Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Inteligentná hra pre mobilné zariadenia

(Technická dokumentácia)

Tím č. 14

Pedagogický vedúci: *Ing. Marek Tomša*
Kontakt: *fiit-team14@googlegroups.com*
Študijný odbor: *SI/IS*
Akademický rok: *2011/2012*
Dátum: *9.11.2011*

Autori: Bc. MátéFejes
Bc. Ľuboš Gelányi
Bc. Ľuboš Masný
Bc. Juraj Mäsiar
Bc. Adam Mihálik
Bc. Dávid Pszota

2. odovzdanie v ZS (1., 2.,3. a 4. šprint)

Obsah

1 Úvod

1.1 Účel dokumentu

Tento dokument je vytvorený pre účel projektovej dokumentácie projektu *Inteligentná hra pre mobilné zariadenia* v rámci predmetu Tímový projekt na FIIT STU. V dokumente sú zdokumentované jednotlivé priebehy šprintov metodiky SCRUM.

1.2 Cieľ projektu

Príchodom internetu a vyspelých komunikačných technológií sa osobný kontakt ľudí pomaly dostáva do útlmu. Čím ďalej, tým viac staviame elektronickú komunikáciu pred osobnú. Ako mladý ambiciózny tím programátorov by sme chceli pomôcť nám ľudom k tomu, aby sme sa socializovali i naďalej prostredníctvom osobného kontaktu. Ako socializačný nástroj by sme chceli využiť inteligentnú hru, ktorá by nás motivovala k stretávaniu sa s ľuďmi. Avšak naším prioritným cieľom je posunúť inteligenciu hier o krok ďalej, aby agenti „rozmýšľali“ a rozhodovali sami za seba.

Cieľom projektu je vytvoriť inteligentnú hru, v ktorej by sa inteligentný agent správal podľa predstáv hráča, ktoré mu postupne vštepil prostredníctvom trénovania. Chceme postaviť používateľa do virtuálneho sveta nie ako hráča, ale ako „učiteľa” za pomoci najnovších poznatkov o umelej inteligencii.

2 História zmien

V tabuľke Tab. 1 História zmien dokumentu***Tab. 1*** sú zobrazené jednotlivé modifikácie dokumentu, ktoré vykonali členovia tímu v danom období.

Dátum	Kapitola	Autor
3.11.2011	Vytvorenie dokumentu	Ľuboš Gelányi
3.11.2011	1. Úvod	Ľuboš Gelányi
3.11.2011	2. História zmien	Ľuboš Gelányi
7.11.2011	3.2 Menu a ovládanie	Ľuboš Gelányi
7.11.2011	3.1 P2P komunikácia	Dávid Pszota
7.11.2011	4.3 Prototypovanie sieťovej komunikácie	Dávid Pszota
7.11.2011	3.3 Stratégie umelej inteligencie	Juraj Mäsiar
7.11.2011	3.3 Stratégie umelej inteligencie	Ľuboš Masný
7.11.2011	4.1 Herné Prostredie	MátéFejes
7.11.2011	4.2 Algoritmus NEXT	Adam Mihalik
5.12.2011	5.1 Výber hráča a mapy	Ľuboš Gelányi
5.12.2011	5.2 Manažment máp	MátéFejes
5.12.2011	5.3 Neurónová sieť so spätným šírením chyby	Ľuboš Masný
5.12.2011	5.4 Markovovské siete	MátéFejes
5.12.2011	5.5 Multiplayer	Dávid Pszota
5.12.2011	5.6 EvilBot	Juraj Mäasiar
5.12.2011	6.1 Výbuch bomby	Dávid Pszota
5.12.2011	6.2 Herné režimy a navigácie	Ľuboš Gelányi
5.12.2011	6.3 Neurónová sieť - vyhodnotenie nasl. kroku	Ľuboš Masný
5.12.2011	6.4 Markovovské siete	Adam Mihalik
5.12.2011	6.5 Grafika, animácia hráča	Ľuboš Gelányi
5.12.2011	6.6 Analyzátor mapy	Juraj Mäsiar
5.12.2011	6.7 Box DestroyerDefensiveBot	Juraj Mäsiar
6.12.2011	7 Opis prototypu	Ľuboš Masný
12.12.2011	Revízia dokumentu	Juraj Mäsiar

Tab. 1 História zmien dokumentu

3 Šprint č.1

Číslo šprintu	**01**
Začiatok šprintu	13.10.2011
Koniec šprintu	26.10.2011
Príbehy (*userstories*)	• P2P komunikácia • Menu a ovládanie • Stratégie umelej inteligencie

3.1 P2P komunikácia

Riešenie sieťovej komunikácie bolo zamerané na testovanie existujúcej implementácie komunikačných protokolov architektúry klient - server. Prvotný prístup bol zameraný na štandardnú *TCP/IP* komunikáciu medzi *PC -PC, PC - Android*, *Android - Android*. Analýza týchto prístupov odhalila mierne nedostatky v technológiách.

3.1.1 Možnosti riešenia

3.1.1.1 Komunikácia cez TCP/IP socket

Použiteľná je len *klient-server* architektúra. Protokol je dostatočne rýchly ale vyskytujú sa chyby pri naviazaní *TCP* spojenia. Je potrebné implementovať vlastný protokol nad *TCP* vrstvou. *Klient-klient* komunikácia je možná v prípade ak sú obidve zariadenia v rovnakej podsieti.

Výhody:

- rýchla výmena dát
- multiplatformové (*TCP/IP* je sieťový štandard)

Nevýhody

- otvorený port na server
- upravený protokol nad TCP

UDPpackety boli taktiež zvážené.

3.1.1.2 PeerDroid

Knižnica vyvinutá pre *peer to peer (P2P)* komunikáciu. Protokol je založený na *JTXA*, ktorý umožňuje multiplatformové využitie. Jedná sa o *klient-klient* komunikáciu, pre ktorú je potrebný "*rendezvous*" server (*RDV*), ktorý zabezpečuje manažment komunikácie. *RDVserver* musí byť spustení na počítači v lokálnej sieti. Pre možnosť hrania cez internet je potrebné spúšťať server na stroji s verejnou IP, alebo cez *DNS server*.

Výhody:

- slabé vyťaženie servera
- reálne *P2P* spojenie

Nevýhody

- implementácia *RDV servera*
- knižnica vyvinutá iba pre *Android*

3.1.1.3 Hessain

Multiplatformová knižnica založená na *HTTP* protokole. Prenos binárnych dát, rôzne možnosti implementácie (*C#, PHP, Java, Objective C*). *iPhone* a *Windows mobile* sú taktiež podporované.

Výhody:

- multiplatformové
- existujúce implementácie pre rôzne platformy

Nevýhody

- nejedná sa o reálnu *P2P* komunikáciu (komunikáciu a výmenu dát zabezpečuje server)

3.1.1.4 AndEngine – multiplayerextension

Framework pre *TCP/IP*. Možnosť herného módu pre viac hráčov cez *Bluetooth*s *AndEngine*.Stále je vo vývoji, nie je možné určiť spoľahlivosť.

Je možné vytvoriť privátnu *Wi-Fi* sieť pomocou *AP* (*Access point*) vytvorené *androidom*. Využiteľnosť overená testovaním. Je potrebná verzia *Androidu 2.2* (*API*level 8). Vytvorenie prístupového bodu nie je možné programátorsky (iba ručne, s používateľovou interakciou).

Výhody:

- podpora *Bluetooth*
- *Framework*

Nevýhody

- *Klient-server* komunikácia a zisťovanie servera cez *Wi-Fi* je možné len ak sú obe zariadenia na spoločnej *LANsieti*.

3.2 Menu a ovládanie

Menu a ovládanie sme sa rozhodli riešiť pomocou grafického *engine*-u pre Android*AndEngine,* ktorý poskytuje omnoho väčšie možnosti ako klasické *Android API*.

3.2.1 Menu

Menu (*Obr. 1*) je potrebné implementovať pre potreby herného prostredia. Hra musí byť orientovaná na šírku (*landscape*) preto sa menu vytvorilo taktiež v horizontálnej podobe. Obsahuje základné bitmapové textúry poskytované v rámci knižnice *AndEngine*. Je rozdelené do piatich položiek, ktoré je možné podľa potreby dopĺňať:

- *Single player* - jednoduchá hra proti AI
- *Control Test* - položka určená pre testovanie
- *Multiplayer* - hra s viacerými hráčmi (jedným ľudským a prípadne ďalšími *botmi* prípadne *AI*)
- *Options* - položka určená pre úpravu nastavení
- *Quit*- ukončenie hry

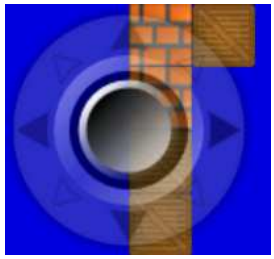
Obr. 1 Menu hry

3.2.2 Ovládanie

Ovládanie sa musí taktiež prispôsobiť možnostiam a povahe vyvíjanej hry. Do úvahy pripadá viacero možných variant ovládania:

- *Ovládanie pomocou Gyroskopu* - nie je potrebné pre 2D hru, náročnejšia implementácia. Nepresné pri krokovaní agenta.
- *Hardvérový Joyistick* - nepripadá do úvahy pretože nie všetky Android zariadenia ho obsahujú.
- *Intuitívne ovládanie* - teda presné určovanie pozície, na ktorú sa má hráč premiestniť. Toto riešenie priamo podporuje *AndEngine* avšak javí sa ako trochu neintuitívne a málo flexibilné pre potreby rýchlej reakcie agenta. Taktiež by mohli nastať konflikty pri obchádzaní prekážok, kde by agent mohol zvoliť rôzne smery (toto je úlohou len inteligentného agenta)
- *Softvérový joystick* - taktiež podporovaný *Andengine-om*. Veľmi intuitívne a presné. Drobné problémy s pokrytím pravého dolného rohu obrazovky. Je možné vyriešiť transparentným komponentom.

Z uvedených možností bol vybratý Softvérový *joystick* (*Obr. 2*) pre jeho kladné vlastnosti a intuitívne ovládanie.



Obr. 2 Ovládanie

Ovládanie sa skladá z dvoch komponentov

- *softvérový joystick* (v ľavej dolnej časti obrazovky)
- tlačidlo pre akcie - uloženie bomby (v pravej dolnej časti obrazovky)

3.3 Stratégie umelej inteligencie

UI sa dostáva ku slovu po tom, ako *Player* zavolá funkciu *signal()*. Nastáva výpočet optimálnej akcie, ktorú následne nastaví do premennej *action*. Následne, keď *Player* zavolá funkciu *getAction()*, vracia UI akciu *action* na vykonanie.

Hlavnú rozhodovaciu funkciu v celej UI hrá neurónová sieť. Naučená neurónová sieť bude v závislosti od vstupov poskytovať nasledujúce akcie v danej situácií. Ako vstupy sú: pozícia oboch agentov, vzdialenosť súpera a prekážky medzi nimi (tieto vstupy budú postupne rozširované, ako sa bude UI vyvíjať). Výstupom z neurónovej siete je rozhodnutie, či je vhodné položiť bombu alebo spraviť pohyb. Ak sa položí bomba, tak sa následné snaží dostať do bezpečia (miesto na mape, kde nie je ohrozený žiadnou bombou).

Samotné učenie bude prebiehať pomocou spätného šírenia chýb, pričom na začiatku sa bude snažiť napodobniť jeden zo základných stereotypov. Učenie, teda zdokonaľovanie hracieho mozgu agenta, sa bude uskutočňovať až po odohraní zápasu. Je to nevyhnutné na optimalizáciu rýchlosti priebehu duelu, keďže je aplikácia určená pre mobilné zariadenia. To znamená, že jednotlivé situácie počas odohrávania zápasu sa zaznamenávajú a vyhodnocujú neskôr. Akcie, ktoré viedli k víťazstvu (môže to byť jedna akcia alebo viacero) budú v danej situácií v nasledujúcom súboji vybrané s väčšou pravdepodobnosťou. Naopak, akcie ktoré viedli k prehre, budú vyberané s menšou pravdepodobnosťou.

Rozhodnutie akým smerom sa agent vyberie má na starosti A* algoritmus. Po vyhodnotení od neurónovej siete, že je nutné dostať sa k súperovi, A* nájde optimálnu cestu (jedno políčko od súpera, ktoré je k súperovi najbližšie). Túto cestu sa snaží dodržať, kým nenastane veľká zmena (napr. bomba v ceste, zmena pozície súpera, atď.).

V prípade, že je agent ohrozený bombou (nezáleží čí je jeho vlastná alebo súperova), tak pomocou algoritmu A* sa snaží nájsť optimálnu cestu na miesto, kde nie je ohrozený. V prípade, že je takýchto miest viacej, snaží sa dostať čo najďalej od súpera (toto je prvotná inteligencia, kedy sa agent snaží byť v čo najmenšom ohrození. V neskoršej fáze bude možnosť ísť aj smerom k súperovi, aby ho mohol čo najskôr opäť atakovať).

3.3.1 Analýza technológií na urýchlenie práce neurónových sietí

Práca neurónových sietí vyžaduje často len základné matematické operácie (sčítanie, odčítanie, násobenie a delenie) vykonávané na veľkom množstve dát. Je teda vhodné použiť SIMD inštrukcie (single instruction, multipledata).

Tieto sa na ARM procesoroch vyskytujú od verzie 6 kde sú transparentné pre OS. Tieto by mali zvládať spracovávať naraz 4 x 8 bitov alebo 2 x 16 bitov.

Od verzie 7 procesorov ARM sú prítomné *NEON* inštrukcie, ktoré už nie sú transparentné pre OS. Tieto zvládajú prácu až s 16-timi registrami s veľkosťou 128 bitov.

Pre použitie SIMD inštrukcií je vhodné použiť niektorú z existujúcich knižníc. Ako najlepšia voľba sa javí opensource C++ knižnica *Eigen*. Táto je momentálne vo verzii 3.0.3 (6.10. 2011) a je učená na prácu s maticami (ideálne pre použitie s neurónovými sieťami). Automaticky využíva dostupné SIMD inštrukcie ako *SSE 2/3/4*, *ARM NEON* a *AltiVec*. Knižnica je pritom tvorená len hlavičkovými súbormi, využívajúc C++ standardlibrary.

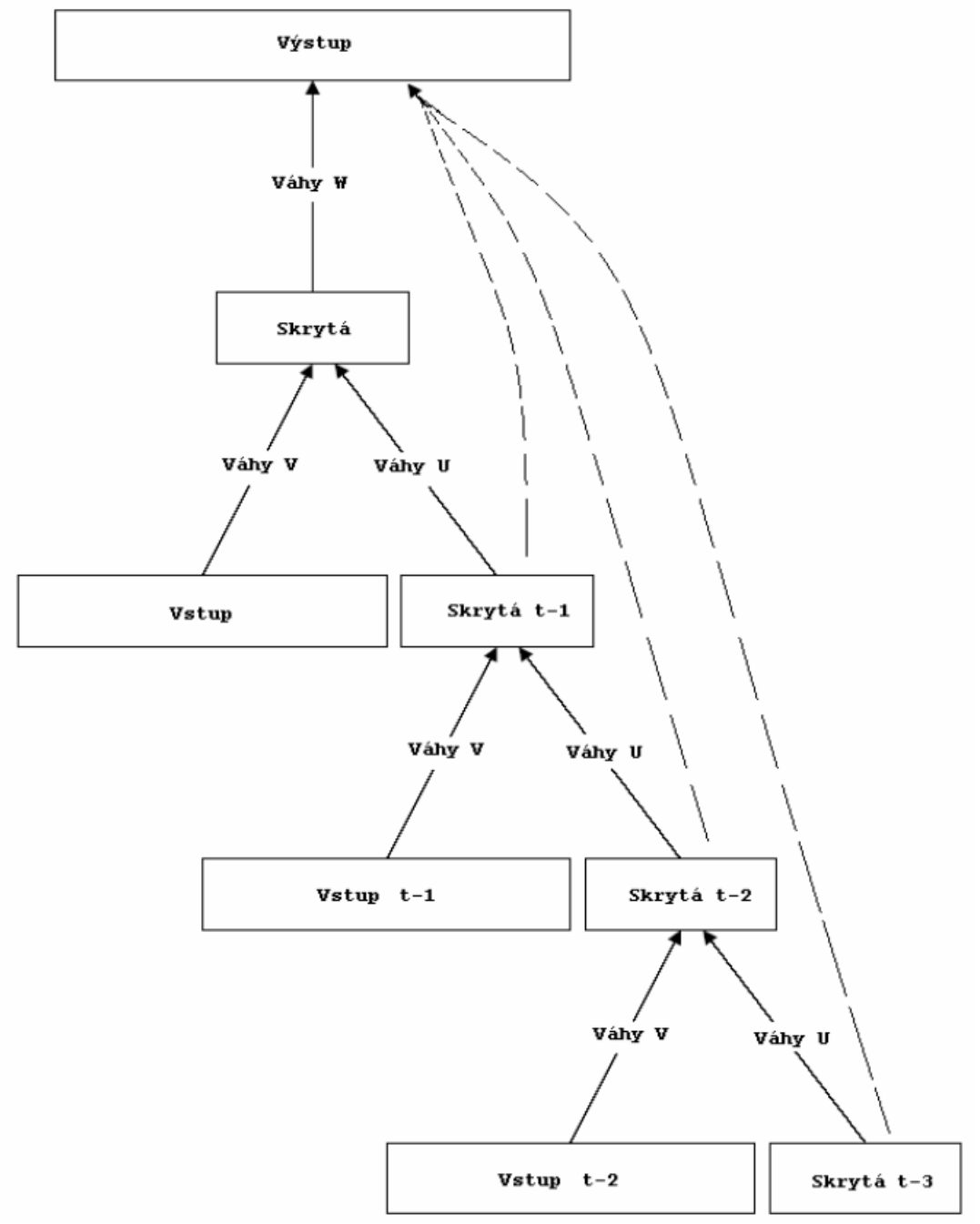
Túto knižnicu by sme teda mohli využiť pri implementácii servera, na ktorom budú prebiehať súboje. Nakoľko ten bude implementovaný v Jave, bude nutné použiť JNI (Javanativeinterface) pre integráciu C++ kódu. Treba podotknúť, že Java momentálne nemá podporu pre SIMD inštrukcie, použitie C++ knižnice je teda takmer jediná možnosť ako urýchliť výpočty matíc (urýchlenie počítania matíc je pritom až niekoľko-násobné).

Viac informácií o technológiách:

Eigen:
http://eigen.tuxfamily.org/index.php?title=Main_Page
ARM NEON:
http://www.arm.com/products/processors/technologies/neon.php

3.3.2 Navrhované fungovanie mozgu agenta

Jednou z možností ako vytvoriť inteligentného agenta je použitím neurónových sietí. Tie sa delia na niekoľko skupín, ja som sa rozhodol pre rekurentnú neurónovú sieť. Tá si uchováva informáciu o svojich aktivitách z predošlých stavov. Je teda vhodné použiť ju pri problémoch s časovou súvislosťou. Jej základné fungovanie môžeme vidieť na obrázku *Obr. 3*.



Obr. 3 Fungovanie rekurentnej neurónovej siete

Môžeme vidieť tri vstupy z troch po sebe idúcich situácií. Podstatou je, že aktuálny výstup neurónovej siete sa použije ako vstup pri ďalšom použití siete spolu s normálnym vstupom. Sieť sa teda rozhoduje nielen na základe aktuálneho vstupu ale aj na základe predchádzajúcich vstupov.

V našom prípade môžu byť vstupom do neurónovej siete tieto vlastnosti aktuálneho stavu:

- som ohrozený bombou
- môžem byť ohrozený bombou
- koľko času ostáva do vybuchnutia bomby
- ako ďaleko je najbližšie bezpečné políčko
- je možné sa dostať k súperovi
- ako ďaleko je políčko, z ktorého je možné ohroziť súpera

Výstupom potom môžu byť nasledujúce stratégie:

- choď na najbližšie bezpečné políčko
- polož bombu na políčko, odkiaľ je možné ohroziť súpera
- polož bombu na políčko, odkiaľ je možné zničiť stenu

Tieto vstupy a výstupy sú len približné, budú sa v priebehu vývoja hry meniť. Hlavne sa budú pridávať nové s pribúdajúcimi možnosťami hráča (hádzať bombu, odkopnúť bombu a iné).

Použitá literatúra

Juraj Koščák, *Experimentálna analýza algoritmu backpropagationthroughtime pre určenie rekurentných neurónových sietí*, dostupné na internete:
http://neuron.tuke.sk/~jaksa/theses/2005/Koscak-Jaksa-MSc05-thesis.pdf

4 Šprint č.2

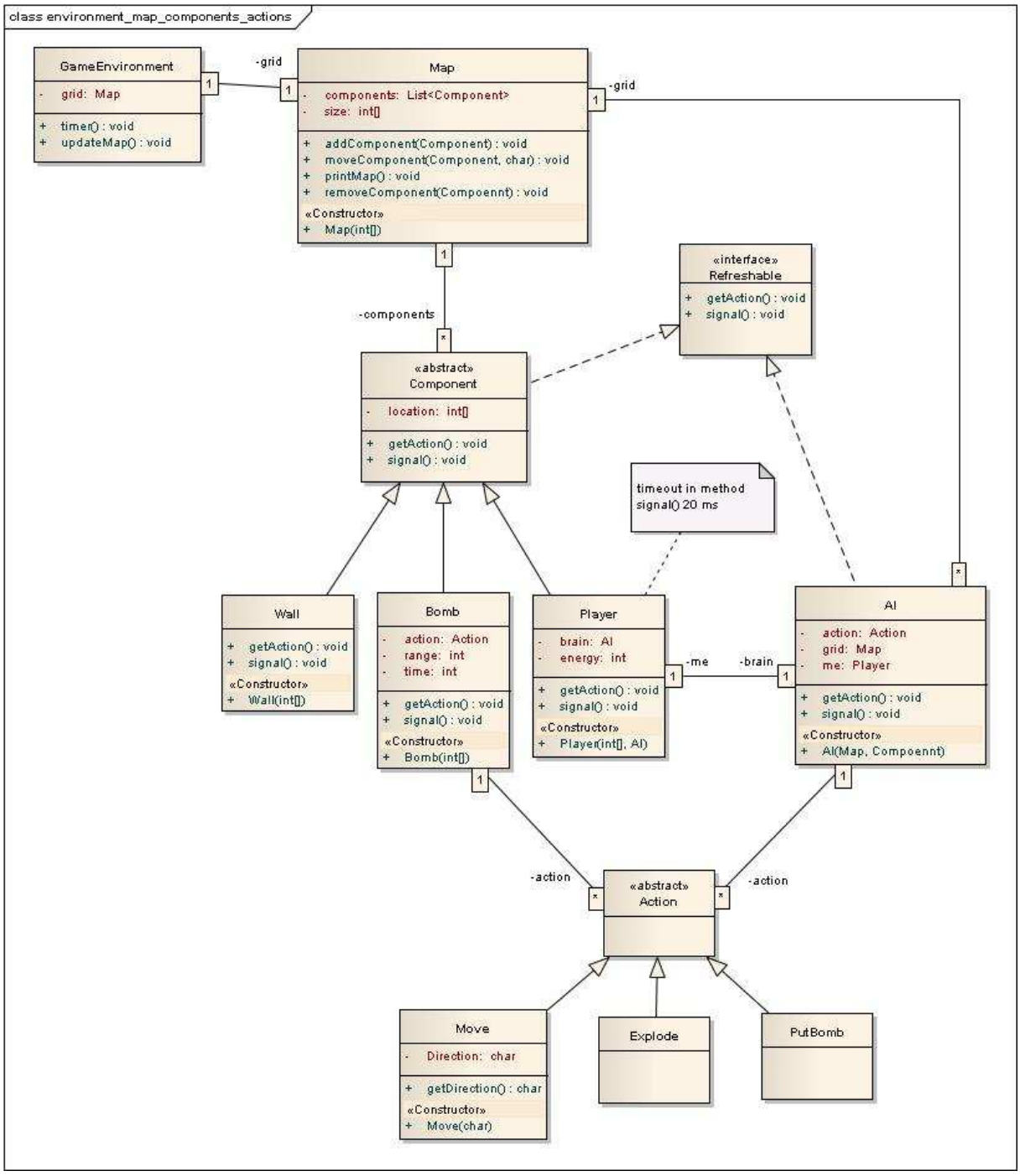
Číslo šprintu	**02**
Začiatok šprintu	26.10.2011
Koniec šprintu	09.11.2011
Príbehy (*userstories*)	• Herné prostredie • NEXT algoritmus (mód hry: získanie vlajky) • Prototypovanie sieťovej komunikácie

4.1 Herné prostredie

Tento návrh slúži na vytvorenie základného rámca hry. Výsledkom implementácie musí byť prototyp, ktorý podporuje nasledujúce služby:

- Vytvorenie mapy a komponentov na nej.
- Vytvorenie herného prostredia, ktoré slúži ako kontrolná vrstva nad všetkými entitami.
- Vytvorenie umelej inteligencie, ktorá má za úlohu riadenie daného komponentu typu hráč.
- Umelá inteligencia musí zvládnuť jedinú základnú úlohu, nájsť cestu medzi dvoma bodmi, prípadne položiť bombu na aktuálne miesto hráča.
- Herné prostredie musí byť schopné spracovať a zosynchronizovať akcie hráčov a iných komponentov.

4.1.1 Základná štruktúra entít



Obr. 4 Relácia tried

Model na *Obr.4* obsahuje len základné objekty. Pri implementácii môžu byť doplnené pomocnými triedami, metódami a atribútmi.

4.1.1.1 GameEnvironment

- Herné prostredie.
- V rámci nej beží jediný časovač.
- Postupne posiela signál komponentom a následne si od nich pýta požadovanú akciu, ktorú vykoná na mape.
- Signál naznačuje komponentu, že je na rade a môže si vygenerovať akciu.
- Atribúty:
  - *grid* – inštancia triedy *Map.*
- Metódy:
  - *updateMap* – vykonanie požadovaných akcií. V rámci tejto metódy sa zavolajú rôzne metódy triedy *Map*. Výber metód závisí od typu objektu *Action,* ktorý je prijatý ako návratová hodnota metódy *getAction* daného komponentu.
    - Pokiaľ je objekt *Action* typu *Move*, zavolá sa metóda *moveComponent* pre príslušný komponent.
    - Pokiaľ je objekt *Action* typu *PutBomb*, zavolá sa metóda *addComponent* pre príslušný komponent (položenie bomby na aktuálne miesto hráča, ktorý akciu vrátil).
    - Pokiaľ je objekt *Action* typu *Explode*, zavolá sa metóda *removeComponent* pre príslušné komponenty (pre bombu, ktorá akciu vrátila a pre všetky komponenty v jej rozsahu).

4.1.1.2 Refreshable

- Rozhranie pre všetky triedy, ktoré priamo alebo nepriamo komunikujú s herným prostredím.
- Metódy:
  - *signal* – naznačenie, že je objekt na rade, môže si vygenerovať akciu a uložiť si do premennej. Metóda *signal* je blokujúca, ale môže trvať len vopred definovaný čas. Metóda signal musí v rámci implementujúcej triedy zavolať ďalšie metódy, ktoré vytvoria a uložia požadovanú akciu.
  - *getAction* – po zbehnutí metódy *signal* daného objektu herné prostredie požiada diskutovaný objekt o poskytnutie vygenerovanej akcie. Návratová hodnota tejto metódy je objekt typu *Acion*.

4.1.1.3 Component

- Abstraktná trieda, reprezentuje všetky predmety, ktoré sa na mape nachádzajú.
- Každý komponent musí komunikovať s herným prostredím, preto táto abstraktná trieda implementuje rozhranie *Refreshable*.
- Atribúty:

- *location* – poloha komponentu. Pole celých čísel, vždy obsahuje 2 prvky (x, y súradnica).

4.1.1.4 Player

- Reprezentuje hráča – môže byť riadený používateľom cez vstupné zariadenie, alebo umelou inteligenciou.
- Atribúty:
  - *brain* – ukazovateľ na umelú inteligenciu (objekt typu AI), ktorá hráča riadi
  - *energy* – celé číslo, reprezentuje energiu (zatiaľ 1, ak je živý, 0, ak je mŕtvy).
- Metódy:
  - Konštruktor: parametrom je poloha (pole celých čísel a ukazovateľ na AI).
  - *signal* – implementovaná z rozhrania *Refreshable*. V prípade hráča signál má časový limit 20ms, týmto sa zabezpečuje umelej inteligencii potrebný čas a generovanie akcie. V rámci tejto metódy sa signál postúpi umelej inteligencii – zavolá sa metóda *signal* z *AI*.
  - *getaction* – získanie vygenerovanej akcie od umelej inteligencie.

4.1.1.5 Bomb

- Bomba položená na určité miesto mapy.
- Atribúty:
  - *range* – celé číslo, vyjadruje rozsah, v ktorom výbuch má vplyv, t.j. v ktorom zničí ostatné komponenty.
  - *time* – čas od položenia do výbuchu.
- Metódy:
  - *signal* – implementovaná z rozhrania *Refreshable*. V rámci tejto metódy sa dekrementuje hodnota premennej *time*.
  - *getAction* – implementovaná z rozhrania *Refreshable*. Ak je hodnota premennej *time* rovná nule, vytvorí a vráti inštanciu triedy *Explode*, v opačnom prípade vráti NULL.

4.1.1.6 Map

- Reprezentuje mriežku daného rozmeru.
- Obsahuje zoznam inštancií triedy *Component.*
- Podporuje pridávanie, mazanie a posúvanie komponentov.
- Atribúty:
  - *components* – zoznam inštancií triedy *Component.*
  - *size* – rozmery mriežky. Pole celých čísel, vždy obsahuje 2 prvky (šírka, výška).
- Metódy
  - *addComponent* – inštanciu typu *Component* prijatú v parametre pridá do zoznamu *components*

- *removeComponent* – zo zoznamu *components* vymaže prvok, na ktorý dostal ukazovateľ v parametre.
- *moveComponent* – v parametre dostane:
  - ukazovateľ na komponent, nad ktorým treba operáciu vykonať
  - znak reprezentujúci smer posunu

Táto metóda zavolá metódu *setLocation* pre daný komponent a prepíše jeho polohu.

4.1.1.7 AI

- Reprezentuje umelú inteligenciu, t.j. mozog daného hráča.
- Má svoju heuristiku, dokáže rozhodovať v danej situácii a posielať požadované akcie hernému prostrediu.
- Prototyp AI musí byť schopný nájsť optimálnu cestu medzi dvoma bodmi a položiť bombu na aktuálne miesto hráča, ktorého riadi.
- Nepriamo komunikuje s herným prostredím cez objekt typu *Player*, preto implementuje rozhranie *Refreshable*.
- Atribúty:
  - *action* – premenná typu *Action*. Tu je uložená vygenerovaná akcia, kým sa nepošle hernému prostrediu.
  - *me* – premenná typu *Player*. Ukazuje na hráča, ktorého riadi.
  - *grid* – mapa, na ktorej sa hráč nachádza.
- Metódy:
  - Konštruktor – uloží ukazovateľ na hráča (komponent), ktorého riadi a na mapu, na ktorej sa hráč nachádza.
  - *signal* – Volá sa z objektu typu *Player* – postúpenie signálu od herného prostredia. Umelá inteligencia zváži situáciu na mape a v závislosti od stavu si nastaví atribút *action* na zodpovedajúcu hodnotu.
  - *getAction* – vráti hodnotu atribútu *action.*

4.1.1.8 Action

- Abstraktná trieda pre všetky možné akcie, ktoré môžu byť v hernom prostredí spracované.

4.1.1.9 Move

- Trieda implementujúca rozhranie *Action.*
- Reprezentuje posun komponentu o 1 miesto.
- Atribúty:
  - *direction* – znak (A, B, C alebo D), vyjadruje smer posunu.
    - *A* – posun hore
    - *B* – posun dole
    - *C* – posun vpravo

- *D* – posun vľavo

4.1.1.10 PutBomb

- Položenie bomby na aktuálne miesto hráča (komponentu), ktorý túto akciu vygeneroval.

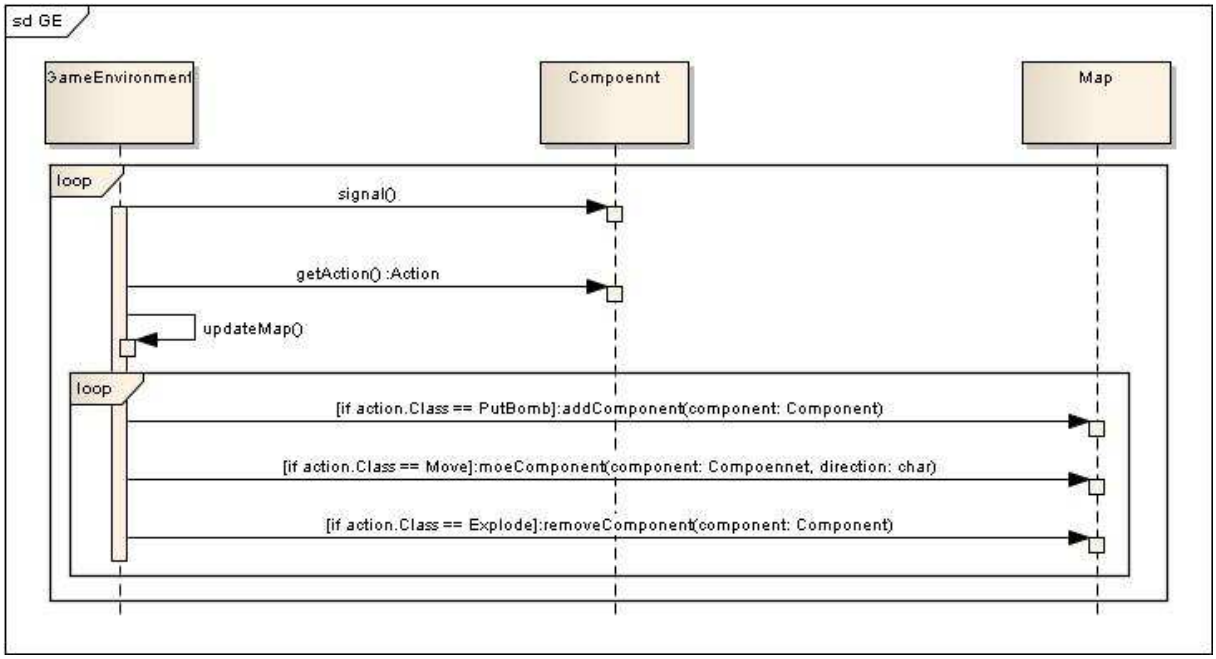
4.1.1.11 Explode

- Vybuchnutie bomby (komponentu), ktorá túto akciu vygenerovala.

4.1.2 Sekvencia krokov

Táto kapitola opisuje postupnosť krokov (poradie volania metód). Prvý diagram znázorňuje základnú činnosť herného prostredia, resp. jeho komunikáciu s komponentmi. Ďalšie diagramy tiež charakterizujú túto činnosť s rozdielom, že každý z nich je špecifický pre jeden vybraný typ komponentu.

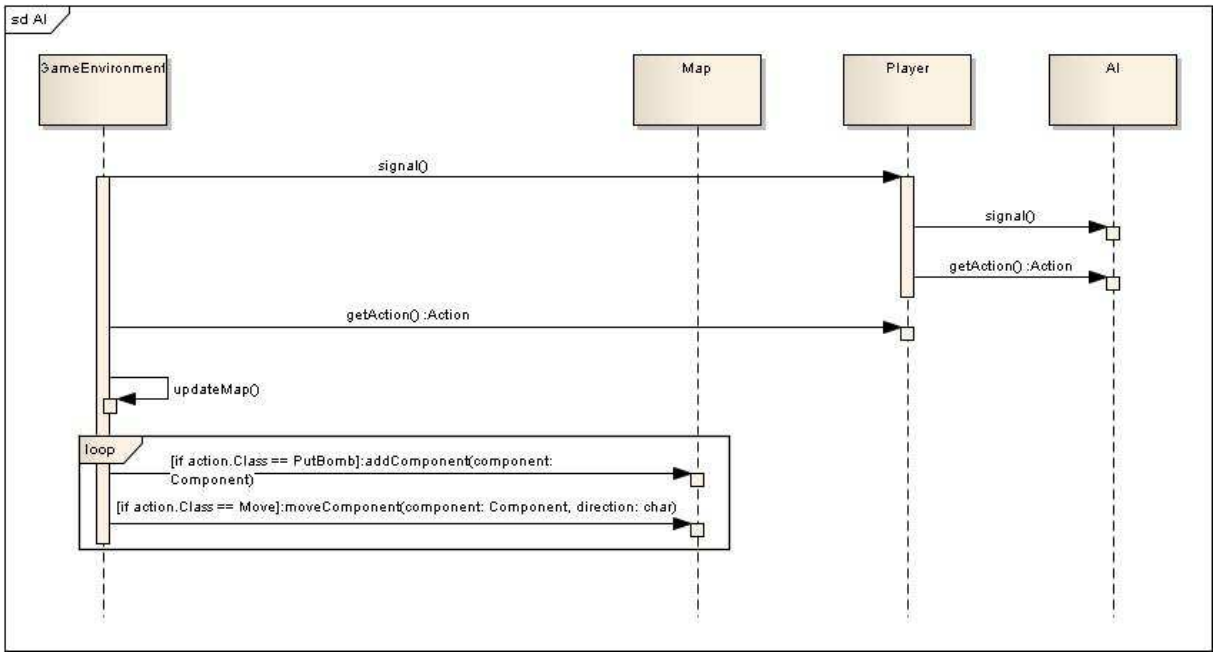
4.1.2.1 Základný postup vykonávania



Obr. 5 Postupnosť krokov v rámci behu

1. Herné prostredie pošle signál komponentu, ktorý je na rade.
2. Komponent si vygeneruje novú akciu.
3. Herné prostredia pošle komponentu žiadosť o poskytnutie akcie (*getAction*).
4. Zavolá sa *updateMap*, v rámci ktorej sa zavolajú metódy mapy, ktoré zodpovedajú typu danej akcie.
5. Opakujú sa kroky vyššie pre každý komponent.

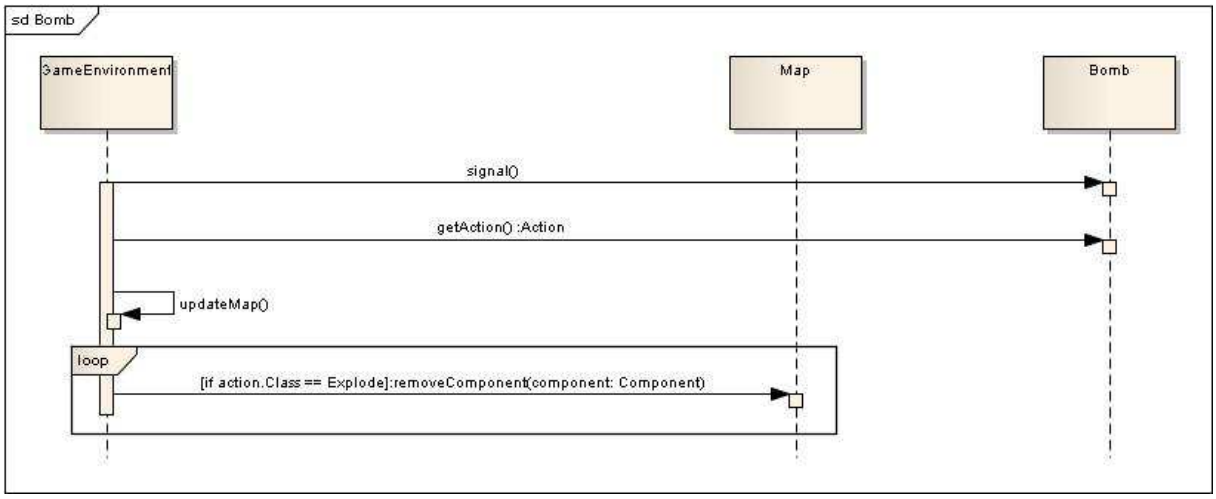
4.1.2.2 Akcie hráča (umelej inteligencie)



Obr. 6 Postupnosť krokov AI

1. Herné prostredie pošle signál komponentu typu *Player*.
2. *Player* postúpi signál umelej inteligencii, ktorá si vygeneruje akciu v závislosti od situácie na mape.
3. *Player* pošle žiadosť umelej inteligencii o poskytnutie akcie.
4. Herné prostredie pošle žiadosť hráčovi o poskytnutie akcie získanej od *AI*.
5. Zavolá sa *updateMap* ...(ďalej viď. na diagrame č. 2, krok 4).

4.1.2.3 Akcie bomby



Obr. 7 Postupnosť krokov AI

1. Herné prostredie pošle signál komponentu typu *Bomb*.
2. Herné prostredia pošle bombe žiadosť o poskytnutie akcie. Bomba vráti akciu typu *Explode*, alebo NULL.
3. Zavolá sa *updateMap*, v rámci ktorej sa zavolá *removeComponent* pre bombu a pre všetky komponenty, ktoré sa nachádzajú v rozsahu bomby.

- som ohrozený bombou
- môžem byť ohrozený bombou
- koľko času ostáva do vybuchnutia bomby
- ako ďaleko je najbližšie bezpečné políčko
- je možné sa dostať k súperovi
- ako ďaleko je políčko, z ktorého je možné ohroziť súpera

Výstupom potom môžu byť nasledujúce stratégie:

- choď na najbližšie bezpečné políčko
- polož bombu na políčko, odkiaľ je možné ohroziť súpera
- polož bombu na políčko, odkiaľ je možné zničiť stenu

Tieto vstupy a výstupy sú len približné, budú sa v priebehu vývoja hry meniť. Hlavne sa budú pridávať nové s pribúdajúcimi možnosťami hráča (hádzať bombu, odkopnúť bombu a iné).

4.2 Algoritmus Next

Základ algoritmu pozostáva v pôvodnom algoritme A*. Počas hľadania najkratšej cesty k dosiahnutiu cieľa je ale v špecifických prípadoch potrebné uvažovať zmenu stavu skúmaného prostredia ale len pre aktuálne skúmaný stav a pre všetky ďalšie stavy vzniknuté z tohto stavu.

4.2.1 Základ algoritmu NEXT pomocou A*

Algoritmus A* je určený na prehľadanie stavového priestoru a jeho cieľom je nájsť určitú cestu od počiatočného stavu k cieľovému. V hľadaní tejto cesty využíva ohodnotenie skúmaného stavu, čím je následne schopný uprednostňovať preskúmanie tých stavov, ktoré potenciálne privádzajú hľadanie k cieľu rýchlejšie ako iné skúmané stavy. Týmto spôsobom neprehľadáva celý stavový priestor, ale iba je podoblasť, v ktorej sa riešenie nachádza. Nutnou podmienkou pre vstup tohto algoritmu je schopnosť ohodnotiť skúmaný stav na základe heuristickej metódy.

Algoritmus A* je možné opísať nasledovnými krokmi:

1. Ohodnoť začiatočný stav
2. Vlož začiatočný stav do zoznamu stavov
3. Pokiaľ nie je nájdený cieľ alebo zoznam stavov je neprázdny, vykonaj:
   3.1. Vyber prvý stav zo zoznamu stavov
   3.2. Vytvor nové stavy aplikovaním pravidiel prostredia na vybraný stav
   3.3. Ohodnoť všetky nové stavy
   3.4. Vlož všetky nové stavy do zoznamu stavov

Zároveň pre algoritmus platí:

- nový stav sa pridá do zoznamu stavov utriedene podľa ohodnotenia stavov. Zoznam stavov preto musí byť v každom stave algoritmu utriedený podľa ohodnotení obsiahnutých stavov.
- nový stav sa pridá do zoznamu stavov predošlým spôsobom iba v prípade, že tento stav ešte nebol skúmaný.
- Ak novovytvorený stav (resp. začiatočný stav) je ohodnotený a zistí sa, že je zároveň cieľovým stavom, algoritmus ihneď končí svoju činnosť s výsledkom reprezentujúcim cestu k cieľu.

Heuristické ohodnotenie skúmaného stavu je modifikovateľné. V súčasnej verzii je použitý vzorec:

$HB = MD + Poplatok$, kde:

HB je hodnota skúmaného bodu (stavu),*MD* je tzv."Mannhattanská" vzdialenosť. Určuje sa absolútnou hodnotou rozdielu vzdialenosti medzi skúmaným a cieľovým stavom získanej súčtom dĺžky dvoch pravouhlých (vodorovnej a zvislej) úsečiek opisujúcich zdanlivú cestu z počiatočného bodu ku koncovému.

Poplatok je heuristické ohodnotenie skúmaného stavu vyjadrujúce výšku "obety" za dosiahnutie tohto stavu. Pre výpočet poplatku sme použili faktory:

- počet nutných posunov pre dosiahnutie stavu
- doba čakania na získanie stavu
- pravdepodobnosť výskytu prekážky ako objektu brániaceho dosiahnutiu stavu
- typ prekážky

Výpočet poplatku teda uvažuje prípady, kedy nie je možné dosiahnuť skúmaný stav. Takéto prípady sú napríklad obsadenie políčka iným hráčom alebo pevným objektom - stena, bomba a pod. Výška poplatku v tomto prípade vzrastie, ak *typ prekážky* na danom políčku je neprekonateľný, v opačnom prípade bude zvyšovať poplatok o prázdnu hodnotu (nezvýši poplatok). Čím je *pravdepodobnosť výskytu prekážky* vyššia, tým vyšší bude poplatok za tento stav. Napríklad výskyt steny je zaručený (100%), hodnota poplatku nebude znižovaná. V prípade, ak je prekážka iná postava v hre, teda je pohyblivý objekt, výskyt nie je zaručený (50%) a preto bude výška poplatku znížená na polovicu.

Ďalej sa uvažujú prípady, kedy je možné prekážku prekonať dostupnými spôsobmi (zničenie, posunutie). Ak by bola prekážka v podobe zničiteľnej steny, výšku poplatku by výrazne ovplyvňovala *doba čakania na získanie stavu*. Predpokladá sa totiž, že by bola použitá bomba na zničenie prekážky, čím by ale hráč musel vykonať istý *počet nutných posunov* a zotrvať v určitej polohe počas *doby čakania na získania stavu* a po odstránení prekážky následne znova vykonať istý *počet nutných posunov*.

4.2.2 Uvažovanie zmeny stavu prostredia počas hľadania cesty

Hľadanie cesty k cieľovému stavu potrebuje v špecifických prípadoch zmeniť stav prostredia na základe dostupného spôsobu prekonania prekážky, v ktorom skúma novovytvorený stav. Ak nastane takáto potreba, zmena nemôže byť aplikovaná na stav prostredia pre ostatné skúmané stavy, nakoľko tie danú zmenu uvažovať ani nemôžu (v ich stavovom priestore takáto zmena nenastala). Je

preto potrebné túto zmenu zaznamenať, aby mohla byť použitá v ohodnocovaní nový stavov vzniknutých zo stavu, ktorý túto zmenu spôsobil. Vytvorili sme preto množinu zmien stavov prostredia, ktorá uchováva zmenené stavy počas hľadania cesty a je prístupná pre všetky skúmané stavy. Zároveň platí, že každý novovzniknutý stav "dedí" svoj stav prostredia od predošlého stavu, z ktorého vznikol. Ak by však dosiahnutie stavu požadovalo zmenu stavu prostredia, táto zmena sa zaznamená a prostredia pre stavy vzniknuté z tohto stavu budú dediť práve tento zmenený stav. Úprava pôvodného algoritmu A* je teda vo forme doplnenia:

1. Ohodnoť začiatočný stav podľa začiatočného stavu prostredia
2. Vlož začiatočný stav do zoznamu stavov s referenciou na stav prostredia použitým pri ohodnotení
3. Pokiaľ nie je nájdený cieľ alebo zoznam stavov je neprázdny, vykonaj:
   3.1. Vyber prvý stav zo zoznamu stavov s referenciou na skúmaný stav prostredia
   3.2. Vytvor nové stavy aplikovaním pravidiel prostredia na vybraný stav. Ak nebolo potrebné vykonať zmenu stavu prostredia, pokračuj krokom 3.4
   3.3. Aplikuj zmenu stavu prostredia, zaznamenaj zmenu a urči zmenený stav prostredia za skúmané prostredie
   3.4. Ohodnoť všetky nové stavy podľa skúmaného prostredia predošlého stavu
   3.5. Vlož všetky nové stavy do zoznamu stavov s referenciou na stav prostredia použitým pri ohodnotení

4.3 Prototypovanie sieťovej komunikácie

4.3.1 *HessianKlient-ServercezHTTP*

Android Klient

- Využitie *Hessdroid* knižnice – Portovaný *hessian* kód na android. Kód skopírovaný z repozitára, kompilovaný lokálne využitý ako referencovaná knižnica v Android projekte.

Server

- *PHP* server je schopný komunikovať s *PC Java klientom* ale len pri použití posledného vydania *hessian*-u. *Hessdroid* spojenie zlyhalo pravdepodobne kvôli nekompatibilite.
- *JSP Servlet (Java EE)* na *Java Serveri*. Testované na localhost (*GlassFish* server). Použitá verzia*hessian-4.0.7* (najnovšie vydanie), funguje spoľahlivo. Oficiálny sever beží na*VM29's Tomcat*serveri, funguje v poriadku.

4.3.2 AndEngine

Wifi C-S

- ak sú zariadenia na spoločnej *LAN* – funguje,
- táto implementácia využíva *TCP socket.*

Wifi C-S

- cez android 2.2 hotspot, nie je možné zapínanie programátorsky, referencie na:http://stackoverflow.com/questions/3023226/android-2-2-wifi-hotspot-api
- Testované

Bluetooth

- Test zlyhal (použité boli zariadenia *Samsung* and *HTC*)
- Automatické zisťovanie *Wi-fi* Servera
- referencie:
  http://www.android-x86.org/documents/virtualboxhowto
  http://www.android-x86.org/documents/debug-howto

Inštalácia *Virtual boxu* prebehla v poriadku. Použil sa premostený sieťový adaptér pre získavanie *IPadresy* zo smerovača.

Testovanie C-S prebehlo v poriadku. Testovanie automatického zisťovania prebehlo v poriadku.

4.3.3 Zhrnutie

Výsledky analýzy sieťovej komunikácie sú nasledovné:

- *Server - Klient* je spoľahlivé prostredníctvom *Hessian*-u (slabé pripojenie ale vysoká latencia)
- *Klient - klient* je spoľahlivé cez *TCP/IP* protokol využívajúc *AndEngine* alebo vlastný protokol (spojenie môže byť vytvorené len keď sú zariadenia na rovnakej podsieti)
- *Klient - Klient* prostredníctvom *Bluetooth*: *AndEnginemultiplayerextension* je stále vo vývoji
- Existuje známa chyba *Bluetooth* modulu v *AndEngineMultiplayerExtension*ktorá zabraňuje korektnej funkčnosti príkladov.

Referencie: http://code.google.com/p/andenginemultiplayerextension/issues/detail?id=1

5 Šprint č.3

Číslo šprintu	**03**
Začiatok šprintu	09.11.2011
Koniec šprintu	23.11.2011
Príbehy (*userstories*)	• Výber hráča a mapy a úprava menu • Manažment máp • Neurónová sieť so spätným šírením chyby • Markovovské siete • Multiplayer • EvilBot

5.1 Výber hráča a mapy a úprava menu

Menu umožňujúce výber hráča a mapy.

5.1.1 Analýza

Pre vylepšenie celkového dojmu z hry je potrebné zapracovať do menu grafické prvky, ktoré môžu mať pre používateľa rôzny význam:

- lepšia orientácia v hre
- zlepšený pocit a celkový dojem
- pri zvolení vhodných doplnkov je možne navodiť základnú pointu a účel hry

5.1.2 Návrh

Pre celkový dizajn hry bola zvolená kombinácia farieb oranžovej a zelenej čo vytvára pozitívnejšie pocity u používateľa ako strohé a nevýrazné farby.

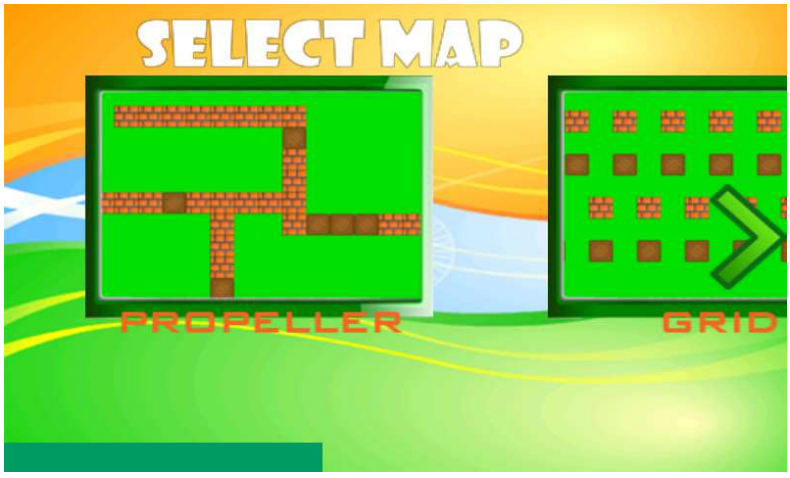
V *Hlavnom menu* sa zobrazujú dve postavy, ktoré zobrazujú človeka a "*humanoida*" v jednom tíme. Samotného *avatara* charakterizuje čiapka a okuliare podobne ako jeho samotná textúra v hre. Položky v menu sa zmenili podľa návrhu režimov hry a navigácie a to nasledovne:

- Quick Game
- Training
- MultiPlayer Game
- OnlineGames
- Options
- Quit

Na obrázku *Obr. 8*je zobrazená ukážka aktuálneho menu.

Obr. 8 Inovované menu

Do Menu bola taktiež pridaná nová obrazovka pre výber mapy. Samotné pozadie a písma v hre sú v ladené do podobnosti so základným Menu. Obrazovka obsahuje dynamický počet položiek na základe aktuálne vytvorených máp. Aktuálne sú v tomto menu 4 položky, po zvolení ktorých sa otvorí príslušná mapa a začne sa hra. Ukážka menu na výber mapy je zobrazená na obrázku *Obr. 9*.



Obr. 9 Obrazovka pre výber mapy

Samotná obrazovka je posuvná, čo je bežná prax pri dotykových telefónoch. Tento fakt indikuje vodorovný posúvač v spodnej časti obrazovky, ktorý sa plynule posúva v protismere posunu obrazovky ako aj zobrazovanie a skrývanie smerových šípok.

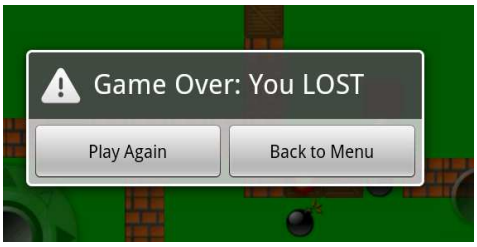
Pri ukončení hry je taktiež potrebné o tomto stave informovať používateľa ako aj vypísať, či používateľ vyhral alebo prehral. Na toto slúž dialóg, ktorý sa vyvolá v dvoch prípadoch:

- ak bol zneškodnený *HumanPlayer*
- ak boli zneškodnení všetci hráči okrem *HumanPlayer*

Dialóg obsahuje dve základné tlačidlá a to:

- *PlayAgain* - po stlačení sa opäť vyvolá hra na definovanej mape
- *Back to Menu* - po stlačení sa objaví hlavné menu

Ukážka dialógu je zobrazená na obrázku *Obr. 10*.



Obr. 10 Game Over dialóg

5.1.3 Implementácia

Pre hlavné menu boli zmenené *TextMenuItems* v triede *MainMenu*. Taktiež bol v tejto triede vytvorený nový *backgroundSprite*, ktorý sa používa ako komponent pre pozadie*mainScene.*

Pre menu na výber mapy bola vytvorená nová trieda *GameMapMenu* s podobnou funkcionalitou ako trieda *MainMenu.*

Dialóg pre ukončenie hry je volaný metódou *removeComponentsFromScene()* kde sa overujú podmienky pre ukončenie a v prípade, že sú tieto podmienky splnené vyvolá sa štandardný dialóg *Android API.*

5.1.4 Testovanie

Názov	Main Menu	**Prípad použitia**	UC Selectlevel
Rozhranie	Obrazovka menu pre výber mapy	**ID testu**	TEST-MENU-01
Účel	Zistiť načítanie správnej mapy		
Vstupné podmienky	Zvolená obrazovka pre výber mapy		
Výstupné podmienky	Hráč sa animuje v správnych smeroch		
Vyhotovil	Ľuboš Gelányi	**Dátum**	22.11.2011

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ stlačí smerové tlačidlo "hore"	Postava hráča sa otočí hore a animuje sa pohyb	Postava hráča sa otočí hore a animuje sa pohyb
2.	Používateľ stlačí smerové tlačidlo "dole"	Postava hráča sa otočí dole a animuje sa pohyb	Postava hráča sa otočí dole a animuje sa pohyb
3.	Používateľ stlačí smerové tlačidlo "doprava"	Postava hráča sa otočí vpravo a animuje sa pohyb	Postava hráča sa otočí vpravo a animuje sa pohyb
4.	Používateľ stlačí smerové tlačidlo "doľava"	Postava hráča sa otočí vľavo a animuje sa pohyb	Postava hráča sa otočí vľavo a animuje sa pohyb

Tab. 2 Akceptačný test pre príbeh Výber hráča a mapy a úprava menu

Názov	Game Over	**Prípad použitia**	UC Play Game
Rozhranie	Obrazovka menu pre výber mapy	**ID testu**	TEST-MENU-02
Účel	Zistiť funkčnosť Game Over dialógu		
Vstupné podmienky	Zneškodnený HumanPlayer alebo všetci ostatní okrem neho		
Výstupné podmienky	Vyvolá sa dialóg a správne zareaguje na voľbu používateľa		
Vyhotovil	Ľuboš Gelányi	**Dátum**	22.11.2011

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Po vyvolaní dialógu používateľ zvolí možnosť Playagain	Načíta sa nová hra, pozície hráčov sa nastavia do prednastavenej polohy	Načíta sa nová hra, pozície hráčov sa nastavia do prednastavenej polohy
2.	Po vyvolaní dialógu používateľ zvolí možnosť Back to Menu	Zobrazí sa hlavné menu	Zobrazí sa hlavné menu

Tab. 3 Akceptačný test pre príbeh Výber hráča a mapy a úprava menu 2

5.2 Manažment máp

Jednoduché reprezentovanie máp v textovom súbore umožňujúce rýchle vytvárania a načítavanie máp.

5.2.1 Analýza

Pri určitých herných módoch používateľ môže chcieť určiť konkrétne prostredie, v ktorom sa hra odohráva. Ak si používateľ zvolil herný mód, ktorý mu umožňuje vybrať mapu, na ktorej si chce zahrať, pred začiatkom samotnej hry sa zobrazí obrazovka obsahujúca všetky ponúkané mapy. Používateľ posúvaním obrazovky (listovaním) a kliknutím na ukážkový obrázok si môže vybrať požadovanú mapu. Následne sa spustí hra na vybranom bojisku.

5.2.2 Návrh

Pre reprezentáciu máp v súbore sme navrhli dátovú štruktúru, ktorá je udržiavaná v obyčajnom textovom súbore. Nižšie je uvedená ukážka obsahu súboru mapy.

```
map-width = 15
map-height = 10
char-0 = empty
char-1 = Wall
char-2 = Crate
char-3 = Player

components_begin
300000000000003
011111111000000
000000002000000
000000001000000
000000001000000
111211111000000
000001001222111
000001000000000
000001000000000
300002000000003
components_end
```

Prvé dva riadky súboru vyjadrujú rozmery mapy. Nasledujúce riadky identifikujú typy objektov, ktoré sú jednotlivými znakmi (číslami) reprezentované. Matica znakov medzi riadkami „*components_begin*“ a „*components_end*“ opisujú mriežku mapy. Znaky z matice sa na mapujú na prázdnu mapu a následne sú nahradené príslušnými objektmi (*Wall, Crate, Player*).

5.2.3 Implementácia

Jednotlivé mapy sú uložené v súboroch typu *.lvl*. Načítanie a extrakcia súboru, následne vytvorenie komponentov na mape prebieha v objekte *MapLoader*. Na vstupe metódy *loadMapFromConfigFile* tejto triedy je objekt *GameConfig*, ktorý reprezentuje súbor na načítanie a množinu všeobecných informácií o mape. Je to statická metóda, ktorá sa volá z triedy *GameServer,* v ktorej sa mapa inicializuje.

5.2.4 Testovanie

Názov	Načítanie mapy	**Prípad použitia**	*UC Selectlevel*
Rozhranie	Obrazovka výberu mapy	**ID testu**	TEST-MAPL-01
Účel	Určiť súbor z ktorého sa načíta mapa		
Vstupné podmienky	Zvolený herný mód umožňuje individuálny výber mapy		
Výstupné podmienky	Spustená hra na zvolenej mape		
Vyhotovil	MátéFejes	**Dátum**	*21.11.2011*

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ si zvolí herný mód, ktorý umožňuje výber mapy	Obrazovka pre výber mapy, zobrazené ukážky ponúkaných máp	Obrazovka pre výber mapy, zobrazené ukážky ponúkaných máp
2.	Kliknutie na ukážku zvolenej mapy	Spustenie hry na zvolenej mape	Spustenie hry na zvolenej mape

Tab. 4 Akceptačný test pre príbeh Manažment máp

5.3 Neurónová sieť so spätným šírením chyby

Cieľom je vytvorenie neurónovej siete, ktorá bude neskôr slúžiť na vyhodnocovanie nasledujúceho kroku.

5.3.1 Analýza

Jednou z možností, ako je možné napodobňovať a simulovať správanie používateľa je učenie *avatara* pomocou neurónových sietí. Výhoda takejto umelej inteligencie spočíva v obrovskej prispôsobivosti na nové situácie. Keďže by bolo veľmi náročné (a často krát nepredstaviteľné a nelogické) učiť *avatara* konkrétne pohyby zo všetkých situácií, je učenie pomocou neurónovej siete nespornou výhodou. Navyše, okrem toho, že bude vykonaný regulárny krok, zachováva sa aj simulovanie správania hráča.

Na učenie pomocou neurónových sieti je potrebné vhodne vytvoriť a naučiť neurónovú sieť. Je veľmi dôležité rozhodnúť, aký bude vstup a výstup samotnej neurónovej siete. Na prvé otestovanie učenia sa pomocou neurónovej siete postačí aj jednoduchší model (sieť obsahuje menej neurónov). Takýto model sa bude dať v budúcnosti jednoducho rozšíriť o ďalšie parametre (a teda aj neuróny). Vytvorí sa sieť, ktorá obsahuje tri vrstvy: vstupnú, skrytú a výstupnú. Na vstupe môžu byť rôzne parametre, ako napr. pozícia hráča, vzdialenosť súpera, vzdialenosť od bomby, vzdialenosť od steny a pod. Výstup môže obsahovať taktiež niekoľko neurónov. Na prvé otestovanie bude postačovať jeden neurón, ktorého výstupom bude pravdepodobnosť, s akou sa položí bomba.

Výsledky z neurónového mozgu sú tým lepšie a presnejšie, čím viac parametrov je zohľadňovaných pri výpočte. To znamená, že keby bola na vstupe namiesto informácie či je *avatar* ohrozený bombou kompletné rozloženie bômb, bolo by to presnejšie a pravdepodobne by to aj lepšie simulovalo správanie sa hráča. Avšak nevýhoda neurónovej siete spočíva v náročnosti učenia na čas a zložitosť. Preto je veľmi dôležité spraviť neurónovú sieť čo najjednoduchšiu, avšak tak, aby vedela simulovať správanie hráča.

5.3.2 Návrh

Prvý prototyp neurónovej siete je navrhnutý jednoduchšie, pričom sa počíta s rozšírením a vylepšením bázy poznatkov, z ktorých sa bude učiť. Počíta sa aj s rozšírením výstupu na viacero neurónov. Sieť je navrhnutá na tri vrstvy, pričom vstupná aj skrytá vrstva obsahuje tri neuróny, výstupná len jeden. Na vstupe sa udávajú tri vlastnosti: či sa *avatar* nachádza vedľa súpera,

či sa *avatar* nachádza vedľa bedne a či je ohrozený bombou. Výstup vyráta pravdepodobnosť, s akou sa položí bomba. V prípade, že sa bomba nepoloží, *avatar* pohyb k najbližšiemu súperovi.

Učenie je navrhnuté nasledujúcim spôsobom:

1. Učenie prebieha pomocou vzorov, podľa ktorých sa *avatar* na začiatku hry správa.
2. S výpočtom a zmenou neurónového mozgu sa počíta po každom cykle učenia (t.j. po skončení tréningu *avatara* sa spustí výpočet nového, mierne upraveného mozgu).
3. Pri hraní proti ostatným *avatarom* sa s rozvojom mozgu *avatara* zatiaľ nepočíta.

Momentálne sa bude mozog trénovať zakaždým odznova. Zmena nastane, keď bude možnosť ukladania neurónového mozgu do súboru a následné načítavanie z neho.

5.3.3 Implementácia

Celá implementácia neurónovej siete je rozdelená do niekoľkých tried. Trieda *Neuron* obsahuje všetky potrebné informácie, ktoré treba uchovávať pre každý jeden neurón na všetkých vrstvách. Táto trieda poskytuje výpočet aktivity neurónov, ich derivácií a zmeny váh medzi jednotlivými neurónmi. Tieto operácie poskytujú nasledujúce metódy:

- *OutputActivity*()
- *Activity*()
- *DerivationActivity*()
- *ChangeOfWeights*()

Trieda *PatternCreatingNeuralNetwork* slúži na vytváranie vzorov, podľa ktorých sa bude učiť neurónová sieť. Tieto vzory sú načítavané pri každom spustení hry. Je vytvorený stereotyp jedného hráča (momentálne fiktívneho), ktorý sa zakaždým načítava. Avšak táto tvorba vzorov bude prebiehať na základe pohybov hráča, keď bude trénovať *avatara*. Vzory ponúkajú očakávaný výstup pre dané vstupy do neurónovej siete. Tieto činností podporujú nasledujúce metódy:

- *getPatternInputs()*
- *getPatternOutput()*

Pomocnou triedou je trieda *WeightOfEachNeuron*. Táto trieda tvorí spojenie medzi neurónmi, ktoré sa nachádzajú v rôznych vrstvách (výstupná vrstva so skrytou a skrytá so vstupnou).Spojenie je uskutočnené vďaka tomu, že každý neurón (na skrytej a vonkajšej vrstve) obsahuje zoznam prvkov tejto triedy. Tieto prvky obsahujú hodnotu váhy spojenia a neurón, s ktorým sú v spojení. To znamená, že k jednotlivým váham je možné ľahko identifikovať neuróny, ktoré sú spojené. Táto vlastnosť je mimoriadne dôležitá pri učení neurónovej siete so spätným šírením chyby.

Trieda *LayerOfNeuralNetwork* slúži na vytvorenie jednotlivých vrstiev neurónovej siete. Obsahuje dva konštruktory, pomocou ktorých sa pridávajú do jednotlivých vrstiev neuróny. Jeden konštruktov je na vytvorenie vstupnej a druhy na vytvorenie skrytej vrstvy. Rozdiel je v tom, že konštruktor pre skrytú vrstvu obsahuje informáciu o tom, že nadväzuje na vstupnú vrstvu.

Poslednou, avšak najdôležitejšou triedou, je trieda *BackwardsNeuralNetwork*. Táto trieda obsahuje celú funkcionalitu učenia neurónovej siete. Obsahuje metódu na načítanie vzorov (podľa ktorých sa bude neurónová sieť učiť), vytvorenie jednotlivých vrstiev siete, inicializácia jednotlivých neurónov a nakoniec metódu na natrénovanie. Taktiež je tu metóda na výpočet aktivity výstupného neurónu z daných vstupov a menenie váh spojení medzi neurónmi.

Spomínané metódy sú nasledujúce:

- *LoadFileWithPatterns()*
- *LayersInitialization()*
- *LearningNeuralNetwork()*
- *CountingActivityOfOutputNeuron()*
- *GlobalChangeOfWeights()*

5.3.4 Testovanie

Testovanie bude prebiehať po dopracovaní v ďalšom šprinte

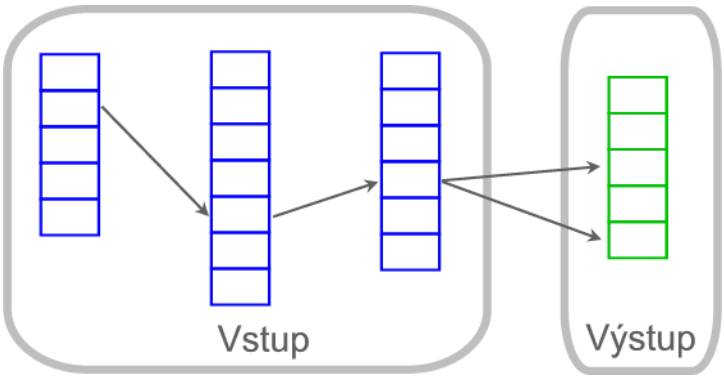
5.4 Markovovské siete

Reprezentácia umelej inteligencie pomocou Markovovských sietí.

5.4.1 Analýza

Pre *avatarov* treba vytvoriť vhodný systém, ktorý bude schopný si zapamätávať kroky používateľa, spracovať ich, vytvoriť podobný herný štýl a následne ho využiť v hre. Musí mať heuristiku, pomocou ktorej sa rozhoduje v rôznych situáciách.

Jedna z možností pre vytvorenie „mozgu“ *avatara* je reprezentácia naučených používateľských krokov v podobe Markovských sietí, ktoré sú zodpovedné za strojové učenie sa, resp. rozhodovanie umelej inteligencie. Na vstupe Markovských sietí sú vstupné údaje rôzneho typu, v závislosti ktorej treba vygenerovať vhodný výstup. Modré stĺpce uvedené na Obr. 1 reprezentujú rôzne kategórie vstupných údajov. Kosoštvorce v stĺpcoch obsahujú konkrétne hodnoty v rámci kategórií. Výstupná hodnota je odvodená z rôznych kombinácií vstupných informácií. V jednej relácii vstup – výstup sa nachádza jedna hodnota z každej vstupnej kategórie a niekoľko výstupných hodnôt (viď. *Obr. 11*). K výstupným hodnotám môžu byť priradené ďalšie hodnoty, ktoré vyjadrujú pravdepodobnosť daného výstupu pre príslušnú kombináciu vstupov.



Obr. 11 Relácia vstupných a výstupných hodnôt

5.4.2 Návrh

5.4.2.1 Architektúra mozgu

Každý modrý stĺpec na obrázku *Obr. 11* reprezentuje jednu vstupnú zložku kategorizácie. Uvažujeme dané zložky:

- Vzdialenosť od cieľa
- Vzdialenosť od protihráča
- Vzdialenosť od bomby – ak sa na mape nachádza viac bômb, počíta sa vzdialenosť od najbližšej
- Prekážka – logická hodnota, je 1, ak sa na mieste, kam sa *avatar* chce posunúť, nachádza prekážka (stena), inak 0

Zelený stĺpec reprezentuje možné výstupy, ktoré môžu nadobúdať hodnoty:

- Posun k cieľu
- Posun k protihráčovi
- Posun od protihráča
- Posun od bomby
- Položenie bomby
- Čakanie na mieste

5.4.2.2 Princíp učenia sa

Samotné učenie sa prebieha pozorovaním používateľa. Pri trénovaní sa každý krok používateľa uloží v podobe relácií uvedených vyššie. Jednotlivé kroky používateľa sa preložia do „jazyka“ zrozumiteľného pre umelú inteligenciu.

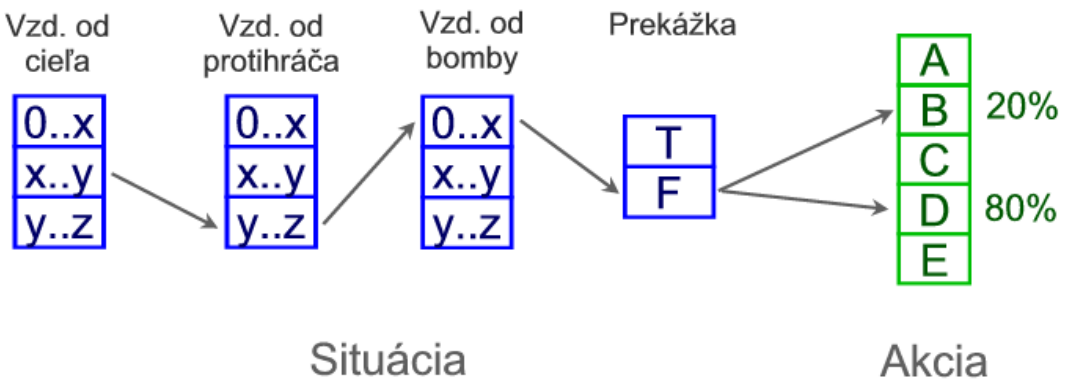
Používateľské akcie	Akcie UI
• **Posun hore / dole / doprava / doľava** • **Položenie bomby**	• Posun k cieľu • Posun k protihráčovi • Posun od protihráča • Posun od bomby • Položenie bomby

Tab. 5Ekvivalencia medzi akciami používateľa a UI

Počas hry umelou inteligenciou sa jednotlivé kroky vyberajú na základe danej situácie pomocou uložených vzorov. Pri výstupoch každej relácie je uvedená číselná hodnota, ktorá reprezentuje počet výberu určitého kroku v danej situácii. Na základe tejto hodnoty stroj vyberie najvhodnejší (používateľom najviac preferovaný) krok.

5.4.2.2.1 Reprezentácia hodnôt

V záujme zvýšenia pravdepodobnosti existencie uloženého kroku pre každú situáciu je nutné pre vstupné hodnoty zvoliť vhodnú granularitu. Granularita musí nahradiť konkrétne číselné hodnoty intervalmi. Ukážka možnej reprezentácie údajov je znázornená na *Obr. 12*.



Obr. 12 Ukážka reprezentácie údajov

x, y, z – celé čísla

T – True

F – False

A, B, C, D, E – akcie (viď. vyššie)

V zmysle obrázku 2 používateľ v situácii, ktorej charakteristiky patria do označených intervalov zvolil akciu B v 20% prípadov a akcia D v 80% prípadov. Toto znamená, že počas hry umelou inteligenciou sa v takejto situácii vykoná akcia D.

5.4.2.3 Spôsob kategorizácie stavu prostredia

Ako zložky kategorizácie sme si určili hlavne vzdialenosti od jednotlivých bodov záujmu. Pre výpočet číselnej reprezentácie zložiek na základe vzdialenosti sme použili rovnicu:

$$NK = PK + VK * 1.618034^{i*0.61}$$

Rovnica pozostáva z hodnôt:

- NK - nasledujúca kategória
- PK - predošlá kategória
- VK - veľkosť kroku
- i - číslo iterácie

Počiatočná hodnota NK je 2. Veľkosť kroku sa vypočíta na základe vzťahu:

$$VK = \frac{MD}{MHK + 1}$$

Rovnica pozostáva z nových hodnôt:

- MD - maximálna dĺžka reprezentácie cesty. Je to hranica sledovanosti vzdialenosti objektu
- MHK - maximálna hodnota kategorizácie

Do cyklu vstupuje číselná vzdialenosť od bodu záujmu, na základe ktorej sa určí pomocou uvedených rovníc kategória vzdialenosti. Výpočet sa uskutoční iteratívnym zvyšovaním hodnoty NK do platnosti tvrdenia:

$$VBZ \leq NK$$

Hodnota VBZ je skúmaná vzdialenosť od bodu záujmu. Iteratívny výpočet sme vybrali z dôvodu, že rovnica použitá pre výpočet NK sa dynamicky mení a výpočet polynomiálnej funkcie by bol pre tento prípad neefektívny (rátame s maximálnym počtom iterácii 5).

5.4.2.4 Spôsob záznamu štatistiky

Výber akcií zaznamenávame ako pravdepodobnosť znovu použitia. Hodnota týchto pravdepodobností je reprezentovaná desatinným číslom od 0 po 1, obe hodnoty vrátane. Zoznam týchto hodnôt je sprevádzaný celým, kladným číslom (vrátane 0), v ktorom je zaznamenaný počet zápisov do štatistiky. Prvým krokom procesu zapisovania je navrátenie skutočných počtov výberu akcií na základe rovnice:

$$PV = PP * CPZ$$

Rovnica pozostáva z hodnôt:

- PV - skutočný počet výberov akcie
- PP - pravdepodobnosť znovu použitia akcie
- CPZ - celkový počet záznamov výberu akcií

V druhom kroku je hodnota CPZ a PV vybranej akcie zvýšená o 1. Následne je vykonaný posledný krok denormalizácie hodnôt rovnicou:

$$PP = \frac{PV}{CPZ}$$

5.4.2.5 Rôzne povahy UI

Rôzne typy umelej inteligencie môžeme vytvoriť priradením priorít k jednotlivým výstupným akciám. Táto priorita je určená poradím zápisu akcií počas trénovania. Každý typ UI má definované poradie uprednostnených akcií. Pri vykonaní kroku používateľa sa pre danú situáciu uloží prvá akcia v poradí, ktorej vykonaný krok zodpovedá. Takýmto spôsobom sa vytvoria umelé inteligencie, ktoré preferujú rôzne herné stratégie. Možné poradia uprednostnenia krokov sú uvedené nižšie.

5.4.2.5.1 Stratégia 1

Hlavným cieľom je dostať sa do cieľa, pritom sa snaží škodiť súperovi. Obrana je na poslednom mieste.

1. Posun k cieľu
2. Posun k protihráčovi
3. Položenie bomby
4. Posun od bomby
5. Posun od protihráča

5.4.2.5.2 Stratégia 2

Ofenzívna povaha, prvoradé je zabiť súpera.

1. Posun k protihráčovi
2. Položenie bomby
3. Posun k cieľu
4. Posun od bomby
5. Posun od protihráča

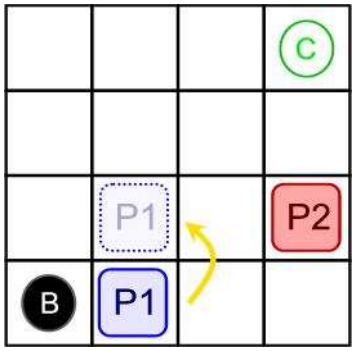
5.4.2.5.3 Stratégia 3

Defenzívna stratégia, neriskuje, najprv sa snaží dostať do bezpečia, potom sa blíži k cieľu.

1. Posun od bomby
2. Posun od protihráča
3. Posun k cieľu
4. Posun k protihráčovi
5. Položenie bomby

5.4.2.6 Príklad zápisu krokov

Na *Obr. 13*je znázornená ukážková situácia hry.



Obr. 13 Ukážková situácia hry

P1, P2 – hráči

B – bomba

C – cieľ

Na ukážke sa hráča P1 používateľ posunul o 1 miesto hore. Táto akcia používateľa sa preloží do nasledujúcich možných akcií UI:

- Posun k cieľu
- Posun k protihráčovi
- Posun od bomby

Všetky 3 kroky zodpovedajú akcii používateľa, do mozgu sa však zapíše len jedna – ktorá z nich je prvá v poradí v závislosti od stratégie (viď. Rôzne povahy UI).

5.4.2.7 Hodnota pravdepodobnosti výberu kroku

Pri danej situácii si používateľ môže zvoliť viac akcií. V záujme určenia preferovanej akcie pri každej akcii uvádzame hodnotu pravdepodobnosti, na základe ktorej sa kroky vyberajú. Navrhli sme dve možnosti na výpočet pravdepodobnosti. Pri príkladoch uvedených nižšie uvažujeme 3 akcie (A, B, C), ktoré boli vykonané v jednej situácii. Uvažujeme opätovné vykonanie akcie A, pričom uvádzame pravdepodobnosť výberu jednotlivých akcií pred a po jeho vykonaní.

5.4.2.7.1 Možnosť I – počet vykonaní

V tomto prípade pri zvolení akcie sa inkrementuje celkový počet jeho vykonaní.

Pred vykonaním akcie A	Po vykonaní akcie A
A: 3	A: 4
B: 5	B: 5
C: 8	C: 8

Tab. 6Prepočet pravdepodobností výberu krokov

Výhody:

- Rozdiel medzi ostatnými hodnotami sa zachová

Nevýhody:

- Možnosť prekročenia pamäte

5.4.2.7.2 Možnosť II – percentuálny podiel

Celková pravdepodobnosť všetkých krokov je 100%. Pri vykonaní daného kroku sa jeho pravdepodobnosť zvýši o x%, následne sa vzniknutý rozdiel odráta z ostatných pravdepodobností v pomere ich aktuálnych hodnôt.

V príklade uvažujeme zvýšenie o 10%.

Pred vykonaním akcie A	Po vykonaní akcie A
A: 30%	A: 30,3%
B: 50%	B: 50 – 5/7*0,3 %
C: 20%	C: 20 – 2/7*0,3 %

Tab. 7Prepočet pravdepodobností výberu krokov

Výhody:

- Šetrenie pamäte

Nevýhody:

- Rozdiely sa nezachovajú

5.4.3 Implementácia

Presúva sa do ďalšieho šprintu

5.4.4 Testovanie

Presúva sa do ďalšieho šprintu

5.5 Multiplayer

Hra umožní súboje pre dvoch či viac ľudských hráčov cez *Bluetooth* či *WiFi*.

5.5.1 Analýza

Prebehla v šprinte *#2*. Viď *kap. 4.3*.

5.5.2 Návrh

Počas prototypovania herného prostredia vznikli ďalšie požiadavky a identifikovali sa chyby existujúceho riešenia. Aby architektúra podporovala hru na jednom, i na viacerých zariadeniach súčasne, bolo nutné ju znovu navrhnúť.

V novej štruktúre pôvodná trieda*SinglePlayerActivity* (prototyp z *AndEngine*-u) bola nahradená triedami:

1. *GameEngine* (*Activity*) - grafické rozhranie programu zodpovedné za zmeny objektu *Map* a aktualizáciu obrazovky. Ďalej slúži na spracovanie, riadenie a pripojenie ovládačov(mozgov) k avatarovi(*AI*, *Human*, *Bot*). Z tejto triedy je možne odvodiť jednotlivé herné módy: *SinglePlayer*, *MultiPlayerServer*, *MultiPlayerClient*
2. *GameServer* (*SocketServer*) – hlavná riadiaca jednotka programu, časovač hry. Riadi pripojenie, prijíma akcie (*Actions*) klientov. Vyhodnotí prijaté akcie a distribuuje zmeny na mape.

5.5.3 Implementácia

5.5.3.1 Herný protokol

1. Vytvorenie spojenia medzi klientom a serverom. Súčasne prebieha aj priraďovanie hráčov na mape ku klientovi resp. k mozgu.
2. *GameServer*hromadne posiela (*broadcasting*)aktuálnyčas a zmeny na mape (nové herné kolo). Časovač je spustený v kontexte *GameEngine*(*Activity*) spätným volaním do triedy *GameServer*.
3. Klientipo prijatí správy o novom kole signalizujú všetky ovládateľné komponenty a posielajú akcie (*Action*), ktoré boli získane od mozgov. Je potrebná revízia umelej inteligencie aby signalizácia nebola blokujúca funkcia a akcie sa mohli posielať*as-on-produced*.
4. Server prijíma akcie a pripraví ich na spracovanie.

V nasledujúcej tabuľke sú uvedené posielané správy medzi klientom a serverom. Použitá skratka BC znamená hromadnú správu (*broadcastmessage*). Správy sú uvedené v slede ich poslania.

Sender	***Receiver***	***Message***	***Sprint#4 master***	***New versionsmayextendwith***
Server	*BC*	*DiscoveryData*	*Server IP and port*	*localplayers (whoelseisconnected to this game)*
Client	*Server*	*ConnectionEstablished*	*Protocolversion (nowit's a constant)*	*Localplayer'suseraccount,* *Changeprotocolversion to game version*
Server	*Client*	*Connection* *Established*	*GameConfigstructure*	*-*
Server	*Client*	*Protocolrejected*	*Serversprotocolversion*	*Changeprotocolversion to game version*
Server	*BC*	*ActionBroadcast*	*Current game time* *New map state* *(seeMapChange)*	*-*
Client	*Server*	*ActionReply*	*Current game timeComponent ID* *Action*	*-*
Client	*Server*	*ConnectionClose*	*-*	*Reason*
Server	*BC*	*ConnectionClose*	*-*	*Reason*

Tab. 8Poslané správy medzi klientom a serverom

5.5.3.2 Aktualizácia hernej mapy a grafického rozhrania

GUI scéna je aktualizovaná v triede *GameEngine* (*Activity*), zmeny sa uplatnia na základe prijatej hromadnej správy(*ActionBroadcast*). Správa obsahuje serializované údaje o zmene objektov mapy (*MapChange*), ktoré majú nasledujúcu štruktúru:

Attribute	*Type*	*Usage*
ID	*Integer*	*Component ID*
Y	*Integer*	*Cellcoordinates*
X	*Integer*	*Cellcoordinates*
ChangeType	*MapChangeType*	*Defines type ofthechange, possibilities:* `PLACE, MOVE, EXPLODE, DISPOSE`
Parameter	*Byte*	*Additional parameter, mainlyusedwithplace*

Tab. 9 Štruktúra prijate správy

Objekt mapy na klientovi (ktorý využíva aj mozog avatarov) je taktiež aktualizovaný podľa tejto štruktúry. Slúži na replikáciu mapy, ktorá je vygenerovaná na *GameServer*.

5.5.3.3 Konfiguračná štruktúra hry

*GameConfig*trieda je používaná na distribúciu iniciálnej konfigurácie hry, mapy, pripojenia klientov, počtu a typu hráčov na mape. Štruktúra je serializovateľná, aby bola kompatibilná s prenosom medzi vzdialeným zariadeniam.

Attribute	*Type*	*Usage*
MapLevel	*String*	*Filenameof game level*
PerClient	*PlayerTypes*[1]	*NumberofPlayerobjects on mapcontrolled by oneclient, withdefinedtypes. Usedforserver'scontroller-playerassignment.*
ClientCount	*Integer*	*Connectedclientscount, datafor server, in SinglePlayermustequal to 1.*
Global	*PlayerTypes**	*ComputednumberofPlayerobjects on map, used in MapLoader. In singlePlayermodeit'sequal to PerClientvalue.*
ControllerData	*List<ControllerData>*	*Usedforserver'splayer-controllerassignment,it'ssent to client and handledinmethodGameEngineActivity.PlayerIdRecieved()*

Tab. 10 Štruktúra triedy Game Config

5.5.4 Testovanie

Názov	Overenie funkčnosti ovládania	**Prípad použitia**	UC Moveplayer UC Putbomb
Rozhranie	Obrazovka bojiska hry (mapa hry)	**ID testu**	TEST-MPE-01
Účel	Overiť funkčnosť ovládania panáčika a položenia bomby		
Vstupné podmienky	Vybraná mapa a spustená hra		
Výstupné podmienky	Ovládateľný panáčik		
Vyhotovil	Dávid Pszota	**Dátum**	20.11.2011

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ používa smerové tlačidlá, t.j. softvérový joystick na obrazovke	Panáčik sa pohne v zodpovedajúcom smere.	Panáčik skočí na novom mieste bez plynulého prechodu
2.	Kliknutie na akciové tlačidlo	Panáčik položí bombu	Zobrazí sa bomba na obrazovke
3.	Používateľ súčasne používa obidva ovládacie prvky (multitouch)	Panáčik položí bombu a pohne sa novým smerom	Systém nevie rozpoznať viac používateľských vstupov naraz.

Tab. 11 Akceptačný test pre príbeh Multiplayer

[1] **PlayerTypes** isstructureforhandlingsimpleintegervalues, thenumberof***Human, Bot***and***AI (Avatar)*** compontents on map (eachisinstanceofPlayer)

Názov	Vytvorenie hry v móde Multiplayer	**Prípad použitia**	UC Create game UC Join game
Rozhranie	Obrazovka *MainMenu* Obrazovka *GameMapMenu* Obrazovka bojiska hry (mapa hry)	**ID testu**	TEST-MPE-02
Účel	Overiť funkčnosť vytvorenia a pripojenia k hre		
Vstupné podmienky	Sieťové pripojenie Wifi LAN, alebo zapnutý WifiHotSpot		
Výstupné podmienky	Vytvorená sieťová hra, synchronizované akcie na všetkých klientoch		
Vyhotovil	Dávid Pszota	**Dátum**	20.11.2011

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ zvolí funkciu *MultiPlayer* v *MainMenu*	Zobrazenie dialógu na výber módu	Zobrazené dialógové okno
2.a	Používateľ zvolí tlačidlo Server	Zobrazuje obrazovka výber mapy	Spustí sa hra, obrazovka výber mapy nedostupná
2.b	Používateľ zvolí tlačidlo *Client*	Zobrazuje sa obrazovka na výber prístupných serverov	Spustí sa hra, obrazovka výber na serverov nedostupná
3.a	Používateľ zvolí mapu	Spustí sa hra, spätná väzba o čakaní na pripojenie	Bez spätnej väzby
3.b	Používateľ zvolí server	Spustí sa hra, spätná väzba o spustenej hry	Bez spätnej väzby
4.	Pokračovanie s testom TEST-MPE-01	Synchronizované udalosti na všetkých pripojených zariadeniach	Udalosti sa objavia na všetkých zariadeniach bez meškania alebo rozdielov

Tab. 12 Akceptačný test pre príbeh Multiplayer 2

5.6 EvilBot

Bot, ktorého základnou prioritou je zlikvidovanie súpera.

5.6.1 Analýza

Na testovanie schopností v budúcnosti vytvorených neurónových sietí je dobré použiť okrem defenzívneho *bota* aj útočného, ofenzívneho. Takýto *bot* sa bude vyznačovať nasledujúcimi vlastnosťami:

- útočí na najbližšieho protihráča
- snaží sa prežiť – vyhýba sa políčkam, ktoré ohrozuje bomba

Takýto ofenzívny typ *bota* overí defenzívne vlastnosti ľubovoľného oponenta, či už pôjde o UI založenú na neurónových sieťach či Markovovských modeloch. Vytvorenie *EvilBot*-a bude vyžadovať analyzátor mapy, ktorý poskytne potrebné informácie o aktuálnej situácii hry na mape.

5.6.2 Návrh

Fungovanie *EvilBot*-a možno definovať nasledujúcim pseudokódom:

```
Začiatok;

ak (je ohrozený bombou)
{
	choď na najbližšie bezpečné políčko;
}
inak
{
	ak (je na políčku, z ktorého môže ohroziť protihráča)
	{
		polož bombu;
	}
	inak
	{
		choď na políčko, odkiaľ môžeš ohroziť protihráča;
	}
}

Koniec;
```

Je vidieť, že fungovanie *EvilBot*-a je založené na veľmi jednoduchých pravidlách, no napriek tomu je náročným protihráčom. Postupne bude rovnako ako *defenzívnytypbota* vylepšovaný pridaním ďalších

pravidiel, čím sa stane dostatočne silným oponentom aj pre náročných protihráčov ako sú *neurónové siete* či človek.

5.6.3 Implementácia

Samotný kód *EvilBot*-a sa nachádza v triede *EvilBot.java*. V budúcnosti bude jeho konštruktor obohatený o parametre nastavujúce jeho vlastnosti aby ho bolo možné čo najlepšie prispôsobiť konkrétnej situácii (napríklad nastavením obtiažnosti). Kód na svoje fungovanie využíva A* algoritmus implementovaný v triede *NextEngine*. Rovnako ako *BoxDestroyerDefensiveBot* využíva pre zisťovanie aktuálnej situácie na mape triedu *MapAnalyzer*, ktorá mu poskytuje všetky dôležité metódy nutné pre jeho správne rozhodovanie.

Jedná sa o nasledovných 5 metód:

- `getMyPosition()`
- `amIThreatenByBomb()`
- `getClosestSafePlace()`
- `isThisPlaceSave(int, int)`
- `getClosestPlayer()`

5.6.4 Testovanie

Názov	*EvilBot* útok	**Prípad použitia**	*UC Play Game*
Rozhranie	Obrazovka bojiska	**ID testu**	TEST-EBOT-01
Účel	Overenie útočných schopností *EvilBot*-a		
Vstupné podmienky	Na mape sa nachádza testovací *EvilBot* a aspoň jeden človekom riadený hráč		
Výstupné podmienky			
Vyhotovil	*Juraj Mäsiar*	**Dátum**	19.11.2011

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ so svojim hráčom stojí a nehýbe sa	*EvilBot* sa presunie na políčko, z ktorého možno ohroziť protihráča, položí bombu a presunie sa na najbližšie bezpečné políčko	*EvilBot* vyhľadal protihráča a položil bombu na miesto, z ktorého priamo ohrozila protihráča a vzápätí sa presunul na najbližšie bezpečné políčko

Tab. 13 Akceptačný test pre príbeh EvilBot

6 Šprint č.4

Číslo šprintu	**04**
Začiatok šprintu	23.11.2011
Koniec šprintu	07.12.2011
Príbehy (*userstories*)	• Výbuch bomby • Herné režimy a navigácia • Neurónová sieť - vyhodnotenie nasledovného kroku • Markovovské siete • Grafika, animácia avatara • Analyzátor mapy • Box DestroyerDefensiveBot

6.1 Výbuch bomby

Bomba v hre spĺňa základné požiadavky ako napríklad zničenie všetkých zničiteľných objektov v definovanom rozsahu ako aj zreťazenie výbuchov či správna animácia výbuchu.

6.1.1 Analýza

V hernom prostredí majú hráči možnosť položiť bomby, ktorých úlohou je zničiť ostané komponenty v danom rozsahu. Položené bomby majú vybuchnúť s oneskorením aby hráč mal čas sa ešte pohnúť na bezpečné políčko ešte pred výbuchom. Algoritmus výbuchu má zahŕňať aj zreťazenie výbuchov, t.j. keď jedna bomba vybuchne a vo svojom rozsahu má aj iné bomby aj tie musia vybuchnúť v tom istom hernom kole.

6.1.2 Návrh

Zmeniteľným parametrami bomby sú rozsah a počet herných kôl do výbuchu, zadávajú sa pri vytváraní objektu, možné dedenie parametrov od panáčika, ktorý bombu položil. Bomba je komponentom mapy čo zmenená, že má definovanú aj polohu na mape. V každom hernom kole sa sleduje čas, kedy má bomba vybuchnúť, pokiaľ čas prešiel, bomba nastaví svoju akciu na výbuch (pozri *Componentsignal* a *getAction*). Šírenie plameňov na osách X a Y sa rieši s rekurzívnym volaním funkcie pre každý dosiahnutý komponent mapy. Medzi nefunkčnými požiadavkami výbuchu bomby patrí aj animácia výbuchu.

6.1.3 Implementácia

Akciu spracováva *GameServer* spolu s akciami ostaných komponentov mapy v metóde *Map.executeActions*().Algoritmus výbuchu vygeneruje *MapChange* objekty, ktoré slúžia na aktualizáciu distribuovaných objektov typu *Map* na pripojených klientov = grafické rozhranie.

Pre všetky komponenty, ktoré boli zasiahnuté plameňom bomby sa vygeneruje *MapChange* objekt s typom *DISPOSE*. Tento objekt je spracovaný klientom, vymaže bombu zo zásobníka komponentov mapy a odstráni ho z grafického rozhrania. Pre všetky polia, ktoré boli zasiahnuté plameňom sa vygeneruje aj objekt s typom *EXPLODE*, grafické rozhranie hry vykresľuje animovanie plameňov.

Poznámka: Problém s modifikovaním zásobníka komponentov (zmazanie a pridanie nových objektov) je riešený s definovaním pomocných zásobníkov *mToRemove* a *mToAdd*, po vykonaní všetkých akcií komponentov hlavného zásobníka je možné bezpečne aktualizovať.

Zdrojový kód:

```
        privatevoidexecuteExplodeAction(Component pComponent) {

                if (mToRemove.contains(pComponent))
                        return; // already handled explosion
                // prevents cyclic recursive call

                int range = pComponent.getExplosionRange();
                intposX = pComponent.getX();
                intposY = pComponent.getY();
                List<Component>infectedComponents = newArrayList<Component>();

                updates.add(newMapChange(pComponent, MapChangeType.DISPOSE));

                mToRemove.add(pComponent);

                // if range is 0 (e.g. player) only self component is added to remove
                for (inti = 0; i< (range * 2 + 1); i++) {
                        for (Component c : this.getCompoentsInCell(posX - range + i, posY)) {
                                // flames x-axis
                                if (!infectedComponents.contains(c))
                                        infectedComponents.add(c);

                        }
                        for (Component c : this.getCompoentsInCell(posX, posY - range + i)) {
                                // flames y-axis
                                if (!infectedComponents.contains(c))
                                        infectedComponents.add(c);
                        }
                        updates.add(newMapChange(posX - range + i, posY,
                                        MapChangeType.EXPLODE));
                        updates.add(newMapChange(posX, posY - range + i,
                                        MapChangeType.EXPLODE));
                }

                for (Component c : infectedComponents) {
                        if (c.isDestructible)
                                executeExplodeAction(c);
                               // recursive explode all components
                }

        }
```

6.1.4 Testovanie

Názov	Logika hry – výbuch bomby	**Prípad použitia**	*UC Putbomb*
Rozhranie	Obrazovka dejiska hry (mapa hry)	**ID testu**	TEST-GLO-01
Účel	Overiť funkčnosť vplyv výbuchu bomby		
Vstupné podmienky	Vybraná mapa a spustená hra		
Výstupné podmienky	Zničené komponenty a animovaný výbuch		
Vyhotovil	Dávid Pszota	**Dátum**	5.12.2011

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Kliknutie na akciové tlačidlo	Panáčik položí bombu	Bomba sa objaví na obrazovke
2.	Používateľ čaká na výbuch bomby	Bomba sa vybuchne a zničí komponenty na mape	Vykresľuje sa animácia výbuchu a dosiahnuté komponenty sa zmiznú z obrazovky

Tab. 14 Akceptačný test pre príbeh Výbuch bomby

6.2 Herné režimy a navigácia

Herné módy a režimy definujú základnú množinu herných situácií podľa definovaných požiadaviek. Hra má vytvorené všetky typy režimov, módov či hráčom na splnenie týchto požiadaviek.

6.2.1 Analýza

Hra by mala pokrývať základy funkcionality, ktoré boli definované pri vytváraní *ProductBacklog*-u. Medzi tie najzákladnejšie patrí možnosť *multiplayer* hry ako aj učenie vlastného *avatara* s "umelým mozgom". Pre pokrytie týchto základných vlastností je potrebné navrhnúť základný model navigácie v hre ako aj herných režimov, či typov hráčov.

6.2.2 Návrh

6.2.2.1 Chronologický proces prechádzania jednotlivých obrazoviek

1. *Splash screen*
2. *Login*
3. *Avatar Selecting*
4. *Main Menu*
    - *Quick Game*
        1. *Game Settings*
        2. *Map*
        3. *Game*
        4. *Game statistics*
    - *Training*
        1. *Game Settings*
        2. *Map*
        3. *Game*
        4. *Game statistics*
    - *Multiplayer Game*
        1. *Create or join game*
            a. *Game mode*
            b. *Map*
        2. *Game*
        3. *Game statistics*
    - *Online Games*
        1. *Online Games statistics*
    - *Options*
        1. *Game options*
        2. *Avatar options*
        3. *Ranking*
    - *Quit*

6.2.2.2 Popis jednotlivých obrazoviek

Splashscreen

Obrazovka, ktorá sa objaví pri štarte aplikácie. Slúži na vyplnenie času potrebné pre nahratie všetkých *resources*. Obsahovať bude grafické pozadie s námetom hry a textovú informáciu o načítavaní hry.

Login

Obrazovka slúžiaca na prihlásenie používateľa. Ak používateľ už je prihlásený zvolí možnosť *Next*a automaticky prejde na ďalšiu obrazovku. Ak nie vyplní svoje prihlasovacie údaje. V prípade zmeny používateľa je na tomto mieste možné ho odhlásiť a prihlásiť iného.

Komponenty na obrazovke:

- Tlačidlo *Register*– Vyvolá totožnú obrazovku, v tejto sekcii sa používateľ zaregistruje
- V prípade neprihláseného používateľa
    - Textové Pole *User*pre zadanie používateľského mena
    - Textové pole *Password* pre zadanie používateľského hesla
    - Tlačidlo *Sign In* – V tomto momente aplikácia prejde do ďalšej sekcie

AvatarSelecting

Obrazovka sa zobrazí len pre práve registrovaného používateľa. V tomto bode si používateľ zvolí základný typ svojho agenta, ktorého bude trénovať. V prípade, že sa jedná o už registrovaného používateľa táto obrazovka sa neobjaví. Prípadná zmena, pridanie či odstránenie *avatara* bude dodatočne možné v položke *Option – AvatarSettings.*

Komponenty na obrazovke:

- Tlačidlá pre jednotlivých avatarov reprezentované grafickými ikonami.

Main Menu

Základné herné menu, ktoré slúži na navigáciu v hre.

Obsahuje grafické pozadie a nasledovné tlačidlá:

- *Quick Game*
- *Training*
- *Multiplayer Game*
- *OnlineGames*
- *Options*
- *Quit*

Game Settings

Obrazovka obsahuje nasledujúce položky:

Difficulty

Obrazovka použitá pri spustení režimov *Quick Game* a *Training.* Zahŕňa tri základné úrovne obtiažností.

Obsahuje tieto položky:

- *Beginner*
- *Advanced*
- *Expert*

Game Mode

Obrazovka určená na zvolenie herného módu. Používa sa pri všetkých herných režimoch okrem *OnlineGames*.

Obsahuje tieto položky:

- *Deathmatch*
- *CaptureTheFlag*

NumberofPlayers

V tejto ponuke si používateľ zvolí počet hráčov na mape. Ponuka bude aktívna len pre režimy hry *Quick Game* a *Training*.

Na výber sú tieto možnosti:

- *2*
- *3*
- *4*

Map

V tejto ponuke si používateľ môže zvoliť mapu bojiska, na ktorom bude prebiehať súboj.

Obsahuje tieto položky.

- Tlačidlá v počte X s názvom daného bojiska a jeho grafického náhľadu.
- Tlačidlo *Back*

Game

Základná obrazovka obsahujúca samotné bojisko. Obsahuje mriežku s rôznymi komponentmi ako sú napríklad stena, zničiteľná stena, hráč, špeciálne predmety, vlajka či bomba. Táto obrazovka slúži ako základné prostredie pre všetky herné režimy a všetky herné módy.

V rámci navigačných položiek pre používateľa obsahuje táto položka nasledovné komponenty:

- Softvérové ovládanie pre pohyb hráča zobrazené v ľavom dolnom rohu
- Tlačidlo pre položenie bomby zobrazené v pravom dolnom rohu
- Tlačidlo *Pause* zobrazené v ľavom hornom rohu

Game Statistics

Obrazovka sa objaví po skončení každého súboja. Zobrazuje základné štatistiky z hry ale hlavne úspešnosť avatara.

Taktiež obsahuje nasledujúce tlačidlá:

- Tlačidlo *SeeRanking* – zobrazí obrazovku *Rankings*, kde sa nachádza aktuálny rebríček hráčov podľa bodov
- Tlačidlo *PlayAgain* – spustí súboj odznova
- Tlačidlo *Menu* – otvorí základné menu

Create/Join Game

Obrazovka pre multiplayer herný mód. Ktorá ponúka používateľovi vytvoriť bojiska, do ktorého sa môže pripojiť iný používateľ alebo samotné pripojenie do vytvoreného bojiska. Ak hráč vytvára hru musí nastaviť parametre súboja na ponúknutých obrazovkách *Game Mode*a *Map.*

Obsahuje nasledovné položky:

- Tlačidlo *Create Game* – vytvorí samotné bojisko a pridá používateľa s jeho *avatarom*. Hra je pozastavená kým sa nepripojí aj druhý tím.
- Tlačidlo *Join Game* – pridá používateľa a *avatara* na bojisko vytvorené iným používateľom.

Online Game statistics

Obrazovka určená pre prezeranie výsledkov online súbojov kde používateľ môže prezerať jednotlivé výsledky svojho *avatara*, prípadne spustiť .

Pre zvolený súboj obrazovka obsahuje nasledovné položky:

- Obrazovku *Game Statistics*, ktorá bude mať nahradené svoje tlačidlá
- Tlačidlo *Replaymatch,* ktoré vyvolá *Game* obrazovku a prehrá záznam súboja
- Tlačidlo *Back*

Game settings

Obrazovka slúžiaca na nastavovanie parametrov pre samotnú hru. Nachádza sa v položke menu *Options.*

Obsahuje nasledovné položky:

- *Sound- on/off*
- *Prefferedcommunication – WiFi/Bluetooth*
- *How to play (tutorial)*
- *About*
- Tlačidlo *Back*

Avatarsetting

Obrazovka slúžiaca pre nastavenie parametrov pre avatara.Nachádza sa v položke menu *Options.*

Obsahuje nasledovné položky:

- *Manageavatarscrew* –Prezeranie všetkých používateľských *avatarov* aj s ich preferenciami a dosiahnutými výsledkami. Jednotlivých *avatarov* je možné odstraňovať alebo pridávať. Pre pridanie sa vyvolá obrazovka *Avatarselecting.*
- *Choose default avatar* – Používateľ si zvolí *avatara* zo svojej nadobudnutej skupiny *avatarov*
- *Alteravatarbehaviour* – Nastavovanie preferencií nastaveného *avatara*
- *OnlineGames periodicity* – Nastavenie frekvencie pripájania *avatara* na online súboje v intervale 1 až X.
- Tlačidlo *Back*

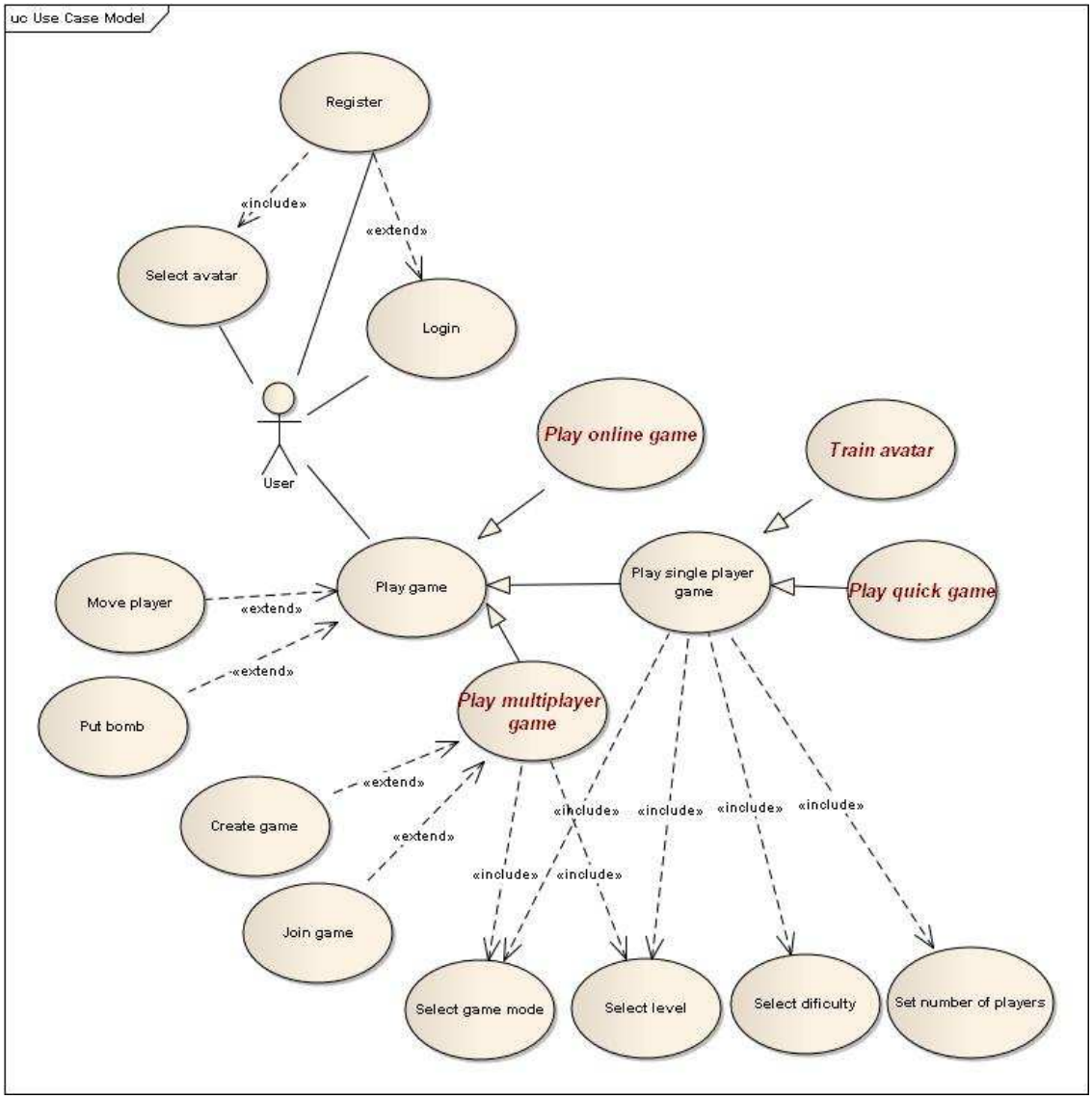
Ranking

Obrazovka obsahujúca zoznam všetkých používateľov na základe počtu bodov aktualizovaný zo servera. V prípade príliš rozsiahleho zoznamu bude zoznam zredukovaný len na prvých 10 používateľov, aktuálneho používateľa prípadne všetkých používateľov, s ktorými sa ocitol v režime hry Multiplayer.

Obrazovka obsahuje aj nasledovné položky:

- Tlačidlo *Back*

Všetky prípady použitia vyplývajúce z interakcií používateľa s hrou sú zobrazené na *UseCase*diagrame na obrázku *Obr. 14*.



Obr. 14UseCase Diagram

Typy hráčov

- ***HumanPlayer –*** hráč bez akejkoľvek inteligencie ovládaný používateľom pomocou GUI.
- ***Avatar –*** Hráč zvolený používateľom s „umelým“ mozgom určený pre trénovanie a následné využitie v súbojoch. Typy avatarov sú dostupné pri registrovaní používateľa ako aj pri manažovaní svojich *avatarov*. Jednotliví *avatari* sa používateľovi odomykajú na základe získaných bodov.
- ***Bot –*** Hráč bez výraznejšej inteligencie, bez schopnosti učiť sa s pevne definovanými reakciami.

6.2.2.3 Herný mód

Herný mód si môže používateľ zvoliť vždy keď ide vstúpiť na bojisko či už s vlastným *avatarom* alebo bez neho. Vo všetkých herných módoch môže nakoniec zavážiť počet získaných bodov, ktoré je možné získať napríklad za nájdenie špeciálnych predmetov ukrytých v zničiteľných stenách.

Základné herné módy sú nasledujúce:

- ***DeathMatch –*** Súboj na život a na smrť kde jediným účelom je zneškodnenie všetkých nepriateľov. Každý hráč má 3 životy a v prípade ich vyčerpania sa po zneškodnení už znova neobjaví na mape. Súboj má taktiež časové obmedzenie a v prípade neukončenia súboja štandardným spôsobom sa víťaz určí na základe väčšieho počtu bodov.

- ***CaptureTheFlag –*** Alebo ukradnutie vlajky protihráča je založené na premiestnenie hráča na bázu protihráča. V tomto prípade majú hráči neobmedzený počet životov a hra sa končí ak hráč získa 3 vlajky svojho protihráča. Po každom získaní vlajky sa pozícia hráča, ktorý ju získal presunie na vlastnú bázu. Ak je hráč zneškodnený presunie sa prednastavenú pozíciu. Hráč vyhráva ak získa potrebné 3 vlajky alebo ak uplynie časový limit a má na svojom konte väčší počet bodov.

6.2.2.4 Režim hry

Režim hry definuje typ účastníkov v rámci jedného súboja. V každom type je nastavené iné zloženie hráčov ako aj iné spôsoby učenia *avatara*.

Quick Game

Jednoduchý súboj určený hlavne pre zaujatie používateľa. Herný mód je zvolený v položke *Game Mode.* V prípade módu *Deatmatch* má každý hráč 3 životy. Hra sa končí ak na mape ostane jeden hráč prípadne ak bude 3 krát zneškodnený *HumanPlayer*.

- **Účastníci** : Hráč ovládaný používateľom teda *HumanPlayer* a počítačový *Bot* bez výraznejšej inteligencie bojujúci len na základe definovaných akcií. Jeho schopnosti sa zvyšujú na základe zvoleného stupňa náročnosti v *Difficult.* Súboja v tomto režime sa môžu zúčastniť maximálne 4 hráči, kde vždy len jeden je ovládaný používateľom a maximálne troch hráčov reprezentujú *boti*. Počet hráčov je možné vyberať v ponuke *NumberofPlayers.*
- **Spôsob učenia:** V tomto režime hry samotný *avatar* nie je prítomný na bojisku napriek tomu využíva sledovanie *HumanPlayera* pre získavanie poznatkov. Teda v tomto režime hry prebieha učenie od používateľa.
- **Bodovanie:** Z tohto súboja sa body do štatistík zapisujú klasickým spôsobom. Teda počet bodov získaných pri hraní tohto režimu sa násobí koeficientom 1.

Training

Režim hry určený pre trénovanie vlastného *avatara*. Herný mód je zvolený v položke *Game Mode.* V prípade módu *Deatmatch* majú hráči neobmedzený počet životov. Hra sa ukončí stlačením vyvolaním *Game Puse Menu* a následným uzavretím hry.

- **Účastníci:** Na mape môžu byť umiestnení maximálne 4 hráči z toho jeden musí byť *HumanPlayer*, jeden *Avatar* a maximálne dvaja hráči môžu byť doplnení *Botmi*. Počet hráčov je možné vyberať v ponuke *NumberofPlayers.* Úroveň *Botov* sa nastaví v ponuke *Difficulty.*
- **Spôsob učenia:** *Avatar* je priamo prítomný na bojisku v tomto prípade sa však neučí od používateľa ale zaznamenáva a vyhodnocuje výsledky svojich vlastných akcií. *Avatar* sa teda učí z vlastných chýb.
- **Bodovania:** Z tohto súboja sa body do celkového poradia hráčov nezaznamenávajú. Teda body získané v tomto súboji sa vynásobia koeficientom 0.

Multiplayer Game

Tento režim je určený pre socializáciu používateľov. Používatelia musia byť v priamom kontakte aby bola možná komunikácia cez *Bluetooth* alebo *WiFi*. Herný mód je zvolený v položke *Game Mode.* V prípade módu *Deatmatch* má každý hráč 3 životy. Hra sa končí ak na mape ostane jeden hráč prípadne ak budú zneškodnený obaja hráči jedného tímu.

- **Účastníci:** Na mape sa nachádzajú práve 4 hráči kde práve dvaja sú *HumanPlayers* a práve dvaja ich vlastní *Avatari*. Na mape teda budú bojovať dva tímy v každom používateľ so svojím *avatarom*.
- **Spôsob učenia:** Aj v tomto prípade sa *avatar* učí na základe vlastných vykonaných akcií a ich následkov.
- **Bodovanie:** Samotná hra má jeden z cieľov socializovať používateľov hier na mobilných zariadeniach. Preto body získané zo súboja sa násobia koeficientom 5.

Online Game

Režim hry určený pre ďalšie získavanie znalostí *avatara* ako aj štatistické overovanie jeho výsledkov. Súboje prebiehajú na serveri teda pre používateľa nie je možné pozorovať tieto súboje v reálnom čase. Z každého súboja však bude možné prezerať štatistiky prípadne spúšťať kompletný „video“ záznam zo súboja. Na serveri budú vytvorené online bojiská (rôzny počet pre rôzne herné módy a rôzny počet hráčov), na ktoré sa budú môcť *avatari* pripájať a zúčastňovať. Tieto procesy budú prebiehať bez zásahov používateľa. Online súboje budú obmedzené na maximálne 5 počas 24 hodín.

- **Účastníci:** Minimálne dvaja a maximálne štyria *avatari*.
- **Spôsob učenia:** Aj v tomto prípade sa *avatar* učí na základe vlastných vykonaných akcií a ich následkov.
- **Bodovanie:** Keďže *avatar* v týchto súbojoch vystupuje sám bez pomoci používateľa ale na druhej strane takýto súboj nespĺňa požiadavku socializácie výsledné body z takýchto súbojov sa budú násobiť koeficientom 3.

6.2.3 Implementácia

Z hľadiska navigácie hry sú na implementované tieto základné obrazovky v menu:

- Main Menu - v triede *MainMenu*
- Map - v triede *GameMapMenu*
- Game - v triede *GameEngineActivity*

Z hľadiska typov hráčov je implementované nasledovné:

- *HumanPlayer* - ovládaný pomocou metódy *getHumanControlled*
- *Avatar* - v súčasnosti funkčný v dvoch rôznych verziách
    - umelá inteligencia pomocou Markovovských sietí
    - umelá inteligencia pomocou Neurónovej siete

Z hľadiska režimov hry je implementované nasledovné:

- *QuickGame*
- *MultiplayerGame*

6.2.4 Testovanie

Testovanie prebieha vždy po implementácii jednotlivých obrazoviek a režimov hry.

6.3 Neurónová sieť - vyhodnotenie nasledovného kroku

Táto časť nadväzuje na vytvorenú neurónovú sieť v predošlom šprinte. Pokračovanie spočíva vo využití neurónovej siete na vyhodnocovanie ďalšieho kroku *avatara*. Ďalší krok sa vyhodnocuje z aktuálnej situácie na mape. Aby sa neurónová sieť mohla využívať na vyhodnocovanie, bolo potrebné ju spojiť s A* algoritmom a analyzátorom mapy.

6.3.1 Analýza

Neurónová sieť potrebuje na vyhodnocovanie ďalšieho kroku zistiť aktuálnu situáciu na mape. Na zistenie aktuálnej situácie bude slúžiť analyzátor mapy, ktorý poskytuje funkcie ako napr. zistenie najbližšieho súpera, zistenie najbližšej steny, vyhodnotí či je *avatar* v ohrození a pod. Takéto informácie budú následne vstupovať do neurónového mozgu, ktorý nakoniec vyhodnotí pravdepodobnosť, s akou položí bombu.

Ak *avatar* vyhodnotí, že je najlepšie spraviť nejaký pohyb a nie položiť bombu, je potrebné, aby dokázal nájsť optimálnu cestu za cieľom. Hľadanie optimálnej cesty zabezpečuje algoritmus A*, ktorý je potrebné spojiť s neurónovou sieťou. A* priamo nevstupuje do jej hlavnej časti neurónovej siete, zavolá sa až po určení, že je potrebné sa pohnúť.

6.3.2 Návrh

Po ukončení trénovania neurónovej siete (trénovanie je uskutočnené v predchádzajúcom šprinte) je potrebné, aby sa na jej vstup dostala aktuálna situácia na mape. Túto aktuálnu situáciu bude vyhodnocovať a poskytovať analyzátor mapy.

Po schopnosti neurónovej sieti dostať aktuálne dáta na vstup je potrebné, aby dokázala z nich vyrátať pravdepodobnosť, s akou sa položí bomba. Následne sa vyhodnotí nasledujúci krok. Pri samotnom vyhodnocovaní, aký bude nasledujúci krok, hrá určitú rolu aj náhoda. Výstup z neurónového mozgu je pravdepodobnosť, s akou sa položí bomba. Avšak táto pravdepodobnosť sa nezaokrúhľuje (čo by mohla byť jedna z možností), ale necháva sa jej pôvodná hodnota (táto hodnota je reálne číslo na uzavretom intervale 0 a 1). Následne sa vygeneruje náhodné číslo a nastáva porovnanie pravdepodobnosti položenia bomby a tohto náhodného čísla. Ak je vygenerované náhodné číslo menšie ako pravdepodobnosť, tak nasledujúci krok je položenie bomby (pretože náhodné číslo spadá do intervalu položenia bomby). V opačnom prípade nastáva pohyb a *avatar* nájde, pomocou analyzátora mapy, najbližšieho nepriateľa. Aby ho mohol zneškodniť, je potrebné, aby sa dostal na

pozíciu, z ktorej ho môže ohroziť. Dostanie sa na túto pozíciu zabezpečí spojenie analýzy mapy s algoritmom na hľadanie optimálnej cesty A*.

Neurónová sieť pri volaní algoritmu A* zadá ako parameter súradnicu, kde sa chce dostať (kde je najbližší súper). Táto súradnica nie je konkrétne miesto *avatara*, ale je to políčko blízko nepriateľa, z ktorého ho je možné zneškodniť. A* vráti všetky jednotlivé kroky, ktoré je potrebné vykonať, aby sa *avatar* dostal ku nepriateľovi. Avšak tento neurónový mozog si vezme len prvý krok, ktorý následne vykoná. Ostatné kroky vymaže. Vymazanie nastáva z toho dôvodu, že sa očakáva pohyb od nepriateľa. To znamená, že polia, z ktorých ho je možné ohroziť, sa časom menia. Navyše týmto mazaním sa zabezpečí aj možná zmena správania v prípade, že bude *avatar* ohrozený bombou nepriateľa.

6.3.3 Implementácia

Vyhodnocovanie ďalšej akcie nastáva až po natrénovaní mozgu. Toto vyhodnocovanie prebieha v triede *BackwardsNeuralNetwork* a je volané pomocou nasledujúcej metódy:

- *returnNextAction*()

Na vstupe neurónovej siete je aktuálna situácia na mape. Táto situácia sa získava pomocou triedy *MapAnalyzer*. Pomocou tejto triedy je *avatar* schopný získať všetky potrebné informácie (momentálne sú to tri informácie): či sa nachádza vedľa nepriateľa, či sa nachádza vedľa steny, ktorú je možné zničiť a či je ohrozený nejakou bombou. Potrebné informácie získavajú nasledujúce metódy:

- *getClosestPlayer*()
- *getClosestBox*()
- *amIThreatenByBomb*()

Pre vykonanie sa zavolá A* algoritmus, ktorý vyráta podľa danej heuristiky najlepšiu cestu do cieľa. Vyhľadanie tejto cesty zabezpečuje nasledujúca metóda:

- *getShortestWay*()

6.3.4 Testovanie

Názov	Neurónová sieť	**Prípad použitia**	*UC Play Game*
Rozhranie	Mapa hry	**ID testu**	TEST-NN-01
Účel	Určiť učenie a vyhodnocovanie neurónovej siete		
Vstupné podmienky	Na mape sa nachádza *avatar* s neurónovým mozgom		
Výstupné podmienky	*Avatar* ničí steny alebo sa snaží zneškodniť nepriateľa		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Spustenie hry	Nezamrznutie hry. Neurónový mozog po natrénovaní vyhodnocuje nasledujúce kroky	Hra na 2-3 sekundy zamrzne. Následne neurónový mozog po natrénovaní (spomínané 2-3 sekundy) vyhodnocuje nasledujúce kroky

Tab. 15 Akceptačný test pre príbeh Neurónová sieť - vyhodnotenie nasledovného kroku

6.4 Markovovské siete

Reprezentácia *avatara* na základe Markovovských sietí.

6.4.1 Analýza

Prebehla v šprinte #3. Viď kapitola 5.4.1.

6.4.2 Návrh

Prebehol v šprinte #3. Viď kapitola 5.4.2.

6.4.3 Implementácia

Inteligencia je zatiaľ obsiahnutá v *Java* triedach *Markowsky.java* a *GoalMarkowsky.java*. Trieda *Markowsky* je abstraktná trieda obsahujúca všetky výpočty potrebné správu pamäte mozgu, pre spracovanie aktuálneho stavu a jeho vyhodnotenie. Metódy určujúce prioritu zapisovania, čím určujú „povahu správania“, sú v triede *GoalMarkowsky*. V budúcnosti plánujeme vytvoriť ďalšie stereotypy na podobnom princípe, ako bola vytvorená *GoalMarkowsky*, a teda dedením od triedy *Markowsky*, pričom doimplementujeme metódy na zapisovanie do pamäte.

Pre svoje výpočty potrebuje triedu *NextEngine.java*, vďaka ktorej si uvedomuje vzdialenosť od objektov záujmu - cieľa, protivníka a bomby. Triedu *MapAnalyzer* používa pre vyhodnotenie stavu mapy a vyhľadanie spomenutých objektov záujmu.

Jeden cyklus výpočtu sa deje v dvoch etapách:

- zapísanie používateľskej reakcie pomocou metódy *setUserReactionOnSituation*.
- vypočítanie vlastnej reakcie na aktuálny stav metódou *generateAction*.

Zápis reakcie používateľa opisuje nasledovný

pseudokód:

setUserReactionOnSituation:

1. Získaj objekt postavičky používateľa
2. Zisti, akú akciu vykonal
3. Zaznamenaj zistenú akciu podľa kategórie predošlej situácie
4. Kategorizuj súčasnú situáciu
5. Kategória predošlej sit. <= Kategória súčasnej sit.

Pre získanie postavičky používateľa je použitá metóda *getPlayerByID* v objekte typu *GameEngineActivity*. Posúdenie aktivity používateľa sa vykonáva metódou *getPlayersAction*, ktorá zatiaľ len rozpoznáva pohyb používateľa, preto napríklad nepoloží bombu. Metóda *saveStatisticByUser* zapisuje reakciu používateľa do mozgu. Zvyšuje tak pravdepodobnosť použitia danej reakcie v podobnej situácii. Následne je ohodnotená súčasná situácia pomocou metódy *rateSituation*. Táto kategorizácia je označená ako predošlá pre ďalší cyklus výpočtu.

Výber vlastnej reakcie prebieha podľa
pseudokódu:

generateAction:

1. Kategorizuj súčasnú situáciu
2. Sprístupni ohodnotenia súčasnej kategorizácii
3. Zisti najpravdepodobnejšie správanie zo záznamov
4. Vykonaj reakciu na základe zisteného správania

Kategorizovanie súčasného stavu je vykonané metódou *rateSituation*. Podľa ohodnotenia sa sprístupnia pravdepodobnosti správaní používateľa. Najpravdepodobnejšie správanie sa zo záznamov získa metódou *getMostSelectedAction* volanej nad objektom správaní používateľa. Reakcia sa určí na základe získaného správania a vykonanie akcie v jeho ponímaní. Napríklad, ak pravdepodobné správanie by bolo pohyb k cieľu, ako reakcia sa určí vykonanie prvého kroku podľa A* cesty smerom k cieľu. Obdobne je to pre správanie pohyb k protivníkovi, od bomby, položenie bomby atď.

6.4.4 Testovanie

Názov	Učenie umelej inteligencie	**Prípad použitia**	*UC Play Game*
Rozhranie	Obrazovka bojiska	**ID testu**	TEST-MN-01
Účel	Overenie funkčnosti a správnosti učenia sa UI Markovovskými sieťami		
Vstupné podmienky	Na mape sa nachádza Hráč a *Avatar*		
Výstupné podmienky	*Avatar* sa pohybuje po mape podľa vlastných rozhodnutí		
Vyhotovil	Adam Mihalik	**Dátum**	5.12.2011

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ spustí súboj s preddefinovanou inteligenciou	Obrazovka súboja	Obrazovka súboja
2.	Používateľ sa pohybuje po mape podľa vlastného uváženia	*Avatar* sa začne pohybovať	*Avatar* sa začal chaoticky pohybovať
3.	Používateľ sa snaží pohybovať po mape pravidelným správaním	*Avatar*-ove pohyby začnú časom pripomínať pohyby a rozhodnutia používateľa	*Avatar*-ove pohyby začali dávať zmysel, boli ucelené, neodpovedali však presne používateľskému správaniu

Obr. 15 Akceptačný test pre príbeh Markovovské siete

6.5 Grafika, animácia hráča

Hráč dostáva podoby, ktoré umožňujú používateľovi rozpoznať jednotlivé typy hráčov na mape. Hráč sa taktiež animuje v správnych fázach podľa zvoleného smeru.

6.5.1 Analýza

Pre potreby grafického prostredia hry je potrebné vytvoriť rozličné textúry pre správnu identifikáciu jednotlivých komponentov na mape ako aj pre zlepšenie celkového pocitu z hry.

Statickú textúru hráča je potrebné nahradiť dynamickou animáciou, ktorá bude vhodne reprezentovať zvolené akcie. Hráč musí byť animovaný vo všetkých smeroch a taktiež musí obsahovať viac podôb podľa jednotlivých typov hráčov. *Andengine* kompletne podporuje animácie *Sprite* objektov na základe posúvania *padding* v jednom *.png* súbore. Takýmto spôsobom je možné nastavovať rozličné animácie pre rozličné akcie.

6.5.2 Návrh

Z návrhu herných režimov vyplynulo niekoľko typov hráčov, ktorí musia byť reprezentovaný rôznymi textúrami. Typy hráčov sú nasledovné:

- Typ *HUMAN* - červený
- Typ *HUMAN* - zelený
- Typ *AVATAR* - červený
- Typ *AVATAR* - modrý
- Typ *AVATAR* - zelený
- Typ *AVATAR* - strieborný
- Typ *BOT* - čierny
- Typ *BOT* - červený
- Typ *BOT* - zlatý

V tabuľke *Tab. 16*sú zobrazené jednotlivé textúry pre hráčov.

Typ Hráča	Textúry
HUMAN	
AVATAR	
BOT	

Tab. 16 Textúry pre jednotlivé typy hráčov

Obr. 16 Polohy hráča

Jednotlivé polohy hráča budú animované pomocou metódy *animation* typu *AnimatedSprite*. Každý hráč obsahuje mriežku 3x4 pre reprezentovanie kompletných pohybov, kde riadky obsahujú jednotlivé kroky (pohyb) a stĺpce jednotlivé smery. Ukážka takéhoto súboru je na obrázku *Obr. 16*.

6.5.3 Implementácia

Jednotlivé textúry sa načítavajú v triede *GameMapEngine* na základe parametrov, ktoré sú získané z triedy *SinglePlayer*, kde je definované akí hráči budú pridaní na mapu. Na základe toho sa zvolia jednotlivé typy hráčov a ich farby a následne sa pridajú na mapu ako *AnimatedSprite*, kde sa pri pohybe spustí metóda *animate.* Animácie sa vykonávajú pri zavolaní metódy *updateComponentsOnMap* kde sa jednotlivé komponenty podľa potreby presúvajú.

6.5.4 Testovanie

Názov	Animácia hráča	**Prípad použitia**	*UC MovePlayer*
Rozhranie	Obrazovka hry	**ID testu**	TEST-GPC-01
Účel	Zistiť správnu animáciu hráča		
Vstupné podmienky	Zvolená mapa a spustenie hry		
Výstupné podmienky	Hráč sa animuje v správnych smeroch		
Vyhotovil	Ľuboš Gelányi	**Dátum**	1.12.2011

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ stlačí smerové tlačidlo "hore"	Postava hráča sa otočí hore a animuje sa pohyb	Postava hráča sa otočí hore a animuje sa pohyb
2.	Používateľ stlačí smerové tlačidlo "dole"	Postava hráča sa otočí dole a animuje sa pohyb	Postava hráča sa otočí dole a animuje sa pohyb
3.	Používateľ stlačí smerové tlačidlo "doprava"	Postava hráča sa otočí vpravo a animuje sa pohyb	Postava hráča sa otočí vpravo a animuje sa pohyb
4.	Používateľ stlačí smerové tlačidlo "doľava"	Postava hráča sa otočí vľavo a animuje sa pohyb	Postava hráča sa otočí vľavo a animuje sa pohyb

Tab. 17 Akceptačný test pre príbeh Grafika, animácia avatara

6.6 Analyzátor mapy

Aktuálny stav na mape je nutné nejakým spôsobom analyzovať. Pre tieto potreby vznikla trieda analyzujúca mapu.

6.6.1 Analýza

Analýza pre vytvorenie neurónovej siete ukázala potrebu vytvorenia univerzálneho nástroja, určeného na analýzu mapy, ktorú by mohla využívať nie len neurónová sieť, ale aj ľubovoľný iný človekom neriadený hráč. Predispozíciou pre vytvorenie analyzátora je algoritmus, hľadajúci najkratšiu cestu do cieľa na ľubovoľnej mape.

6.6.2 Návrh

Analyzátor mapy bude fungovať ako samostatná trieda, ktorej metódy budú riešiť rôzne problémy. Ktoré všetky to budú sa teraz určiť nedá, budú sa postupne pridávať podľa potreby. Na začiatok budú implementované metódy riešiace tieto problémy:

- zistiť pozíciu hráča – človeka
- zistiť pozíciu hráča – toho, ktorý bol predaný cez konštruktor
- zistiť pozíciu najbližšieho protihráča
- nájsť najbližší box
- overiť, či je hráč ohrozený bombou
- nájsť najbližšie bezpečné políčko
- nájsť najbližšie bezpečné políčka
- vrátiť moju pozíciu na mape
- zistiť, či je dané políčko bezpečné
- vrátiť pozíciu najbližšej bomby
- vrátiť pozíciu najbližšieho políčka (políčok), ktoré je zároveň čo najďalej od protihráča
- overiť, či na aktuálnej pozícii môžem byť ohrozený protihráčom
- vrátiť políčka, ktoré budú zasiahnuté danou bombou
- zistiť, či bude dané miesto zasiahnuté bombou ak ju položím na aktuálnu pozíciu

Tieto problémy je často možné riešiť viacerými spôsobmi, na začiatok bude pre zistenie najbližšieho políčka použitá metóda *manhattanovksej vzdialenosti*. Najvhodnejšie by ale bolo použiť niektorý z algoritmov na hľadanie najefektívnejšej cesty do daného miesta a počet krokov by sa bral ako

vzdialenosť. Toto však vyžaduje veľmi efektívny algoritmus, ktorý zatiaľ nie je dostupný (jeho implementácia je naplánovaná na neskôr).

6.6.3 Implementácia

Celý kód analyzátora mapy je uložený v súbore *MapAnalyzer.java*. Tento súbor obsahuje triedu *MapAnalyzer*, z ktorej možno vytvoriť objekt použitím konštruktora s parametrami *map* a *player* (kde *map* je objekt obsahujúci všetky objekty na mape a *player* je objekt mapy reprezentujúci hráča, pre ktorého budú vykonávané jednotlivé metódy).

Trieda obsahuje tri *private* metódy :

- *getDangerousPlaces(Component)*
- *isThatPlaceEmpty(int, int)*
- *getAllDangerousPlaces()*

A 14 *public* metód:

- *getHumanPosition()*
- *getClosestPlayer()*
- *amIThreatenByBomb()*
- *getClosestSavePlace()*
- *getClosestSavePlaces()*
- *getMyPosition()*
- *isThisPlaceSave(int, int)*
- *getClosestBomb()*
- *getClosestSavePlaceFarthestFromOponent()*
- *getClosestSavePlacesFarthestFromOponent()*
- *canIBeThreatenByOponent()*
- *getHitPositions(Coordinates, int)*
- *getClosestBoxPosition()*
- *canIHitThisPlace(Coordinates, int)*

Všetky metódy majú napísaný opis ich činnosti v anglickom jazyku vo formáte kompatibilnom s nástrojmi *Javadoc* a *Doxygen*.

6.6.4 Testovanie

Testovanie prebieha priamo na jednotlivých *AI*, či *botoch*, ktorí analyzátor mapy využívajú.

6.7 Box DestroyerDefensiveBot

Hra obsahuje defenzívneho *bota*, ktorý neútočí na ostatných (len keď sa cíti byť ohrozený) a jeho základná priorita je prežiť.

6.7.1 Analýza

Na testovanie v budúcnosti vytvorených neurónových sietí je dobré použiť napevno vytvoreného strojovo riadeného hráča, *bota*, ktorý preverí pripravenosť neurónovej siete na špecifický typ protihráčov. Jedným z nich je *bot*, ktorý sa vyznačuje nasledovnými vlastnosťami:

- neútočí na protihráčov
- snaží sa prežiť – ak je ohrozený bombou, snaží sa ujsť na bezpečné políčko
- ak sa k nemu priblíži protihráč na vzdialenosť z ktorej ho môže ohroziť, sám preventívne položí bombu a presunie sa na najbližšie bezpečné políčko, ktoré je čo najďalej od protihráča
- aby len tak nestál, ničí postupne všetky boxy na mape

Takýto defenzívny typ *bota* overí ofenzívne vlastnosti ľubovoľného protihráča.

6.7.2 Návrh

Fungovanie *bota* možno definovať nasledujúcim pseudokódom:

```
Začiatok;

ak (je ohrozený bombou)
{
	choď na najbližšie bezpečné políčko, ktoré je čo najďalej od protihráča;
}
inak
{
	ak (je na mape ešte nejaký box no nie je pri ňom dostatočne blízko)
	{
		choď na najbližšie políčko, odkiaľ tvoja bomba zasiahne box;
	}
	ak (je na mieste odkiaľ jeho bomba zasiahne box)
	{
		polož bombu;
	}
}

Koniec;
```

Je vidieť, že fungovanie *bota* je založené na niekoľkých jednoduchých pravidlách. Tieto tvoria základný pilier jeho správania. Dopĺňaním ďalších pravidiel možno vytvoriť lepšieho hráča pripraveného čeliť aj ťažším súperom.

6.7.3 Implementácia

Napriek tomu, že základný návrh vyzerá jednoducho, implementácia je pomerne náročná, vzhľadom na použitie množstva netriviálnych metód riešiacich pre človeka triviálne problémy.

Samotný kód *bota* sa nachádza v triede *BoxDestroyerDefensiveBot.java*. V budúcnosti bude jeho konštruktor obsahovať parametre nastavujúce jeho vlastnosti aby ho bolo možné prispôsobiť konkrétnej situácii. Kód na svoje správne fungovanie využíva A* algoritmus implementovaný v triede *NextEngine*. Ďalej využíva ešte jednu triedu, *MapAnalyzer*, ktorá mu poskytuje všetky dôležité metódy nutné pre jeho správne rozhodovanie. Konkrétne sa jedná o nasledovné:

- `getMyPosition()`
- `getClosestSafePlacesFarthestFromOponent()`
- `canIBeThreatenByOponent()`
- `amIThreatenByBomb()`
- `getClosestBoxPosition()`
- `isThisPlaceSave(int, int)`
- `canIHitThisPlace(Coordinates, int)`

Trieda*MapAnalyzer* a jej metódy tvoria kľúčovú časť celého *bota* a kvalita jej výsledkov priamo ovplyvňuje jeho výsledky.

6.7.4 Testovanie

Názov	Obranné schopnosti	**Prípad použitia**	*UC Play Game*
Rozhranie	Obrazovka bojiska	**ID testu**	TEST-DBOT-01
Účel	Overenie obrannej schopnosti		
Vstupné podmienky	Na mape sa nachádza testovací *bot* a aspoň jeden človekom riadený hráč		
Výstupné podmienky			
Vyhotovil	Juraj Mäsiar	**Dátum**	5.12.2011

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ sa priblíži svojim hráčom k testovaciemu *botu*	*Bot* položí bombu a posunie sa na najbližšie bezpečné políčko umiestnené čo najďalej od protihráča	Po priblížení sa k *botovi* hráčom *bot* položil bombu a presunul sa na najbližšie bezpečné políčko, ktoré bolo najďalej od hráča zo všetkých ostatných najbližších bezpečných políčok

Tab. 18 Akceptačný test k príbehu Box DestroyerDefensiveBot

7 Opis Prototypu

Prototyp je na konci 4. šprintu funkčný. Poskytuje základnú hrateľnosť, kde je možné otestovať logiku hry, umelú inteligenciu jednotlivých avatarov ako aj funkčnosť menu a multiplayer hry.

7.1 Menu

Je intuitívne a jednoducho ovládateľné. Sú pripravené všetky tlačidlá, ktoré sa očakávajú vo finálnej aplikácií. V terajšom prototyp sú avšak funkčné len *QuickGame* a *Multiplayer*, ktoré vyvolávajú príslušné režimy hry.

Quick Game – poskytuje možnosť vyskúšania základnej hrateľnosti celej hry. Po výbere tohto režimu hry je možné zvoliť si rôzny typ mapy, na ktorom chceme odskúšať hrateľnosť. Tento režim taktiež poskytuje možnosť otestovania si inteligenciu botov a avatarov.

Multiplayer – poskytuje možnosť odskúšania si hrania dvoch hráčov proti sebe. Hra sa odohráva medzi dvoma hráčmi a ich dvoma mobilnými zariadeniami. Spojenie je možné vytvoriť s pomocou bezdrôtovej siete *Wifi*, keď sa dané zariadenia sú pripojené do jednej sieti. Navrhnutý systém umožňuje aj uskutočnenie pripojenie s pomocou technológie *bluetooth*. Implementácia umožňuje aj Jedno mobilné zariadenie funguje ako *server*, druhé ako *client* – preto sú postačujúce len dve zariadenia. Výber kto bude *server* a kto *client* ostáva na samotných hráčoch, podmienkou len je, že jeden musím mať vytvorený *server*, druhý sa ako *client* pripojí. Ako už bolo spomenuté, tento režim umožňuje hráčovi so svojím avatarom hrať proti inému hráčovi a jeho avatarovi.

7.2 Ovládanie hry

Ovládanie pohybu hráča je spravené pomocou šípkového posúvania (pomocou tzv. softvérového *joystick*-u). Je situovaný v ľavej dolnej časti obrazovky. V pravej časti sa nachádza tlačidlo, ktoré slúži na položenie bomby.

7.3 Herné módy

V prototype sa nachádza zatiaľ len jeden, základný herný mód. Týmto módom je zlikvidovanie súpera a ostať ako jediný na mape. Pri dosiahnutí takého stavu sa hráč (alebo avatar) stáva víťazom. Naopak, ak je zlikvidovaný, je automaticky porazený.

7.4 Mapa

Základom mapy je mriežka, na ktorej sa odohráva súboj. Všetky doteraz vytvorené mapy majú rovnakú mriežku, avšak je tu možnosť vytvorenia aj mriežky, s inými rozmermi. Na mape (na jednotlivých poliach mriežky) sa nachádza niekoľko rôznych komponentov. Základnými komponentmi sú stena a škatuľa. Tieto komponenty poskytujú jednotlivým mapám rôznorodosť. Stena nie posuvná a ani zničiteľná. Naopak, škatuľu je možné aj posunúť aj zničiť. Avšak nie je možné posunúť viac ako jednu škatuľu naraz. Ďalším komponentom je bomba. Bombu môže položiť akýkoľvek hráč, či už sa jedná o človeka alebo počítač. Bomba ničí všetky komponenty v dosahu (okrem steny). V prípade zasiahnutia výbuchom bomby je hráč porazený a odstránený z mapy.

Ďalšími komponentmi sú už spomínaný hráči. Nachádzajú sa tu tri rôzne typy hráčov: *human*, *avatar* a *bot*. Hráča *human* ovláda človek pomocou už spomínaného ovládača. *Avatar* je hráč, ktorý je riadený umelou inteligenciou. *Bot* je natvrdo naprogramovaný, nemenný hráč.

Ako už bolo spomínané, je na výber viacero máp. Tieto mapy sú načítavané zo súboru. Výber je na hráčovi. Štartovacia pozícia jednotlivých hráčov sa môže meniť v závislosti od mapy. Každá mapa má však označené štyri miesta, kde môže byť umiestnený hráč.

7.5 Umelá inteligencia

Markovovské siete – metóda heuristiky prototypu umelej inteligencie založenej na Markovovských sieťach funguje pomocou pozorovania používateľa. Zapamätáva si kroky vykonané prostredníctvom ovládača, ktoré interpretuje do vlastnej štruktúry a ukladá si do pamäte. Následne ich dokáže využívať pri samostatnej hre, tým získa herný štýl používateľa. Reprezentácia týchto poznatkov je založená na kombinácii vstupných informácií rôznych kategórií. Vstupné údaje reprezentujú situáciu hry, na ktorú používateľ daným spôsobom reagoval. V relácii sa s kombináciami vstupov nachádzajú rôzne výstupy, ktoré vyjadrujú určité akcie hráča. K danej kombinácii vstupov (situácii) môže byť priradený viac akcií v prípade, ak používateľ v podobných stavoch vykonal rôzne akcie. V záujme

rozlišovania týchto výstupov sú k nim priradené hodnoty, ktoré vyjadrujú početnosť zvolenia akcie, t.j. pravdepodobnosť voľby daného kroku. Táto metóda sa nerozhoduje o vhodnosti, resp. nevhodnosti jednotlivých krokov, vždy napodobňuje herný štýl používateľa.

Neurónové siete – v tomto prototype sa nachádza jeden typ neurónovej siete – so spätným šírením chyby. Táto neurónová sieť slúži na naučenie sa, ako sa má správať avatar, za vopred definovaných okolností. Výhodou je, že avatar sa dokáže zorientovať aj v situáciách, na ktoré nebol nikdy trénovaný a dokáže do istej mieri improvizovať a prekvapiť. Učenie prebieha len od hráča – avatar sa sám zatiaľ nezdokonaľuje. Momentálne sa avatar učí z vopred definovaných vzorov, pričom tieto vzory budú v budúcnosti vytvárané na základe pohybu používateľa.

Bot–ako už bolo spomenuté, tak tieto typy hráčov nie sú riadené umelou inteligenciou. Napriek tomu ich spomenieme, keďže sa vyskytujú v samotnej hre.

Evil – je typ hráča, ktorý je zameraný čisto na útočenie a zneškodnenie súpera. Má jednoduchý algoritmus, avšak je veľmi účinný. Ak nie je ohrozený žiadnou bombou, nájde najbližšieho súpera. Po nájdení najbližšieho hráča, nájde najbližšie miesto, odkiaľ ho môže ohroziť a následne sa na toto miesto snaží dostať. Keď je na mieste, odkiaľ môže ohroziť súpera, položí bombu a prejde na miesto, ktoré nie je ohrozené žiadnou bombou.

Box destroyerdefensive – je to typ hráča, ktorý neútočí na protivníkov ale na škatule. Položí bombu len keď je v ohrození iným hráčom.

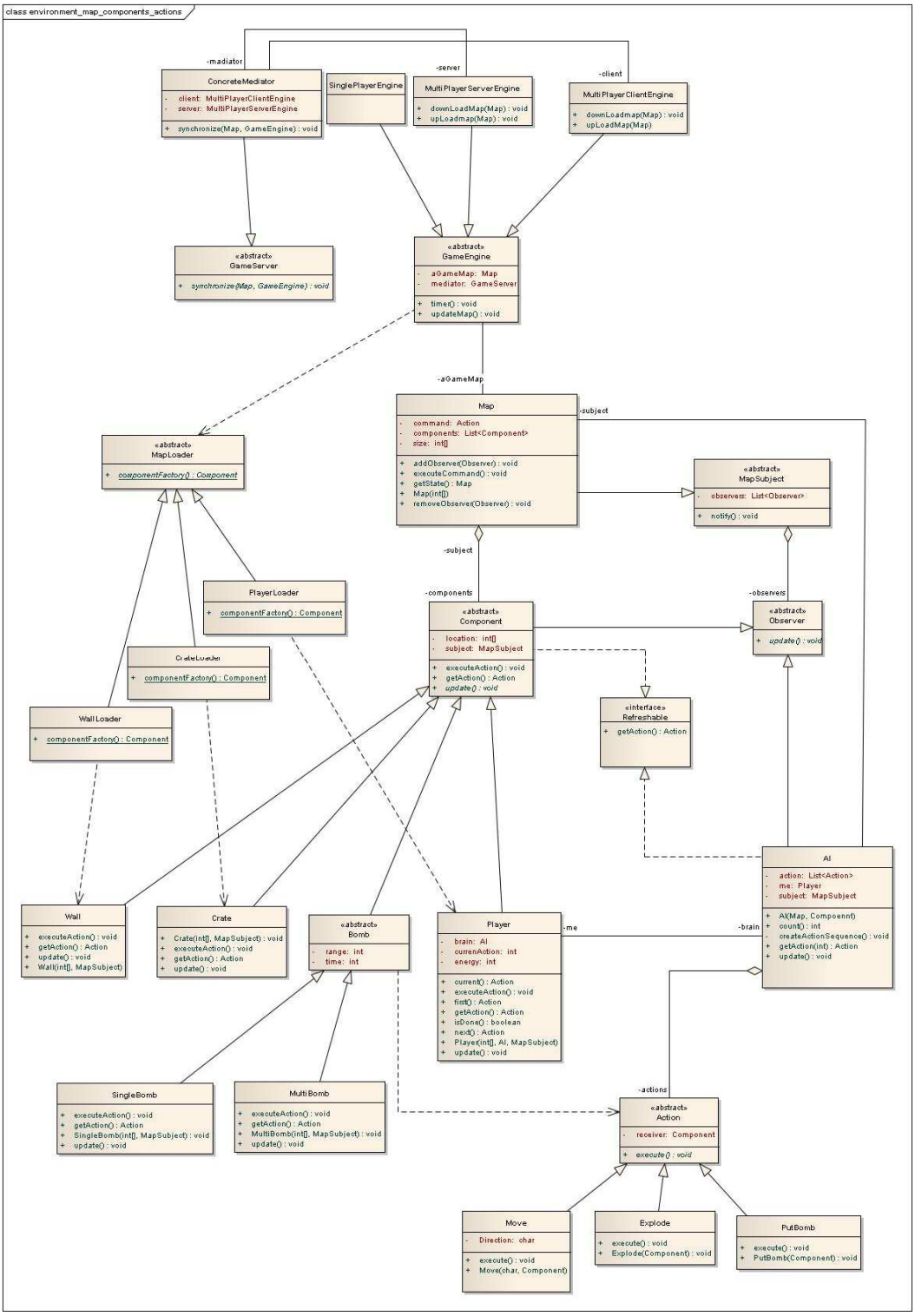
7.6 Plánované vylepšenia

V rámci budúceho semestra plánujeme dokončiť nasledovné funkcionality:

- podpora všetkých herných režimov (*Quick Game, Training, ...*)
- podpora všetkých herných módov (*Deathmatch, CapturetheFlag*)
- *on-line* súboje na serveri
- štatistiky o používateľoch a ich *avataroch*
- podpora *multitouch*
- nový *avatar* riadený novou (inou) neurónovou sieťou

7.7 Diagram Tried

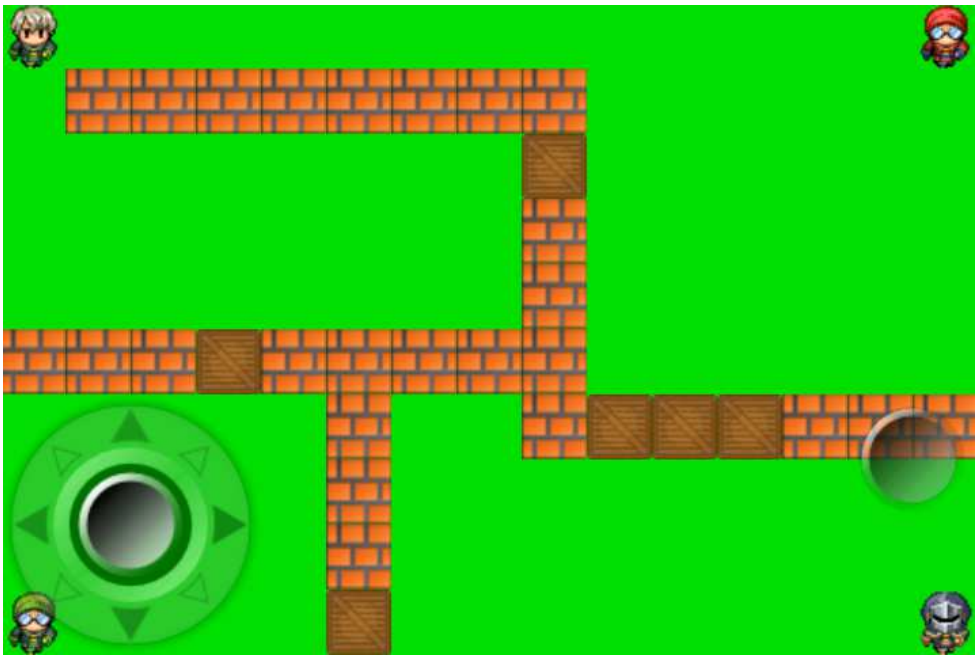
Na obrázku *Obr. 17* sa nachádza Class Diagram implementovaného prototypu



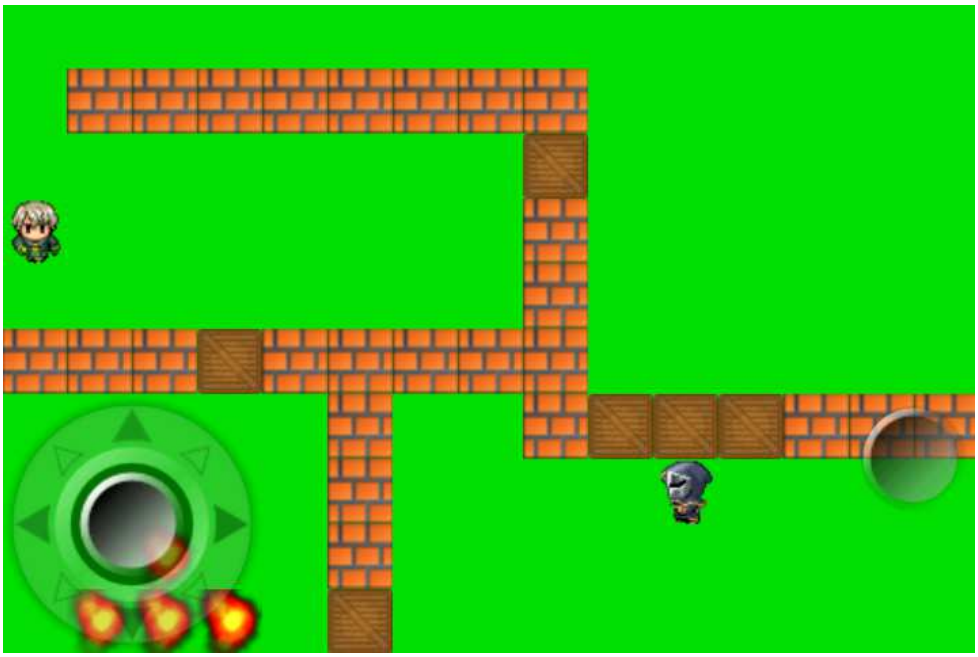
Obr. 17Class Diagram implementovaného prototypu

7.8 Ukážka hry

Na obrázkoch *Obr. 18* a *Obr. 19* zobrazené ukážky z hry.



Obr. 18 Ukážka z hry 1



Obr. 19 Ukážka z hry 2