeden znak rovnosti „=“. Tento nedostatok bol však rýchlo odstránený a načítanie konfigurácie funguje bezproblémovo.

9.3 Návrh a implementácia policajných línií a ich postupu

9.3.1 Úloha

Rozmiestnenie policajtov, ich pohyb a rozširovanie rozostupov medzi nimi v prípade rozširujúcej sa ulice. Vytváranie policajných línií v závislosti od šírky ulice.

9.3.2 Implementácia

Pohyb poriadkových zložiek v rozširujúcej sa ulici bol implementovaný už v predchádzajúcom šprinte. V tomto šprinte sme doplnili funkcionalitu vytvárania policajných línií. Policajné zložky sú schopné vytvárať línie podľa šírky ulice. Ak sa ulica, ktorou idú, zužuje, tak sú schopní sa rozdeliť a vytvoriť z jednej línie policajtov viacero línií bez zmeny smeru postupu. Zároveň sú príslušníci policajných línií schopní preskupovať sa medzi líniami. Funkcionalita bola implementovaná tak, že každá inštancia triedy ***AgentPolicajt*** má atribút ***lineNumber***, v ktorom si uchováva líniu, v ktorej sa nachádza v rámci jedného policajného šíku. Tento atribút mení svoju hodnotu podľa toho, ako tesne sa policajti pri sebe nachádzajú. Ak sa hustota policajtov pri pochode ulicou zväčšuje, tak sa policajt rozhodne prejsť do ďalšej línie. To je implementované ako zmena hodnoty atribútu ***lineNumber***.
Policajti z jedného policajného šíku sú v každom kroku simulácie delení podľa hodnoty atribútu do línií, v ktorých sa budú nachádzať. Toto delenie prebieha v cykle, ktorý iteruje cez všetkých policajtov v šíku a rozdelí ich, pričom sú ukladaní do dátovej štruktúry ***Map<Integer, ArrayList<AgentPolicajt>>***. Prvý parameter dátovej štruktúry je číslo línie policajtov a druhý je zoznam policajtov, ktorí sa v danej línii nachádzajú. Dátová štruktúra je nápomocná preto, lebo vieme pristupovať ku všetkým policajtom v šíku ako aj v línii bez problémov.
Pre jednotlivé policajné línie je implementované nie len samotná zmena rozostupov v rámci línie ale aj samostatné formovanie do línie policajtov.

9.3.3 Testovanie

Počas testovania sa zistili určité nedostatky v pohybe a formovaní policajných línií, ktoré sú spôsobené nastavením parametrov RVO. Policajti sa v líniách snažia navzájom vyhýbať a zároveň sú nútení vytvárať líniu. Línia je síce dostatočne pevná, aby zabránila prechodu demonštrantom, ale vizuálny efekt je sprevádzaný neustálym hmýrením policajtov počas pohybu. Preto počas nasledujúcej stabilizácie produktu upravíme parametre RVO tak, aby policajné zložky pri formovaní a pohybe vizuálne lepšie pôsobili.

10 Deviaty šprint

V nižšie uvedenej tabuľke (Tabuľka 10.1) je uvedený zoznam úloh, ktoré sa počas deviateho šprintu, ktorý sme interne pomenovali Námestie nebeského pokoja, vyriešili a osoby, ktoré sa danými úlohami primárne zaoberali. Zoznam úloh a k nim prislúchajúcich riešiteľov.

Tabuľka 10.1 Zoznam úloh a k nim prislúchajúcich riešiteľov.

Názov úlohy	Zodpovedná osoba
Konfigurácia vlastného scenára	Filip Pakan

10.1 Konfigurácia vlastného scenára

10.1.1 Úloha

Cieľom tejto úlohy je napísať návod na konfiguráciu vlastného scenára v prostredí našej simulácie. Taktiež v tejto časti priblížim postup, ako pridať vlastný parameter do scenára.

10.1.2 Vytvorenie konfiguračného súboru scenára

Vytvorenie scenára prebieha vytvorením vlastného konfiguračného súboru pre daný scenár (napríklad ***Pochod.properties***) a jeho umiestnením do projektového adresára ***simTeam/scenarios***.

10.1.3 Nastavenie parametrov scenára

Agenti (demonštranti, policajti, Boženy) sa pridávajú do scenára vo forme skupín. Skupina je abstrakcia zoskupujúca viacerých agentov, ktorí majú spoločné parametre. Do scenára je možné pridať viacero skupín agentov. Pred pridaním každej skupiny je potrebné uviesť počet skupín a jej typ, napríklad:

```
numOfGroupsOfRioters=2
```

Parametre sú uchovávané vo formáte kľúč-hodnota. Nastavovanie parametrov prebieha jednoduchým priradením hodnoty danému kľúču. Vždy je potrebné uviesť všetky parametre skupiny, no nie je potrebné dodržať poradie.
Napríklad:

```
numOfRioters=50
```

10.1.3.1 Parametre demonštrantov

Pre jednu skupinu demonštrantov sú definované nasledujúce vlastnosti:

- Počet agentov v skupine (***numOfRioters***)

- Počiatočná pozícia skupiny agentov (***startPosX***, ***startPosY***)
- Cieľová pozícia skupiny agentov (***destPosX***, ***destPosY***)

Všetci agenti v danej skupine majú rovnakú štartovaciu aj cieľovú pozíciu a pre každú skupinu je definovaný počet agentov v nej. Ak je požiadavka na pridanie demonštrantov do scenára s rôznou cieľovou pozíciu, vytvoria sa dve skupiny demonštrantov a každej skupine sa nastaví iný cieľ.
Príklad pre vytvorenie dvoch skupín demonštrantov s rôznou počiatočnou pozíciou:

```
# Pocet skupin demonstrantov
numOfGroupsOfRioters=2

# Skupina demonstrantov 1
numOfRioters=50
startPosX=400
startPosY=300
destPosX=230
destPosY=620

# Skupina demonstrantov 2
numOfRioters=50
startPosX=200
startPosY=300
destPosX=230
destPosY=620
```

10.1.3.2 Parametre policajtov

Pre jednu skupinu policajtov sú definované nasledujúce vlastnosti:

- Počet agentov v skupine (***numOfCops***)
- Počiatočná pozícia prvého policajta v skupine (***startPosFirstX***, ***startPosFirstY***)
- Posunutie každého ďalšieho policajta od toho predošlého v počiatočnej línii (***startOffsetX***, ***startOffsetY***)
- Cieľová pozícia prvého policajta v skupine (***destPosFirstX***, ***destPosFirstY***)
- Posunutie každého ďalšieho policajta od toho predošlého v cieľovej línii (***destOffsetX***, ***destOffsetY***)

Príklad pre vytvorenie jednej skupiny policajtov rozostavených do horizontálnej línie:

```
# Pocet skupin policajtov
numOfGroupsOfCops=1

# Skupina policajtov 1
numOfCops=20
startPosFirstX=280
startPosFirstY=507
startOffsetX=2
startOffsetY=0
destPosFirstX=280
destPosFirstY=507
destOffsetX=2
destOffsetY=0
```

Ak by sme chceli rozostaviť policajtov do vertikálnej línie, upravili by sme posunutie každého ďalšieho policajta od toho predošlého tak, aby sa nemenila súradnica X, ale Y, pretože pri vertikálnej línii policajtov majú policajti rôznu Y súradnicu. Napríklad:

```
startOffsetX=0
startOffsetY=2
destOffsetX=0
destOffsetY=2
```

Nastavením rovnakého posunutia v začiatočnej aj cieľovej línii sa docieli, že po príchode do cieľa sa rozostúpia rovnako. Vo vyššie uvedenom príklade však majú aj štartovaciu a cieľovú pozíciu nastavenú rovnako, takže ostanú na mieste.

10.1.3.3 Parametre Boženy

Pre jednu Boženu sú definované nasledujúce vlastnosti:

- Počiatočná pozícia Boženy (***startPosX***, ***startPosY***)
- Cieľová pozícia Boženy (***destPosX***, ***destPosY***)

Ich význam je rovnaký ako v predošlom prípade. Do scenára je možné pridať viacero Božien.
Príklad pre vytvorenie jednej Boženy:

```
numOfBozenas=1

# Bozena 1
startPosX=167
startPosY=450
destPosX=200
destPosY=300
```

V prípade, že nemáte záujem pridať do scenára všetky zložky demonštrácie, tak jednoducho môžete konfiguráciu danej zložky (napríklad policajtov) vymazať. Prípadne môžete nastaviť počet skupín danej zložky na 0 (napríklad numOfGroupsOfCops=0) a konfiguráciu skupín v súbore ponechať.

10.1.4 Pridanie vlastných parametrov

1. Pre pridanie vlastného parametra skupine agentov je potrebné pridať daný parameter do triedy reprezentujúcej danú skupinu (***GroupOfRioters***, ***GroupOfCops***, ***GroupOfBozena***). Vzhľadom na to, že demonštranti, policajti aj Boženy sú agenti, je možné využiť hierarchiu tried a parameter, ktorý má platiť pre všetkých agentov, pridať do abstraktnej triedy ***GroupOfAgents***.
2. Ďalej je potrebné pre vlastný parameter vymyslieť kľúč, podľa ktorého sa vyhľadá v konfiguračnom súbore. Samozrejme je potrebné nastaviť hodnotu nového parametra v konfiguračnom súbore.
3. Pri spracovaní konfiguračného súboru sa daný parameter načíta do asociatívneho poľa, odkiaľ je možné k nemu pohodlne pristúpiť podľa kľúča a nastaviť hodnotu parametra v danej triede, do ktorej bol pridaný v prvom kroku.

4. Posledný krok je rozšíriť šablónový scenár tak, aby sa nastavil nový parameter generovaným agentom.

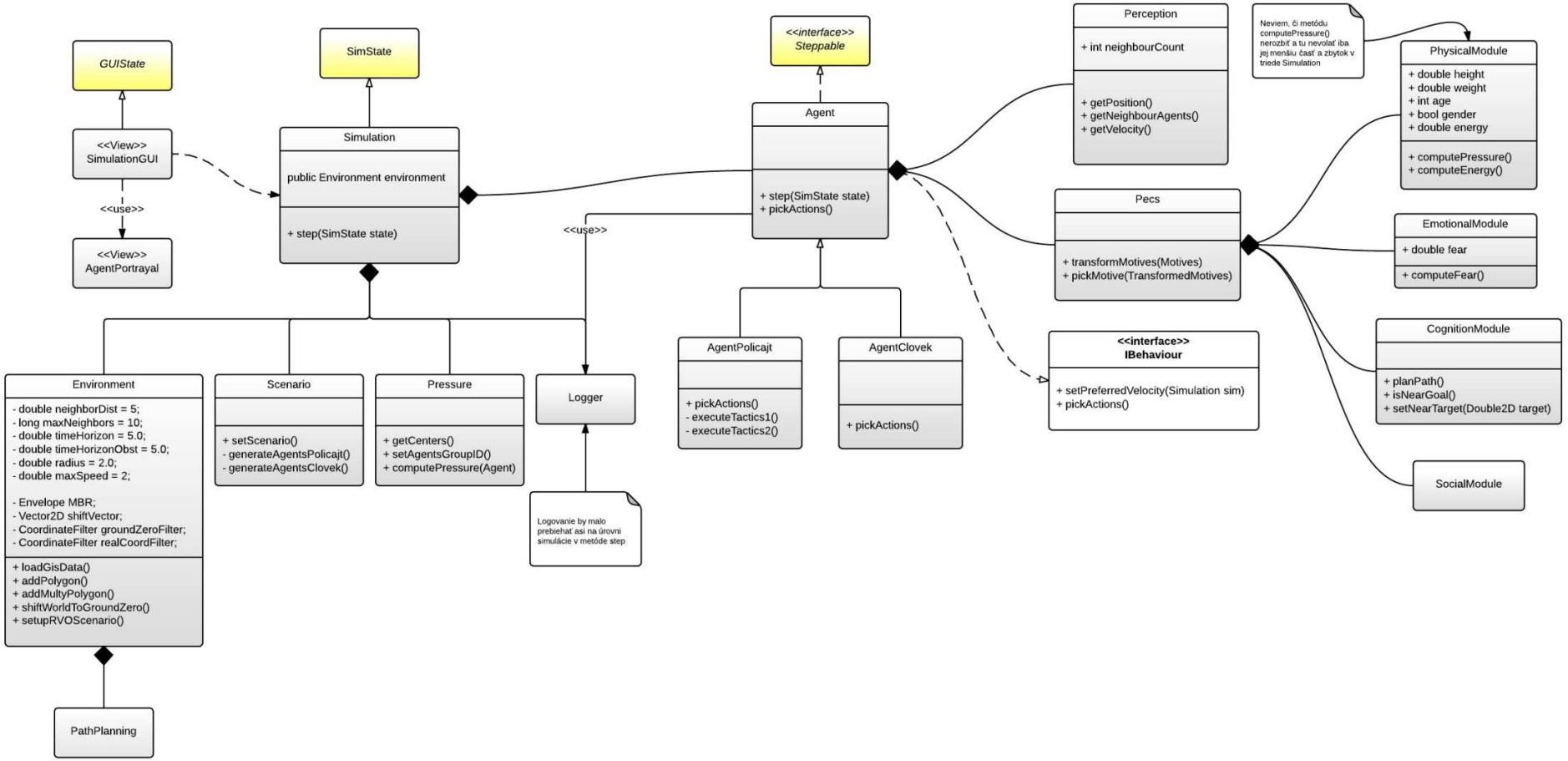
11 Opis prototypu

11.1 Úvod

V ostatných rokoch sme svedkami veľkého počtu protestných akcií nielen na Slovensku ale aj v celej Európe. Demonštrácie majú väčšinou rovnaký cieľ – dosiahnuť spoločenské uznanie názoru demonštrujúceho davu. Sprievodným javom hlasného presadzovania si názoru bývajú aj demolácie verejných budov alebo zranení obyvatelia. Na udržanie demonštrujúceho davu pod kontrolou za dosiahnutia minimalizácie neželaných efektov protestných akcií sú zodpovedné poriadkové zložky. Na zlepšenie predvídania správania demonštrantov a predpovedania reakcií na obranné akcie poriadkových zložiek sa demonštrácie simulujú. Na simulácii je možné nastaviť očakávané správanie davu a zistiť, ako by sa dav počas reálnej demonštrácie správal. Na základe takýchto predikcií je následne možné zvoliť potrebné opatrenia na minimalizáciu škôd spôsobených protestujúcimi. Simulácie demonštrácií preto môžu ušetriť ľudské životy a financie, preto je nevyhnutné sa nimi zaoberať.

11.2 Architektúra prototypu

Pri tvorbe architektúry projektu sme sa snažili vytvoriť architektúru, ktorá umožňuje bude jednoduchá ale zároveň dostatočne robustná na flexibilné zmeny v implementácii správania sa agentov. Architektúra je znázornená ako diagram tried nižšie (Obr. 11:1). Trieda *Simulation* a *SimulationGUI* zabezpečujú zobrazenie a vykonávanie krokov simulácie. Scenár simulácie je plne konfigurovateľný, čo dokazuje aj existencia samostatnej triedy *Scenario*. Agenti v prototype majú svoje špecifické správanie ovplyvnené emóciami a kognitívnymi schopnosťami. Na správanie agentov vplýva, tak ako v reálnom živote, aj ich fyzická zdatnosť a sociálne pomery. Správanie agentov je implementované v 4 moduloch, ktoré korešpondujú s modulmi referenčného modelu správania *PECS*, ktorý je v simulácii použitý na pokrytie znakov správania agentov.



Obr. 11:1 Architektúra prototypu.

11.3 Agenti

Agent predstavuje model človeka účastného na demonštrácii. V prototype simulácie demonštrácie rozlišujeme agentov na dve skupiny:

- demonštranti,
- príslušníci poriadkových zložiek.

Agent ako model človeka je reprezentovaný niekoľkými črtami ľudského správania. V opačnom prípade by sme nesimulovali realitu. Črty, ktoré spomíname, predstavujú súbor psychologických motívov a na ne naviazaných akcií, ktoré dodávajú agentom prvky ľudského správania sa. Psychologické motívy sa líšia medzi demonštranti a príslušníkmi poriadkových zložiek, ale každý jeden agent v našej simulácií sa správa podľa psychologického modelu príznačného pre konkrétnych jedincov.

11.3.1 Demonštranti

11.3.1.1 Sumár psychologických motívov a akcií vyplývajúcich z nich

Simulácia demonštrácie, ktorú sme realizovali stavia na aspektoch a moduloch, ktoré popisuje psychologický model správania PECS. Využili sme všetky jeho časti a správania demonštrantov možno sumarizovať nasledovne.

Agent sa snaží dostať na miesto určenia, čo predstavuje určitý koridor kadiaľ má demonštrácia viesť. Popri tom na neho pôsobia silnejšie či slabšie psychologické vplyvy, ktoré sú založené na vnímaní svojho okolia. Demonštrant reaguje na prítomnosť tzv. **buričov**, ktorí v dave rozosievajú agresivitu, a tým znižujú u daného demonštranta citlivosť resp. mieru strachu pred **vytláčaním zariadením Božena**. Demonštranti prirodzene tvoria tzv. skupinky, ktoré majú určité ciele a v rámci tejto skupiny sa šíri okrem agresivity aj emócia strachu.

Demonštranta ďalej charakterizujú ďalšie vlastnosti, napr. tep srdca, ktorý samozrejme ovplyvňuje tak strach ako aj miera tlaku na osobu. Rýchlosť človeka pri pohybe ulicami ovplyvňuje aj vydanú energiu, ktorá sa pomocou sofistikovaných metód, ktorých základ pochádza z biologických štúdii stotožňuje s vydanými kalóriami.

Pre objasnenie ako sa vonkajšie podnety premietajú do motívov na základe ktorých agent vykonáva konkrétne aktivity je nutné v skratke rozobrať princíp fungovania modelu PECS. Ten tvorí niekoľko vrstiev, pričom informácie tečú nielen zhora dolu, ale aj v rámci vrstvy.

- *1. vrstva* – vnímanie (pozícia agenta v prostredí, agenti okolo mňa atď.)
- *2. vrstva* – telo agenta (moduly – emocionálny, fyzický, sociálny, kognitívny)
- *3. vrstva* - generátor motívov (generuje z modulov a ich stavových premenných tzv. motívy)
- *4. vrstva* – vykonávanie aktivít (vykonáva samotné pozorovateľné aktivity napr. zmena pohybu)

Motívy tvoria podklad pre vykonanie aktivity. ***Generátor motívov*** na základe rôznych stavových premenných obsiahnutých v tele agenta vygeneruje motívy, ktoré taktiež normalizuje, aby bolo možné porovnávať ich závažnosť a povedať, ktorý aktuálne prevažuje a ktoré aktivity ho budú uspokojovať. Motívy sú často transformované

komplexnými matematickými funkciami (typicky exponenciálne/logaritmické krivky), aby odrážali závažnosť motívov, napr. vzdialenosť od vytláčacieho stroja Božena. V prípade, že je agent ďaleko, tak je tento motív zanedbateľný avšak s rastúcou blízkosťou rastie, nie však lineárne.

V našom projekte sme identifikovali nasledovné motívy a aktivity, ktoré z nich vyplývajú:

a) ***Motív (základný)***
Tento motív charakterizuje základnú potrebu agenta dostať sa do určeného cieľa

b) ***Motív (strach)***
Tento motív sa dostáva k slovu v momente, ak miera strachu agenta je relatívne vysoká a porazí ostatné motívy. Aktivity, ktoré vyplývajú z tohto motívu sú nasledovné:
- *Vysoká miera strachu* – **oscilácia cieľa** (chvíľková dezorientovanosť a panika)
- *Nízka miera* – ***zrýchlenie pohybu*** (simulácia vzbudeného stavu demonštranta)

c) ***Motív (Božena)***
Tento motív začína prevažovať, ak je demonštrant relatívne blízko stroja Božena. V takom prípade, sa chce od nej dostať preč, resp. do bezpečnejšej vzdialenosti. Ako náhle ju dosiahne prehodnotí novú pozíciu a eventuálne pokračuje v ofenzíve proti tomuto stroju.
- ***choď do bezpečnej vzdialenosti od stroja Božena***

d) ***Motív (tlak)***
V prípade, že demonštranti tlačia na stroj Božena resp. na policajné zátarasy a tento tlak sa stane pre daného agenta neznesiteľný, tak na túto situáciu reaguje.
- ***snaha dostať sa preč od okolitých demonštrantov – únik z davu***

e) ***Motív (energia)***
Agent (demonštrant) disponuje určitou zásobou energie, čo predstavuje analógiu fyzickej kondície u človeka. Pri pohybe sa táto energia míňa a pri spomalení naopak obnovuje. Rýchlosť výdaja energia ako aj obnova sú prepojené s vekom demonštranta.
- *Príliš nízka* – ***spomaľ výraznejšie***
- *Relatívne nízka* – ***spomaľ iba trochu***

11.3.1.2 Plánovanie trasy

Na zvýšenie reálnosti simulácie sme do prototypu implementovali aj možnosť hľadania trasy do cieľového bodu. V reálnych demonštráciách sa stáva, že policajné zložky zatarasia cestu k budove, ku ktorej chcú demonštranti ísť. Vtedy sa snažia policajné zátarasy obísť.

Plánovanie trasy predstavuje pre agentov pomocný nástroj k tomu, aby sa úspešne dostali do svojho cieľa. Cesta agenta do cieľa nie je priamočiara. Agent sa musí pri tom vyhýbať prekážkam, ktoré v simulácii reprezentujú budovy. Plánovanie trasy má za úlohu odstrániť problém uviaznutia agenta pred prekážkou.
Plánovanie trasy je v simulácii implementované pomocou modifikovaného A* algoritmu. Pri hľadaní trasy do cieľa sa zohľadňuje vzdialenosť od najbližšej budovy. Rovnako boli do algoritmu pridané určité prvky náhodnosti. To zabezpečí, že výsledná trasa nebude vždy najkratšou trasou, ako pri štandardnom A* algoritme.
Agent zapína plánovanie trasy až v momente, kedy sa prestane približovať k svojmu cieľu. Plánovanie vytvorí súbor bodov od pozície agenta až k jeho cieľu. Agent je následne pomocou týchto bodov navigovaný do cieľa.
Dynamická zmena cieľov agentov, či už na základe motívov alebo iných vonkajších vplyvov, je jednoducho uskutočniteľná vďaka možnostiam plánovania trasy.
Hlavné vlastnosti plánovania trasy sú:

a) rieši problém uviaznutia agentov
b) modifikovaný A* algoritmus
c) hľadanie cesty až do cieľa
d) hľadanie cesty od bodu, v ktorom agent uviazol
e) počas plánovania trasy agent berie do úvahy vzdialenosť od budov
f) obsahuje isté prvky náhodnosti
g) výsledná trasa tvorí súbor bodov (tzv. waypointov)

11.3.2 Policajné zložky

Policajné zložky v prototype simulácie demonštrácie v meste slúžia na udržanie demonštrujúceho davu na bezpečnom, dostatočne veľkom priestranstve za účelom ochrany verejného majetku a zdravia osôb. Policajné zložky svoju taktiku koordinujú tak, aby dosiahli požadovaný cieľ. Niekedy, keď je masa demonštrantov príliš veľká, je nevyhnutné použiť vytláčací stroj božena.
Božena je obrnené vozidlo, ktoré poskytuje možnosti na usmernenie pohybu davu v uliciach a zastavaných oblastiach. Zhromaždený dav osôb je možné nasmerovať do požadovaného smeru alebo zón. Božena predstavuje efektívny nástroj aj na zamedzenie prístupu do určitých oblastí.
Demonštrácie sú často sprevádzané presunom veľkých skupín osôb. Aj v simulácii, ktorú sme realizovali, majú agenti nastavení cieľ pohybu, do ktorého sa snažia dostať. Tento cieľ sa počas simulácie dynamicky mení, na základe vonkajších vplyvov a motívov, ktoré vplývajú na agenta. Kľúčovým prvkom simulácie je teda samotný pohyb agentov. Tento pohyb agentov je pre minimalizáciu rôznych rizík potrebné určitým spôsobom regulovať. Regulácia pohybu agentov je v simulácii realizovaná dvomi spôsobmi:

- formáciami policajtov,
- vozidlom Božena

Simulácia umožňuje pridanie ľubovoľného počtu modelov Boženy. Každý model má stanovený vlastný cieľ trasy. Konkrétna trasa do cieľa je získaná pomocou plánovania

trasy. Počas presunu Boženy do cieľa agenti reagujú na blízkosť Boženy. Blízki agenti sú odtláčaní Boženou, čím je zabezpečené usmerňovanie agentov do požadovaných zón. Model Boženy dokáže v simulácii realizovať nasledovné akcie:

a) odtláčať a usmerňovať dav agentov
b) rozširovať a zmenšovať vytláčacie rameno
c) otáčať sa na mieste
d) otáčať sa podľa smeru cesty
e) vyhýbať sa budovám
f) nájsť trasu do cieľa

Model Boženy má rôzne konfigurovateľné parametre, čo zabezpečí jeho použitie v rozličných scenároch. Konfigurovať je možné:

a) veľkosť modelu
b) rýchlosť
c) cieľ
d) rýchlosť otáčania
e) rýchlosť a veľkosť rozšírenia ramien

11.4 Záver

Demonštrácie sú v ostatných rokoch veľmi rozšírený prostriedok na vyjadrenie vlastného názoru. Nepriaznivým znakom demonštrácií však bývajú násilnosti a škody na majetku občanov. Takýmto neželaným vplyvom treba predchádzať, a preto existujú simulácie demonštrácií, ktoré aspoň čiastočne pomáhajú pri predpovedaní vývoja demonštrácie, čo môže jej nepriaznivé sprievodné znaky eliminovať.

Prototyp simulácie demonštrácie v meste bol modelovaný s cieľom napodobniť reálnu demonštráciu. Preto majú agenti psychologické motívy, ktoré ovplyvňujú ich správanie a konanie, vedia plánovať trasu do svojho cieľa. Príslušníci policajných zložiek zabraňujú prechodu demonštrantom na miesta, na ktorých by mohlo dôjsť k stretu s civilným obyvateľstvom nezapojeným do demonštrácie. V prototype sú použité aj moderné spôsoby a nástroje používané pri výtržnostiach a demonštráciách po celom svete.

Prototyp simulácie demonštrácie však neobsahuje všetky motívy správania osôb, ktoré ľudia majú a nezvažuje špeciálne taktiky policajných zložiek. Tie sa od demonštrácie k demonštrácii môžu líšiť. Aj napriek tomu si myslíme, že prototyp vykazuje všetky znaky simulácie a pripravený na ďalšie štúdium a rozpracovanie.

12 Zhrnutie

Projekt simulácie demonštrácie v meste považujeme za veľmi zaujímavý a podnetný. Vďaka nemu sme mali možnosť poodkryť rúško tajomstva nad ľudskou psychikou. Práca na projekte nás nútila bližšie preštudovať články a niektoré publikácie zaoberajúce sa psychológiou človeka. Mali sme možnosť zistiť, aké podnety ovplyvňujú správanie jednotlivca v dave, vo vypätých situáciách, pod psychickým tlakom. Spoznávali sme tak samých seba, pričom sme obohatili svoje vedomosti o poznatky, ktoré nám môžu byť v budúcnosti užitočné.

Projekt obohatil náš profesijný život o skúsenosť práce v tíme. Mnohí z nás takúto skúsenosť zatiaľ nemali. A pritom je tím neodmysliteľnou súčasťou prác na vývoji softvéru. Softvér nerobia jednotlivci ale tímy odborníkov. Projekt nám poskytol teoretické a praktické znalosti a skúsenosti práce v tíme. Ostatné dva semestre sme mali možnosť rozvíjať svoje schopnosti komunikácie s kolegami participujúcimi na vývoji toho istého softvérového produktu, deľby práce za účelom dosiahnutia synergického efektu. Získané znalosti uplatníme v našom ďalšom profesijnom raste.

Po technickej stránke sme sa niektorí opätovne po niekoľkých rokoch vrátili k programovaniu v programovacom jazyku Java. Zlepšenie schopností programovania v tomto jazyku nám môže zvýšiť pomôcť v budúcnosti, lebo rozšírenosť jazyka v praxi je nespochybniteľná. Oboznámili sme sa s knižnicami na modelovanie a simulovanie multiagentových systémov MASON a na koordináciu vzájomného pohybu agentov v multiagentovom prostredí RVO2.

Aj napriek plnému nasadeniu počas dvoch semestrov sú v projekte miesta, v ktorých vidíme možnosť práce v budúcnosti. Na samom začiatku sme sa domnievali, že tak dlhý čas na vývoj simulácie demonštrácie ani nebude potrebný. Až neskôr, po získaní mnohých teoretických vedomostí a ich praktickom použití, sme zistili, že rozsah prác prekračuje obdobie, počas ktorého sme na vývoji produktu pracovali. Preto sme si vedomí nesplnených cieľov.

Projekt sme od začiatku uchopili ako demonštráciu davu, ktorý nie je príliš agresívny a nemá sklony k prehnanému násiliu. Tu je príležitosť doplniť projekt aj o prvky násilnosti, tak časté na mnohých demonštráciách na Slovensku ale aj v zahraničí. V súčasnom stave projektu tieto prvky nie sú zakomponované a nedajú sa zmenou nijakých parametrov pridať. Nemyslíme si však, že projekt nie je konfigurovateľný a dostatočne flexibilný. Práve naopak. Simulácia davu používateľovi umožňuje nastavovať veľké množstvo parametrov na ovplyvňovanie správania demonštrantov ale aj akcií policajných zložiek. Zo všetkých spomeniem napríklad počet susedov, ktorých správanie pôsobí na jednotlivca alebo maximálnu hustotu osôb na danom území. Zmena parametrov simulácie umožňuje používateľom prispôsobiť model reality podľa potreby. Agresia, ako sme už uvádzali, nie je jedným z prvkov, ktorý je konfigurovateľný a v súčasnosti je agresivita vo všeobecnosti nízka. Z toho vyplýva nepoužívanie prostriedkov schopných telesne ublížiť iným demonštrantom alebo príslušníkom poriadkových zložiek.

Pri ďalších prácach vidíme aj možnosť hlbšej analýzy psychológie človeka a davu. Počas analýz, ktoré sme vykonali, sme zistili, že oblasť štúdie ľudskej psychiky nie je uzavretá, a preto nemôžeme od simulácie očakávať integráciu všetkých čŕt ľudskej psychiky. V budúcnosti by bolo možné chýbajúce črty hlbšie rozpracovať a integrovať.
Z vyššie uvedeného len v krátkosti zhrniem, že projekt simulácie demonštrácie v meste považujeme za veľmi zaujímavý a opodstatnený. Jeho opodstatnenosť vidíme najmä v potenciálnom praktickom použití simulácie. V súčasnom stave, kedy má projekt mapový podklad zhodný s mapou mesta Bratislava, vyvstáva otázka jeho reálneho použitia ešte viac. Policajné zložky Slovenskej republiky by boli schopné predpovedať priebeh vopred ohlásenej demonštrácie, čo by pomohlo pri rozhodovaní kde rozostaviť policajné zložky v meste a aké obranné prostriedky proti demonštrantom použiť. Takáto pomoc policajnému zboru zlepší zvládanie masy demonštrantov príslušníkmi policajného zboru. Lepšie zvládanie môže skrátiť čas demonštrácie ako aj pomôcť udržať demonštrujúci dav na dostatočne veľkom priestranstve bez väčšieho zásahu do každodenných povinností nedemonštrujúcim obyvateľom. V konečnom dôsledku by zlepšenie predvídania akcií demonštrantov a zrýchlenie odpovede na akcie demonštrantov mohli znížiť počet ranených a aj šetriť finančné prostriedky vynaložené na udržanie demonštrujúcej masy ľudí pod kontrolou. Myslíme si však, že aj napriek dôležitosti reálneho použitia simulácie demonštrácie, je projekt v súčasnom stave vo fáze, kedy by bolo vhodné prizvať odborníkov na psychológiu jedincov v stresových situáciách a odborníkov na psychológiu davov, aby sa súčasné akcie a motívy akcií demonštrantov patrične doplnili. Myslíme si však, že úlohu odborníkov môžu čiastočne zastúpiť naši nasledovníci, ktorí budú na projekte pokračovať a pokúsia sa dostatočne precízne naštudovať psychologické modely ľudí. Ak by získané poznatky naimplementovali do aktuálneho riešenia, výrazne by to posunulo celý projekt simulácie demonštrácie dopredu. Toto však vyžaduje, aby téma simulácie demonštrácie neupadla do zabudnutia a aby bol súčasný projekt znovu použitý. Domnievame sa, že by bolo škoda, ak by sa v problematike simulácie davov a demonštrácií nepokračovalo.
Prácu na projekte v tíme počas ostatných dvoch semestrov pokladáme za čas, ktorý obohatil naše vedomosti a poskytol nám priestor profesijného rastu získavaním teoretických znalostí ale aj praktických skúseností.

13 Použitá literatúra

1. Guided tour of Subversion. *NetBeans.* [Online] Oracle, 2012. http://netbeans.org/kb/docs/ide/subversion.html.
2. GeoMASON. *George MASON University.* [Online] 25. 06 2012. http://cs.gmu.edu/~eclab/projects/mason/extensions/geomason/.
3. Redmine guide. *Redmine.* [Online] 2012. http://www.redmine.org/guide.
4. The Trac User and Administration Guide. *trac integrated SCM & Project Management.* [Online] 2012. http://trac.edgewall.org/wiki/0.12/TracGuide.
5. Trello Documentation. *Trello Documentation.* [Online] 2011. https://trello.com/docs/.
6. **Atlassian.** JIRA Documentation . *JIRA Documentation* . [Online] 24. 04 2012. https://confluence.atlassian.com/display/JIRA051/JIRA+Documentation.
7. Parameter Overview. *RVO2 Library Documentation.* [Online] 2012. http://gamma.cs.unc.edu/RVO2/documentation/2.0/params.html.
8. OpenStreetMap. *OpenStreetMap.* [Online] 2012. http://www.openstreetmap.org/.
9. QGIS. *Quantum GIS.* [Online] 2012. http://www.qgis.org/.
10. .*Crowd simulation for emergency response using BDI agent based on virtual reality.* **Ameya Shendarkar, Karthik Vasudevan, Seungho Lee, Young-Jun Son.** s.l. : Simulation Conference, 2006. WSC 06. Proceedings of the Winter, IEEE, 2006. 1-4244-0501-7.
11. .*Crowd Modeling and Simulation Technologies.* **SUIPING ZHOU, DAN CHEN, WENTONG CAI, LINBO LUO, MALCOLM YOKE, HEAN LOW and FENG TIAN.** 4, New York, USA : ACM Journal, 2010, Zv. 20.
12. .*Improved Benchmarking for Steering Algorithms.* **Mubbasir Kapadia, Matthew Wang, Glenn Reinmam, Petros Faloutsos.** s.l. : Springer Berlin Heidelberg, 2011, Zv. 7060.
13. **Heppenstall, A.J., a iní.** *Agent-Based Models of Geographical Systems.* s.l. : Springer, 2012. 978-90-481-8927-4.
14. .*Robust crowd behaviors for distributed simulations.* **Eric W. Weisel, Yiannis E. Papelis, Lisa Jean Moya, Catherine E. Easterling.** Edinburg : Proceedings of the 2008 Summer Computer Simulation Conference , 2008.
15. .*The Reference Model SimPan – Agent- based Modelling of Human Behaviour in Panic Situations.* **Schneider, Bernhard.** s.l. : Computer Modeling and Simulation, 2008. UKSIM 2008, 2008.
16. .*Modelling of Human Behaviour The PECS Reference Model.* **Schmidt, Bernd.** s.l. : Proceedings 14th European Simulation Symposium, 2002.
17. .*The hierarchical behavior model for crowd simulation.* **Libo Sun, Yan Liu, Jizhou Sun, Lin Bian.** s.l. : Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry, ACM, Ney York, 2009.
18. .*Multi-agent simulation model of pedestrians crowd based on psychological theories* . **Beltaief, O., El Hadouaj, S. a Ghedira, K.** Hammamet, Tunis : Logistics (LOGISTIQUA), 2011 4th International Conference on, IEEE, 2011.

19. .*Agent-based human behavior modeling for crowd simulation.* **Linbo Luo, Suiping Zhou, Wentong Cai, Malcolm Yoke Hean Low, Feng Tian, Yongwei Wang, Xian Xiao, Dan Chen.** 3,4, Chicester : John Wiley and Sons Ltd, 2008, Zv. 9.
20. .*Human Behavior Models for Agents in Simulators and Games: Part I: Enabling Science with PMFserv.* **Barry G. Silverman, Michael Johns, Jason Cornwell, Kevin O'Brien.** 2, s.l. : Presence: Teleoperators and Virtual Environments, MIT Press Journals, 2006, Zv. 15.
21. Psychológia a kariéra. *Klub Harmony.* [Online] 2009. http://www.klubharmony.eu/sk/clanok/13/2383/davova-psychoza.
22. Psychológia masy podľa Gustave Le Bona. *Psychológia.* [Online] http://psychologia.studentske.eu/2008/09/psycholgia-masy-poda-gustave-le-bona.html.
23. Kolektívne správanie. *Psychológia.* [Online] http://psychologia.studentske.eu/2008/09/kolektvne-sprvanie_9269.html.
24. **Hajdúk, Michal.** *Fandím, teda som chuligán? Sonda do problematiky správania sa a prežívania futbalových fanúšikov na Slovensku.* Bratislava, 2009.
25. .*ESCAPES - Evacuation Simulation with Children, Authorities, Parents, Emotions, and Social comparison.* **Jason Tsai, Natalie Fridman, Emma Bowring, Matthew Brown, Shira Epstein, Gal Kaminka, Stacy Marsella, Andrew Ogden, Inbal Rika, Ankur Sheel,Matthew Taylor,Xuezhi Wang, Avishay Zilka, Milind Tambe.** s.l. : International Foundation for Autonomous Agents and Multiagent Systems, 2011.
26. Psychológia masy . *Psychológia masy* . [Online] http://referaty.atlas.sk/odborne-humanitne/psychologia/10150/?print=1.
27. .*Evoking Panic in Crowd Simulation.* **T. Mao, Q. Ye, H. Jiang, S. Xia, and Z. Wang.** s.l. : ACM, New York, 2011.
28. .*Emotional Decision Making in Large Crowds. Advances on Practical Applications of Agents and Multi-Agent Systems.* **Sharpanskykh, A., Zia, K.** s.l. : Springer Berlin Heidelberg, 2012.
29. Panic. *International Encyclopedia of the Social Sciences.* [Online] 2008. http://www.encyclopedia.com/topic/panic.aspx.
30. Complexity and the madness of crowds – lessons from disaster prevention. [Online] http://reszatonline.wordpress.com/2012/10/06/complexity-and-the-madness-of-crowds-lessons-from-disaster-prevention/.
31. **Viano, D. C., King, A. I.** Biomechanics of Chest and Abdomen Impact. *The Biomedical Engineering Handbook: Second Edition.* CRC Press LLC, 2000.
32. .*Macroscopic effects of microscopic forces between agents in crowd models.* **Colin M. Henein, Tony White.** s.l. : ELSEVIER, 2007, Zv. 373.
33. **ORACLE.** Java SE Development Kit 7 Downloads. *Java SE Development Kit 7 Downloads.* [Online] 2012. http://www.oracle.com/technetwork/java/javase/downloads/jdk7u9-downloads-1859576.html.
34. Java Client Technology Downloads. *Java Client Technology Downloads.* [Online] ORACLE, 2012.

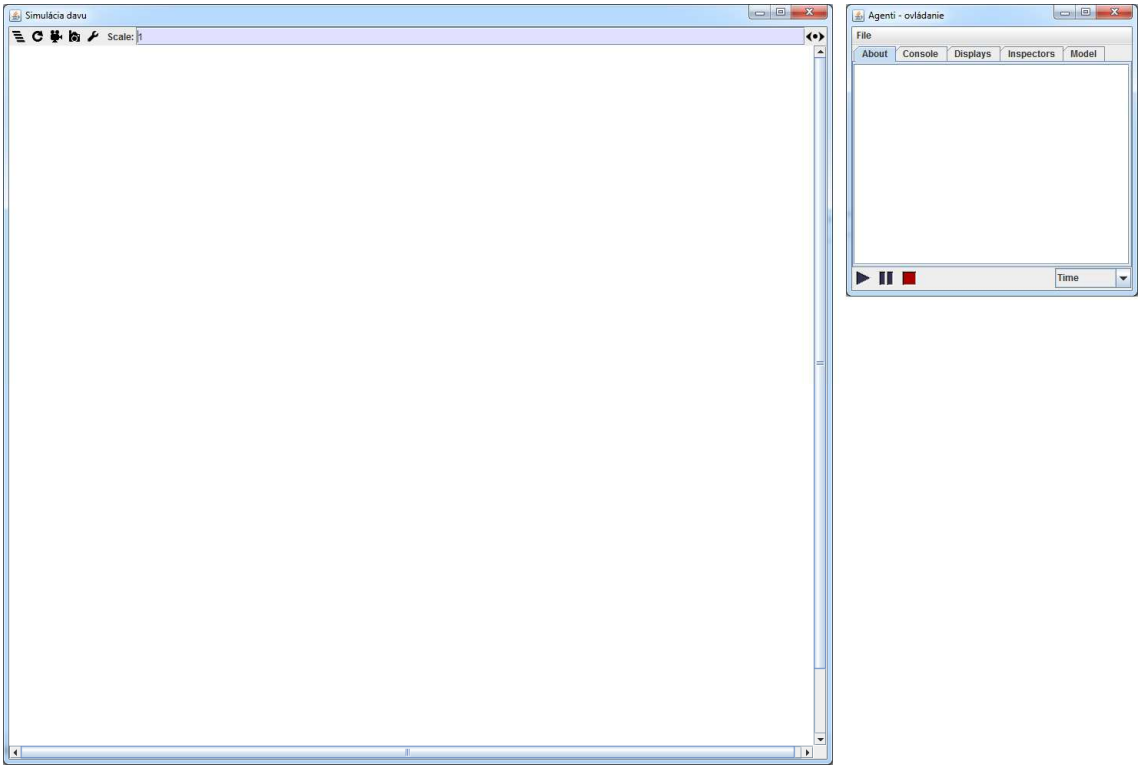
http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-java-client-419417.html#java3d-1.5.1-oth-JPR.
35. **Microsoft.** Visual C++ Redistributable for Visual Studio 2012. *Download Center.* [Online] 2012. http://www.microsoft.com/en-us/download/details.aspx?id=30679.
36. **Wheeler, David A.** Comments on Open Source Software / Free Software (OSS/FS) Software Configuration Management (SCM) / Revision-Control Systems. *Comments on Open Source Software / Free Software (OSS/FS) Software Configuration Management (SCM) / Revision-Control Systems.* [Online] 18. 05 2005. http://www.dwheeler.com/essays/scm.html.
37. **Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato.** Version Control with Subversion . *Version Control with Subversion* . [Online] 2012. 06 2004. http://flylib.com/books/en/2.139.1.2/1/. 0-596-00448-6.
38. Subversion Source Control General Information. *Subversion Source Control General Information.* [Online] 2010. http://www.clear.rice.edu/comp310/Eclipse/Subclipse/subversion.html.
39. .*Coordinated police agents to manage pedestrian flow in an urban crisis situation.* **M. J. Lyell, M. Mejia-Tellez, R. Flo, L. Haynes, R. Levy.** San Diego, USA : Proceedings of the 2009 Spring Simulation Multiconference , 2009.
40. .*Composite agents.* **H. Yeh, S. Curtis, S. Patil, J. van den Berg, D. Manocha, M. Lin.** Aire-la-Ville, Switzerland : Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '08), 2008.
41. **Linda C. Malone, T.L. Clarke, D.J. Kaup, Rex Oleson II, Mario Rosa, Jennifer Reedy.** Statistical Validation of Crowd Simulation (2 Case Studies - Proof of Concept). *Statistical Validation of Crowd Simulation.* [Online] http://www.simmbios.ist.ucf.edu/Portals/8/DHBPubs/Posters/DresdenPoster2007.pdf .
42. .*THE CAUSES AND PREVENTION OF CROWD DISASTERS.* **Fruin, John J.** Londýn : Elsevier Science Publishers, 1993.
43. Základy fyziky - elektronický materiál k videoanalýze fyzikálnych dejov. *Žilinská univerzita, Katedra fyziky.* [Online] 04. 05 2011. http://fyzika.uniza.sk/sk/zaklady/zaklady/02.pdf.
44. **Helbing, Dirk, Farkas, Illés a Vicsek, Tamás.** Simulating Dynamical Features of Escape Panic. *Nature.* 2000, Zv. 407.
45. **Fruit, Rotten.** *London scene of massive student fury.* [Video: http://www.youtube.com/watch?v=ln0B7CRNPRs] London, 2010.
46. **RussianToday.** *Nuevas protestas en Madrid en torno al Congreso de los Diputados.* [Video: http://www.youtube.com/watch?v=ix7PPiro3YM&list=PL8C30336D19544198] Madrid, 2012.
47. **Kenower a Mike.** Urban Riots. *Urban Riots.* [Online] http://l3d.cs.colorado.edu/systems/agentsheets/New-Vista/urban-riots/.
48. BOŽENA RIOT CONTROL. [Online] 2010. [Dátum: 15. 03 2013.] http://www.bozena.eu/sk/vseobecne-brc/.

49. Multiagent Simulation And the MASON Library. [Online] http://cs.gmu.edu/~eclab/projects/mason/manual.pdf.

Príloha A Používateľská príručka

A.1 Spustenie programu simulácie

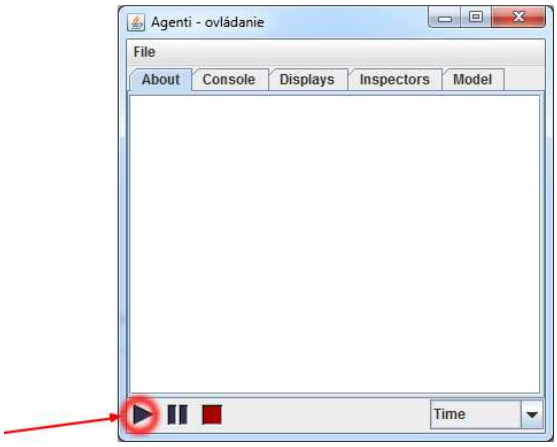
Spustenie simulácie je možné dvojklikom na súbor *start.bat* z priloženého CD nosiča. Po spustení súboru *start.bat* z priloženého CD nosiča sa spustia dve okná (Obr. A:1). Prvé z nich, označené ako *Simulácia davu* slúži na vizuálne zobrazenie simulácie demonštrantov v meste. Druhé otvorené okno označené ako *Agenti-ovládanie* slúži na spustenie simulácie a prezeranie vlastností jednotlivých agentov počas behu simulácie. Zobrazenie a prezeranie vlastností agentov počas behu simulácie rozoberáme neskôr.



Obr. A:1 Úvodný vzhľad aplikácie po spustení.

A.2 Spustenie simulácie

Simulácia sa spúšťa stlačením tlačidla v piktogramom v tvare trojuholníka v okne *Agenti-ovládanie* (Obr A:2).

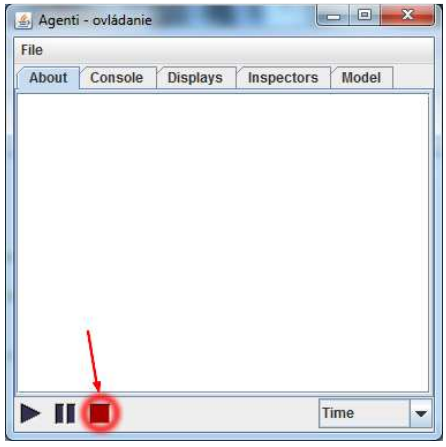


Obr A:2 Spustenie simulácie.

Po spustení simulácie je možné priebeh simulácie demonštrácie v meste sledovať v okne označenom ako *Simulácia davu*.

A.3 Zastavenie simulácie

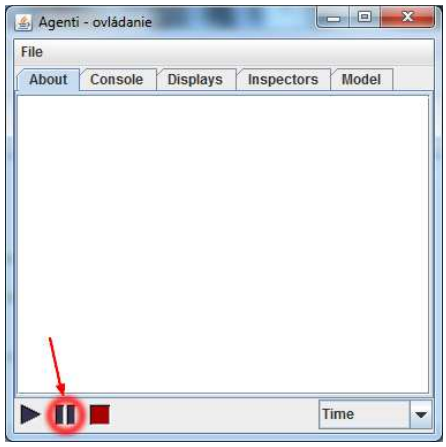
Na zastavenie simulácie slúži červené tlačidlo s piktogramom štvorca v okne *Agenti-ovládanie* (Obr. A:3). Celá simulácia sa zastaví v okamihu stlačenia tlačidla. Po následnom spustení simulácie sa celý priebeh simulácie reštartuje od začiatku. Zastaviť simuláciu je možné aj zatvorením okna *Agenti-ovládanie* kliknutím na červený krížik v pravom hornom rohu okna.



Obr. A:3 Zastavenie simulácie.

A.4 Dočasné pozastavenie simulácie

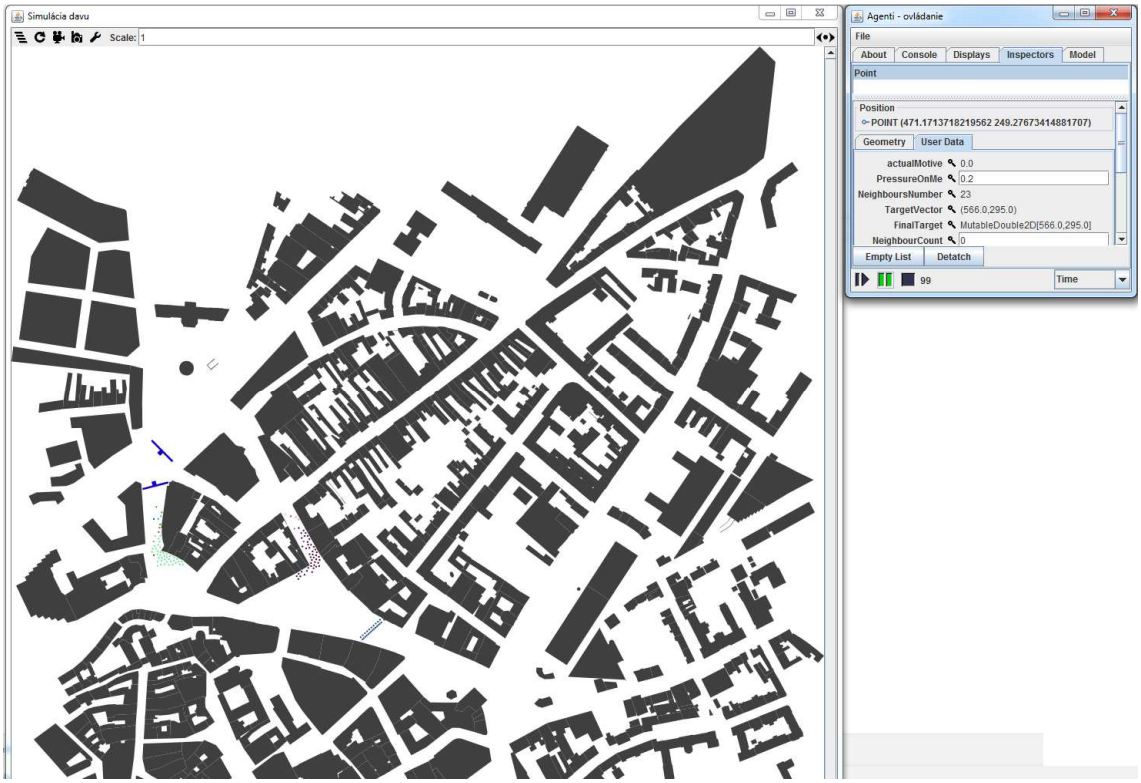
Na dočasné pozastavenie simulácie slúži tlačidlo s piktogramom dvoch rovnobežných čiar v okne *Agenti-ovládanie* (Obr. A:4). Celá simulácia sa zastaví v okamihu stlačenia tlačidla. Simulácia po následnom spustení pokračuje od miesta, v ktorom sa pozastavila.



Obr. A:4 Dočasné pozastavenie simulácie.

A.5 Sledovanie parametrov agentov simulácie

Simulácia demonštrácie je multiagentový systém. Každý agent reprezentujúci policajta alebo demonštranta má svoje charakteristické vlastnosti. Okrem nich prislúchajú ku každému agentovi aj atribúty vyplývajúce z priebehu simulácie, napríklad tlak na agenta, pozícia agenta. Tieto a mnohé ďalšie charakteristické prvky príznačné pre agenta sa dajú sledovať počas priebehu simulácie. Sú dostupné v okne *Agenti-ovládanie* v záložke *Inspectors*. Počas behu simulácie je potrebné kliknúť myšou na ľubovoľného agenta, ktorého parametre sa zobrazia v záložke *Inspectors*. (Obr. A:5).

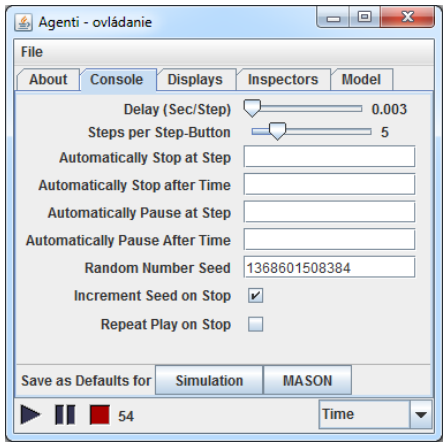


Obr. A:5 Sledovanie parametrov agentov počas simulácie.

A.6 Nastavenie parametrov simulácie

V záložke *Console* v okne *Agenti-ovládanie* je možné nastavovať parametre simulácie (Obr. A:6). V tejto záložke sa dá nastaviť dĺžku kroku simulácie a automatické

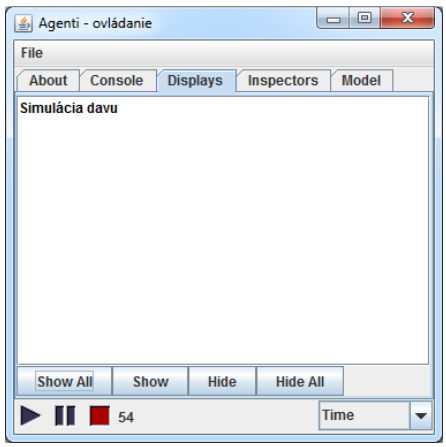
pozastavenie alebo úplné zastavenie simulácie po niekoľkých krokoch simulácie. Kroky simulácie sú viditeľné v okne *Agenti-ovládanie* vedľa tlačidla na zastavenie simulácie.



Obr. A:6 Nastavovanie parametrov simulácie.

A.7 Zobrazenie vizuálneho priebehu simulácie

V záložke *Displays* v okne *Agenti-ovládanie* je možné riadiť zobrazenie okna *Simulácia davu* (Obr. A:7). V záložke je možné zobraziť okno *Simulácia davu* stlačením tlačidla *Show All*. Skryť okno s vizuálnou ukážkou simulácie je možné stlačením tlačidla *Hide All*. Okno Simulácia davu je možné zo záložky *Displays* opätovne otvoriť aj v prípade, že bolo okno *Simulácia davu* zatvorené červeným krížikom v pravom hornom rohu okna symbolizujúci zatvorenie okna. Skrytie a zobrazenie okna s vizualizáciou simulácie nemá vplyv na beh simulácie. Simulácia prebieha až do jej zastavenia alebo pozastavenia (A.3 Zastavenie simulácie, A.4 Dočasné pozastavenie simulácie).



Obr. A:7 Ovládanie zobrazenia simulácie.

Ostatné záložky okna *Agenti-ovládanie*, *About* a *Model* neposkytujú žiadnu pridanú funkcionalitu ani možnosť ovládania simulácie používateľom, preto ich nebudeme bližšie popisovať.

Príloha B Inštalačná príručka

Inštalačná príručka slúži ako návod na inštaláciu produktu za účelom jeho ďalšieho používania.
Aplikácia funguje pod operačným systémom Windows 7. Odporúčaná je 64 bitová verzia tohto operačného systému. Minimálne požiadavky na systémové zdroje pre samotnú aplikáciu nie sú stanovené. Aplikácia však vyžaduje nainštalovanie softvérov tretích strán:

- JDK vo verzii x86
- Java 3D
- Visual C++ 2012 Redistributable vo verzii x86

Inštalácia JDK
JDK (Java Development Kit) je implementácia platformy Java SE. Táto platforma je nevyhnutná na spustenie priloženého produktu. Na správne fungovanie produktu je nevyhnutné inštalovať 32 bitové JDK verzie 7, podverzie aspoň 9. Najaktuálnejšie verzie JDK sú dostupné na stránkach výrobcu:
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Inštalácia Java 3D
Java 3D umožňuje plnohodnotné využívanie trojdimenzionálnych grafických aplikácií. Pre produkt je nevyhnutná verzia aspoň 1.5.1 dostupná na stránke:
http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html

Inštalácia Visual C++ 2012 Redistributable
Visual C++ Redistributable for Visual Studio 2012 je sada komponentov pre Visual C++ knižnice, ktoré je nevyhnutné pre beh aplikácií vyvíjaných s použitím najmodernejších programovacích nástrojov. Visual C++ Redistributable for Visual Studio 2012 je v odporúčanej 32 bitovej verzii stiahnuteľný na stránkach výrobcu:
http://www.microsoft.com/en-us/download/details.aspx?id=30679

Samotný produkt sa neinštaluje. Pre spustenie simulácie demonštrácie v meste je potrebné len skopírovať celý obsah priečinka *Aplikácia* z priloženého CD nosiča a rozbaliť súbor *simTeam.zip* do ľubovoľného adresára. Následne je potrebné upraviť cestu k nainštalovanej verzii JDK v kroku č. 1 v súbore „start.bat“. To je možné vykonať pomocou ľubovoľného textového editora. Po otvorení súboru „start.bat“ v textovom editore, sa zobrazí nasledujúci text:
„*Cesta-k-JDK*“ *-Djava.library.path=./lib –jar* „*SimTeam.jar*“

Časť súboru „*Cesta-k-JDK*“ je potrebné nahradiť absolútnou cestou k nainštalovanému JDK v kroku č. 1. Ak bolo dané JDK napríklad nainštalované na disk D, do adresára Programy, cesta k JDK by mohla vyzerať nasledovne:
„*D:\Programy\Java\jdk1.7.0_09/bin/java*“.

Súbor by teda po úprave danej cesty mal takýto formát:
„D:\Programy\Java\jdk1.7.0_09/bin/java“ -Djava.library.path=./lib -jar „SimTeam.jar“

Simulácia sa spúšťa pomocou súboru *start.bat*.

Príloha C Obsah elektronického média

Priložený CD nosič má nasledovnú adresárovú štruktúru:

Priečinky:

\Aplikacia\	- spustiteľná aplikácia a projekt s aplikáciou
\Aplikacia\libraries\	- knižnice tretích strán potrebné k simulácii
\Aplikacia\maps\	- mapové podklady k simulácii
\Aplikacia\mason\	- zdrojové kódy v jazyku Java a knižnice prostredia MASON a knižnice RVO2
\Aplikacia\simLatest\	- podklady k prostrediu MASON
\Aplikacia\SimTeam\	- zostavené triedy zo zdrojového kódu
\Aplikacia\Produkt\	- spustiteľný produkt
\Aplikacia\Produkt\start.bat	- súbor na spustenie simulácie
\Aplikacia\navod.txt	- textový súbor s názvami softvérov nevyhnutných na spsutenie simulácie.
\Dokument\	- dokumenácia k Tímovému projektu
\Dokument\Dielo\	- dokumentácia k inžinierskemu dielu
\Dokument\Riadenie\	- dokumetácia k riadeniu v tíme
\Web stranka\	- statická forma web stránky