SLOVENSKÁ TECHNICKÁ UNIVERZITA BRATISLAVA
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Tímový projekt

SIMULÁCIA DAVU

Dokumentácia k inžinierskemu dielu

Bc. Michal Fornádeľ
Bc. Adam Pomothy
Bc. Lukáš Pavlech
Bc. Marek Hlaváč
Bc. Daniel Petráš
Bc. Martin Košický

Vedúci tímového projektu: Ing. Peter Lacko, PhD.
Akademický rok: 2011/12

Obsah

1 Úvod **1-1**

1.1 Účel a rozsah dokumentu 1-1

1.2 Prehľad dokumentu 1-1

2 Simulácia davu v podaní iných spoločností **2-1**

2.1 PathFinder 2-1

2.1.1 Spôsoby komunikácie 2-1

2.1.2 Simulačné prostredie 2-2

2.2 PedGo 2-2

2.2.1 Prostredie 2-3

2.2.2 Plánovanie cesty na základe buniek 2-4

2.2.3 Rozhodovanie agenta 2-4

2.2.4 Pohyb a aktualizácia 2-5

2.3 Fire Dynamics Simulator and Smokeview 2-5

2.3.1 Rozšírenia FDS 2-6

2.3.2 HPC (Hight Performance Computing) 2-6

2.3.3 Nástroje vyvinuté tretími stranami 2-7

2.4 SimWalk 2-8

3 Šprint#1 **3-1**

3.1 Základná vizualizácia 3-1

3.2 Protokol agentových akcií a akcií prostredia 3-3

3.3 Komunikácia prostredia a agentov 3-4

3.4 Základná funkcionalita agenta 3-5

3.5 Vypracovanie metodiky Code Guidelines 3-6

3.6 Zapracovanie analyzovaných riešení do šablóny dokumentu riadenia . 3-6

3.7 Integrovanie SVNka s vývojovým prostredím 3-6

3.8 Zistenie formátu mapy v projekte VirtFIIT 3-7

3.9 Nájdenie nástroja pre konverziu OSM -> CAD 3-7

3.10 Základná implementácia mapy 3-8

3.11 Analýza formátov máp 3-9

3.12 Základná funkcionalita prostredia 3-11

4 Šprint#2 **4-1**

4.1 Simulovanie pohľadu agenta 4-1
4.2 Hľadanie dverí zo strany agenta 4-1
4.3 Plánovanie pohybu agenta 4-2
4.4 Integrácia vyhotovenej dokumentácie do šablóny dokumentu riadenia 4-3
4.5 Vytvorenie koncepcie oddelených modulov (DLL knižnice) 4-4
4.6 Detekcia pretínajúcich sa stien 4-4
4.7 Riešenie kolízií agentov 4-5
4.8 Generovanie agentov do mapy 4-5

Kapitola 1

Úvod

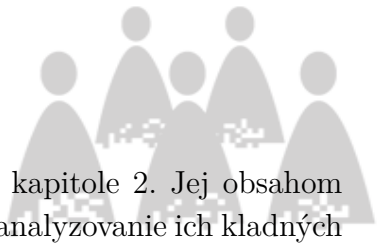
Dokument sa zaoberá problematikou simulovania davu a poskytuje komplexný objektívny pohľad na zadanú problematiku spolu s návrhom a realizáciou riešenia. Aplikácia sa zameriava na simuláciu davu v rámci interiéru a jej výsledkom je schopnosť simulácie stoviek až tisícov agentov umiestnených do prostredia predstavujúceho priestor budovy alebo iného objektu definovaného prostredníctvom mapy. Pri jej vývoja sa tím riadil agilným spôsobom vývoja softvéru SCRUM.

1.1 Účel a rozsah dokumentu

Účelom predkladaného dokumentu je dospieť k analýze, návrhu a implementácii riešenia, ktoré by svojím ponímaním čo najefektívnejšie a najlepšie uspokojilo potreby používateľa požadujúceho produkt z oblasti simulácie davu.

Dokument je výsledkom práce členov tímu v rámci predmetu Tímový projekt na Fakulte informatiky a informačných technológií Slovenskej Technickej Univerzity v Bratislave.

Dokument je určený pre každého používateľa alebo vývojára oboznámeného s problematikou simulovania davu.



1.2 Prehľad dokumentu

Simulácia davu v podaní iných spoločností je rozobratá v kapitole 2. Jej obsahom predstavenie štyroch zaujímavých konkurenčných riešení a zanalyzovanie ich kladných a záporných aspektov. Počas ich analýzy bol hlavný dôraz kladený na pochopenie ich koncepcie a spôsobu fungovania.

Kapitoly 3 a 4 predstavujú jednotlivé úlohy v šprintoch z hľadiska analýzy problému, návrhu riešenia, implementácie a testovania.

Kapitola 2

Simulácia davu v podaní iných spoločností

Tématikou simulovania davu sa dodnes zaoberá množstvo spoločností. Táto oblasť je pomerne rozsiahla a nedá sa povedať, že by sa každé z predstavovaných riešení sústreďovalo na rovnakú problematiku. Rovnako ponímanie spracovávanej problematiky je rôznorodé a mnohokrát optimalizované pre odlišné úrovne výpočtovej sily počítača. Táto kapitola sa zaoberá analýzou troch konkrétnych prevedení a poukazuje rovnako na ich výhody ako aj nevýhody.

2.1 PathFinder

Pathfinder je jeden z moderných evakuačných simulátorov, ktorý pracuje na základe najnovších poznatkov z oblasti počítačových technológií zameraných na modelovanie pohybu jednotlivcov vo vnútorných prostrediach. Pathfinder poskytuje nástroje nevyhnutné pre tvorbu správnych rozhodnutí vyplývajúcich z vytvárania bezpečnostných a evakuačných systémových návrhu stavieb. Obsahuje podporu viacerých simulačných módov a možnosť nastavenia parametrov agentov podľa určitého scenáru. Keďže sa jedná o simulátor založený na agentoch, tak každý agent pracuje na základe sady parametrov a rozhoduje sa nezávisle počas celého trvania simulácie.

2.1.1 Spôsoby komunikácie

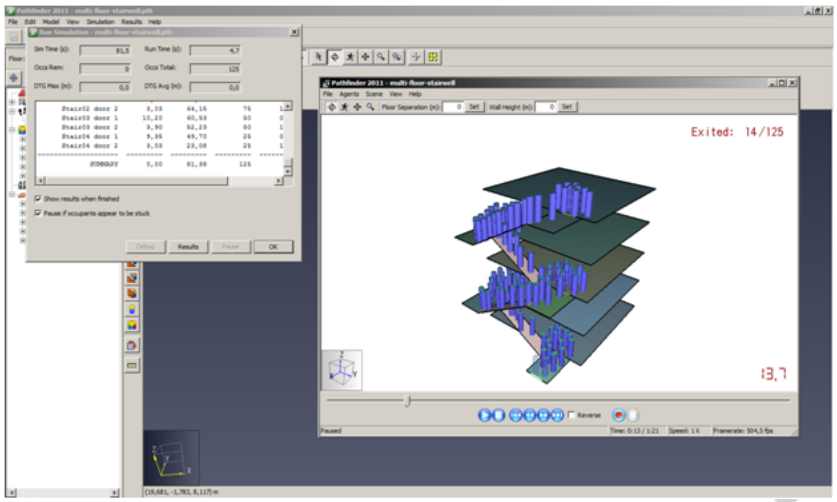
Komunikácia medzi simulačným prostredím a agentmi je možné rozdeliť na dve časti. Prvá sa týka toho ako prostredie vníma agentov a druhá ako agenty vnímajú prostredie. Simulačné prostredie sleduje pohyb agentov priebehom simulácie a poskytuje informácie ohľadom jednotlivých agentov (napr. pozícia). Naopak, agenty nepoznajú všetky východy z budovy a ich rozhodovanie je založené na kritériách vytvorenými

používateľom (napr. agenty môžu poznať hlavný východ, ale iba niektorí budú poznať sekundárne východy), informáciami z poschodia, skúsenosťou a v niektorých prípadoch aj informáciami ohľadom agentov, ktorí sú okolo neho. Pohyb agentov je v značnej miere ovplyvnený hustotou ostatných agentov v prostredí. Pri výpočte nasledujúcich akcií sa berú do úvahy tri fakty: vzdialenosť medzi agentmi, prekážky v prostredí a ohraničenia stavby.

2.1.2 Simulačné prostredie

Dôležitou časťou je editor s prehľadným používateľským rozhraním poskytujúci vytvorenie vlastného scenáru v 3D vizualizácii (obr. 2.1). Štruktúra prostredia je riešená takým spôsobom, že jednotlivé poschodia sú rozdelené do 2D plôch, ktoré umožňujú pohyb agentom z jedného bodu do iného v prostredí budovy. Na pohyb používajú agenty sadu určitých pravidiel, ktorá počíta s viacerými parametrami, medzi ktorými je možné započítať vzdialenosť od nepriechodných buniek, aktuálny stav agenta a iné. Po spustení simulácie sa prepočíta celá simulácia a výsledkom je interaktívna prezentácia, ktorá zobrazuje v 3D vizualizácií priebeh simulácie.

Jeden z problémov PathFinderu je jeho obmedzená časová použiteľnosť vo voľne dostupnej verzii a zdĺhavejší spôsob konfigurácie simulácie.



Obr. 2.1: Ukážka simulácie v programe PathFinder

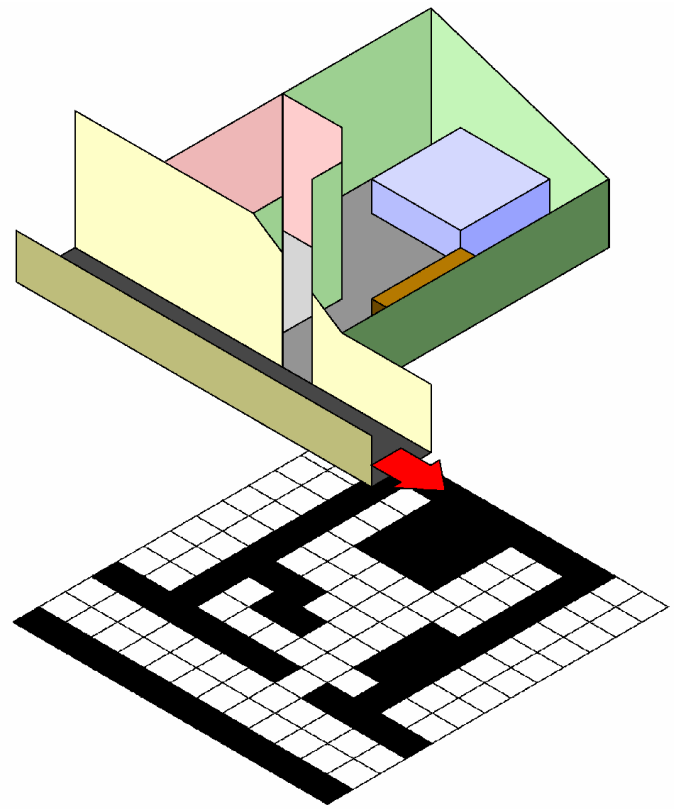
2.2 PedGo

PedGo predstavuje riešenie, ktoré svojimi malými nárokmi príliš nezaťažuje počítač. Využíva multi-agentový model a dokáže simulovať tisícky ľudí už na procesoroch

Pentium o frekvencii 3 500 Mhz. To je umožnené aj minimalistickou vizuálnou stránkou - simulácia je zobrazená len v 2D priestore. Aplikácia je zameraná na simuláciu evakuácie pri rôznych katastrofických scenároch (napr. požiar).

2.2.1 Prostredie

Prostredie je vytvárané transformáciou plánu do dvojrozmernej mriežky buniek. Ukážka tvorby plánu/pôdorysu budovy je na obr. 2.2. Aplikácia rozoznáva 3 druhy buniek:



Obr. 2.2: Transformácia plánu budovy do 2D mriežky buniek

- voľná bunka – agent sa na ňu môže v nasledujúcej jednotke času presunúť
- stena – agent sa nemôže na túto bunku presunúť (predstavuje akúkoľvek prekážku pre pohyb)
- východ – táto bunka predstavuje napr. únikový východ, každý agent sa snaží dosta práve na túto bunku
- dvere – tieto bunky zmenšujú pripustnosť toku agentov, znižujú aj ich rýchlosť

- schody – taktiež znižujú rýchlosť agentov

Model človeka – tzv. agent sa pohybuje po týchto bunkách. Na pohyb moze vyžiť akúkoľvek bunku okrem tej, označenej ako stena. Na jednej bunke môže byť v jednom čase iba jeden agent.

2.2.2 Plánovanie cesty na základe buniek

Každá bunka je ohodnotená podľa vzdialenosti od východu. Čím je bližšie, tým má väčší potenciál. Východom sa nemyslí len definitívny východ znamenajúci úspešnú evakuáciu, ale aj všetky bunky označené ako dvere a schody. Potenciaál sa takto širi po bunkách, až kým nie sú všetky ohodnotené. Agent pri rozhodovaní pozerá na potenciál okolitých buniek a podľa toho si volí cestu.

Každý agent ma na začiatku simulácie náhodne pridelené parametre:

- rýchlosť – udáva, koľko buniek môže prejsť agent za jednotku času
- trpezlivosť – ako dlho bude agent stáť na jednom mieste predtým, ako si zvolí inú cestu
- nevypočítateľnosť – aká je pravdepodobnosť, že sa nebude držat svojej cesty
- náhodnosť – aká je pravdepodobnnosť, že sa bude držať svojho smeru(daného potenciálom) a nezvolí si iný
- reakčný čas – ako dlho trvá agentovi reagovať na evakuačný signál
- spomalenie – pravdepodobnosť, že sa agent zastaví počas jednotky času a zostane stáť až do konca tejto jednotky
- prilnavosť – vyjadruje, ako veľmi sa agent drží v nejakej (danej) skupine agentov

Aplikácia ponúka dve predvolené nastavenia, ktore dané parametre prisposobujú podľa zvoleného času dňa – noc/deň.

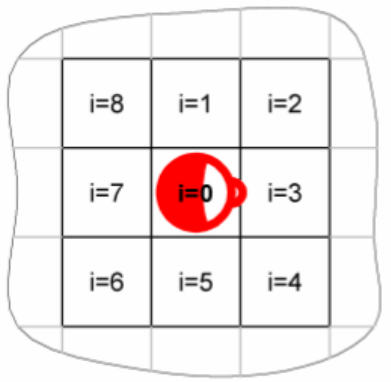
2.2.3 Rozhodovanie agenta

Na obr. 2.3 je znazornený agent (v strede) a okolité bunky. Agent si môže vybrať pohyb do 8 rôznych smerov v nasledujúcom časovom kroku. Pravdepodobnosť, že si vyberie bunku i sa počíta nasledujúcim vzorcom:

$$p_i = e^{-\frac{(P_i - P_0) + s}{S}},$$

pričom p_i vyjadruje pravdepodobnosť vybratia i-tej bunky, P_i potenciál bunky i, P_0 potenciál bunky 0 a S parameter označujúci nevypočítateľnosť.

Po vypočítaní pravdepodobnosti výberu smeru je táto hodnota vynásobená hodnotou náhodnosti.



Obr. 2.3: Znázornenie agenta a možnosti jeho voľby

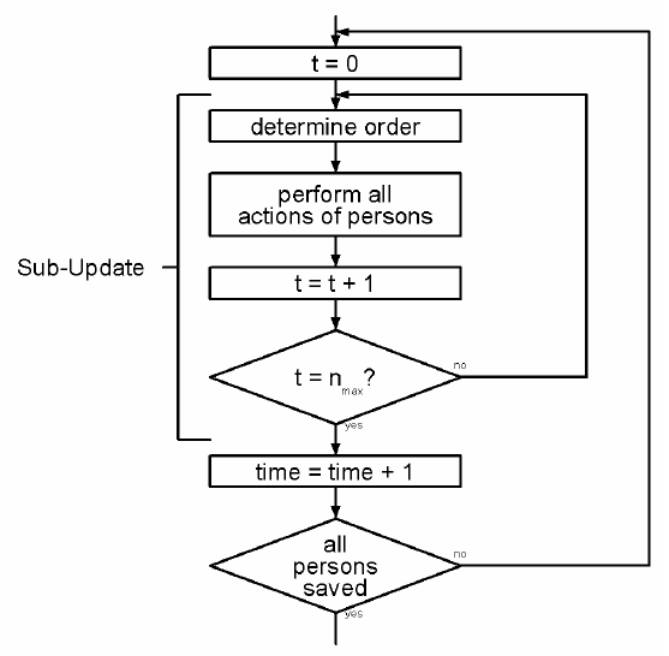
2.2.4 Pohyb a aktualizácia

Na obrázku 2.4 je znázornený diagram aktivít pre aktualizáciu celej populácie agentov, kde t je počet mini-aktualizácií, n rýchlosť celej populácie (koľko buniek sa môžu agenti posunúť za jednotku času) a time časový skok. Algoritmus pohybovania jediného agenta je zobrazený na obrázku 2.5.

2.3 Fire Dynamics Simulator and Smokeview

Fire Dynamics Simulator je CFD (Computational fluid dynamics) model požiaru. Softvér numericky rieši tvar Navier-Stokesových rovníc, ktoré sú vhodné pre pomalý, tepelne riedený tok s dôrazom na transport dymu a tepla z ohňa.

Smokeview (SMV) je vizualizačný program, ktorý sa používa na zobrazenie výstupu z FDS a CFAST simulácií. FDS je voľný softvér vyvinutý NIST-om (National Institute of Standards and Technology of the United States Department of Commerce) v spolupráci s VTT výskumným technickým centrom vo Fínsku. Fire Dynamics Simulator spoločne s Smokeview boli oficiálne predstavený v Februári 2000. FDS číta vstupné dáta z textového súboru, vypočíta numerické riešenie riadiacich rovníc a výsledné dáta zapíše do súborov. Smokeview na základe týchto dát vytvára animáciu na obrazovke.



Obr. 2.4: Diagram aktivít pre aktualizáciu celej populácie agentov

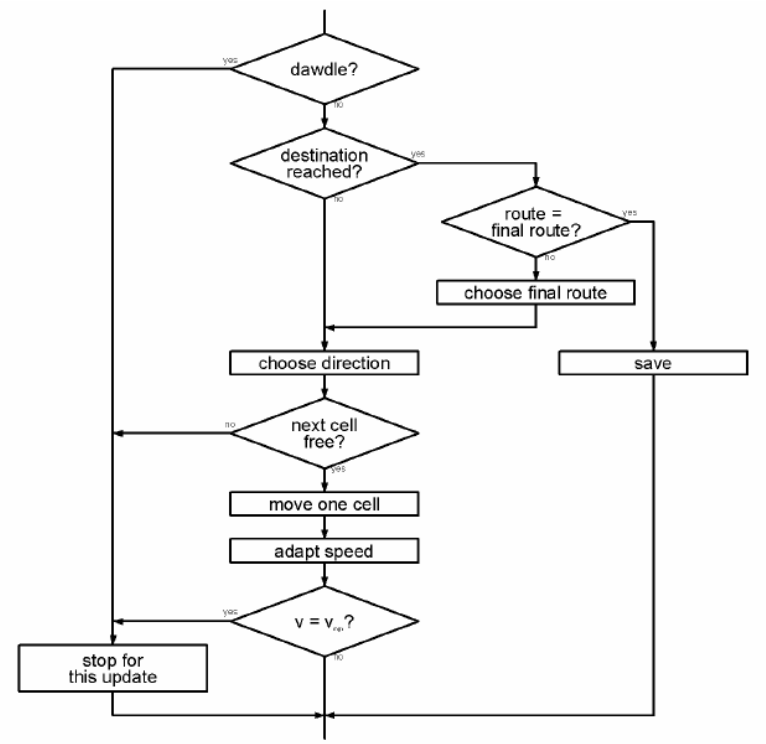
2.3.1 Rozšírenia FDS

FDS + Evac – je evakuačný simulačný model pre FDS. Softvér je využívaný na simuláciu pohybu ľudí pri evakuačných situáciach. Hlavné črty sú:

- Simulácia ľudí založená na agentoch
- Pohybový algoritmus založený na pohybe Panic
- Post-processing pomocou softvéru SMV
- Účinky ohňa sú prepočítavané pomocou FED (Fractional Effective Dose)

2.3.2 HPC (Hight Performance Computing)

FDS je napísaný v programovacom jazyku Fortran 90/95, ale malá časť kódu je napísaná v jazyku C. Momentálne je jedným z issues aj preskúmanie možností na prepis tejto malej časti kódu do Fortránu. V súčastnej dobe existujú dve verzie FDS: sériová a paralelná. Sériová verzia využíva len jeden proces na výpočet riešenia riadiacich rovníc. Paralelná verzia využíva MPI (Message Passing Interface) na spustenie viacerých procesov na rôznych strojoch. Od verzie 5.4 Christian Rogsch z univerzity vo Wuppental v Nemecku implementoval OpenMP direktívy do FDS. OpenMP umožňuje aby FDS bežalo na jednom PC, ale využívalo viacero procesorov a jadier procesora.



Obr. 2.5: Algoritmus pohybovania agenta

Táto verzia je sa oficiálne stále iba testuje, ale OpenMP by mal ostať štandardom pre FDS.

2.3.3 Nástroje vyvinuté tretími stranami

Pre FDS bolo vyvinutých niekoľko nástrojov pre zjednodušenie práce. Tieto nástroje môžeme rozdeliť do kategórií:

- GUI pre FDS – napr. Pyrosim, AspireSDS, BlenderFDS
- Nástroje pre konverziu CAD súborov – napr. 3dsolid2fds, acad2fds,
- Nástroje pre textový editor a tabuľkový procesor – napr. FDS v5 MESH Size Calculator, FDS 5 Syntax file
- Nástroje videa a obrazu – VideoEncoderGUI
- Post-processing a reportovacie nástroje - FDS Reporter
- Nástroje na modelovanie evakuácie - STEPS

2.4 SimWalk

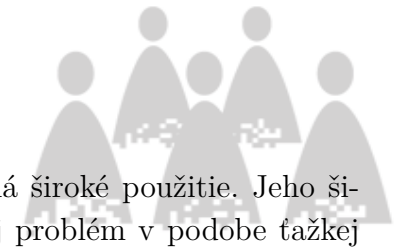
Simwalk je pravdepodobne najviac komerčne používaným nástrojom v rámci simulácie davu za účelom zvýšenie efektivity v rámci logistiky. Logistiku môžeme charakterizovať ako je proces plánovania, simulovania, implementovania a ovládania účinného, bezpečného a efektívneho toku chodcov. Riešenie okrem spomínanej simulácie davu ponúka aj simulovanie lietadiel, vozidiel mestskej hromadnej dopravy a vlakov idúcich z bodu A do bodu B pre účel účinných a bezpečných operácii. Simulovaním veľkého počtu ľudí a analýzami SimWalk pomáha vyriešiť problémy s neúčinnou pešou logistikou a prepravou. Nástroj podporuje všetky fázy projektovania logistiky a tiež umožňuje realistickú štúdiu a vylepšenie toku pasažierov. SimWalk PRO verzia rieši okrem iného aj evakuácie, štandardné letiskové operácie a problémy s prepravou handicapovaných ľudí.

Typické použitie SimWalk PRO

- Kapacita chodcov a účinnostná analýza komplexných zariadení
- Analýza kríženia sa chodcov v dopravných scenároch
- Analýza toku davu vo verejných priestranstvách a mestských plánovaniach
- Štúdia priechodnosti v architektúre
- Kapacita a prechodová analýza na letiskách
- Simulácia evakúacie

Výstupy aplikácie

- Kapacita
- Rýchlosť prechodov
- Počty a toky
- Úrovne služieb
- Strata rýchlosti



Toto riešenie sa javí ako veľmi užitočný softvér, ktorý má široké použitie. Jeho širokospektrálne zameranie predstavuje na druhú stranu aj problém v podobe ťažkej orientácie medzi ponúkanou funkcionalitou. Pre nového používateľa to môže znamenať v úvodnej fáze práce so softvérom početné komplikácie a problémy. Z dôvodu, že Simwalk je komerčným nástrojom, sa nepodarilo nájsť žiadne zmienky o modularite v

zmysle pridávania napríklad vlastných agentov v podobe knižnice alebo dostupného interfejsu. Azda najväčšou výhodou je umožnenie 2D aj 3D simulácie na jednom počítači bez využitia distribuovaného počítania.

Kapitola 3

Šprint#1

ID	Názov úlohy	Zodpovedný
1	Základná vizualizácia	Martin Košický
2	Protokol agentových akcií a akcií prostredia	Daniel Petráš
3	Komunikácia prostredia a agentov	Daniel Petráš
4	Základná funkcionalita agenta	Michal Fornádeľ
5	Vypracovanie metodiky Code Guidelines	Adam Pomothy
6	Zapracovanie analyzovaných riešení do šablóny dokumentu riadenia	Michal Fornádeľ
7	Integrovanie SVNka s vývojovým prostredím	Adam Pomothy
8	Zistenie formátu mapy v projekte VirtFIIT	Lukáš Pavlech
9	Nájdenie nástroja pre konverziu OSM -> CAD	Marek Hlaváč
10	Základná implementácia mapy	Marek Hlaváč
11	Analýza formátov máp	Marek Hlaváč.
12	Základná funkcionalita prostredia	Lukáš Pavlech

3.1 Základná vizualizácia

Analýza problému
Existuje niekoľko spôsobov vykresľovania. Aplikácia je primárne tvorená pre Windows, takže možnosti sú vykresľovanie cez Direct3D, OpenGL , Windows GDI.

Direct3D je súčasťou DirectX SDK a má silnú podporu pre tvorbu grafických aplikácii. Má v podstate takú funkcionalitu ako OpenGL akurát, že Direct 3D nie je portabilné a funguje iba na Windows systémoch primárne. Súčasťou DirectX SDK sú aj knižnice pre prácu so zvukom, práca so vstupom klávesnice, myši a iných zariadení.

OpenGL je knižnica ktorá tiež ako Direct 3D umožňuje pracovať s grafickou kartou

na nízkej úrovni a tak využívať 3D akceleráciu. Výhoda je, že OpenGL je portabilná knižnica, ktorá beží na viacerých operačných systémoch, dokonca aj na iných zariadeniach. Nevýhoda je, že to je iba knižnica na grafiku a teda vstup zklávesnice a iných zariadení treba riešiť osobitne. Jednou z alternatív je knižnica SDL vďaka ktorej sa dá tiež pohodlne vytvoriť okno pre vykresľovanie 3D. Ak by sa nepoužilo SDL, je ešte možnosť použiť Windows funkcie na zachytávanie aktuálne stlačených kláves a detekcia myši. Ale ak by sa nepoužilo SDL tvorba okna by musela byť manuálna.

Windows GDI umožňuje kresliť základné geometrické útvary ako čiary, body, obdĺžniky. Nemá žiadnu 3D akceleráciu a keď chceme vykresľovať veľký počet agentov v reálnom čase tak by GDI nestačilo.

Návrh riešenia
Vizualizácia bude používať SDL a OpenGL ako základ na nízkej úrovni. Poskytujú vysoký výkon a pohodlne sa používajú. Vizualizácia bude fungovať v troch fázach: incializácia, cyklus, deinicializácia, ktorá súčasne ukonči celú aplikáciu.

Opis implementácie
Vizualizácie používa knižnice OpenGL pre vykresľovanie a SDL pre vytváranie okna.

Vizualizácia prebieha v troch fázach: inicializácia, Cyklus pre každý frám a deinicializácia.

Pri inicializácii sa vytvorí okno o rozmeroch 640x480 pixloch s opengl vykresľovacím kontextom. Nastaví sa perspektíva, základný tieňovací model, ktorý sa momentálne aj tak nepoužíva, nastaví farba pozadia, porovnávacia funkcia na hĺbku a alokuje pamäť na grafickej karte, kde sa natiahne mapa, ktorá je momentálne natvrdo zapísana v poli.

Každý frám sa vykonáva vyberanie všetkých udalostí cez SDL, medzi ktoré patria aj stláčenie kláves. Pokiaľ je jedna z udalostí stlačenie klávesy ESC tak sa prejde k deinicializácii, inak sa vymaže hĺbkovy buffer a vykreslí sa odznova mapa cez VBO a v cykle sa prechádzajú všetci agenti a tí sa vykresľujú na určenú pozíciu. Agenti vstupujú do vizualizácie cez komunikačný kanál z prostredia.

Deinicializácia spôsobí dealokácia vykresľovacieho kontextu a vyhodí výnimku, ktorá signalizuje ukončenie aplikácie, nie je to chyba.

3.2 Protokol agentových akcií a akcií prostredia

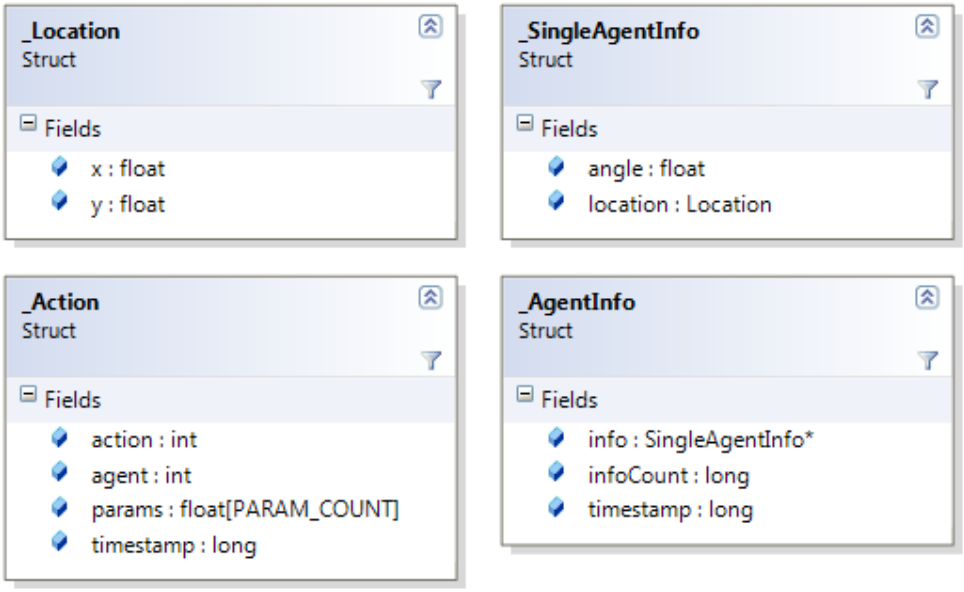
Analýza problému
Základným problémom bolo navrhnúť protokol, pomocou ktorého si budú môcť prostredie a agenty vymieňať informácie. Agent potrebuje vedieť, kde v prostredí sa nachádza a kde sa nachádzajú ostatné agenty. Prostredie zase potrebuje vedieť, akú akciu sa snaží agent vykonať. Základné informácie o agentovi sú jeho poloha a natočenie (uhol). Potrebné sú aj doplnkové informácie, ako napríklad kedy bola daná akcia vygenerovaná a kým.

Návrh riešenia
Informácia o pozícii všetkých agentov sa dá jednoducho zaznamenať pomocou poľa. Prvkami poľa budú dátové štruktúry obsahujúce práve informácie o polohe a natočení agenta. K týmto informáciám je pridaný aktuálny čas kedy bola informácia zaslaná agentom.

Akcia, ktorú posiela agent prostrediu, bude definovaná svojim identifikátorom. Podľa toho, o akú akciu pôjde, môže ďalej obsahovať niekoľko parametrov. Takisto bude obsahovať čas kedy bola akcia zaslaná prostrediu ako aj identifikátor agenta, ktorý akciu vygeneroval.

Opis implementácie
Protokol je implementovaný pomocou niekoľkých dátových štruktúr, ktoré sú zobrazené na obrázku 3.1.
Maximálny počet parametrov v štruktúre Action je 3. V prípade potreby je možné tento počet kedykoľvek zmeniť. Pomocou tohto prístupu je kedykoľvek možné pridať ďalšiu akciu bez toho, aby sa musel meniť komunikačný protokol. Zadefinovane akcie sú:

- ACTION_TURN (id = 1)
  - Akcia otočenia sa v smere hodinových ručičiek (v prípade kladného uhla).
  - Počet parametrov: 1
  - Prvý parameter: uhol (v stupňoch) zmeny aktuálneho natočenia z intervalu (-360.0; 360.0)
- ACTION_STEP (id = 2)
  - Akcia kroku vpred, samotný smer závisí od aktuálneho natočenia.
  - Počet parametrov: 0



Obr. 3.1: Štruktúra prenášaného protokolu

3.3 Komunikácia prostredia a agentov

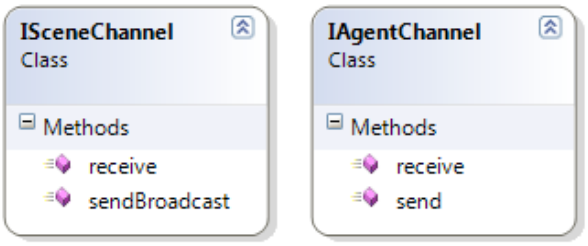
Analýza problému

Bolo treba navrhnúť spôsob, akým budú spolu komunikovať prostredie a agenty bez toho, aby boli naviazaní na konkrétnu implementáciu komunikácie. Spôsob komunikácie sa bude počas vývoja projektu meniť (napr. distribuované počítanie a pod.), takže bolo treba navrhnúť všeobecný model, s ktorého rozhraním budú agenty a prostredie narábať. Ďalšou úlohou bolo implementovať komunikačný kanál na základe navrhnutého modelu. Prvotná implementácia mala byť čo najjednoduchšia, kde agenty existovali v rovnakom procese ako samotné prostredie, takže sa jednalo iba o výmenu informácií medzi objektmi v rámci jedného programu.

Návrh riešenia

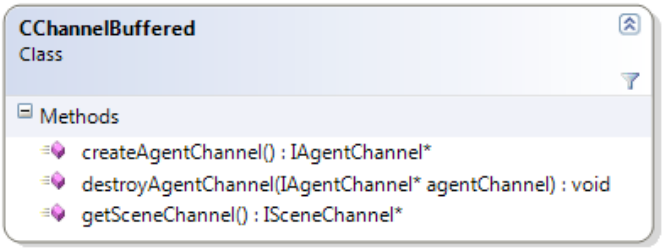
Riešenie spočíva v zadefinovaní dvoch rozhraní. Jedno rozhranie slúži na odosielanie správ všetkým agentom a získavanie správ od agentov (ISceneChannel). Druhé rozhranie na získavanie správ od prostredia a zasielanie sprav prostrediu (IAgentChannel). To akým spôsobom sa správy dostanú z jednej strany na druhu už bude vecou implementácie. Metódy ktoré sa používajú na zasielanie správ sú asynchrónne, tj. neblokujúce, zatiaľ čo metódy na získanie zaslaných informácií, sú blokujúce až kým nepríde odpoveď. Obe rozhrania sú znázornené na obrázku 3.2.

Opis implementácie



Obr. 3.2: Rozhranie komunikácie prostredia a agentov

Prostredie zapisuje správy ktoré chce poslať do vyrovnávacej pamäte. Ak sa pamäť minie začne prepisovať najstaršie správy. Vyrovnávacia pamäť je implementovaná pomocou poľa správ. Každý IAgentChannel si pamätá z ktorej pozície (poľa) naposledy čítal, a po precitaní novej správy sa o jedna posunie. Ak narazí na koniec, skočí na začiatok. Agenty zapisujú svoje správy na koniec fronty a prostredie ich načítava zo začiatku fronty. Z opisu je zrejmé, že obe implementácie rozhraní majú spoločné dátové štruktúry (pole a frontu). To je dosiahnuté pomocou továrne, ktorá na požiadanie vytvorí kanál pre agentov alebo prostredie. Keďže v simulácií vystupuje iba jedno prostredie, aj kanál pre prostredie bude iba jeden. Tieto kanále zdieľajú spomínané dátové štruktúry práve od tejto továrne. Model továrne je zobrazený na obrázku 3.3.



Obr. 3.3: Model továrne

3.4 Základná funkcionalita agenta

Analýza problému

Za účelom overenia funkčnosti prostredia a komunikačného protokolu vznikla potreba dopracovania agenta. Agenty v počiatočnej fáze sú generované na ľubovoľnom mieste na mape a ich správanie nie je ovplyvnené žiadnymi faktormi.

Návrh riešenia

Prvotný návrh bolo vytvorenie primitívneho agenta, ktorého správanie bolo špecifikované generovaním náhodných krokov (v zmysle doľava, doprava, vpred, vzad) alebo uhla natočenia.

Implementácia
Implementácia agenta obsahovala jeho základný vnútorný stav - polohu, kde sa nachádza a natočenie. Každý vytvorený agent bol navrhnutý, aby neustále zásoboval prostredie požiadavky na pohyb. V priemere každá piata požiadavka bola v pohode pootočenia o uhol v rozmedzí -30 až 30 stupňov. Agenty sa pohybovali po mape nezávisle a nebrali do úvahy rozmery, ani umiestnenie na obrazovke.

3.5 Vypracovanie metodiky Code Guidelines

Pre dosiahnutie čo najväčšej kvality produktu z vývojárskeho hľadiska je veľmi dôležité stanoviť pravidlá, ktoré vývojári dodržiavajú pri implementácii. Výsledkom je zefektívnenie vývoja, väčšia produktivita a lepšia komunikácia medzi členmi vývojárskeho tímu. Pravidlá použité pri vývoji tejto aplikácie sa nachadzajú dokumente riadenia v prílohe A.

3.6 Zapracovanie analyzovaných riešení do šablóny dokumentu riadenia

Analyzované riešenia boli z dôvodu štruktúry jednotlivých dokumentov presunuté do dokumentu k inžinierskemu dielu. Tu sa následne doposiaľ vytvorené časti písané v prostredí Microsoft Word upravovali pre následnu integráciu do prostredia šablóny.

3.7 Integrovanie SVNka s vývojovým prostredím

Na plné využitie výhod verziovacieho systému (v našom prípade Subversion) je potrebná aj jeho integrácia do vývojového prostredia. Už pri výbere systému sme brali do úvahy existenciu voľne dostupného doplnku do nami zvoleného vývojového prostredia (Microsfot Visual Studio 2010), ktorý by bol kompatibilný so systémom Subversion. Ako vhodný doplnok sme vybrali AnkhSVN, ktorý podporuje všetky pokročilé funkcie manažovania spoločného prístupu do zdrojových súborov. Doplnok umožňuje aktualizovať, uploadovať ale aj spájať (angl. merge) zdrojový kód priamo z vývojového prostredia. Vďaka tomu sa výrazne zefektívni implemetácia a predíde sa množstvu problémom pri zasahovaní do rovnakých súborov so zdrojovými kódmi.

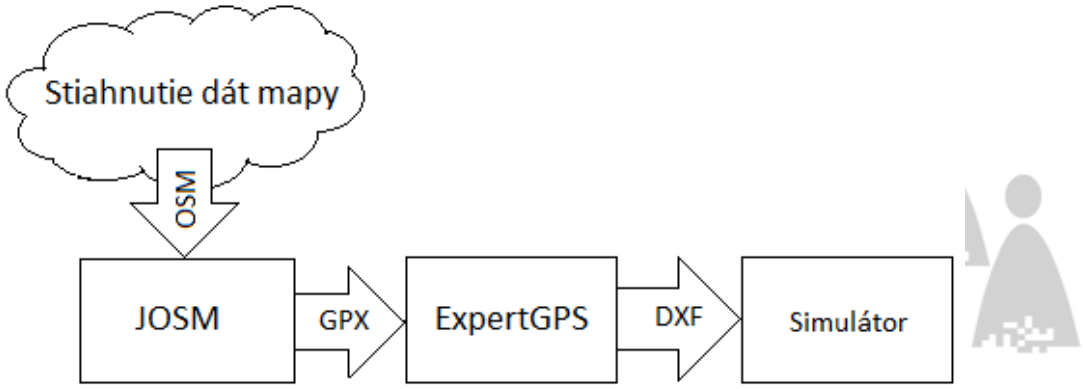
3.8 Zistenie formátu mapy v projekte VirtFIIT

Analýza problému
Projekt Virtualna FIIT bol riešený v ročníku 2010/2011 tímom Mgr. Aleny Kovárovej. V rámci tohto projektu ako výsledok bola vytvorená 3d web prezentácia školy. Model školy ako som sa dozvedel od Bc. Samuela Forusa (člena tímu virtuálna FIIT II.) je vytvorený v nástroji 3ds Max. Tento formát môže byť výhodný pre náš tým a to kvôli tomu, že formát 3ds max sa dá konvertovať do rôznych formátov ako napr. AutoCad, ArchiCad atď.

3.9 Nájdenie nástroja pre konverziu OSM -> CAD

Analýza problému
Pre prácu s formátom OSM sme použili softvér JOSM (JavaOpenStreetMap Editor), ktorý umožňuje stiahnuť reálne údaje a následne s nimi pracovať. Ďalší krok spočíval v konverzii do DXF formátu na základe výberu formátu máp, ktorý je uvedený v Tab.1. v kapitole Analýza formátov máp. Keďže priamy export z JOSM do formátu DXF nie je možný, tak sme boli nútený spraviť prieskum cielený na možnosti konverzie OSM formátu do formátu DXF.

Výsledok prieskumu je, že priama konverzia nie je podporovaná žiadnym voľne šíriteľným nástrojom. Ďalšou možnosťou je nájdenie tretieho formátu, ktorý by slúžil ako sprostredkovateľ konverzie medzi týmito formátmi, čo by znamenalo, že by bola podporovaná konverzia z OSM do tretieho formátu a následne ďalšia konverzia do formátu DXF. Ako sprostredkovateľ konverzie bol vybraný softvér ExpertGPS, ktorý umožňuje export do DXF formátu a import GPX súborov, ktoré je možné vyexportovať prostredníctvom JOSM (obr. 3.4).



Obr. 3.4: Konverzia formátu OSM do DXF

3.10 Základná implementácia mapy

Analýza riešenia
Vzhľadom na základnú funkcionalitu simulácie je nutné do mapy implementovať prvky, ktoré budú poskytovať potrebné informácie. Každý typ prvkov je v mape umiestnený na vlastnej vrstve. Dôvodom oddelenia je ľahšie spracovanie údajov pri importe mapy do vytváranej aplikácie. Základné mapy obsahujú 5 vrstiev:

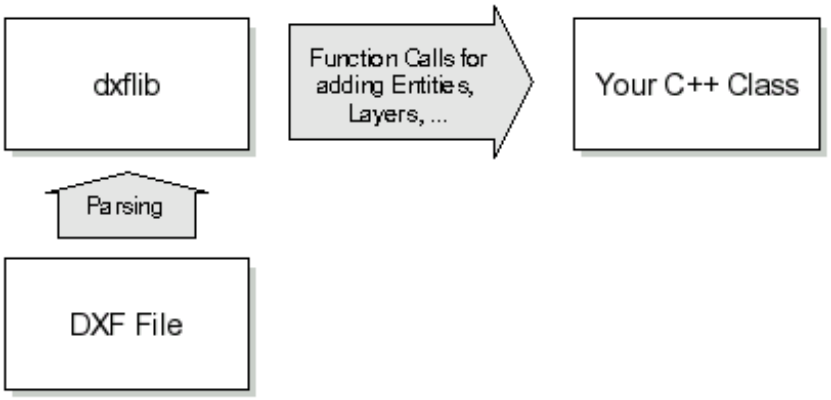
- Steny – reprezentujú ohraničenie budovy, v ktorej sa má simulácia odohrávať.
- Prekážky – reprezentujú objekty v budove, cez ktoré nie je možné prejsť.
- Štarty (angl. spawns) – sú plochy, v ktorých sa generujú agenty na začiatku simulácie.
- Dvere – reprezentujú oblasti medzi miestnosťami, ktoré uľahčujú navigáciu agentov.
- Východy – sú špeciálny druh dverí, ktoré po dosiahnutí agentom ukončujú jeho simuláciu.

Všetky prvky sú vo zvolenom DXF formáte reprezentované ako krivky (angl. polylines), ktoré sú reprezentované postupnosťou bodov, pričom prvý a posledný bod je rovnaký. Takýmto spôsobom sme schopní navrhnúť vlastné jednoduché mapy, po prípade použiť koverziu medzi rôznymi formátmi.

Návrh riešenia
Knižnica dxflib je open source C++ knižnica zameraná na čítanie a následné spracovanie informácií z textových súborov (angl. parsing) formátu DXF. Knižnica pri spracovávaní informácií volá funkcie, ktoré sú zadefinované v používateľskej triede. Tieto funkcie sa volajú na základe entít, ktoré sú aktuálne spracovávané, to znamená, že pre každý typ entity môžeme vytvoriť unikátne spracovanie. Ďalšou dôležitou vlastnosťou je jednoduché spracovávanie údajov prostredníctvom vrstiev. Knižnica dxflib taktiež umožňuje zápis údajov, ale pri používaní sa predpokladá hlbšia znalosť formátu DXF. Štruktúra knižnice dxflib je zachytená na obrázku 3.5.

Implementácia
Pre prvé jednoduché simulácie sme vytvorili nasledujúce mapy, ktorých cieľom je overenie základnej funkcionality simulácie vzhľadom na jednotlivé oblasti:

- spracovávanie informácií aktuálneho stavu simulácie prostredím,
- plánovanie agentov,



Obr. 3.5: Štruktúra knižnice dxflib

- navigácia agentov,
- vizualizácia agentov v prostredí,
- výpočet vnútornej mapy agentov,
- generovanie agentov,
- konfliktné situácie agenta s agentom alebo agenta s prekážkami, či stenami,
- odchod agentov zo simulácie.

3.11 Analýza formátov máp

OpenStreetMaps
OpenStreetMaps (OSM) je celosvetový projekt, ktorého cieľom je vytváranie voľne dostupných zemepisných informácií a ich následná transformácia do máp so zámerom využitia pre topografickú vizualizáciu. OSM dáta sa vytvárajú na základe záznamov z GPS (Global Positioning System) prijímačov alebo iných digitálnych zariadení, ktoré sú licenčne kompatibilné. OSM využíva vlastný formát dát, ktorý je postavený na značkovacom jazyku XML. Primitívnymi konštrukciami formátu OSM sú:

- Uzly – body, ktoré musia obsahovať unikátny identifikátor a pozíciu reprezentovanú prostredníctvom zemepisnej šírky (angl. latitude) a dĺžky (angl.longitude). <node id=”15680” lat=”61.808395385742” lon=”10.849707603454” visible”true” timestamp=”2005-07-30T14:27:12+01:00”/>
- Cesty – postupnosti uzlov, ktoré znázorňuje krivku alebo uzavretú krivku – polygon. Musia obsahovať zoznam referencií na konkrétne uzly a označenie prostredníctvom špeciálneho elementu <tag/>.

<way id=”35” visible=”true” timestamp=”2006-03-14T10:07:23+00:00” user=”johnz”>
<nd ref=”156804”/>
<nd ref=”156805”/>
<nd ref=”156806”/>
<tag k=”highway” v=”secondary”/>
</way>

- Relácie – skupiny uzlov, ciest a ďalších relácií, ktorým zámerom je opísať tieto skupiny.
<relation id="77"visible="true"timestamp="2006-03-14T10:07:23+00:00üser="fred»
<member type="way"ref="343"role="from"/>
<member type="node"ref="911"role=ät"/>
<member type="way"ref="227"role="to"/>
<tag k="type"v="turn_restriction"/>
</relation>

Na základe týchto konštrukcií je možné vytvoriť komplexné topografické mapy. Tieto mapy je možné použiť pri najrôznejších typoch vizualizácií a simulácií, pričom je možné mapy jednoduchým spôsobom rozšíriť o špecifické vlastnosti. Problémom pri OSM je, že satelitné informácie, ktoré sú vstupom pre konverziu do OSM je možné zachytiť len vonkajší pohľad, a teda sme značne obmedzení pri vytváraní simulácie vo vnútornom prostredí štruktúr a taktiež použitím 3D máp.

Drawing Exchange Format
Drawing Exchange Format (DXF) je CAD (Computer Aided Design) formát vyvinutý spoločnosťou Autodesk, ktorý umožňuje výmenu dát medzi softvérom AutoCAD a inými druhmi softvérov.

DXF formát je poskladaný z dvojíc, kde kódu prislúcha konkrétna hodnota. Skupiny kódov (angl. group codes) indikujú typ hodnoty, ktorá za nimi nasleduje. DXF súbor je organizovaný na základe skupín kódov s príslušnými hodnotami do sekcií, ktoré sú poskladané zo záznamov, pričom jednotlivé záznamy sa skladajú zo skupín kódov a dátových položiek. Každá skupina kódu a hodnota je na jednom riadku v DXF súbore. DXF súbor obsahuje viacero textových sekcií:

- Hlavička – obsahuje nastavenia premenných hodnôt, ktoré súvisia s výkresom.
- Triedy – uchovávajú informácie pre aplikačné definície tried, ktorých inštancie sa vyskytujú v sekciách bloky, entity a objekty.
- Tabuľky – obsahujú definície mien základných prvkov zobrazenia.
- Bloky – obsahujú definície prvkov obsiahnutých v blokoch výkresu.

- Entity – obsahujú všetky prvky súboru vrátane umiestenia blokov.
- Objekty – obsahujú informácie aplikované na negrafické objekty.
- Náhľad výkresu
- Koniec súboru

Prostredníctvom DXF formátu sa dá znázorniť akákoľvek mapa, či už 2D alebo 3D. Toto zobrazenie je realizované využitím základných geometrických primitív (body, úsečky, krivky, kružnice, atď.), ktoré je možné skladať do väčších celkov v jednotlivých vrstvách výkresu. Takýmto spôsobom je možné vytvoriť komplexné a ľahko rozšíriteľné mapy, či už vonkajších alebo vnútorných prostredí. DXF je výmenný formát, takže je možné vykonávať konverziu mnohých formátov z alebo do DXF formátu.

Výber formátu
Pri výbere formátu sme sa riadili 4 aspektmi pohľadu:

- Použiteľnosť – reprezentuje hodnotu použitia formátu vo vonkajších a vnútorných prostrediach, podporu knižníc a ďalších nástrojov pri získavaní informácií.
- Konverzia – reprezentuje podporu konverzie medzi formátmi, čím je možné v budúcnosti používať aj mapy v iných formátoch.
- 3D – podpora reprezentácie mapy v 3-dimenziálnom zobrazení.
- Čitateľnosť – charakterizuje úroveň pochopenia čítaného textového formátu dát človekom.
- Vrstvy – podpora vrstevnej reprezentácie mapy.

Formát	Použiteľnosť	Konverzia	3D	Čitateľnosť	Vrstvy
OSM	++	+	- - -	+++	+++
DXF	+++	+++	+++	-	+++

3.12 Základná funkcionalita prostredia

Návrh riešenia
V prvotnom prototype prostredie vykonáva nasledujúce činnosti:

- pri spustení programu musí umiestniť agentov
- príjma od jednotlivých agentov správy, ktoré obsahujú informácie o zmene natočenie, polohy

- prepočítava nové pozície, natočenia pre agenta
- posiela agentom informácie o ich súčastnom umiestnení

Opis implementácie
Pri štarte aplikácie sa vytvárajú agenti. V rámci vytvárania agenta sa agentovi generuje náhodná pozícia vo vnútri budovy (prostredie už predtým získa údaje o mape) a natočenie agenta. Potom sa spustí metóda void CSceneImpl::simulate() v ktorej sa nachádza nekonečný cyklus while. V tomto cykle sa pri každom prechode pošlú inforácie všetkým agentom. Každý agent vykoná svoju úlohu a pošle naspäť prostrediu výslednú akciu. Prostredie dekóduju túto správu podľa Protocol.h a vykoná náležité zmeny.

Kapitola 4

Šprint#2

ID	Názov úlohy	Zodpovedný
1	Simulovanie pohľadu agenta	Martin Košický, Michal Fornádeľ
2	Hľadanie dverí zo strany agenta	Adam Pomothy
3	Plánovanie pohybu agenta	Marek Hlaváč
4	Integrácia vyhotovenej dokumentácie do šablóny dokumentu riadenia	Michal Fornádeľ
5	Vytvorenie koncepcie oddelených modulov (DLL knižnice)	Martin Košický
6	Detekcia pretínajúcich sa stien	Daniel Petráš
7	Riešenie kolízií agentov	Lukáš Pavlech
8	Generovanie agentov do mapy	Lukáš Pavlech

4.1 Simulovanie pohľadu agenta

Úloha v čase ukončenia šprintu nebola splnená.

4.2 Hľadanie dverí zo strany agenta

Analýza problému

Každý agent má k dispozícii svoju lokálnu mapu. Agent sa vďaka nej dokáže rozhodnúť, aký bude jeho nasledujúci krok. Ak v danom momente nepozná bezpečnostnú zónu, je preňho prvoradé nájsť všetky dostupné dvere. O dverách potrebuje vedieť, či sa nachádzajú v rovnakej miestnosti ako on a potrebuje poznať aj ich aktuálnu vzdialenosť.

Návrh riešenia
Proces hľadania dverí pozostáva z nasledujúcich krokov:

1. Získanie zoznamu všetkých dverí z aktuálnej mapy agenta.
2. Pre každé dvere zistiť, či sa medzi nimi a agentom nenachádza prekážka (stena alebo iná prekážka)
3. Vypočítanie vzdialenosti ku každým dverám.
4. Vytvorenie zoznamu dverí s ich vzdialenosťami od agenta a informáciou o prípadnej prekážke medzi nimi a agentom.

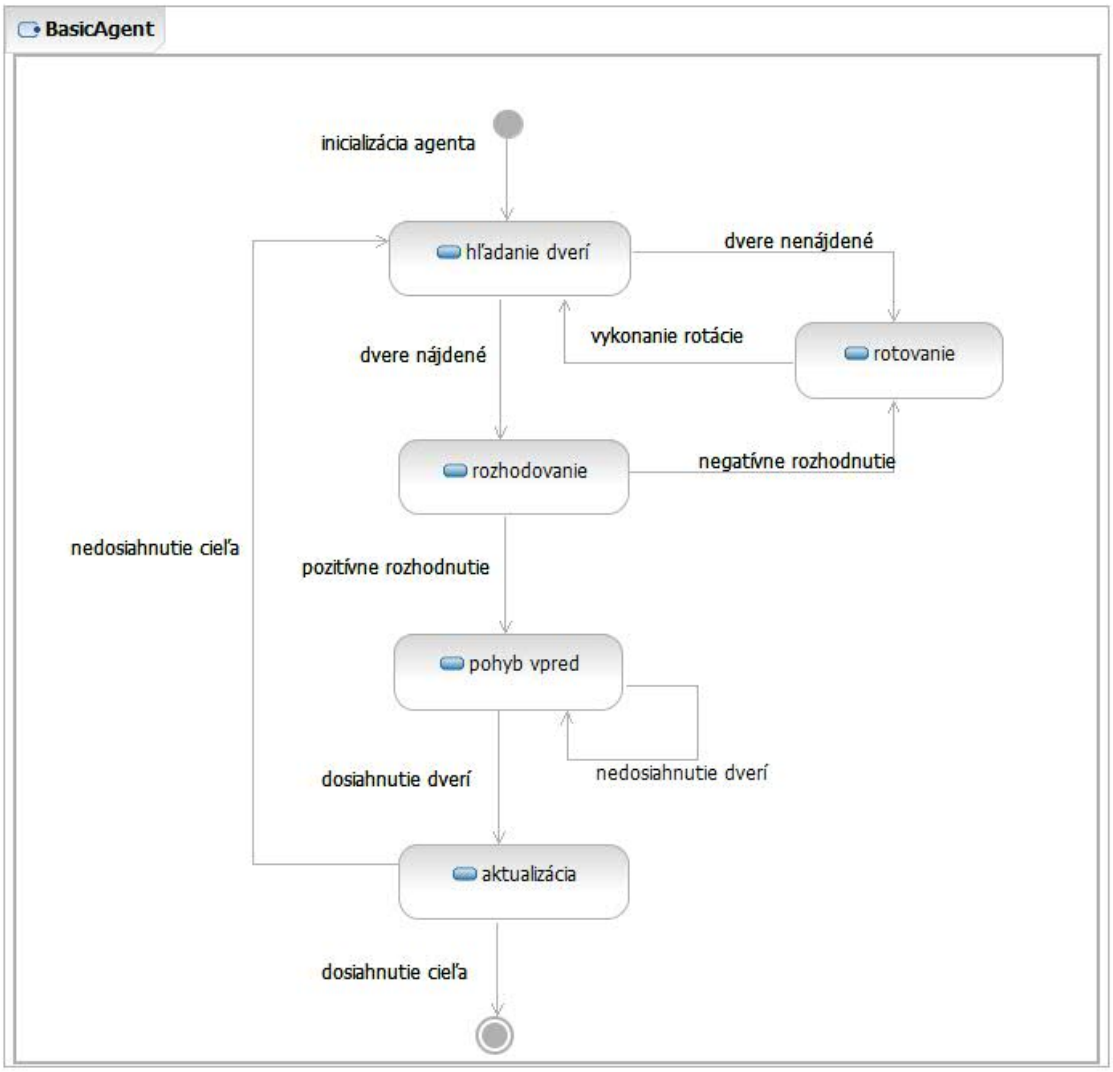
Opis implementácie
Pri vykonávaní procesu je potrebné najprv vypočítať súradnice stredu dverí, nakoľko sú reprezentované ako obdĺžnik. Potom sa vzdialenosť agenta od dverí počíta ako dĺžka spojnice aktuálnej polohy agenta a vypočítaného stredu obdĺžníka reprezentujúceho dvere. Pri počítaní vzdialeností je ďalej potrebné zistiť, či sa spojnica agenta a dverí nepretína so stenou alebo inou prekážkou - čo znamená, že sa dvere nachádzajú v inej miestnosti. Na to je potrebné prejsť všetky prekážky a steny (reprezentované ako vektory) a uistiť sa, že nemajú žiadny spoločný bod so spojnicou agent-stred dverí. Výstup tohoto procesu je zoznam dverí obsahujúci súradnice dverí, ich aktuálnu vzdialenosť k agentovi a informáciu o prípadných prekážkach medzi nimi a agentom.

4.3 Plánovanie pohybu agenta

Návrh riešenia
Po inicializácii sa agent nachádza v stave hľadania dverí. Dôvod je ten, že mapa je pre neho neznáma a vzhľadom na tento fakt je nutné, aby sa porozhliadol po miestnosti so zámerom nájdenia dverí. Dvere slúžia agentovi ako navigácia po mape. V prípade, že agent nevidí dvere, tak rotuje svoju pozíciu, až kým nejaké nenájde. Ako náhle sa mu podarí nájsť dvere, tak sa dostáva do stavu rozhodovania, v ktorom sa snaží rozhodnúť, či pokračovať v hľadaní ďalších dverí alebo nie. Cieľom rozhodovania je zamedzenie pohybu agentovi cez rovnaké dvere a umožniť mu prechádzať cez ešte nenavštívené dvere efektívnym spôsobom. Heuristika rozhodovania môže byť rozšírená o znalosť mapy, ktorá umožňuje simulovať pokročilejšie správanie agentov.

V prípade, že agent je rozhodnutý pokračovať k vybraným dverám, tak sa aktivuje pohyb smerom vpred vzhľadom na natočenie agenta. V každej iterácií pohybu sa kontroluje dosiahnutie dverí. Po úspešnom dosiahnutí sa aktualizuje stav agenta doplnením zoznamu novou položkou dverí, ktoré boli dosiahnuté. Následne sa kontroluje

dosiahnutie cieľa, resp. východu. Pri úspešnej kontrole sa agentova simulácia ukončuje, v opačnom prípade začína celý cyklus odznovu.
Stavový diagram rozhodovania agenta je znázornený na obrázku 4.1.



Obr. 4.1: Stavový diagram rozhodovania agenta

4.4 Integrácia vyhotovenej dokumentácie do šablóny dokumentu riadenia

Úloha spočívala v korekcii a prepise vyhotovených častí dokumentácií do prostredia šablóny dokumentácie riadenia. Niektoré časti boli z koncepčného hľadiska presunuté do dokumentácie ku inžinierskemu dielu.

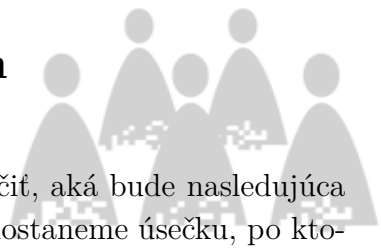
4.5 Vytvorenie koncepcie oddelených modulov (DLL knižnice)

Analýza problému
Existuje niekoľko prístupov k riešeniu projektov na ktorom pracuje viacej ľudí. Môže to byť jedna veľká aplikácia, čo môže viesť k problémom keď veľa ľudí súčasne zasahuje do toho istého projektu a jedna z požiadaviek pre simuláciu davu je, že bude fungovať distribuovane. Existuje ďalší prístup a to je rozdeliť aplikáciu na samostatné aplikácie, respektíve samostatné moduly. Ak by boli samostatné aplikácie bol by problém so zdielaním prostriedkov a pamäte, keďže všetky bežiace inštancie aplikácie majú vlastný pamäťový priestor. Tretí prístup je tvorba modulov, pričom budú mať prístup do jednej pamäte a bude menší problém so zdielaním informácii a dá sa jednoducho testovať.

Návrh riešenia
Navrhujeme vytvoriť 4 projekty, 3 moduly, jeden pre simuláciu agentov, jeden pre prostredie a jeden pre vizualizáciu a jeden obaľovač ktorý bude spúšťať moduly. Moduly budú komunikovať cez spoločný komunikačný kanál ktorý bude dostatočne abstraktný. V prípade snahy rozdeliť aplikáciu na samostatné aplikácie a nie moduly bude treba jedine zmeniť konfigurácia projektu a zmeniť implementáciu kanála.

Opis implementácie
Projekt vizualizácia davu bol rozdelený na 4 samostatné projekty. 3 dynamické knižnice a jeden obalovač ktorý tieto knižnice spúšta. Komunikácia medzi knižnicami sa robí cez vytvorený komunikačný kanál. Jeden je na strane prostredia a pre každý modul inicializovaný sa vytvorí kanál pre agenta a vizualizáciu osobitne. Keďže je predpoklad, že moduly ktoré budú simulovať agentov bude spúšťať prostredie tak prostredie dostáva na vstupe pri konštrukcii alokátor, ktorý spustí modul pre simuláciu agenta. Tiež dostane na vstupe alokátor, ktorý priradí prostredím spusteným modulom kanál, cez ktorý bude prostredie spolu komunikovať s modulom.

4.6 Detekcia pretínajúcich sa stien



Analýza problému
Pomocou uhla natočenia agenta a veľkosti kroku vieme určiť, aká bude nasledujúca poloha agenta. Spojením pôvodnej a novej pozície agenta dostaneme úsečku, po ktorej sa bude pohybovať. Táto úsečka sa nesmie pretínať so žiadnou stenou v mape prostredia. Vylepšenie tohto algoritmu spočíva namiesto použitia jednej pohybovej úsečky dvoch, kde každá z nich začína a končí na bokoch agenta. Tým sa vylúči, že

by agent prešiel cez medzeru, do ktorej sa v podstate nezmestí.

Návrh riešenia
Prvoradé je vytvorenie nástroja, ktorý nám jednoducho povie, či sa dve úsečky pretínajú alebo nie. Pomocou tohto nástroja jednoducho implementujeme potrebnú funkcionalitu. K samotnému nastaveniu pozície dôjde pomocou LocationController-a. To je trieda zodpovedná za zmenu pozície agenta. Každý agent ma niekoľko definovaných niekoľko kontrolerov, pričom v opise implementácie bude pozornosť upriamená iba na jeden samostatný kontroler.

Opis implementácie
Detekcia pretínajúcich sa úsečiek bola realizovaná pomocnou triedou CUtils, pomocou ktorej sa dá jednoducho zistiť, či sú úsečky rovnobežné, totožné, alebo či sa pretínajú. Ak sa pretínajú, je možné zistiť aj bod, v ktorom k tomu dochádza. Pre každého agenta sú následne určené jeho bočné body predstavujúce v abstraktnom ponímaní pozície ramien človeka. Tie závisia od pozície agenta v prostredí a jeho aktuálneho natočenia. Z týchto bodov boli skonštruované úsečky v smere natočenia o veľkosti jedného kroku. Každá z týchto úsečiek je následne porovnávaná so stenami v mape. Ak je zistený prienik, pohyb agenta skončí na tej stene, ktorá je k nemu najbližšie. Nakoniec treba vykonať úpravu pozície tak, aby sa agent dotýkal steny, ale zároveň dodržal naplánovaný smer. Tento odstup od konkrétneho priesečníka závisí od uhla agenta vzhľadom na stenu, na ktorú narazil. Pri kolmom náraze je táto úprava najväčšia, pri malom uhle je minimálna.

4.7 Riešenie kolízií agentov

Úloha nebola v čase ukončenia šprintu splnená.

4.8 Generovanie agentov do mapy

Návrh riešenia
V mape, ktorá sa inicializuje pri štarte aplikácie sa nachádzajú plochy určené na generovanie agentov. Tieto plochy (obdĺžniky) sú rozdelené na mriežku, kde jedna bunka má veľkosť 0,5 metra. Táto veľkosť reprezentuje maximálnu šírku agenta. Do kontajnera startPositions sú pridané všetky možné štartovacie pozície agentov. Následne sa v časti generovania štartovacej pozície generuje číslo v intervale 0 po veľkosť kontajnera startPositions.

Opis implementácie

Pri štarte aplikácie sa cez parameter konštruktora inicializuje mapa pre prostredie. Pomocou tejto mapy sa vypočítajú všetky možné umiestnenia agentov (tak aby nenastavali kolízie) a tieto pozície sa uložia do kontajnera startPositions. Pri vytváraní jednotlivých agentov sa skontroluje, či sa v kontajneri startPositions nachádzajú ešte voľné pozície - ak áno, tak sa vygeneruje náhodná pozícia v danom kontajneri. Z tejto pozície sa vyberú informácie o pozícií na mape a priradia sa novému agentovi. Daná pozícia sa následne odstráni z kontajnera startPositions.