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: *13.04.2012*

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

1. odovzdanie v LS (5., 6., 7., 8 a 9. 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 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
10.4.2012	8.1 Registrovanie používateľa	Máté Fejes
10.4.2012	8.2 Prihlásenie používateľa	Máté Fejes
10.4.2012	8.3 Herné nastavenia	Ľuboš Gelányi
10.4.2012	8.4 Finalizácia UI založenej na Markovských sieťach	Adam Mihalik
10.4.2012	8.5 Analýza implementačných nástrojov pre	Adam Mihalik

	WP	
10.4.2012	9.2 Manažment hry	Ľuboš Gelányi
10.4.2012	9.2 Základ hry pre WP	Adam Mihalik
10.4.2012	9.2 3 úrovne pre Evil-Bota	Juraj Mäasiar
10.4.2012	9.4 Herný server	Máté Fejes
10.4.2012	9.5 Učenie sa od používateľa	Ľuboš Masný
10.4.2012	10.1 Správa dát používateľa	Máté Fejes
10.4.2012	10.2 Quick Game pre WP	Adam Mihalik
10.4.2012	10.3 Dokončenie A*	Juraj Mäsiar
10.4.2012	10.4 Správa polohy používateľa	Máté Fejes
10.4.2012	10.5 Geoplatforma	Ľuboš Gelányi
10.4.2012	10.6 Uloženie neurónového mozgu	Ľuboš Masný
10.4.2012	11.1Nová Neurónová sieť	Juraj Mäsiar
10.4.2012	11.2 Rebríček používateľov	Máté Fejes, Ľuboš Gelányi
10.4.2012	11.3 Manažment Multiplayer Režimu	Ľuboš Gelányi
10.4.2012	11.4 MapLoader pre WP	Ľuboš Masný
10.4.2012	11.5 Dummy Bot pre WP	Ľuboš Masný
10.4.2012	12.1 Informačný panel pre WP	Adam Mihalik
10.4.2012	12.2 Správa avataov	Máté Fejes, Ľuboš Gelányi
10.4.2012	12.3 Herné štatistiky	Máté Fejes
10.4.2012	12.4 Online Tournament avatarov	Dávid Pszota
10.4.2012	12.5 Neurónová sieť pre WP	Ľuboš Masný
10.4.2012	13 Opis prototypu	Máté Fejes
10.4.2012	Úprava dokumentu	Ľuboš Gelányi
13.4.2012	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 či 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.

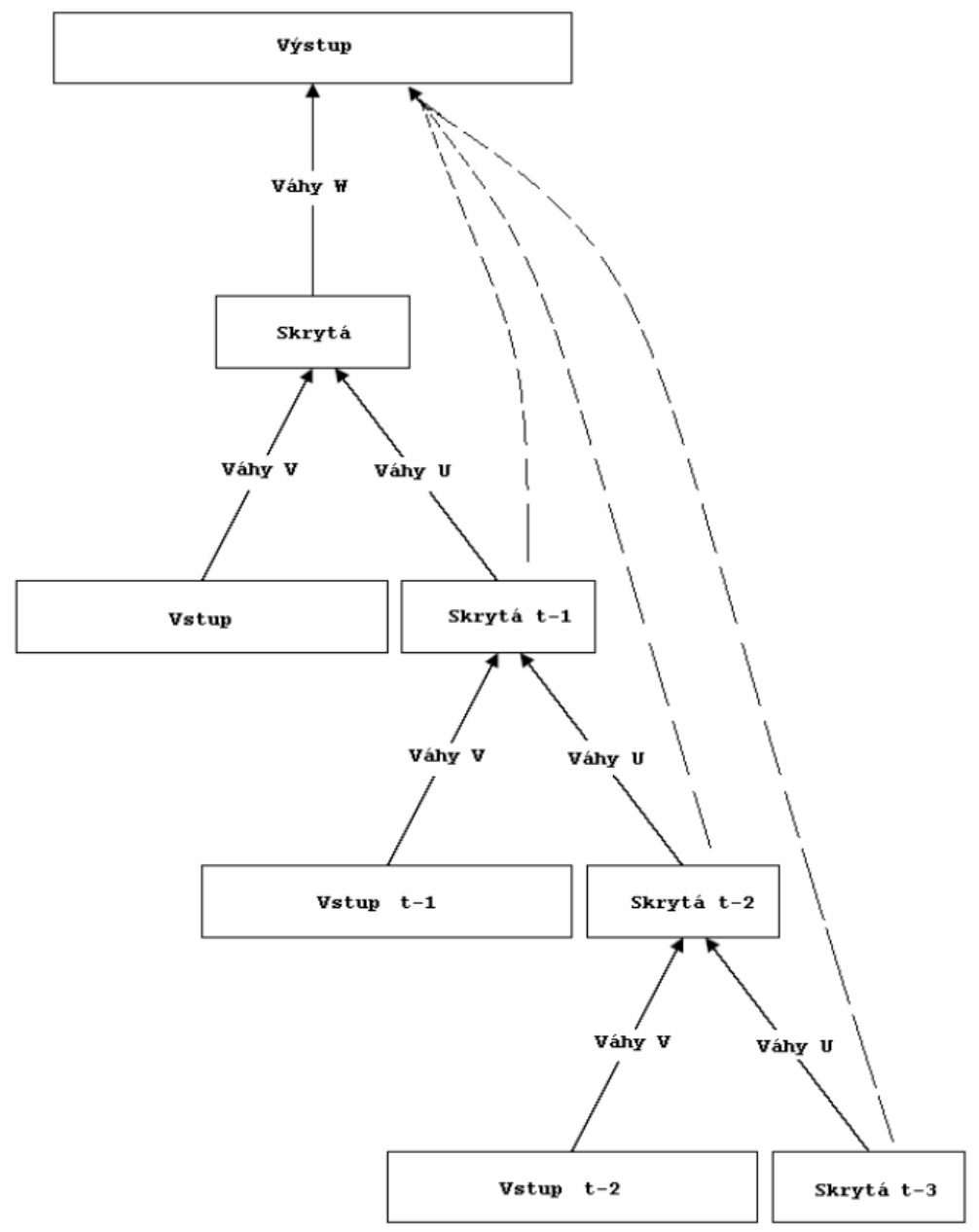
Túto knižnicu by sme teda mohli využiť pri implementácii servera, na ktorom budú prebiehať súboje. Nakoľko ten bude implementovaný v Jave, bude nutné použiť JNI (Javanativeinterface) pre integráciu C++ kódu. Treba podotknúť, že Java momentálne nemá podporu pre SIMD inštrukcie, použitie C++ knižnice je 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

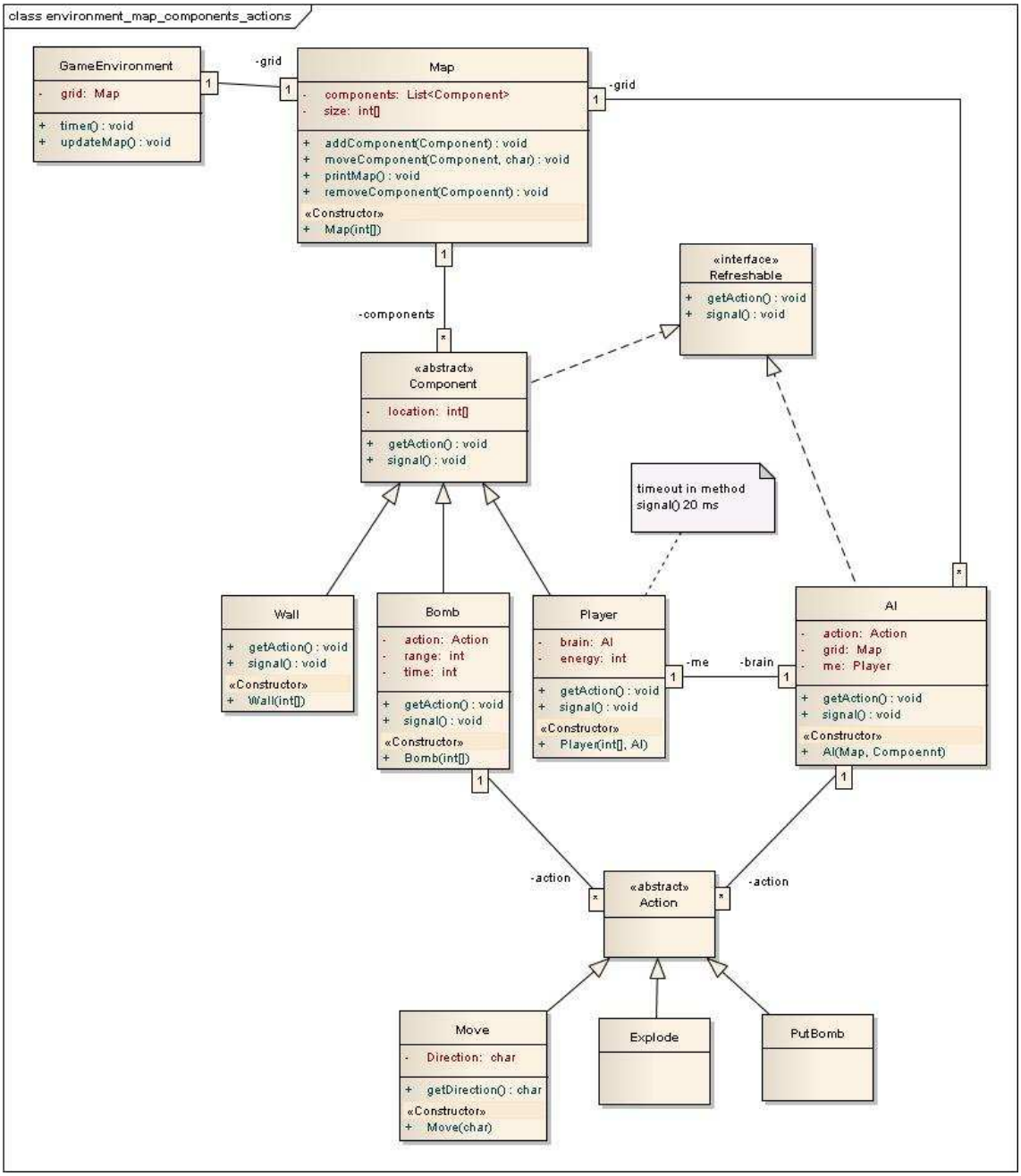
Číslo šprintu	**02**
Začiatok šprintu	26.10.2011
Koniec šprintu	09.11.2011
Príbehy (*userstories*)	• Herné prostredie • NEXT algoritmus (mód hry: získanie vlajky) • Prototypovanie sieťovej komunikácie

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.

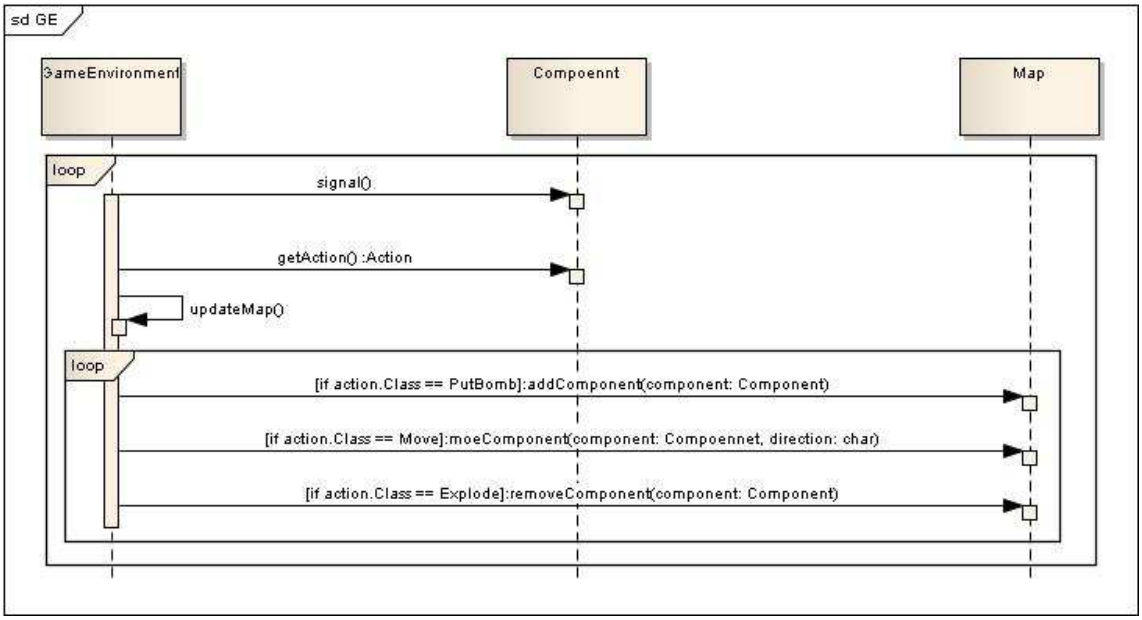
4.1.1.11 Explode

- 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.

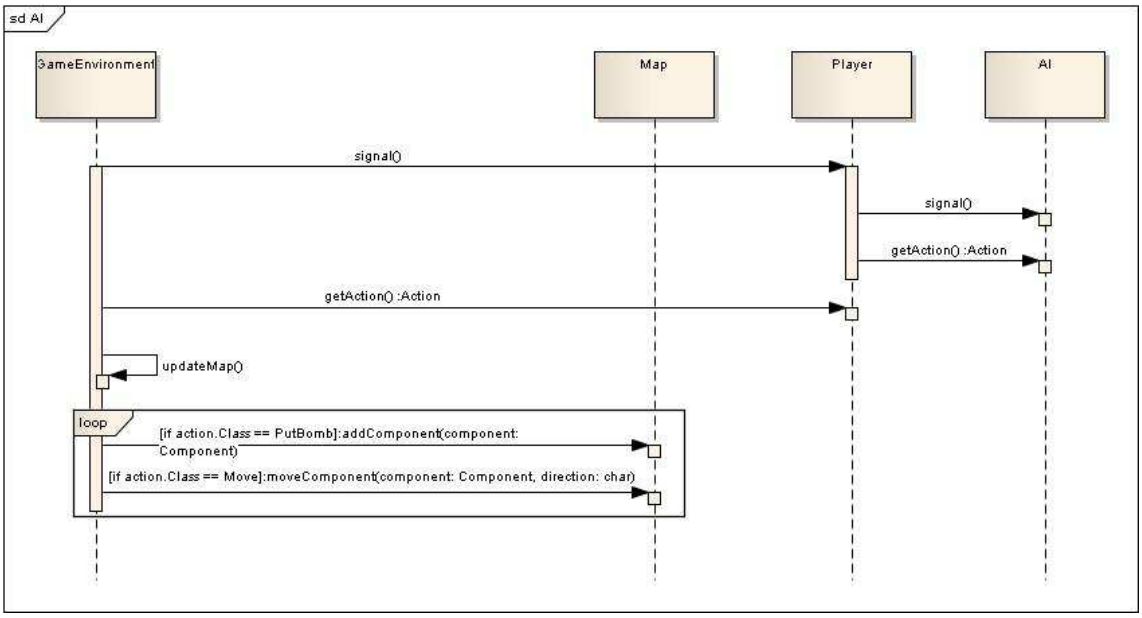
4.1.2.1 Základný postup vykonávania



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.

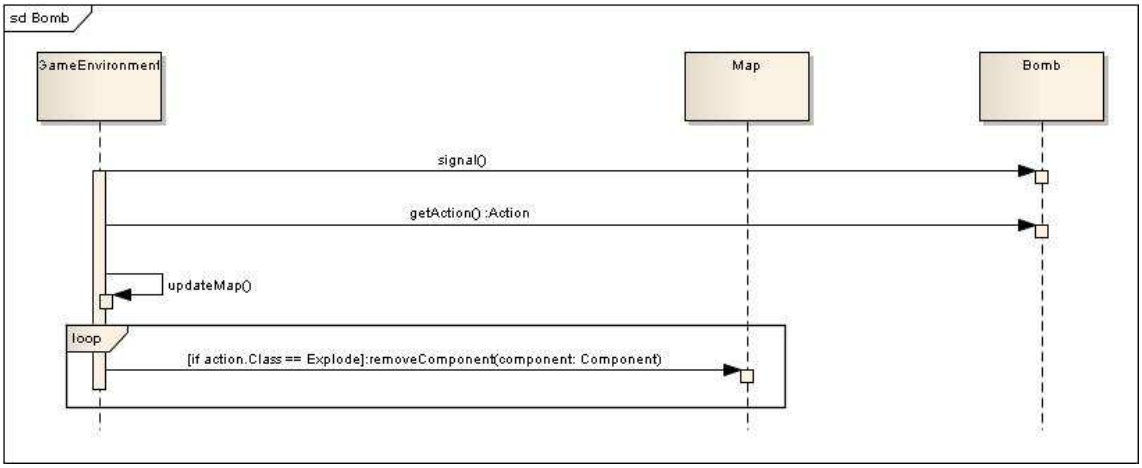
4.1.2.2 Akcie hráča (umelej inteligencie)



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.

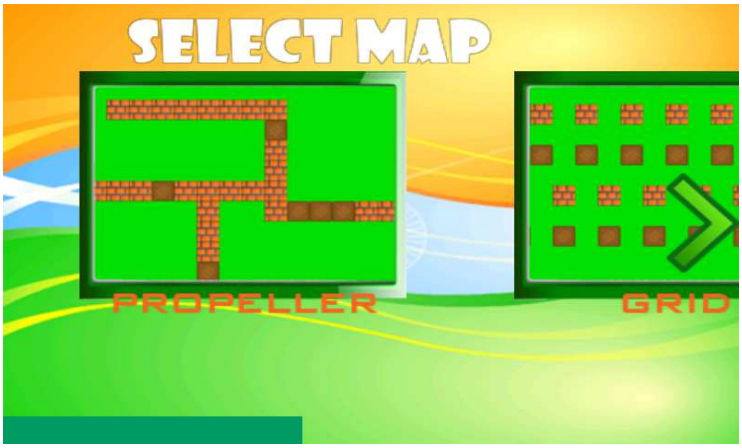
V *Hlavnom menu* sa zobrazujú dve postavy, ktoré zobrazujú človeka a "*humanoida*" v jednom tíme. Samotného *avatara* charakterizuje čiapka a okuliare podobne ako jeho samotná textúra v hre. Položky v menu sa zmenili podľa návrhu režimov hry a navigácie a to nasledovne:

- 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.

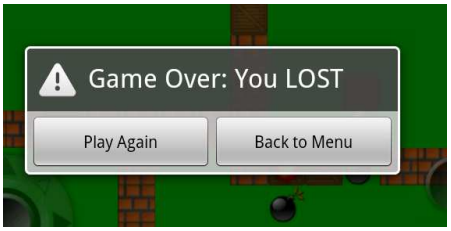
Pri ukončení hry je taktiež potrebné o tomto stave informovať používateľa ako aj vypísať, či používateľ vyhral alebo prehral. Na toto slúž dialóg, ktorý sa vyvolá v dvoch prípadoch:

- 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

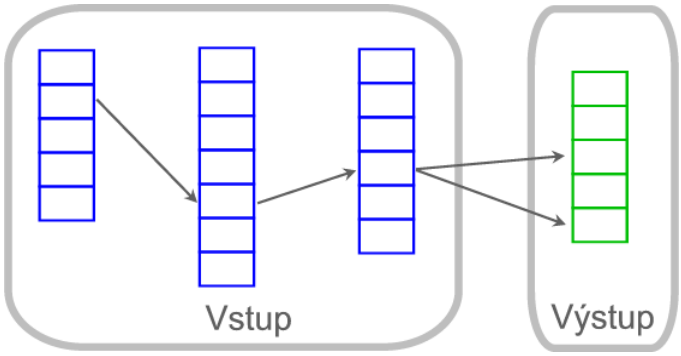
5.4 Markovovské siete

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.

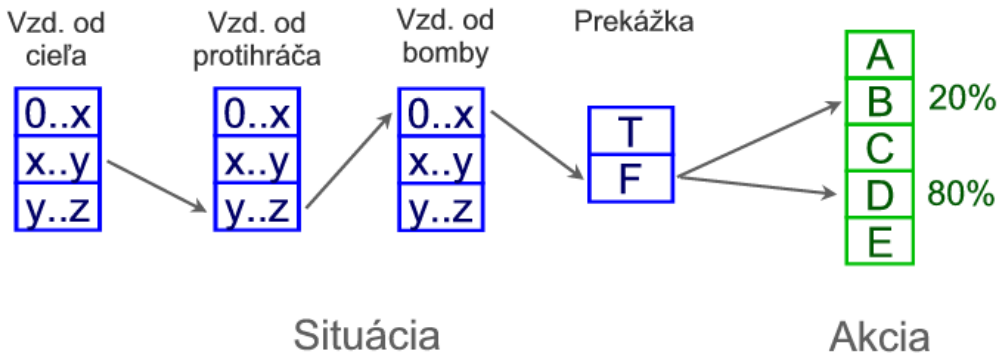
Používateľské akcie	Akcie UI
• **Posun hore / dole / doprava / doľava** • **Položenie bomby**	• Posun k cieľu • Posun k protihráčovi • Posun od protihráča • Posun od bomby • Položenie bomby

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

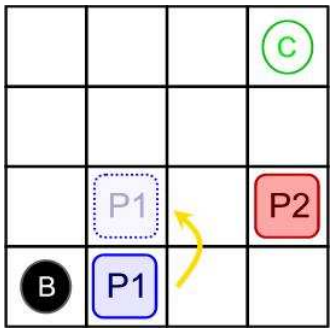
5.4.2.5.3 Stratégia 3

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*

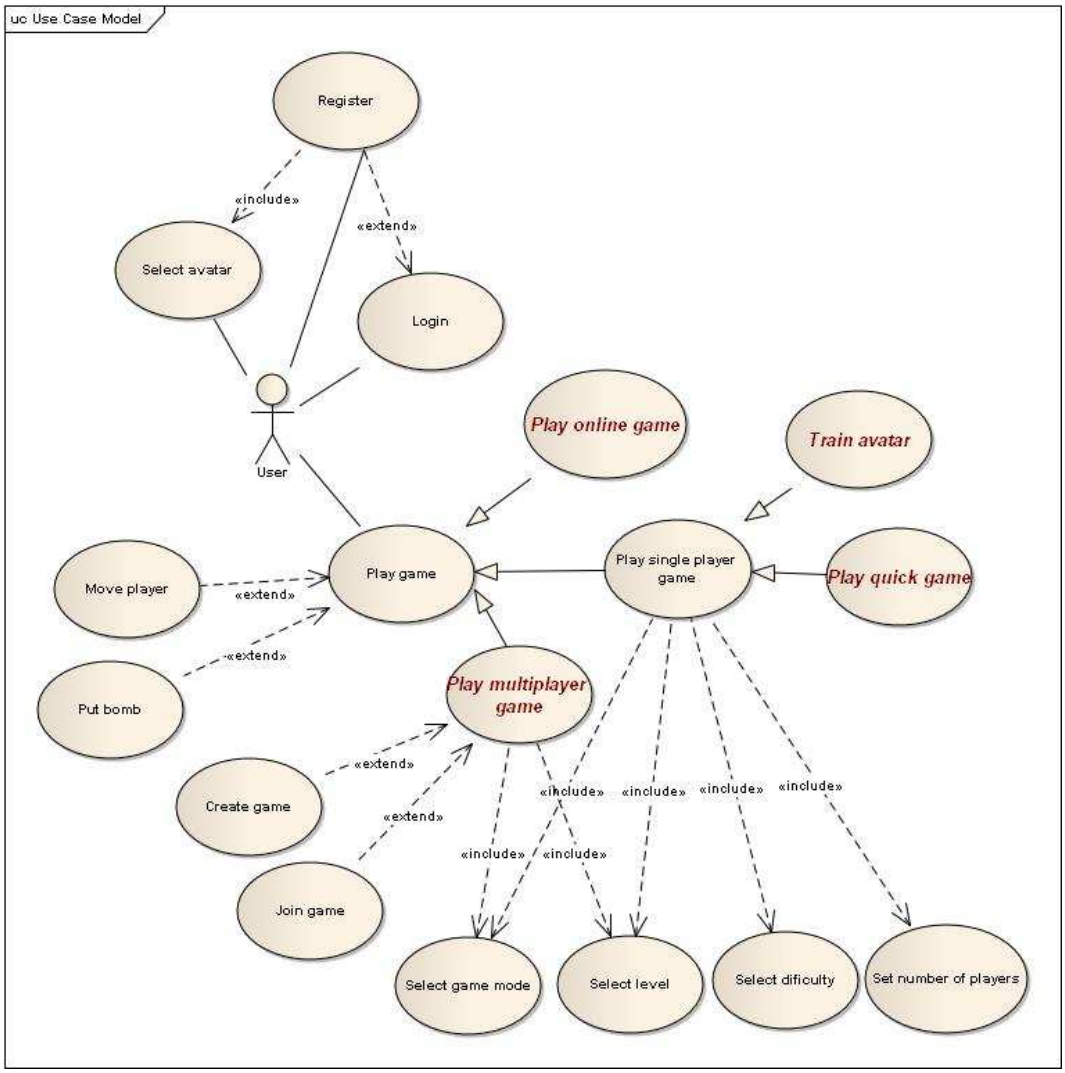
Ranking

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 ava*t*arov 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 po zimnom semestri

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.

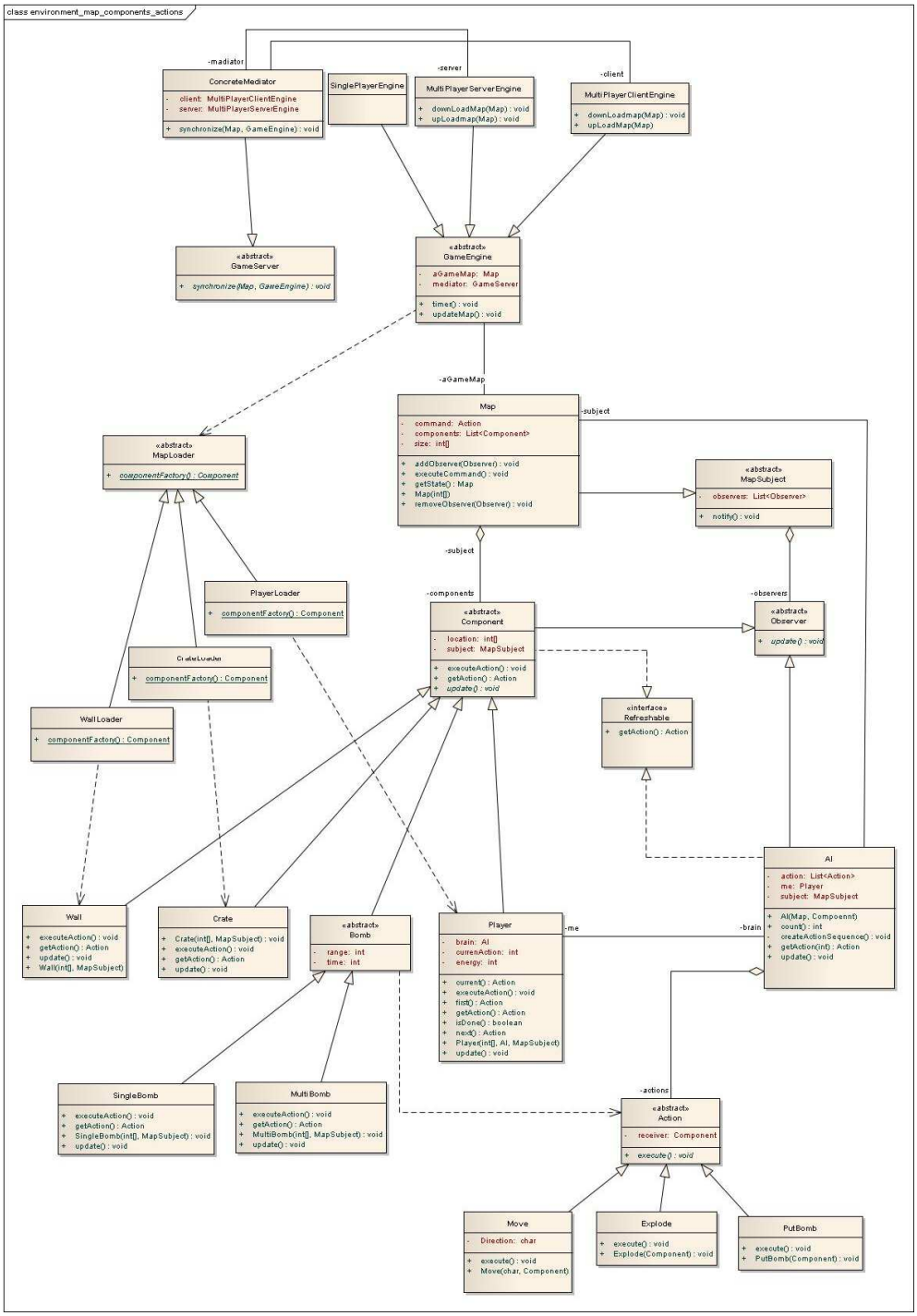
7.6 Plánované vylepšenia

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

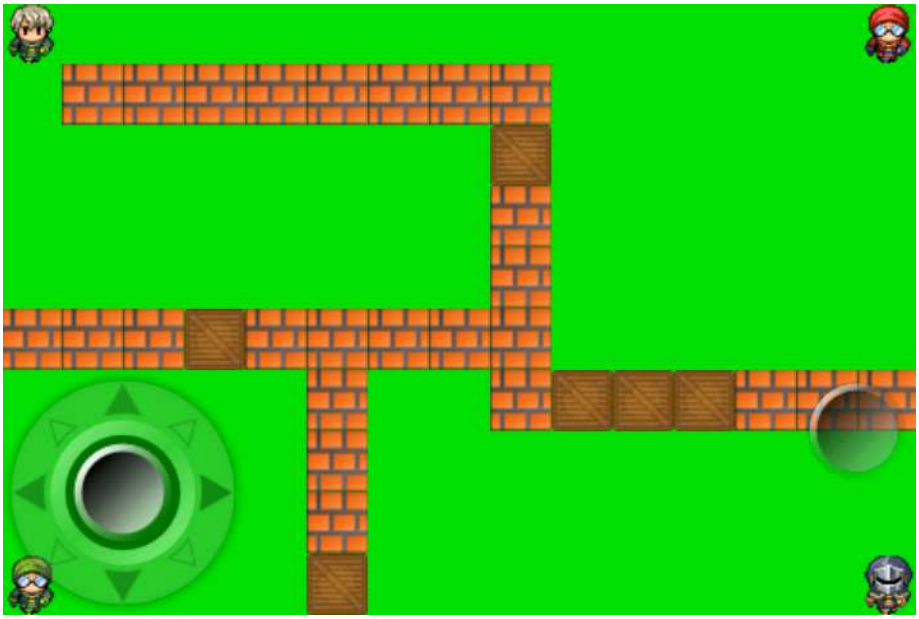
Na obrázku *Obr. 17* sa nachádza Class Diagram implementovaného prototypu



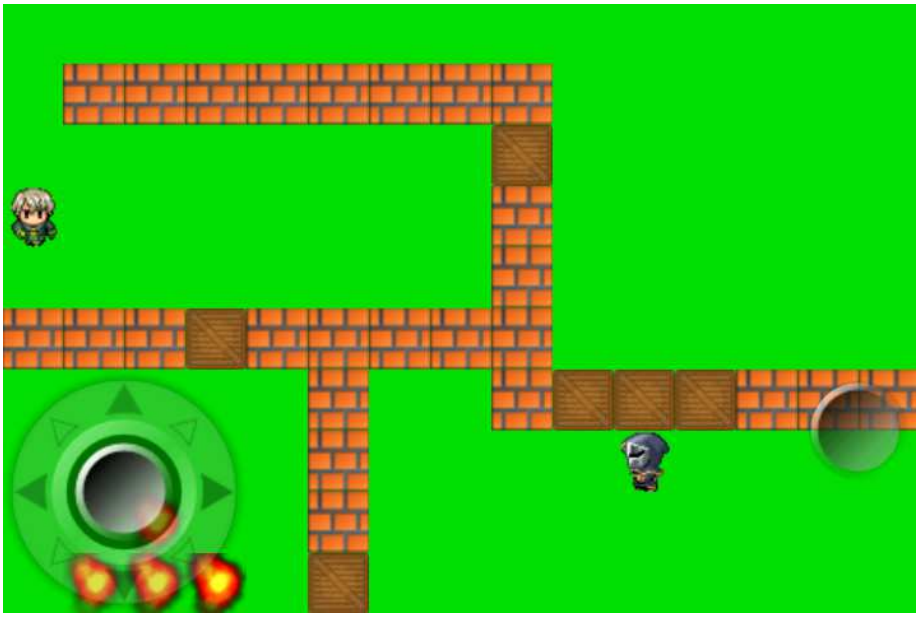
Obr. 17Class Diagram implementovaného prototypu

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

8 Šprint č.5

Číslo šprintu	**05**
Začiatok šprintu	13.2.2012
Koniec šprintu	27.2.2012
Príbehy (*userstories*)	• **Registrácia používateľa** • **Prihlasovanie používateľa** • **Herné nastavenia** • **Finalizácia UI založenej na Markovských sieťach** • **Analýza implementačných nástrojov pre Windows Phone**

8.1 Registrovanie používateľa

8.1.1 Analýza

Nový používateľ si pri prvom spustení hry musí založiť používateľské konto. Pri registrácii sa k nemu priradí používateľské meno a heslo, ktoré budú používané pri prihlasovaní.

8.1.2 Návrh

Po zadaní zvoleného používateľského mena a hesla sa z mobilného zariadenia zavolá príslušná metóda služby. V parametri sa odovzdáva meno a heslo používateľa. Na serveri sa skontroluje spĺňanie požiadaviek, ako jedinečnosť zvoleného mena a dĺžka hesla. O úspešnom zápise používateľa, prípadne o odmietnutej registrácii server pošle spätnú väzbu.

8.1.3 Implementácia

Na lokálnom zariadení sa zo *ServerAPI* volá metóda *signUpUser()*. Po kontrole údajov sa z *Java Servlet* zavolá metóda na zápis nového záznamu.

8.1.4 Testovanie

Názov	Registrácia používateľa	**Prípad použitia**	*UC User SignUp*
Rozhranie	Obrazovka registrácie	**ID testu**	TEST-US-01
Účel	Overovanie registrácie používateľa		
Vstupné podmienky	Prihlasovacie meno a dva krát prihlasovacie heslo		
Výstupné podmienky	*Používateľ je registrovaný a môže sa prihlásiť do aplikácie*		
Vyhotovil	Máté Fejes	**Dátum**	18.2.2012

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ použije tlačidlo "Create account"	Obrazovka registrácie	Obrazovka registrácie
2.	Používateľ zadá prihlasovacie meno a dva krát prihlasovacie heslo a klikne na " Create account "	Používateľ je registrovaný a zobrazí sa obrazovka prihlasovania	Používateľ je registrovaný a zobrazí sa obrazovka prihlasovania

Tab. 19 Akceptačný test k príbehu Registrácia používateľa

8.2 Prihlasovanie používateľa

8.2.1 Analýza

Používateľ sa po registrácii musí prihlásiť pomocou jeho používateľského mena a hesla. Pri ďalších spusteniach hry bude používateľ automaticky prihlásený, až kým sa neodhlási.

8.2.2 Návrh

Po zadaní mena a hesla sa tieto údaje prostredníctvom príslušnej metódy pošlú na server, kde sa skontroluje ich správnosť. O úspešnosti prihlásenia server pošle spätnú väzbu. Pokiaľ bolo prihlásenie úspešné, aktuálne prihlásený používateľ sa dočasne (do odhlásenia) uloží do pamäte lokálneho zariadenia.

8.2.3 Implementácia

Na lokálnom zariadení sa zo *ServerAPI* volá metóda *loginUser()*. Z *Java Servlet* sa zavolajú metódy na kontrolu existencie zadaného používateľa, následne kontrola správnosti hesla.

Tab. 20 Obrazovka Login

8.2.4 Testovanie

Názov	Registrácia používateľa		**Prípad použitia**	*UC User Login*
Rozhranie	Obrazovka prihlasovania		**ID testu**	TEST-UL-01
Účel	Overovanie prihlásenia používateľa			
Vstupné podmienky	Registrovaný používateľ			
Výstupné podmienky	*Používateľ je registrovaný a môže sa prihlásiť do aplikácie*			
Vyhotovil	Máté Fejes		**Dátum**	18.2.2012

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ zadá nesprávne prihlasovacie údaje a klikne na "Sign In"	Login neprebehne, zobrazí sa chybová správa	Login neprebehne, zobrazí sa chybová správa
2.	Používateľ zadá správne prihlasovacie údaje a klikne na "Sign In"	Používateľ je prihlásený a zobrazí sa menu	Používateľ je prihlásený a zobrazí sa menu

Tab. 21 Akceptačný test k príbehu prihlásenie používateľa

8.3 Herné nastavenia

8.3.1 Analýza

Pre potreby prispôsobenia hry používateľom a ich zručnostiam je potrebné samotnú hru upraviť jednotlivými nastaveniami. Medzi najdôležitejšie časti pátri úroveň jednotlivých *botov* pre zabezpečenie hrateľnosti pre rôznych používateľov. Taktiež je veľmi dôležité aby herné prostredie poskytovalo používateľovi istú rozmanitosť k čomu slúžia rôzne typy máp.

8.3.2 Návrh

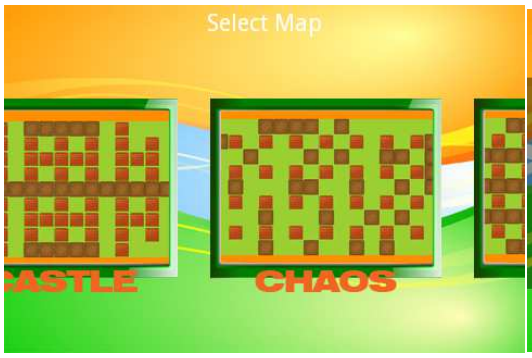
Pre rôzne úrovne *botov* boli navrhnuté 3 náročnosti, ktorým je venovaný ďalší používateľský príbeh v nasledujúcom šprinte. Mapy boli navrhnuté na základe klasického *grid* rozloženia pevných prekážok, ktoré boli rôzne doplňované posúvateľnými a zničiteľnými prekážkami. Pre mapy ako aj nastavenie úrovne náročnosti slúžia samostatné obrazovky.

Na obrazovke s nastaveniami je taktiež možné definovať aj počet hráčov na mape od 2 do 4. Pre jednotlivé herné režimy to znamená, že hra je spustená s potrebným počtom účastníkov prirodzených pre daný herný režim a zvyšný počet sa doplní *botmi.*

8.3.3 Implementácia

Samotné mapy boli implementované vytvorením súboru so znakmi reprezentujúcimi jednotlivé objekty mapy tak ako to bolo definované pre triedu *Maploader* v predošlých šprintoch. Aktuálne je

vytvorených 10 rôznych máp. Nasledujúce *screenshot*-y týchto máp sú zobrazené priamo na klientskom zariadení v podobe horizontálneho zoznamu. Po zvolení mapy sa zobrazí obrazovka pre výber úrovne náročnosti hry a výberu *avatara* ak je táto možnosť pre daný režim hry dostupná. V prípade, že predvolený *avatar* nie je dostupný (v prípade ak sa zúčastňuje *online turnaja*) nie je možné spustiť hru. Používateľ musí tohto *avatara* z turnaja stiahnuť alebo si zvoliť iného. Po vyplnení všetkých položiek môže byť hra spusten. Táto obrazovka bola implementovaná pre všetky dostupné režimy hry okrem *Online hry*.



Obr. 20 Herné nastavenia

Obr. 21 Výber Mapy

8.3.4 Testovanie

Názov	Nastavenia hry	**Prípad použitia**	*UC Game SetUp*
Rozhranie	Obrazovka máp a nastavenia hry	**ID testu**	TEST-GS-01
Účel	Overenie hry spustenej so správnymi nastaveniami		
Vstupné podmienky	Prihlásený používateľ, výber niektorého z herných režimov		
Výstupné podmienky	*Hra je spustená so správnymi nastaveniami*		
Vyhotovil	Ľuboš Gelányi	**Dátum**	23.2.2012

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ vyberie na obrazovke s mapami konkrétnu mapu	Na obrazovke s nastaveniami hry sa zobrazí správny názov mapy, následne je hra spustená na správnej mape	Na obrazovke s nastaveniami hry sa zobrazí správny názov mapy
2.	Na obrazovke s nastaveniami sa zvolí úroveň náročnosti *botov*	Hra je spustená v správnej úrovni náročnosti	Hra je spustená v správnej úrovni náročnosti
3	Používateľ zvolí počet hráčov na mape	Hra je spustená so správnym počtom hráčov	Hra je spustená so správnym počtom hráčov
4	Používateľ zvolí *avatar*a (v prípade, že aktívny)	Hra je spustená so správnym *avatarom*	Hra je spustená so správnym *avatarom*

Tab. 22 Akceptačný test pre príbeh nastavenia hry

8.4 Finalizácie UI založenej na Markovských sieťach

8.4.1 Analýza

Prebehla v predošlých šprintoch.

8.4.2 Návrh

Prebehol v predošlých šprintoch.

8.4.3 Implementácia

Z praktického hľadiska bolo potrebné mierne upraviť základnú štatistiku *Markovovských* sietí kvôli miernym nedostatkom z pohľadu používateľa ako hráča. Problém s pôvodným riešením inteligencie bol v tom, že na začiatku hry sa *avatar* vôbec nepohyboval po mape a predstavoval tak jednoduchý cieľ pre súpera. Rovnako aj prvé výsledky učenia sa prejavovali omnoho neskôr, ako boli očakávané.

Riešenie opísaného problému spočívalo v zmene definovaných počiatočných hodnôt štatistických záznamov. Zmeny sú uvedené v nasledujúcej tabuľke:

	Pôvodné riešenie	Aktuálne riešenie		
Situácia	každá situácia	vedľa súpera	vedľa bomby	inak
Akcia	Pravdepodobnosť			
Pohyb od bomby	16,667%	2,5%	70%	2,5%
Pohyb od súpera	16,667%	2,5%	2,5%	2,5%
Pohyb k súperovi	16,667%	2,5%	2,5%	20%
Pohyb k cieľu	16,667%	20%	20%	70%
Položenie bomby	16,667%	70%	2,5%	2,5%
Státie	16,667%	2,5%	2,5%	2,5%

Tab. 23 Popis riešenia pre počiatočné vyplnenie *Markovovských* sietí podľa situácie hry

Výsledkom tejto zmeny bolo rýchlejšie adaptovanie sa *avatara* na rozhodnutia používateľa, čo sa dobre odrazilo na prvotnom učení. *Avatar* tak začal zreteľne opakovať používateľove kroky omnoho skôr, takmer podľa očakávaní.

Následné vylepšenie ešte stále existujúceho problému (rýchlosť prvotného zreteľného učenia) vidíme vo vytvorení/predtrénovaní tejto inteligencie manuálnym hraním hry. To ale nie je nateraz prioritou, nakoľko je to úlohou používateľa ako hráča.

8.4.4 Testovanie

Názov	Učenie *avatara* na pohyb	**Prípad použitia**	*UC Learning*
Rozhranie	*Android*	**ID testu**	10
Účel	Overiť učenie sa *avatara* s inteligenciou *„Markovovske siete“*		
Vstupné podmienky	Spustená hra s *avatarom* s inteligenciou *„Markovovske siete“*		
Výstupné podmienky	Spustená hra, *avatar* opakuje spôsob boja		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ sa presunie na mape do vzdialenosti 2 políčka od *avatara*.	Postavička sa posunie podľa pokynov používateľa.	Postavička sa posunula podľa pokynov používateľa.
2.	Používateľ sa posunie na mape do vzdialenosti približne 10 políčok od *avatara*.	Postavička sa posunie podľa pokynov používateľa.	Postavička sa posunula podľa pokynov používateľa.
3.	Používateľ zopakuje test od kroku 1 práve 20 krát.	*Avatar* sa začne zreteľne pohybovať smerom k používateľovi.	*Avatar* sa začal presúvať k používateľovi, každým cyklom testu zreteľnejšie.

Tab. 24 Akceptačný test pre sledovanie učenia sa štýlu boja od používateľa

Názov	Zabúdanie *avatara*	**Prípad použitia**	*UC Forgetting*
Rozhranie	*android*	**ID testu**	11
Účel	Overiť preučenie *avatara* s inteligenciou *„Markovovske siete“*		
Vstupné podmienky	Spustená hra s naučeným *avatarom* s inteligenciou *„Markovovske siete“*		
Výstupné podmienky	Spustená hra, *avatar* sa prestal hýbať		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ stojí a čaká na mieste.	*Avatar* sa prenaučí z pohybu k používateľovi na akciu stáť.	Spočiatku sa avatar pohyboval k používateľovi, neskôr zastal. Preučenie trvalo dlhšie ako učenie, bolo však razantné.

Tab. 25 Akceptačný test pre sledovanie preučenia sa štýlu boja na iný podľa používateľa

8.5 Analýza implementačných nástrojov pre Windows Phone

8.5.1 Analýza

V tomto šprinte sme sa pre hru *Smart bomber* začali zaoberať rozhraním Windows Phone. Začali sme základnou analýzou dostupných riešení pre tvorbu herných aplikácií. Analyzovali sme päť vývojových prostredí pre ovládanie grafiky:

- WMGL - WM PPC Graphics Library, programový jazyk C++/C#, voľná licencia
- Gapidraw - robustnejšia knižnica, implementované funkcionality, platená licencia
- Direct3D Mobile - takmer základná použiteľnosť, voľná licencia, programovanie v 3D
- Cocos2DX - vyhovujúca funkcionalita pre našu aplikáciu, open source, alpha verzia
- Základný XNA framework - plná podpora na všetkých zariadeniach, voľná licencia, podpora pre tvorbu základného algoritmu hier

Nakoniec sme sa rozhodli pre vytvorenie základnej architektúry hry postavenej na XNA. Implementačným jazykom prostredia XNA je samozrejme C#. Nakoľko neplánujeme vytvorenie plnej funkcionality pre platformu WP7, nie je potrebné využívať iné externé knižnice pre vývoj hry.

9 Šprint č.6

Číslo šprintu	**06**
Začiatok šprintu	27.2.2012
Koniec šprintu	13.3.2012
Príbehy (*userstories*)	• **Manažment hry** • **Základ hry pre Windows Phone** • **3 úrovne *Evil-bota*** • **Herný Server** • **Učenie sa od používateľa**

9.1 Manažment hry

9.1.1 Analýza

Pre ovládanie samotnej hry nie je nutné len ovládanie hráča ale aj ostatné informačné panely a možnosť hru prerušiť a znova spustiť. Vo väčšine hier sa zobrazuje skóre, čas hry ako aj pauza pre hru.

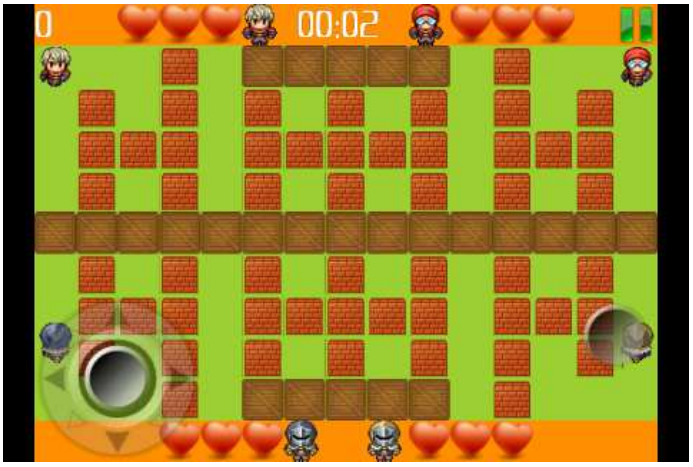
9.1.2 Návrh

Pre manažment hry boli navrhnuté dva informačné panely, ktoré sú umiestnené v hornom a dolnom rade obrazovky. Horný rad obsahuje informáciu o skóre, počte životov pre prvých dvoch hráčov ako aj čas hry. Nakoniec v pravom hornom rohu je umiestnené tlačidlo pauzy. Spodný informačný panel obsahuje informáciu o životoch ďalších dvoch hráčov v prípade, že sa nachádzajú na mape.

9.1.3 Implementácia

Všetky navrhnuté komponenty sú umiestnené na scénu v navrhnutých pozíciách. Pri každej akcii sa overuje výbuch hráča či boxu pre pripisovanie bodov. Ak je hráč zasiahnutý, v prípade, že ešte má viac ako jeden život, pozícia sa mu nastaví na predvolenú pozíciu a počet životov sa mu zníži o jeden. V opačnom prípade sa z mapy stratí. Celkový manažment hry prebieha na konečnej vykresľovanej vrstve, kde sa overujú prítomnosti jednotlivých textúr. Všetky potrebné informácie sa však zapisujú do jednotlivých parametrov v triede *Player.*

Tlačidlo pauza vyvoláva upravený *android* dialóg, ktorý obsahuje dve tlačidlá a to pokračovanie v hre alebo ukončenie hry. V prípade pauzy je jednoducho pozastavený *gameEngine* a v prípade ukončenia hry je úplne zastavený a následne sa hra naviguje do základného menu.



Obr. 22 Informačné panely v hre

9.1.4 Testovanie

Názov	Manažment hry	**Prípad použitia**	*UC Game Management*
Rozhranie	Obrazovka hry	**ID testu**	TEST-GM-01
Účel	Overenie ovládania a poskytovania informácií v hre		
Vstupné podmienky	Prihlásený používateľ spustí jeden z herných režimov		
Výstupné podmienky	*Hra zobrazuje korektné informácie a reaguje správne na používateľove vstupy*		
Vyhotovil	Luboš Gelányi	**Dátum**	10.2.2012

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ spustí a hrá hru	Informačné panely zobrazujú správne informácie	Informačné panely zobrazujú správne informácie
2.	Používateľ stlačí pauzu	Hra je pozastavená	*Hra je pozastavená*
3	Používateľ vyberie možnosť "Resume Game"	Hra pokračuje, čas hry začne znovu bežať	Hra pokračuje, čas hry začne znovu bežať
4	Používateľ vyberie možnosť "End Game"	Zobrazí sa hlavné menu	Zobrazí sa hlavné menu

Tab. 26 Akceptačný test pre Manažment hry

9.2 Základ hry pre Windows Phone

9.2.1 Analýza

Prebehla v predošlých šprintoch

9.2.2 Návrh

Základ aplikácie na WP

Cieľom tohto šprintu v oblasti vytvorenia základu aplikácie bolo implementovať do značnej miery architektúru hry a to po logickej úrovni, adekvátne k hre na platforme *Android*. Nakoľko bolo naplánované vytvorenie základu len pre oblasť módu jedného hráča (ďalej *Single Player*), táto úloha bola splnená nad rámec zadania. Okrem jednoduchého základu hry sa tak podarilo vytvoriť funkčný prototyp *Single Player*-a (posúvanie boxov, pohyb postavy, položenie bomby, kolízia objektov a výbuch bomby) bez inteligencie súpera. Po grafickej stránke bol tento prototyp zhodný s aplikáciou na platforme *Android*.

9.2.3 Implementácia

Prototyp hry na platforme Windows Phone nebol v plnom súlade s existujúcim riešením pre platformu *Android* a to po architektonickej stránke. Tento nedostatok sme riešili počas celého šprintu, pretože *Android* aplikácia je značne robustná pre požiadavky prototypu na WP. Vynechali sme väčšiu časť riešení, ktoré sa netýkali priamo *Single Player* módu. Z funkčného hľadiska sme neobohatili prototyp aplikácie na WP, napriek tomu sme úspešne znížili implementačné rozdiely medzi platformami.

9.2.4 Testovanie

Testovanie prebehlo v ďalšom šprinte

9.3 3 úrovne Evil-bota

9.3.1 Analýza

Praktické testy ukázali, že vytvorený *EvilBot* je príliš silným súperom pre človeka, i *avatarov*. Je nutné vytvoriť minimálne tri úrovne náročnosti, aby si hru proti *EviBot*-ovi mohol zahrať aj menej skúsený hráč prípadne nenatrénovaný *avatar*.

9.3.2 Návrh

Aktuálny *EvilBot* predstavuje najlepšieho možného bota, ostatné úrovne potom vzniknú zhoršením jeho vlastností a zmenou jeho chovania.

Návrh počíta s vytvorením týchto piatich kategórií:

- *hard* – pôvodný *EvilBot*
- *normal* – pôvodný *EvilBot* ale pri každom jeho kroku je 25% pravdepodobnosť, že vykoná náhodný krok (urobí chybu)
- *easy* – rovnako ako pri *normal* ale pravdepodobnosť chyby je až 50%
- *boxdestroyer* – neútočí na ostatných hráčov, iba sa bráni ak je ohrozovaný
- *random* – všetky akcie sú náhodné, určené len na testovacie účely

Týchto päť úrovní pokryje jednak požiadavky na testovanie *avatarov* a jednak požiadavky na normálne hranie.

9.3.3 Implementácia

Celý kód sa nachádza v triede *EvilBotAI.java* pričom na svoje fungovanie využíva funkcie triedy *AstarAnalyzer.java*. Požadovaná úroveň sa nastavuje v konštruktore ako *enum* hodnota premennej *difficulty*.

9.3.4 Testovanie

Názov	Nastavenie easy úrovne	**Prípad použitia**	*UC Bot Difficulty*
Rozhranie	Obrazovka bojiska	**ID testu**	TEST - BD - 01
Účel	Overenie chybového správania *EvilBot*-a po nastavení úrovne easy.		
Vstupné podmienky	Na mape sa nachádza aspoň jeden *EvilBot* s nastavenou úrovňou easy. Na mape sa nachádza ľubovoľný iný hráč, pričom navzájom sa nachádzajú na rovnakom riadku alebo stĺpci a medzi nimi sa nenachádza žiadna prekážka.		
Výstupné podmienky			

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Žiadna.	*EvilBot* sa približuje k súperovi no jeho cesta nie je najkratšia možná (robí chyby)	*EvilBot* sa začal približovať k súperovi no počas cesty niekoľko krát zastal, vrátil sa o políčko naspäť a vybočil aj dole.

Tab. 27 Akceptačný test pre príbeh 3 úrovne *Evil-Bota*

9.4 Herný sever

9.4.1 Analýza

Nakoľko hra *Smart Bomber* ponúka funkcionality, ako súťaženie *avatarov*, lokalizáciu používateľov a pod., určité činnosti sú podporované vzdialeným serverom, s ktorým jednotlivé zariadenia komunikujú prostredníctvom internetu. Bez sieťového pripojenia tieto činnosti nie je možné vykonať. Vzdialený server má dve hlavné komponenty*: centrálne úložisko dát* a *online súťaženie avatarov*. Na lokálnych zariadeniach sú dočasne uložené len vybrané údaje, ako napr. aktuálne prihlásený používateľ a pod. Pri určitých činnostiach používateľa sa zmeny relevantných dát automaticky aktualizujú na serveri.

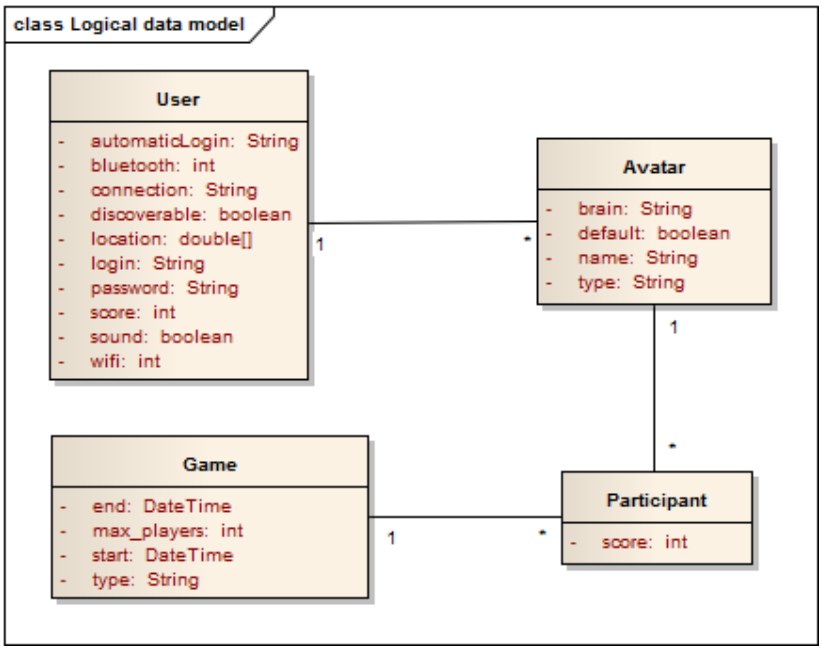
9.4.2 Návrh

9.4.2.1 Centrálne úložisko

V záujme vytvorenia centrálneho úložiska všetkých používateľských dát sme navrhli dátovú štruktúru, ktorá spĺňa požiadavky na udržiavanie relevantných informácií. Ďalej bolo potrebné zabezpečiť prenos dát medzi severom a mobilnými zariadeniami.

9.4.2.2 Dátová štruktúra

Na obr. 23 je znázornený logický diagram dátovej štruktúry, ktorá zodpovedá organizácie dát na vzdialenom serveri.



Obr. 23 Logický dátový model

V uvedenom diagrame sú zahrnuté štyri dátové entity, ktoré sú v hre prítomné. Entita *User* udržiava údaje používateľa, ďalej agreguje skupinu *avatarov*, ktoré sú reprezentované inštanciami entity *Avatar*. Entita *Game* reprezentuje odohrané hry bez ohľadu na jej typ. *Participant* tvorí relačnú entitu medzi *Avatar* a *Game*. Vyjadruje, ktorý *avatary* sa zúčastnili danej hry, zároveň udržiava počet získaných bodov.

9.4.2.3 Prenos dát

Pre umožnenie vhodného spôsobu komunikácie sme identifikovali činnosti používateľa, ktoré vyžadujú internetové pripojenie, následne sme ich zoskupovali a špecifikovali na nižšej úrovni tak, aby zodpovedali navrhnutej dátovej štruktúre. Identifikované akcie sme realizovali v podobe webovej služby. Zoznam činností podporovaných webovou službou je uvedený nižšie:

- Registrovanie používateľa
- Prihlasovanie používateľa
- Správa dát používateľa
- Získavanie zoznamu používateľov
- Správa polohy používateľa
- Správa *avatarov*
- Získavanie herných štatistik

9.4.3 Implementácia

Pre komunikáciu s webovou službou sme vytvorili rozhranie v triede *ServerAPI*. Rozhranie obsahuje abstraktné metódy, ktoré pokryjú vyššie diskutované činnosti. Metódy rozhrania sú implementované v *Java Servlet*, ktorý ďalej volá metódy triedy *DatabaseConnector*, ktoré podporujú primitívne operácie nad databázou.

Webová služba a *online* súboje sú implementované v jazyku *Java EE*, komunikácia medzi serverom a mobilnými zariadeniami je zabezpečená binárnym protokolom pre webové služby *Hessian*. Komunikácia s databázou je realizovaná pomocou objektovo-relačného mapovača *Hibernate*.

9.4.4 Testovanie

Funkcionalita herného servera sa testuje pri iných používateľských príbehoch, ktoré ho využívajú.

9.5 Učenie sa od používateľa

Pre naše potreby je veľmi dôležité, aby sa *avatari* počas tréningu dokázali učiť od používateľa. *Avatari* sledujú pohyby používateľa. Nesnažia sa vytrénovať na dokonalých *avatarov*, ale na takých, aby sa čo najviac priblížili hernému štýlu svojmu trénerovi – používateľovi.

9.5.1 Analýza

Učenie od používateľa nastáva len počas herného módu *Training*. Počas hrania tohto módu sa *avatar* snaží sledovať aktuálnu situáciu používateľa a k nemu príslušné kroky.

Keďže sledovanie, analyzovanie a vyhodnocovanie celej mapy by bolo veľmi zložité, *avatar* si bude všímať len blízke okolie hráča. Bude sledovať, či sú v blízkosti iný hráči, škatule alebo bomby a prislúchajúce reakcie používateľa.

V prípade, že používateľ na rovnakú situáciu vykoná rôzne kroky, bude sa brať do úvahy tá reakcia, ktorá bola vykonávaná najčastejšie. Túto reakciu sa bude snažiť aplikovať pri vlastnom hraní. Avšak vplyvom určitej náhodnosti je možné, aby *avatar* vykonal aj iný krok, ako bol prioritne naučený. Tým sa zabezpečí, že sa naučí najčastejšiu reakciu na danú situáciu, ale nevylučujú sa ani ostatné reakcie.

9.5.2 Návrh

Učenie bude prebiehať v dvoch fázach. Prvá fáza bude pozorovanie, druhá preučenie *avatarovho* mozgu.

Pri pozorovaní bude *avatar* sledovať aktuálnu situáciu a príslušnú reakciu. Tieto údaje bude počas tréningu zbierať. Na konci tieto údaje spravuje a vytvorí vzor, podľa ktorého sa bude preúčať na nové správanie.

Preučenie prebehne po skončení tréningu a vytvorení vzoru správania. *Avatar* si načíta doterajší mozog, s ktorým sa následne snaží čo najviac priblížiť novým vzorom správania. Tento mozog si postupne upravuje. Upravovaním sa približuje najnovšiemu správaniu používateľovi. Avšak vďaka tomu, že vychádza zo svojho pôvodného mozgu si zachováva aj časť starých vlastností.

9.5.3 Implementácia

Celé učenie *avatara* sa odohráva v triede *BackwardsNeuralNetwork.java*. Učenie stojí na dvoch základných metódach. Prvá je *setUserMove(Player)*, ktorá vyhodnocuje aktuálnu situáciu

a používateľovu reakciu. Reakciu vyhodnocuje na základe analyzovania postavenia hráča po vykonaní ďalšieho kroku.

Následne sa po skončení celého tréningu vytvoria nové vzory správania sa, podľa ktorých sa bude neurónový mozog preúčať. Vytváranie týchto vzorov zabezpečuje funkcia settingNewPatterns(). Následne sa len zavolá klasické učenie, pričom sa vychádza zo starého (pôvodného) *avatarovho* mozgu, pričom sa snaží naučiť na nové správanie.

9.5.4 Testovanie

Názov	Učenie *avatara*	**Prípad použitia**	**UC Relearning**
Rozhranie	Herná obrazovka	**ID testu**	**TEST-GLO-01**
Účel	Overiť učenie *avatara*		
Vstupné podmienky	Spustený tréning a *avatar* s neurónovým mozgom		
Výstupné podmienky	Upravený (preučený) mozog *avatara*		
Vyhotovil	Ľuboš Masný	**Dátum**	6.4.2012

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Vytvorenie *avatara* s neurónovým mozgom	*Avatar* pridaný do stajne *avatarov*	Nový *avatar* v stajni *avatarov*
2.	Spustenie tréningu s novým *avatarom*	Spustenie hry, kde sa nachádza hráč a novovytvorený *avatar*	Spustená hra, kde je hráč a *avatar*
3.	Víťazstvo hráča	Koniec hry. *Avatar* sa preučí pohyby od hráča	Zmenil sa súbor, kde je uložený neurónový mozog. To znamená, že u *avatara* nastalo učenie

Tab. 28 Akceptačný test pre príbeh Výbuch bomby

10 Šprint č.7

Číslo šprintu	**07**
Začiatok šprintu	13.3.2012
Koniec šprintu	19.3.2012
Príbehy (*userstories*)	• **Správa dát používateľa** • **Quick Game pre Windows Phone** • **Dokončenie A* algoritmu** • **Správa polohy používateľa** • **Geoplatforma** • **Uloženie Neurónového mozgu**

10.1 Správa dát používateľa

10.1.1 Analýza

Používateľ má možnosť na prezeranie a editovanie svojich údajov, zároveň systém je schopný zmeniť určité nastavenia používateľa.

Táto funkcionalita podporuje:

- Nastavenie WiFi ID
- Aktualizovanie skóre
- Zmenu prihlasovacích údajov
- Zmenu vlastnosti *discoverable* na mape

10.1.2 Návrh

Pri žiadosti o poskytnutie údajov používateľa server pošle na mobilné zariadenie celý obsah záznamu tabuľky *User*, ktorý reprezentuje prihláseného používateľa. Pri spätnom posielaní záznamu používateľa sa zmenené údaje prepíšu v centrálnej databáze.

10.1.3 Implementácia

Na získavanie údajov používateľa sa volá zo *ServerAPI* metóda *getUser()*, ktorá vráti inštanciu objektu *User*. Metóda *setUser()* triedy *ServerAPI* aktualizuje údaje používateľa odovzdaného v parametri.

10.1.4 Testovanie

Názov	Správa dát používateľa	**Prípad použitia**	*UC User Management*
Rozhranie	Nastavenia hry, Multiplayer, Hra	**ID testu**	TEST-UM-01
Účel	Overenie uloženia zmenených dát		
Vstupné podmienky	Prihlásený používateľ		
Výstupné podmienky	*Nahraté dáta sú rovnaké aké boli naposledy pri ich uložení*		
Vyhotovil	Máté Fejes	**Dátum**	15.3.2012

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ zmení akékoľvek nastavenia v aplikácií, následne sa odhlási a prihlási	Nahraté dáta sú rovnaké aké boli naposledy pri ich uložení	Nahraté dáta sú rovnaké aké boli naposledy pri ich uložení

Tab. 29Akceptačný test pre správu dát používateľa

10.2 Quick Game pre Windows Phone

10.2.1 Analýza

Prebehla v predošlých šprintoch

10.2.2 Návrh

Prebehol v predošlých šprintoch

10.2.3 Implementácia

Prototyp hry pre platformu WP bolo potrebné obohatiť z aspektu kvality o nasledovné doplnky:

- Animácia pohybu a akcií hráčov a objektov
- Animácia efektu výbuchu bomby
- Oddialenie výbuchu bomby a trvanie pôsobenia vzniknutej explózie
- Viac - dotykové ovládanie virtuálnym ovládačom

Multitouch ovládanie postavičky používateľa spočívalo v preimplementácii čítania dotyku používateľa z XNA triedy *VirtualPad*. Pôvodné riešenie spracovávalo dotyky zo zariadenia pomocou tzv. „gest“. Uvažované boli gestá typu „tap“ (ťuknutie, ťapnutie) a „free drag“ (posunutie prsta po displeji). Čítanie vstupu z displeja pomocou gest však nepodporovalo stlačenie viacerých virtuálnych kláves naraz, preto sme znížili úroveň čítania na získavanie bodov dotyku pomocou volania metódy „TouchPanel.GetState()“. Reprezentácia výstupu tejto metódy bola množina stlačených bodov na displeji v čase volania. Vďaka tomu sme vedeli určiť stlačenie všetkých tlačidiel na obrazovke aj pri použití viacerých prstov.

Zmeny týkajúce sa animácie pohybu a výbuchov úzko spolu súvisia. Pre ich implementáciu sme museli mierne zmeniť architektúru vykonávania inštrukcií na úrovni aktualizovania stavu prostredia a jeho vykresľovania. Pôvodné riešenie aktualizovalo stav hry každých 300 *ms* a najmä vykreslenie zmeny mapy sa uskutočnilo obdobne každých 300 *ms*. Aktuálne riešenie sme navrhli a implementovali ako vykonávanie desiatich menších cyklov trvajúcich 30 *ms*, pričom počas posledného menšieho cyklu sa vykonali tie inštrukcie (aktualizácia mapy), ktoré boli vykonané každých 300 *ms* v pôvodnom riešení.

Vďaka vytvoreniu týchto menších cyklov sme boli schopný vykresľovať zmenu mapy v animácii. Pohyb bol implementovaný zapamätaním si aktuálneho a predošlého stavu (z ktorého vznikol aktuálny). Po dobu ďalších 270ms (9 menších cyklov) sa postupne vykresľovala zmena pozície postavičiek na mape.

Po dosiahnutí 10-teho menšieho cyklu sa postavička vykreslila na aktuálnom umiestnení a na predošlé umiestnenie sa zabudlo - ukončila sa animácia pohybu.

10.2.4 Testovanie

Názov	Ovládanie postavičky	**Prípad použitia**	UC Markow
Rozhranie	Windows Phone 7	**ID testu**	TEST - MT - 01
Účel	Overiť ovládateľnosť pridelenej postavičky a pozíciu kamery		
Vstupné podmienky	Hra spustená s virtuálnym ovládačom		
Výstupné podmienky	Posunutá postavička v mape a posunutá kamera zobrazenia		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ stlačí tlačidlo pre smer doprava na virtuálnom ovládači	Postavička sa posunie o jedno políčko doprava, postavička bude stále v strede obrazovky	Postavička sa posunula o jedno políčko doprava animovaným pohybom, postavička je stále v strede obrazovky
2.	Používateľ stlačí tlačidlo pre smer doľava na virtuálnom ovládači	Postavička sa posunie o jedno políčko doľava, postavička bude stále v strede obrazovky	Postavička sa posunula o jedno políčko doľava animovaným pohybom, postavička je stále v strede obrazovky
3.	Používateľ stlačí tlačidlo pre smer dole na virtuálnom ovládači	Postavička sa posunie o jedno políčko dole, postavička bude stále v strede obrazovky	Postavička sa posunula o jedno políčko dole animovaným pohybom, postavička je stále v strede obrazovky
4.	Používateľ stlačí tlačidlo pre smer hore na virtuálnom ovládači	Postavička sa posunie o jedno políčko hore, postavička bude stále v strede obrazovky	Postavička sa posunula o jedno políčko hore animovaným pohybom, postavička je stále v strede obrazovky

Tab. 30 Akceptačný test pre overenie funkčnosti virtuálneho ovládača

Názov	Výbuch bomby	**Prípad použitia**	**UC Bomb Explode**
Rozhranie	Windows Phone 7	**ID testu**	**TEST - BE - 01**
Účel	Overiť oneskorenie výbuchu bomby a trvanie výbuchu		
Vstupné podmienky	Spustená hra na rozhraní Windows Phone 7		
Výstupné podmienky	Výbuch bomby a zničenie okolitých objektov		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ použije tlačidlo pre umiestnenie bomby v hre.	Na mape sa objaví bomba v aktuálnej pozícii postavičky.	Na mape sa objavila bomba v aktuálnej pozícii postavičky.
2.	Používateľ použije virtuálny ovládač pre posun postavy mimo dosah bomby.	Postavička sa posunie podľa pokynov používateľa.	Postavička sa posunula podľa pokynov používateľa.
3.a	Používateľ počká dobu 3 sekundy od času položenia bomby.	Bomba vybuchne a na mape sa v celom dosahu bomby objavia miesta explózie.	Na mape sa objavila explózia v celom rozsahu dosahu bomby.
3.b		Objekty nachádzajúce sa v dosahu bomby zmiznú v čase výbuchu.	Objekty v dosahu bomby boli zničené.
4.	Používateľ počká dobu 2 sekundy od času výbuchu bomby.	Explózia zmizla.	Zobrazenie explózie je animované, po uplynutí doby pôsobenia zmizla.

Tab. 31 Akceptačný test pre overenie funkcionality bomby a animácie výbuchu

Názov	Pôsobenie explózie na hráča	**Prípad použitia**	UC Bomb impact
Rozhranie	Windows Phone 7	**ID testu**	TEST - BI -01
Účel	Overiť pôsobenie explózie na postavičku používateľa		
Vstupné podmienky	Spustená hra, postavička s viac ako 1 životmi		
Výstupné podmienky	Pokračovanie hry, postavička na pôvodnom mieste		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ sa presunie v mape na miesto rôzne od pôvodného.	Postavička sa posunie podľa pokynov používateľa.	Postavička sa posunula podľa pokynov používateľa.
2.	Používateľ použije tlačidlo pre umiestnenie bomby v hre.	Na mape sa objaví bomba v aktuálnej pozícii postavičky.	Na mape sa objavila bomba v aktuálnej pozícii postavičky.
3.a	Používateľ počká dobu 3 sekundy od času položenia bomby.	Bomba vybuchne a na mape sa v celom dosahu bomby objavia miesta explózie.	Na mape sa objavila explózia v celom rozsahu dosahu bomby.
3.b		Postavička sa objaví na pôvodnom mieste, kamera je vycentrovaná na postavičku.	Kamera a postavička sa presunula na pôvodné miesto v čase výbuchu, explózia je stále viditeľná, nachádza sa v mieste výbuchu korektne.

Tab. 32 Akceptačný test pre overenie správnej funkcionality pre výbuch bomby a trvanie explózie

11 Dokončenie A* algoritmu

11.1.1 Analýza

Doterajšie riešenie využívajúce analyzátor mapy *MapAnalyzer.java* v kombinácii s *A** algoritmom *NextEngine.java* sa ukázalo ako nedostatočné pre mnohé bežné situácie. Napríklad často používaná funkcia na nájdenie najbližšieho bezpečného políčka používala manhattanovksú vzdialenosť a tak nájdené políčko vôbec nemuselo byť najbližšie prípadne nemuselo byť dostupné. Navyše sa vykonávalo množstvo operácií (prehľadávaní mapy), ktoré zdržovali výpočet. Toto bola motivácia k vytvoreniu nového analyzátora mapy, kombinujúceho vlastnosti oboch spomínaných tried.

11.1.2 Návrh

Myšlienka spočíva v dvoch kľúčových vylepšeniach – použiť hľadanie do šírky a vytvoriť si niekoľko pomocných máp (aktuálne 3). Hľadaním do šírky objavíme všetky políčka, na ktoré sa hráč dokáže dostať. Ku každému takémuto stavu si pamätáme aj postupnosť krokov, ktoré do neho viedli. Týchto stavov je konečné množstvo a na malej mape relatívne malé (okolo 50), algoritmus teda nie je výpočtovo náročný. Takýmto spôsobom vieme nájsť cestu do každého políčka na mape (pokiaľ existuje). Aby sme však vedeli, napríklad ktoré políčka sú bezpečné, prípadne z ktorých vieme ohroziť súpera, box a podobne, je výhodné urobiť si mapu políčok pre každú z požadovaných funkcií. Následne fungovanie metódy, napríklad pre nájdenie najbližšieho bezpečného políčka, bude spočívať v postupnom prechádzaní stavov vytvorených pri hľadaní do šírky a testovaní daného políčka, či je bezpečné. Keďže stavy sú zoradené v poradí, v akom boli vytvorené, takmer vždy stačí prejsť iba prvých pár stavov a riešenie je nájdené. Takto po jednom analyzovaní mapy (vytvorení stavov a máp) je možné získať množstvo informácií s malým počtom operácií.

11.1.3 Implementácia

Celá funkcionalita je implementovaná v triede *AstarAnalyzer.java*. Na uchovanie jednotlivých stavov je použitá vnorená trieda State, ktorá obsahuje súradnice hráča, cestu do daného políčka a kľúč (univerzálny identifikátor daného stavu), ktorý sa počíta z aktuálnych súradníc vzorcom:

$$key = x \cdot 100 + y$$

Fungovanie celého objektu je nasledovné. V konštruktore sa prejde zoznam komponentov na mape a vytvoria sa pomocné mapy, konkrétne:

- mapa nebezpečných políčok
- mapa políčok, z ktorých viem ohroziť súpera
- mapa políčok, z ktorých viem ohroziť box

Následne sa spustí hľadanie do šírky pričom sa navštívia všetky prístupné políčka a vytvorí sa zodpovedajúci počet stavov. Týmto končí inicializácia objektu a ten je možné následne využívať použitím nasledujúcich metód:

- *getClosestOponentsThreatPlace()*
- *getClosestBoxThreatPlace()*
- *getClosestSavePlace()*
- *amIThreatenByBomb()*
- *isThisPlaceSave (int, int)*
- *canHitOponent()*
- *canHitBox()*
- *getMyPosition()*

11.1.4 Testovanie

Názov	AstarAnalyzer	**Prípad použitia**	*UC Astar*
Rozhranie	Obrazovka bojiska	**ID testu**	TEST - AS - 01
Účel	Overenie fungovania AstarAnalyzátora. Nakoľko je s touto triedou silne previazaný EvilBot, jej správne fungovanie bude overené cez neho.		
Vstupné podmienky	Na mape sa nachádza EvilBot s úrovňou high alebo BoxDestroyer a niekoľko boxov prípadne ďalší hráč.		
Výstupné podmienky			

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Žiadna.	EvilBot sa posúva po mape, dáva bombu ku boxom alebo k iným hráčom.	EvilBot postupne ničil boxy a následne aj hráča.

Tab. 33 Akceptačný test pre príbeh Dokončenie A* algoritmu

11.2 Správa polohy používateľa

11.2.1 Analýza

Používateľ má možnosť na sledovanie polohy ostatných používateľov v záujme osobného stretávania a hrania *multiplayer* hier.

11.2.2 Návrh

Pokiaľ si používateľ zvolil možnosť zobrazenia jeho polohy, na serveri sa v pravidelných časových intervaloch aktualizuje údaj reprezentujúci polohu (atribút *location* v tabuľke *User*). Nakoľko sa tento údaj posiela a zapisuje do databázy častejšie ako ostatné, táto akcia sa nevykonáva prostredníctvom metódy na aktualizáciu celého záznamu používateľa – navrhli sme metódu, ktorá spravuje iba polohu, nezaťažuje systém prenosom a zápisom zbytočne veľkého množstva dát.

Pokiaľ používateľ chce sledovať polohu ostatných používateľov, lokálne zariadenie pošle žiadosť na server o poskytnutie zoznamu všetkých používateľov, následne sa na lokálnom zariadení zobrazí mapa s označenými miestami ostatných používateľov v bližšom okolí.

11.2.3 Implementácia

Pre aktualizáciu polohy používateľa sa volá zo *ServerAPI* metóda *setUserLocation()*. V parametre sú odovzdané súradnice určené pomocou GPS zariadenia. *Z Java Servlet* sa zavolá príslušná metóda na zápis polohy. Zoznam všetkých používateľov vracia metóda *getUsers().*

11.2.4 Testovanie

Testovanie prebehne v rámci testovania *Geoplatformy*.

11.3 Geoplatforma

11.3.1 Analýza

V rámci dodržania jedného z hlavného cieľa hry a teda socializácie bolo potrebné navrhnúť *framework*, ktorý by umožnil hráčom jednoduché stretávanie. V *android* zariadeniach je možné využívať GPS lokalizáciu ako aj podporu Google máp.

11.3.2 Návrh

Pomocou spomenutých prostriedkov je teda možné vytvoriť jednoduchú *Geoplatformu*, na ktorej by hráči mohli pozorovať polohu iných hráčov. Taktiež by bolo vhodné rozdeľovať hráčov na tých, ktorí sú dostupný pre *Multiplayer* hru a tých ktorí nie. Väčšina zariadení podporuje Android Google API, čo nám umožňuje importovať klasickú Google Mapu do aplikácie a vykonávať nad ňou rôzne operácie.

11.3.3 Implementácia

Geoplatforma je implementovaná ako potomok triedy *MapActivity*, ktorej jedinou úlohou je zobrazovanie mapy. Pri spúšťaní obrazovky trieda *MyLocation* spúšťa overovanie polohy zariadenia, ktoré prebehne v dvoch krokoch. V prípade, že je na zariadení zapnuté GPS, získa sa poloha priamo zo satelitu. V opačnom prípade sú informácie získané z *WiFi* siete alebo poskytovateľa mobilného internetu. Následne sa poloha používateľa uloží na server a vykreslí sa na mapu. Po určení polohy sa spustí vyhľadávanie používateľov, ktorí tento už vykonali a majú zapnutú vlastnosť *isDiscoverable.* Všetci títo používatelia sa vykreslia na mapu. Hráči sú na mape rozlíšení farebne a to nasledovne:

- červený hráč - pozícia prihláseného používateľa
- zelený hráč - pozícia hráča používateľa, ktorého je možné vyzvať na súboj cez *WiFi*
- čierny hráč - pozícia používateľa, ktorý nie je dostupný na *Multiplayer* súboj

Obr. 24 *Geoplatforma*

11.3.3.1 Testovanie

Názov	*Geoplatforma*		**Prípad použitia**	*UC Geoplatform*
Rozhranie	Obrazovka *Geoplatformy*		**ID testu**	TEST-GP-01
Účel	Správne zobrazenie hráčov na mape			
Vstupné podmienky	Prihlásený používateľ			
Výstupné podmienky	*Nahraté dáta sú rovnaké aké boli naposledy pri ich uložení*			
Vyhotovil	Ľuboš Gelányi		**Dátum**	18.3.2012
Krok	**Akcia používateľa**	**Očakávaná reakcia systému**		**Skutočná reakcia systému**
1.	Používateľ spustí mapu	Používateľ vidí seba a ostatných hráčov na správnych pozíciách		Používateľ vidí seba a ostatných hráčov na správnych pozíciách

Tab. 34 Akceptačný test pre príbeh *Geoplatforma*

11.4 Uloženie neurónového mozgu

Uloženie slúži na uchovávanie natrénovaného mozgu *avatara*. V prípade hrania s *avatarom* sa tento mozog načíta zo súboru. Pri tréningu sa načíta mozog zo súboru, pretrénuje sa a uloží sa pretrénovaná verzia.

11.4.1 Analýza

Ukladanie mozgu je potrebné pre zaznamenávanie progresu jednotlivých *avatarov*. Je potrebné vytvoriť jednoduchú, ale účinnú formu ukladania mozgu. Najvhodnejším a najjednoduchším spôsobom by bolo ukladanie prahov všetkých neurónov (okrem vstupných neurónov) a ukladanie jednotlivých váh medzi pospájanými neurónmi.

Načítavanie mozgu bude prebiehať analogickým spôsobom, akurát sa nebudú dáta ukladať, ale nahrávať.

11.4.2 Návrh

Ukladanie neurónového mozgu do súboru bude prebiehať tak, že sa budú ukladať prahy všetkých neurónov (okrem neurónov, ktoré budú na vstupnej vrstve) a jednotlivé váhy, ktoré spájajú neuróny. Tieto čísla sa transformujú na jednoduchý textový reťazec, ktorý bude ukladaný do súboru.

Samotné načítavanie bude prebiehať obdobným spôsobom. Načíta sa súbor vo forme textového reťazca, ten sa spracuje a premení na čísla. Po vytvorení prázdnej neurónovej siete sa z týchto spracovaných čísel doplnia váhy medzi neurónmi a ich prahy.

11.4.3 Implementácia

Proces ukladania neurónového mozgu sa nachádza v triede *BackwardsNeuralNetwork.java*. V prípade, že sa vytvorí nový *avatar*, ktorý bude mať mozog založený na neurónovej sieti, vytvorí sa nový mozog a uloží sa do nového súboru. Tento mozog je inicializovaný náhodne a aby dosahoval výsledky, je potrebné ho natrénovať.

Ak sa jedná o načítanie *avatara*, ktorý už bol vytvorený v minulosti, tak sa načíta jeho mozog zo súboru.

11.4.4 Testovanie

Názov	Uloženie mozgu	**Prípad použitia**	*UC SaveBrain*
Rozhranie	Obrazovka na vytvorenie nového *avatara*	**ID testu**	TEST-GLO-01
Účel	Overiť ukladanie nového neurónového mozgu		
Vstupné podmienky	Zvolený *avatar* s neurónovým mozgom		
Výstupné podmienky	Vytvorený neurónový mozog (súbor, kde je uložený)		
Vyhotovil	Ľuboš Masný	**Dátum**	30.3.2012

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Kliknutie na vytvorenie nového *avatara*	Zobrazí sa ponuka možných *avatarov* na vytvorenie	Obrazovka na vytvorenie *avatara* je zobrazená
2.	Zvolenie typu mozgu neurónová sieť	*Avatar* je úspešne vytvorený	*Avatar* je úspešne vytvorený a je pridaný do stajne *avatarov* používateľa

Tab. 35 Akceptačný test pre príbeh Uloženie mozgu

12 Šprint č.8

Číslo šprintu	**08**
Začiatok šprintu	19.3.2012
Koniec šprintu	26.3.2012
Príbehy (*userstories*)	• **Nová Neurónová sieť** • **Rebríček používateľov** • **Manažment *Multiplayer* režimu** • ***MapLoader* pre Windows Phone** • ***Dummy Bot* pre Windows Phone**

12.1 Nová neurónová sieť

12.1.1 Analýza

Máme implementovaných už dvoch *avatarov* pričom jeden z nich funguje s použitím neurónovej siete. Dôvodom prečo vytvoriť ďalšieho, je použitie úplne iného spôsobu analyzovania mapy, akcií a spôsobu učenia. Ako vstup sa použije okolie hráča, výstupom bude len jedna akcia a učenie bude zabezpečené pomocou genetického algoritmu. Navyše neurónová sieť bude obsahovať aj okno do minulosti čo jej umožní vytvárať stratégie a pri rovnakom vstupe reagovať rôzne v závislosti od predchádzajúcich akcií.

12.1.2 Návrh

Neurónová sieť je navrhnutá ako dopredná neurónová sieť avšak je doplnená o okno do minulosti. Veľkosť tohto okna nie je pevne daná, rovnako ako váhovanie vstupov neurónov ju nastavuje genetický algoritmus. Myšlienkou je ukladať počas tréningu avatara akcie a stav mapy v okolí hráča a následne po konci hry vytvoriť genetickým algoritmom takú neurónovú sieť, ktorá pri vstupoch zodpovedajúcich uloženým stavom, dá na výstupe rovnaké akcie ako vykonal hráč. Takýmto spôsobom je možné natrénovať avatara jedinou hrou.

12.1.2.1 Fungovanie genetického algoritmu

Vysvetlenie základných pojmov:

- jedinec – je súčasťou populácie, reprezentuje jedno riešenie, v našom prípade jednu neurónovú sieť
- populácia – je zložená z niekoľkých jedincov, má stále rovnakú veľkosť, v každom cykle sa vytvorí nová
- kríženie – vytváranie nového jedinca z dvoch „rodičovských“ jedincov
- mutácia – náhodná zmena hodnôt jedinca, uplatňuje sa pri krížení
- fitnes – ohodnotenie jedinca, väčšia fitnes znamená lepší jedinec

12.1.2.1.1 Vytváranie novej populácie

Nová populácia vzniká zo starej v troch krokoch:

- nahradením niekoľkých najhorších jedincov nanovo vygenerovanými (nová krv)
- skopírovaním niekoľkých najlepších jedincov (elitárstvo)
- krížením jedincov

12.1.2.1.2 Kríženie a mutácia

Kríženia sa zúčastňujú dvaja jedinci, teda dve neurónové siete. Ich výber z populácie zabezpečuje spravodlivá ruleta. V našom prípade pri krížení vzniká nová neurónová sieť. Táto obsahuje rovnaký počet neurónov ako jej „rodičia“ pričom zdedí v každej vrstve náhodne niektoré neuróny od jedného a niektoré od druhého rodiča. Zdedenie rodičovského neurónu u nás znamená skopírovanie váh vstupov a okna do minulosti. Keby sa krížil jedinec sám so sebou, vznikol by rovnaký jedinec.
S krížením je spojená aj mutácia, náhodná zmena hodnôt. Môže sa prejaviť s pravdepodobnosťou 4% pri každej kopírovanej hodnote.

12.1.2.1.3 Ruleta

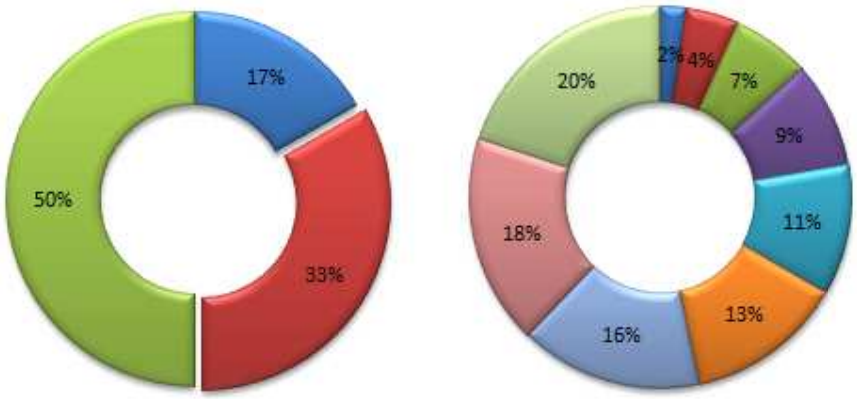
Úlohou rulety je náhodne vybrať dvoch jedincov z populácie, ktorý sa budú krížiť. Samotný výber je náhodný len do určitej miery pretože lepšie jedince (s väčšou fitnes) musia dostať väčšiu príležitosť na kríženie. Fungovanie rulety je nasledovné.

Najskôr sa jedinci v populácii usporiadajú podľa svojej fitnes od najhorších po najlepšie. Povedzme, že k nim môžeme pristupovať podľa ich indexov kde 0 je index najhoršieho a *n* je index najlepšieho (pričom *n + 1* je počet všetkých jedincov v populácii). Podstatou spravodlivej rulety je, aby každý jedinec dostal taký výsek „koláča“, aký je jeho *index + 1*. Koláč sa teda rozdelí na toľko častí, koľko je súčet o jeden zvýšených indexov jedincov a následne sa prvému pridelí jeden kúsok, druhému dva kúsky, tretiemu tri, atď. Následne sa vygeneruje náhodné číslo *x* v rozsahu 0 až *počet častí koláča* a po dosadení tohto čísla do nasledujúceho vzorca, dostaneme index vybraného jedinca.

$$J = \left\lfloor \frac{-1+\sqrt{1+8x}}{2} \right\rfloor$$

Vzorec vlastne hovorí koľko čísiel musíme postupne sčítať, aby sme sa dostali po náhodne vygenerované číslo x. Napríklad pre náhodné číslo 6 dostaneme výsledok 3 čo znamená že musíme sčítať 1 + 2 + 3 aby sme sa dostali po číslo 6.

Takýmto spôsobom bude pravdepodobnosť vybratia každého jedinca rovnomerne rozdelená s prihliadnutím na jeho fitnes. Graficky je to zobrazené na dvoch príkladoch na *Obr. 25* (v ľavo pre 3 jedincov, v pravo pre 9 jedincov).



Obr. 25 Rozdelenie pravdepodobnosti vybratia každého z troch (v ľavo) či deviatich (v pravo) jedincov ruletou

12.1.3 Implementácia

Na vytvorenie neurónovej siete som nepoužil žiadny existujúci *framework* či knižnicu, nakoľko som nenašiel žiadnu, ktorá by fungovala na platforme *Android*. Vytvoril som si teda vlastnú neurónovú sieť.

Celá funkcionalita sa nachádza v triedach *GeneticNeuralNteworkBasedAvatar.java, GeneticNeuralNetwork.java, Layer.java a Neuron.java*.

Neurónová sieť obsahuje jednu vstupnú, tri skryté a jednu výstupnú vrstvu. Všetky neuróny sú medzi vrstvami prepojené spôsobom „každý s každým“. Výstupná vrstva je tvorená iba jedným neurónom, ktorého hodnota je transformovaná na výslednú akciu podľa nasledujúcej tabuľky:

Výstupná hodnota neurónu	Zodpovedajúca akcia
-1,00 až -0,66	polož bombu
-0,66 až -0,33	choď hore
-0,33 až 0,00	choď dole
0,00 až 0,33	choď doprava
0,33 až 0,66	choď doľava
0,66 až 1,00	nevykonaj žiadnu akciu

Vstupná vrstva neurónovej siete obsahuje spolu 24 neurónov, nakoľko vstupom je časť mapy okolo hráča – rozsah 5x5 políčok. Situácia na mape je transformovaná na rozsah -1 až 1 podľa nasledujúcej tabuľky:

Obsah políčka	**Vstupná hodnota**
bomba	-1,00 až -0,66
hráč	-0,66 až -0,33
box	-0,33 až 0,00
stena	0,00 až 0,33
prázdne políčko	0,33 až 0,66
políčko mimo mapy	0,66 až 1,00

Výstup každého neurónu je počítaný spojitou funkciou (*sigmoida*) sumy váhovaných vstupov.

Vzorec pre funkciu sigmoida: $y = \frac{1}{1+e^{-x}}$

Čo sa týka genetického algoritmu, jeho parametre sú nastavené nasledovne. Populácia má veľkosť 10 jedincov, z toho 1 je náhodne vytvorený, traja najlepší sú skopírovaný z predošlej populácie a zvyšných 6 je vytvorených krížením. Nové populácie sa vytvárajú dovtedy, kým sa nevytvorí požadovaná neurónová sieť.

12.1.4 Testovanie

Názov	Učenie sa od hráča	**Prípad použitia**	*UC Neural Network*
Rozhranie	Obrazovka bojiska	**ID testu**	TEST - NN - 01
Účel	Overenie fungovania učenia sa od hráča.		
Vstupné podmienky	Na mape sa nachádza *avatar* reprezentujúci neurónovú sieť a hráč riadený používateľom.		
Výstupné podmienky			

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ so svojim hráčom príde k boxu, položí k nemu bombu a vybuchne spolu s ňou.	Hra skončí, prebehne učenie neurónovej siete genetickým algoritmom a nová neurónová sieť sa zapíše do súboru a odošle na server.	Hra skončila, niečo sa stalo.
2.	Používateľ opäť spustí hru a nevykonáva ďalej žiadnu akciu.	*Avatar* príde k boxu, položí bombu a vybuchne spolu s ňou.	*Avatar* prišiel k boxu, položil bombu a vybuchol.

Tab. 36 Akceptačný test pre príbeh Nová neurónová sieť

12.2 Rebríček používateľov

12.2.1 Analýza

Používateľ má možnosť na prezeranie rebríčku všetkých používateľov, t.j. ich poradia podľa celkového počtu získaných bodov.

12.2.2 Návrh

Návratová hodnota od servera na požiadavku klienta je zoznam inštancií typu *User* pre všetkých existujúcich používateľov.

12.2.3 Implementácia

Z klienta sa volá funkcia *getUsers()* definovaná v *ServerAPI*. V implementácii metódy (*Java Servlet*) sa dopytuje na všetky záznamy tabuľky *User*.

Na zariadení je rebríček implementovaný v položke *Settings* pod záložkou *User Ranking*. Na tejto obrazovke sa pri každom spustení načíta zoznam všetkých používateľov, ktorí sa následne zoraďuje podľa počtu získaných bodov.

12.2.4 Testovanie

Názov	Zobrazenie rebríčka		**Prípad použitia**	*UC User Ranking*
Rozhranie	Obrazovka *User Ranking*		**ID testu**	TEST - UR - 01
Účel	Overenia načítania rebríčka			
Vstupné podmienky	Prihlásený používateľ			
Výstupné podmienky				
Krok	**Akcia používateľa**	**Očakávaná reakcia systému**		**Skutočná reakcia systému**
1.	Používateľ prejde do sekcie *User Ranking*	Zobrazený sú všetci používatelia zoradení podľa najvyššieho skóre		Zobrazený sú všetci používatelia zoradení podľa najvyššieho skóre

Tab. 37 Akceptačný test pre Rebríček používateľov

12.3 Manažment Multiplayer režimu

12.3.1 Analýza

Pre vytvorenie *multiplayer* hry je potrebné istým spôsobom ošetrovať identifikáciu klienta a servera. V pôvodnej implementácii z predošlých šprintov bolo možné pripojiť na server ľubovoľné zariadenie, ktoré bolo na rovnakej podsieti. Čo však v niektorých prípadoch nemusí byť ideálne. Obidvaja používatelia musia byť jasne identifikovaný napríklad aj kvôli skóre.

12.3.2 Návrh

Android platforma poskytuje API na pracovanie s *Wifi* pripojením. Preto je vhodné toto využiť na identifikovanie hráčov v jednej podsieti a následne vykonávať výzvy na súboj.

Po zvolení *Multiplayer* režimu sú používateľovi zobrazené 3 ponuky. *Create game, Join game, Geoplatform.* V *Create Game* používateľ prejde všetkými nastaveniami rovnako ako v iných herných režimoch a nakoniec prejde na obrazovku výberu oponenta. Pri možnosti *Join Game* je používateľ presmerovaný priamo na výber oponenta. Obrazovka *Geoplatform* bola rozobratá v predošlých šprintoch

12.3.3 Implementácia

Obrazovka na výber oponenta pre klient a server vyzerá rovnako, funkčne sa však zásadne líši. V obidvoch prípadoch však kliknutím na *WiFi* ikonu aplikácia zmizne a objaví sa obrazovka pre pripojenie na *WiFi* sieť. Rozdiel spočíva v zozname dostupných oponentov. Pre server sa zobrazujú všetci používatelia, ktorí sú na rovnakej podsieti. Toto sa overuje na základe *WiFi BSSID*, ktoré má každý používateľ ako svoj parameter. Na strane klienta sa v zozname zobrazujú len hráči, ktorí

používateľa vyzvali na súboj. Toto sa zisťuje pomocou IP adresy klienta, ktorú si server uloží ako parameter.

Tento prístup zabezpečuje spojenie len dohodnutých používateľov a teda nemôže sa stať, že používateľ nepozná svojho oponenta.

Obr. 26 Výber oponenta pre *Multiplayer*

12.3.4 Testovanie

Názov	Zobrazenie oponentov	**Prípad použitia**	*UC Multiplayer*
Rozhranie	Obrazovka *User Ranking*	**ID testu**	TEST - MG - 01
Účel	Overenia načítania dostupných hráčov		
Vstupné podmienky	Prihlásený používateľ zapnutá *WiFi*		
Výstupné podmienky			

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ prejde do sekcie *Choose Opponent*	Zobrazení sú všetci hráči na rovnakej podsieti	Zobrazení sú všetci hráči na rovnakej podsieti

Obr. 27 Akceptačný test pre príbeh Manažment *Multiplayer* režimu

12.4 MapLoader pre Windows Phone

Aby bola zabezpečená kompatibilita máp na WP a *Android*, bolo potrebné vytvoriť funkciu na načítavanie máp zo súboru.

12.4.1 Analýza

Načítavanie mapy pre WP platformu bude prebiehať rovnakým spôsobom ako pre načítavanie mapy pre *Android*. Jednotlivé mapy sú uložené v súboroch s koncovkou *„.lvl“*. V súbore sa nachádzajú čísla od 0 – 5.

Číslo *0* predstavuje na mape prázdne políčko. *1* je nerozbitná stena. *2* je zničiteľná škatuľa, ktorá sa dá posúvať. Číslo *3* predstavuje hráča, *4* je *Bot* a *5 avatar*.

12.4.2 Návrh

Načítavanie mapy prebieha načítaním a spracovaním súboru s koncovkou *„.lvl“*. Následne sa budú vytvárať jednotlivé komponenty podľa toho, aké číslo sa na políčku nachádza.

12.4.3 Implementácia

Načítanie mapy nastáva pri inicializácií celej hry. Najskôr sa otvorí a načíta súbor. Textová podoba sa rozdelí na jednotlivé znaky, ktoré reprezentujú čísla. Tieto znaky sa premenia na číselnú hodnotu, ktorá vyjadruje, aký komponent sa má na mape vytvoriť.

12.4.4 Testovanie

Názov	Načítanie mapy	**Prípad použitia**	*UC MapLoad*
Rozhranie	Herná obrazovka	**ID testu**	TEST-GLO-01
Účel	Overiť načítavanie mapy zo súboru		
Vstupné podmienky	Spustená hra		
Výstupné podmienky	Vytvorená mapa podľa súboru		
Vyhotovil	Ľuboš Masný	**Dátum**	**30.3.2012**

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Kliknutie na spustenie aplikácie	Spustená aplikácia s mapou vytvorenou podľa načítavaného súboru	Aplikácia bola spustená s vytvorenou mapou podľa súboru

Tab. 38 Akceptačný test pre príbeh Výbuch bomby

12.5 Dummy Bot pre Windows Phone

Dummy bot je jedným z možných *botov*. Slúži na odskúšanie aplikácie a taktiež poskytuje možnosť hrania. *Dummy bot* je jeden z najprimitívnejších *botov* v našej hre.

12.5.1 Analýza

Dummy bot predstavuje takého *bota*, ktorý zakaždým vykoná náhodnú akciu. Akcie, ktoré môže vykonávať, sú *následovné*:

- Pohyb hore
- Pohyb vľavo
- Pohyb vpravo
- Pohyb dole
- Žiadna akcia
- Položenie bomby

12.5.2 Návrh

Dummy bot bude vykonávať náhodné akcie s rovnakou pravdepodobnosťou pre každú akciu. Tieto akcie budú vybrané zakaždým, keď za zavolá metóda na vybratie ďalšieho kroku. *Dummy bot* nemá zadaný žiadny cieľ, len náhodne vykonáva akcie.

12.5.3 Implementácia

Aktualizovanie nasledujúcej akcie sa spraví po každom volaní funkcie *Update*(*GameTime*). Tieto akcie nie sú nijako zaznamenávané a zakaždým sa vykonajú s rovnakou pravdepodobnosťou.

12.5.4 Testovanie

Názov	*Dummy bot*	**Prípad použitia**	*UC DummyBot*
Rozhranie	Herná obrazovka (mapa)	**ID testu**	TEST-GLO-01
Účel	Overiť synchronizované pohyby hráča a *bota*		
Vstupné podmienky	Spustená hra		
Výstupné podmienky	Synchronizované a korektné pohyby hráča a *bota*		
Vyhotovil	Ľuboš Masný	**Dátum**	**5.4.2012**

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Kliknutie na spustenie aplikácie	Spustená aplikácia	Aplikácia bola spustená
2.	Pohybovanie hráčom pomocou *controlera*	Pohyby hráča a *bota* sú synchronizované a korektné	Pohyby obidvoch, hráča aj *bota*, boli synchronizované. Taktiež nenastala situácia, kedy by nastal konflikt pohybov.

Tab. 39 Akceptačný test pre príbeh Výbuch bomby

13 Šprint č.9

Číslo šprintu	**09**
Začiatok šprintu	26.3.2012
Koniec šprintu	11.4.2012
Príbehy (*userstories*)	• **Informačný panel hry vo Windows Phone** • **Správa Avatarov** • **Herné Štatistiky** • **Online Tournament Avatarov** • **Neurónová sieť pre Windows Phone**

13.1 Informačný panel hry na WP

13.1.1 Analýza

Prebehla v predošlých šprintoch

13.1.2 Návrh

Prebehol v predošlých šprintoch

13.1.3 Implementácia

Počas hrania hry je pre používateľa potrebné vedieť o aktuálnom stave hry. Ako hráč sa zaujíma o stav hry z troch základných aspektov:

- Vlastný stav životov
- Stav životov súpera/súperov
- Trvanie hry

Pre zobrazenie týchto hodnôt sa vybrali uvedené umiestnenia na obrazovke:

- Vlastný stav životov - vždy v ľavom hornom rohu obrazovky
- Stav životov súpera/súperov - v ostatných rohoch obrazovky podľa počtu súperov
- Trvanie hry - v hornej časti obrazovky v strede (medzi zobrazením stavu životov hráča a súpera) vo formáte „HH:MM“

Táto forma zobrazenia bola spočiatku implementovaná len na *Android* verzii aplikácie. Kvôli použiteľnosti a potrebám používateľa bola preto implementovaná aj na platforme WP.

13.1.3.1 WP rovnaké grafické prevedenie

Keďže sa jedná o jednu hru, bolo potrebné zabezpečiť rovnaký výzor hry aj na druhej platforme. Všetky obrázky a obrazovky používané na *androide* bolo potrebné použiť aj na WP. Taktiež bolo potrebné spraviť rovnaké správanie a vykresľovanie aplikácie, rovnaké ovládacie tlačidlá, rovnaké ovládanie a rovnaké vyhodnocovanie hry.

13.1.4 Testovanie

Názov	Znižovanie životov hráča	**Prípad použitia**	UC Game WP
Rozhranie	Windows Phone 7	**ID testu**	TEST - GWP - 01
Účel	Overiť kalkuláciu životov postavičky používateľa.		
Vstupné podmienky	Spustená hra, postavička s viac ako 1 životom		
Výstupné podmienky	Zobrazená obrazovka „Game over“		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ použije tlačidlo pre umiestnenie bomby v hre.	Na mape sa objaví bomba v aktuálnej pozícii postavičky.	Na mape sa objavila bomba v aktuálnej pozícii postavičky.
2.a	Používateľ počká dobu 3 sekundy od času položenia bomby.	Bomba vybuchne a na mape sa v celom dosahu bomby objavia miesta explózie.	Na mape sa objavila explózia v celom rozsahu dosahu bomby.
2.b		Stav počtu životov postavičky sa zníži o jeden.	Počet životov sa znížil o jeden
3.	Používateľ zopakuje testovanie od bodu 1 až do dosiahnutia 0 životov.	Zobrazí sa „Game over“ obrazovka.	Zobrazila sa „Game over“ obrazovka, nebol zobrazený virtuálny ovládač (korektné).

Tab. 40 Akceptačný test pre overenie kalkulácie životov postavičky

Názov	Výhra používateľa v hre	**Prípad použitia**	UC GAME WP
Rozhranie	Windows Phone 7	**ID testu**	TEST - GWP - 02
Účel	Overiť kalkuláciu životov protihráčov		
Vstupné podmienky	Spustená hra, počet protihráčov 1 a viac, bez inteligencie, životov 1		
Výstupné podmienky	Zobrazená obrazovka „You win“		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ sa presunie v mape na miesto vedľa protihráča.	Postavička sa posunie podľa pokynov používateľa.	Postavička sa posunula podľa pokynov používateľa.
2.	Používateľ použije tlačidlo pre umiestnenie bomby v hre.	Na mape sa objaví bomba v aktuálnej pozícii postavičky.	Na mape sa objavila bomba v aktuálnej pozícii postavičky.
3.	Používateľ použije virtuálny ovládač pre posun postavy mimo dosah bomby.	Postavička sa posunie podľa pokynov používateľa.	Postavička sa posunula podľa pokynov používateľa.
	Používateľ počká dobu 3 sekundy od času položenia bomby.	Bomba vybuchne a zničí protihráčov v dosahu.	Postavička vedľa bomby zmizla, neobjavila sa na pôvod. mieste, jej životy sú nastavené na stav 0.
	Používateľ zopakuje testovanie od bodu 1., pokiaľ je na mape ešte ďalší protihráč.	Zobrazí sa obrazovka „You win“.	Zobrazila sa obrazovka „You win“.

Tab. 41 Akceptačný test pre overenie kalkulácie postavičiek súperov

Názov	Pauzovanie hry	**Prípad použitia**	UC GAME WP
Rozhranie	Windows Phone 7	**ID testu**	TEST - GWP - 03
Účel	Overiť pozastavenie hry a vplyv pauzy na jej beh		
Vstupné podmienky	Spustená hra		
Výstupné podmienky	Hra je pozastavená, čas hrania meška od skutočného času behu aplikácie		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.a	Používateľ počas behu hry stlačí tlačidlo „Pauza“ v dolnej časti obrazovky.	Zobrazí sa obrazovka pauza s nadpisom „Game paused“ a s menu obsahujúcim tlačidlá „Resume“ a „Exit“.	Zobrazilo sa menu s korektným nadpisom a s položkami v menu.
1.b		Čas hrania sa zastavil, animácie hry sa zastavili a nevytvárajú sa ďalšie akcie postavičiek a objektov.	Čas postál, animácie sa zastavili, celá hra zastala. Bomba nevybuchla, explózia je stále zobrazená.

Tab. 42 Akceptačný test pre overenie funkčnosti pozastavenia behu hry

Názov	Obnovenie hry z pauzy	**Prípad použitia**	UC GAME WP
Rozhranie	Windows Phone 7	**ID testu**	TEST - GWP - 04
Účel	Overiť znovuobnovenie behov hry po vrátení sa do hry z menu pauza		
Vstupné podmienky	Hra spustená a následne pozastavená pauzou		
Výstupné podmienky	Hra pokračuje, čas hrania meška od skutočného času behu aplikácie		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.a	Používateľ počas behu hry stlačí tlačidlo „Pauza“ v dolnej časti obrazovky.	Zobrazí sa obrazovka pauza s nadpisom „Game paused“ a s menu obsahujúcim tlačidlá „Resume“ a „Exit“.	Zobrazilo sa menu s korektným nadpisom a s položkami v menu.
1.b		Čas hrania sa zastavil, animácie hry sa zastavili a nevytvárajú sa ďalšie akcie postavičiek a objektov.	Čas postál, animácie sa zastavili, celá hra zastala. Bomba nevybuchla, explózia je stále zobrazená.
2.	Používateľ stlačí tlačidlo „Resume“ v menu pauzy	Obrazovka pauzy zmizne, dokončia sa animácie cyklu hry a pokračuje sa ďalšou interakciou objektov.	Hra sa obnovila v bode, v ktorom bola pozastavená. čas adekvátne meškal od skutočného času behu aplikácie, bola umožnené ďalej hrať.

Tab. 43 Akceptačný test pre overenie správneho obnovenia hry z pauzy

Názov	Vypnutie hry	**Prípad použitia**	UC GAME WP
Rozhranie	Windows Phone 7	**ID testu**	TEST - GWP - 05
Účel	Overiť reakciu hry na vypnutie		
Vstupné podmienky	Hra spustená a následne pozastavená, zobrazené menu pauzy		
Výstupné podmienky	**Hra je vypnutá**		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.a	Používateľ počas behu hry stlačí tlačidlo „Pauza“ v dolnej časti obrazovky.	Zobrazí sa obrazovka pauza s nadpisom „Game paused“ a s menu obsahujúcim tlačidlá „Resume“ a „Exit“.	Zobrazilo sa menu s korektným nadpisom a s položkami v menu.
1.b		Čas hrania sa zastavil, animácie hry sa zastavili a nevytvárajú sa ďalšie akcie postavičiek a objektov.	Čas postál, animácie sa zastavili, celá hra zastala. Bomba nevybuchla, explózia je stále zobrazená.
2.	Používateľ stlačí tlačidlo „Exit“ v menu pauzy.	Hra je ukončená a zariadenie zobrazí menu v OS.	Hra sa vypla.

Tab. 44 Akceptačný test pre overenie funkcionality menu pauzy

Načítanie mapy

Názov	Spustenie hry	**Prípad použitia**	UC GAME WP
Rozhranie	Windows Phone 7	**ID testu**	TEST - GWP - 06
Účel	Zistiť načítanie správnej mapy		
Vstupné podmienky	Spustená hra na rozhraní Windows Phone 7		
Výstupné podmienky	Spustí sa hra so správne načítanou mapou		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ si vyberie mapu podľa názvu a stlačí "Play"	Hra sa spustí so zvolenou mapou a s troma ďalšími hráčmi	Hra sa spustí so zvolenou mapou a s troma ďalšími hráčmi

Tab. 45 Akceptačný test pre načítanie mapy zo súboru na platforme WP

13.2 Správa avatarov

13.2.1 Analýza

Používateľ môže vytvárať, editovať a mazať vlastných *avatarov*, zároveň je systému umožnená aktualizácia údajov *avatarov*, ktoré sa menia počas *online súbojov* a mnohých ďalších činností.

Táto funkcionalita zahŕňa:

- Vytvorenie a zmazanie *avatara*
- Poslanie *avatara* na *online súboj*
- Aktualizovanie mozgu *avatara* po trénovaní
- Nastavenie *avatara* ako preddefinovaného

13.2.2 Návrh

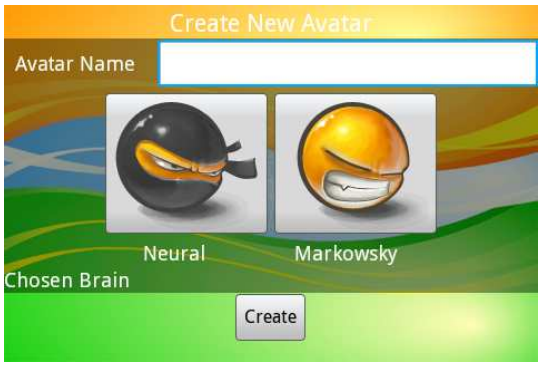
Pri požiadavke klienta o poskytnutie údajov o *avataroch* sa zo servera pošle zoznam inštancií všetkých *avatarov*, ktoré patria k používateľovi prihlásenému na žiadajúcom mobilnom zariadení. Pri zápise sa pošle na server jedna inštancia *avatara* so zmenenými údajmi, ktorým sa prepíše záznam v centrálnej databáze.

13.2.3 Implementácia

Vyššie uvedené funkcionality sú v *ServerAPI* definované a v *Java Servlet* realizované v metódach:

- *addAvatar()* – vytvorenie nového *avatara*
- *removeAvatar()* – zmazanie *avatara*
- *setAvatar()* – aktualizácia údajov *avatara*
- *getAvatarList()* – získavanie zoznamu všetkých *avatarov*, zároveň *avatarov* žiadajúceho používateľa

Na zariadení je správa *avatarov* implementovaná ako stajňa *avatarov* - *Avatar Stable* kde je možné vytvárať nových *avatarov* a výber svojho predvoleného *avatara*. Na obrazovke *Online Games* sú zobrazení *avatri*, ktorí sú aktuálne v *online tournament*-e. Kliknutím na *avatara* je možne ho stiahnuť späť a kliknutím na *add avatar to tournament* pridať do turnaja. Ak sa medzi týmito *avatarmi* nachádza aj default *avatar* nie je možné ho použiť v tréningu. V tomto prípade je tréning zablokovaný až kým používateľ nestiahne *avatara* z turnaja alebo si nezvolí nového predvoleného *avatara*. Tieto úlohy je možné v tomto prípade vykonať priamo na obrazovke nastavenia tréningu.



Obr. 28 Vytvorenie nového *avatara*

Obr. 29 Stajňa *avatarov*

13.2.4 Testovanie

Názov	Vytvorenie avatara		**Prípad použitia**	UC AVatar Management
Rozhranie	*Avatar Stable*		**ID testu**	TEST - AM - 01
Účel	Overenie správneho vytvorenia *avatara*			
Vstupné podmienky	Prihlásený používateľ			
Výstupné podmienky				
Krok	**Akcia používateľa**	**Očakávaná reakcia systému**		**Skutočná reakcia systému**
1.	Používateľ klikne na *Create New Avatar* - následné zvolí meno a mozog *avatara*	V stajni *avatarov* pribude príslušný *avatar* so správnym menom a ikonou		V stajni *avatarov* pribude príslušný *avatar* so správnym menom a ikonou

13.3 Herné štatistiky

13.3.1 Analýza

Používateľ má možnosť na prezeranie jeho herných štatistík.

13.3.2 Návrh

Po ukončení hry akéhokoľvek typu systém pošle na server výsledky hry (účastníci, získané body a pod.), ktoré sa uložia do databázy. Pri žiadosti o poskytnutie herných štatistík sever spracuje uložené výsledky v databáze. Ako odpoveď na žiadosť klientovi pošle počet odohraných hier a počet výhier používateľa a zvlášť preddefinovaného *avatara* pre všetky typy hry.

13.3.3 Implementácia

Po ukončení danej hry sa na klientovi vytvorí inštancia typu Game, pridajú sa účastníci a ich získané body, následne sa zavolá metóda *addNewGame()* zo *ServerAPI*, ktorá v parametre odovzdá vytvorenú inštanciu. *Java Servlet* prijatú inštanciu uloží do databázy.

Pri žiadosti o poskytnutie herných štatistík sa na klientovi zavolá metóda *getGameStatistics()*, ktorá vracia pole celočíselných hodnôt špecifikovaných vyššie.

13.3.4 Testovanie

Testuje sa pri *online hrách*.

13.4 Online tournament avatarov

13.4.1 Analýza

Refaktoring - zmena základnej architektúry hry umožní používať existujúci kód hry ako *framework* pre vývoj oddelene od systému *Android*. Taktiež samotný *framework* je možný použiť oddelene od grafického rozhrania, prípadne od alternatívnych rozhraní ako konzolová verzia, Java *Applet* alebo *frame*.

Zabezpečenie týchto podmienok umožnili implementáciu programu, ktorý bude schopný simulovať hru bez používateľského vstupu – súboj *avatarov*. Inštancie takýchto programov môžu byť nasedené súčasne na viaceré servery alebo osobné počítače, kde bežia na pozadí a vykonávajú simuláciu hry – *škálovateľnosť*.

13.4.2 Návrh

Primárnym účelom programu je zabezpečenie tzv. *online súbojov*, kde môžu súťažiť natrénovaný *avatary*. Cieľom je vytvoriť program, ktorý spĺňa nasledujúce požiadavky:

- Vytvorí prostredie hry
  - Načíta mapu zo súboru
  - Načíta stav „mozgu“ zo súboru
- Riadenie hry
  - Pridanie hráčov do hry
  - Spustí a zastaví simuláciu
- Zabezpečuje komunikáciu s webovými službami:
  - prenos máp
  - prenos mozgov
  - prenos výsledkov hry

13.4.3 Implementácia

Samotný program sa skladá z jedného cyklu, ktorý po jeho skončení sa znova spustí.

V počiatočnej fáze program komunikuje s webovými službami (*GameLevelAPI*, *TournamentAPI*), a získané dáta sa uložia na disk. V prípade zlyhania komunikácie alebo nedostatku zdrojov / voľných avatarov daný cyklus sa zruší a naplánuje sa oneskorený štart ďalšieho.

Druhá fáza je samotné vykonanie súboja. Vytvorenie inštancii *GameEngine*, *ControllerAI* pomocou získaných súborov. Simulácia beží v reálnom čase – bez časovača, krok za krokom. Hra sa skončí, ak:

a) Bude dosiahnutý počet maximálnych kôl
b) Existuje len jeden (alebo menej) živý hráč na mape.

Po skončení hry program komunikuje s webovou službou – odoslanie štatistiky.

13.4.4 Testovanie

Funkcie programu sú implementovane priamo v knižnici (*framework*) *SBEngine* a sú pokryté s *JUnit* testami.

Názov	Súťaž *avatara* v *online turnaji*	**Prípad použitia**	*UC Online Games*
Rozhranie	Android, WebService	**ID testu**	TEST - OG - 01
Účel	Overiť funkčnosť online súbojov		
Vstupné podmienky	Prihlásenie do hry cez *Android*. Vytvorený a natrénovaný *avatar*		
Výstupné podmienky	Používateľ získa body, štatistické informácie o odohratých hrách		

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Používateľ stlačí tlačidlo pre poslanie *avatara* do *online turnaja*.	Na obrazovke sa zmení stav *avatara*	Na obrazovke sa zmení stav *avatara*
2.a	Používateľ si zvolí z menu funkciu pre zobrazenie štatistík	Na obrazovke sa zobrazia informácie o odohratých hrách	Žiadne zmeny – *avatar* sa ešte nezúčastnil *online* hry
2.b	Používateľ si zvolí z menu funkciu pre zobrazenie štatistík	Na obrazovke sa zobrazia informácie o odohratých hrách	Nové zmeny – *avatar* sa zúčastnil hry, získal body.
3	Používateľ si zvolí z menu funkciu pre zobrazenie detailov hry	Na obrazovke sa zobrazia informácie o turnaji	Funkcia nedostupná

13.5 Neurónová sieť pre Windows Phone

Tak ako na *androide*, tak bolo potrebné aj na WP vytvoriť mozog na základe neurónovej sieti.

13.5.1 Analýza

Vychádza sa z rovnakých potrieb a predpokladov ako pri *androide*. Sledujú sa tie isté znaky situácie a neurónová sieť sa vytvára rovnakým spôsobom.

13.5.2 Návrh

Architektúra návrhu je úplne identická ako pri neurónovom mozgu na *androide*.

13.5.3 Implementácia

Hlavná časť neurónového mozgu sa nachádza v triede *NeuralNetwork.cs*. Implementácia je taktiež úplne identická, ako pri androide. Jediný rozdiel je, že mozog sa nevytvára zo súboru, ale zo staticky vytvorenej predlohy.

13.5.4 Testovanie

Názov	WP neurónový mozog	**Prípad použitia**	*UC WPNeuralBrain*
Rozhranie	Herná obrazovka	**ID testu**	TEST-GLO-01
Účel	Overiť neurónového mozgu		
Vstupné podmienky	Spustená hra		
Výstupné podmienky	*Avatar* sa sám nezabíja a snaží sa zneškodniť súpera		
Vyhotovil	Ľuboš Masný	**Dátum**	8.4.2012

Krok	**Akcia používateľa**	**Očakávaná reakcia systému**	**Skutočná reakcia systému**
1.	Kliknutie na spustenie aplikácie	Spustená aplikácia s *avatarom*, ktorý sa snaží zvíťaziť tak, že zneškodní súpera	*Avatar* sa pohybuje k súperovi a ohrozuje ho svojimi bombami

Tab. 46 Akceptačný test pre príbeh Výbuch bomby

14 Opis Prototypu po letnom semestri

Na konci letného semestra sa nám podarilo splniť všetky požiadavky a ciele, ktoré sme si stanovili na začiatku zimného a spresnili na začiatku letného semestra. Podarilo sa nám vytvoriť hru, ktorú majú možnosť používatelia hrať sami, s priateľmi alebo s novými kamarátmi.

14.1 Motivácia

Našou najväčšou motiváciou bolo vytvoriť hru, ktorá by dokázala hráčom ponúknuť okrem zábavy ešte niečo navyše. Často krát sa stretávame s prípadom, kedy sú hráči odrezaný od reality a takmer vôbec sa nestretávajú s ľuďmi. Preto bolo naším cieľom vytvoriť takú hru, ktorá by okrem zábavy motivovala hráčov k socializácií.

Okrem samotnej socializácie sme chceli vytvoriť hru, ktorá by bola svojou hrateľnosťou výnimočná. Preto sme sa rozhodli, že v našej hre nebudú hráči jediný, ktorí budú zastávať hlavnú úlohu. Okrem hráčov, budú túto hlavnú úlohu hrať postavy vytvorené hráčmi, tzv. *avatari*. Týchto *avatarov* budú môcť používatelia trénovať. Čím lepšie bude *avatar* natrénovaný, tím väčší úspech bude mať pri súbojoch s ostatnými hráčmi.

14.2 Stručný opis hry

Smart Bomber je hra založená na klasickej hre *Bomberman*. Je to akčná hra, ktorej dej sa odohráva na mape (pôdorys mapy je mriežka), na ktorej sa nachádzajú dvaja alebo viacerí hráči. Každý hráč je schopný položiť bombu, ktorá po čase vybuchne. Svojím výbuchom dokáže ničiť objekty vo svojom blízkom okolí. Cieľom každého hráča je zneškodnenie ostatných súperov, vtedy sa stáva víťazom.

Hra poskytuje používateľom rôzne módy hrania. Hráči môžu hrať sami proti automatizovanému hráčovi (tzv. *bot*), môžu trénovať svojho *avatara*, môžu nechať súperiť svojich *avatarov* alebo môžu hrať medzi sebou. Taktiež je možné vyslať svojho *avatara* do virtuálneho sveta, kde súťažia proti sebe *avatari* ostatných používateľov.

14.3 Avatar

Avatar je typ hráča, ktorého si vytvára a trénuje používateľ. Môže si vybrať z niekoľkých rôznych typov *avatarov*, pričom ich rozdiel je v spôsobe učenia. Novo vytvoreného *avatara* je nutné natrénovať. Prvé overenie jeho naučených schopností môže spraviť hneď počas trénovania, kedy používateľ súperi so svojím *avatarom*.

Keď bude používateľ chcieť otestovať kvality svojho natrénovaného *avatara*, môže ho poslať do *online súboja*. Tu sa stretne s ostatnými *avatarmi*, kde budú medzi sebou súperiť a porovnávať si svoje naučené schopnosti.

14.4 Bot

Je to ďalší typ hráča, ktorý sa môže zúčastniť súbojov. Je to automatizovaný hráč, ktorý sa správa podľa vopred definovanej heuristiky. Na rozdiel od *avatara*, tohto hráča nemôže používateľ nijako trénovať alebo modifikovať. Používa sa v takých prípadoch, keď si chce používateľ zahrať s niekým iným ako svojím *avatarom* a nemá pri sebe žiadneho ľudského protihráča.

14.5 Konkrétne typy hráčov

14.5.1 Human

Hráč *human* predstavuje samotného používateľa. Tento hráč nemá v sebe žiadnu umelú inteligenciu a celé ovládanie zabezpečuje používateľ. Pohybovanie *human* hráča je ovládané pomocou smerových tlačidiel (tzv. *joystick*). Okrem týchto smerových tlačidiel má používateľ k dispozícií druhé tlačidlo, ktoré slúži na položenie bomby.

14.5.2 Evil bot

Evil bot predstavuje hráča, ktorý patrí do skupiny *botov*. Jeho štýl hry podlieha pravidlám, aby čo najviac a najčastejšie ohrozoval súpera, pričom prvoradé je pre neho, aby si dával pozor sám na seba. *Evil bot* hrá štýlom - ak nie je sám ohrozený, snaží sa ohroziť súpera.

Keďže samotný *evil bot* sa správal ako veľmi dobrý a silný súper, boli sme nútený vytvoriť viacero úrovní náročnosti. Vytvorili sme tri typy: *hard*, *normal* a *easy*. Pre nových používateľov je najvhodnejšie začať s *easy evil botom* a postupne skúšať ťažšie úrovne.

- *Samotný evil bot sa riadi svojou vlastnou implementovanou heuristikou. Tá mu hovorí o tom, že ak sa nenachádza v ohrození, tak má nájsť také pole, ktoré je najbližšie k nemu, pričom z neho dokáže ohroziť súpera. Následne po vykonaní svojho kroku opätovne skontroluje svoje ohrozenie a vyhľadá políčko, z ktorého dokáže ohroziť najbližšieho súpera.*

Rozdelenie do jednotlivých úrovni sme spravili nasledovným spôsobom. Hard evil bot sa správa presne podľa naprogramovanej heuristiky. Normal evil bot sa správa presne tak isto ako hard evil bot, ale s tým, že je 25%-ná pravdepodobnosť zmeny jeho rozhodnutia (urobenia chyby). Easy evil bot funguje na takom istom princípe, akurát s tým, že vykoná chybný krok s 50% pravdepodobnosťou.

14.5.3 Neurónový avatar

Neurónový hráč patrí do skupiny *avatarov*. To znamená, že používateľ ho môže trénovať. Tento *avatar* používa ako mozog neurónovú sieť. Táto neurónová sieť má na starosti vyhodnocovanie ďalšieho kroku, ktorý vykoná *avatar*.

Na začiatku je tento neurónový hráč takmer úplne nepoužiteľný. V podstate jediné, čo robí, sú náhodné kroky. Po prvých tréningoch sa pomaličky zlepšuje a prestáva sa správať nepredvídateľne. Postupným trénovaním začína *avatar* hrať podobným štýlom, akým hrá aj používateľ. V prípade, že používateľ zistí nedostatky v natrénovaní, môže svojho *avatara* jednoducho preučiť a pretrénovať ho na novú stratégiu hrania.

Keďže naším cieľom bolo vytvorenie takých hráčov, ktorý by v čo najväčšej miere dokázali napodobniť herný štýl hráča, samotný *avatar* sa učí len a len od používateľa. To znamená, že sleduje jeho situáciu, v akej sa nachádza a následnú reakciu. Učenie a zdokonaľovanie avatara prebieha len za herného módu *Training*. Za iných okolností *avatar* svoje správanie nijako neupravuje a nemení a hrá len podľa toho, čo sa naučil v tréningoch so svojím používateľom.

- *Samotná neurónová sieť sa skladá z troch vrstiev. Na vstupnej vrstve sa nachádza informácia o situácií okolo* avatara*. Na výstupnej vrstve sa nachádza informácia o tom, aký bude nasledujúci krok* avatara

Ako bolo spomenuté, samotné učenie prebieha len od používateľa a len počas herného módu Training. Počas tohto módu avatar sleduje reakcie používateľa vzhľadom na jeho danú situáciu. Následne si túto situáciu a reakciu zapamätá. Na konci tréningu sa podľa týchto zapamätaných situácií a reakcií preučí mozog avatara.

14.5.4 Markovovský avatar

Je to ďalší typ *avatara*, ktorého je možné trénovať používateľom. Tento *avatar* robí pohyby na základe štatistík získaných od používateľa. Podobne, ako *neurónový avatar*, je aj tento *avatar* bez natrénovania nepredvídateľný.

Priebeh trénovania *avatara* je veľmi podobný, ako u prvého spomínaného *avatara*. Používateľ si v tréningu zahrá niekoľko hier, z ktorých sa *avatar* snaží čo najviac priblížiť hernému štýlu používateľa. Samotné vyhodnocovanie krokov prebieha na základe zozbieraných údajov. Pri zbieraní údajov sa sleduje aktuálna situácia hráča a jeho následná reakcia. Ak nastanú za rovnakej situácie dve alebo viac rôznych reakcií, tak sa vychádza z pravdepodobnosti. Pravdepodobnosť, s akou sa vykoná daná reakcia záleží od toho, ako často túto reakciu v danej situácií využíval používateľ.

- *Učenie prebieha v dvoch fázach. Prvou fázou je sledovanie používateľa pri hernom móde Training. Počas tohto tréningu si avatar robí záznamy, ako sa používateľ správa za určitých okolností. Druhá fáza je využívanie týchto zozbieraných údajov pri hraní.*

  V tomto prípade nenastáva učenie, ako to bolo pri neurónovom avatarovi. Zozbierané údaje počas tréningu slúžia avatarovi pri rozhodovaní svojej vlastnej reakcie. Postupuje tak, že najskôr si rozanalyzuje svoju vlastnú situáciu. Následne sa vo svojich záznamoch snaží vyhľadať, či sa používateľ počas tréningu nenachádzal v rovnakej situácií. Ak nie, tak sa snaží nájsť čo najviac podobnú situáciu používateľa so svojou. Potom, ako nájde zhodnú alebo podobnú situáciu s používateľovou, začne skúmať, aké mal reakcie. Ak bola na danú situáciu len jedna, tak ju vykoná. V prípade výskytu viacerých reakcií sa snaží vykonať tú, ktorú robil používateľ častejšie, s väčšou pravdepodobnosťou. S menšou pravdepodobnosťou sa vykonajú reakcie, ktoré používateľ používal menej.

14.5.5 Genetický avatar

Posledným *avatarom*, ktorý sa nachádza v prototype hry je *genetický avatar*. Tento *avatar* je riadený experimentálnou *doprednou neurónovou sieťou*, ktorá má potenciál naučiť sa stratégiu hráča už po jednej hre.

V tomto prototype je tento *genetický avatar* v napodobňovaní správania sa používateľa najmenej úspešný.

- *Vstupom tejto neurónovej siete sú políčka okolo hráča v rozsahu 5x5 s hráčom uprostred. Výstupom je akcia avatara, teda pohyb alebo položenie bomby. Táto neurónová sieť navyše obsahuje aj okno do minulosti, vďaka čomu môže na rovnakú situáciu (rovnaké vstupy v inom čase) reagovať odlišne, v závislosti od predchádzajúcich situácií. Váhy tejto neurónovej siete nie sú nastavované systémom spätného šírenia chýb, ale pomocou genetického algoritmu. Genetický algoritmus vytvára populácie jedincov, kde každý jedinec reprezentuje jednu neurónovú sieť – jedno riešenie. Nové populácie vznikajú na základe starých riešení tromi spôsobmi:*
  - *skopírovaním najlepších jedincov*
  - *vytvorením náhodných nových jedincov*
  - *pseudonáhodným krížením jedincov (väčšiu šancu na kríženie majú úspešnejšie jedince)*

Úspešnejší jedinec je ten, ktorý dokáže lepšie opakovať kroky hráča. Po skončení algoritmu vznikne jedinec – neurónová sieť, ktorá pri postupnosti vstupov zhodujúcej sa tej pri učení, dá na výstupe rovnakú postupnosť akcií, aké zvolil hráč.

14.6 Prihlasovacia obrazovka a registrácia

Pri prvom spustení hry sa každému používateľovi zobrazí prihlasovacia obrazovka. Je potrebné, aby vyplnil svoje prihlasovacie údaje, ktorými sú prihlasovacie meno a heslo.

V prípade, že používateľ chce hrať hru po prvý krát, je potrebné, aby si vytvoril nové používateľské konto. Konto si vytvorí pomocou registrácie, kde zadá požadované prihlasovacie meno a heslo. Ak nie je požadované prihlasovacie meno priradené inému používateľovi, vytvorí sa mu konto.

14.7 Menu

Menu poskytuje výber herného módu a nastavenia hry.

Na začiatku si používateľ vyberie, aký typ hry chce odohrať. Má na výber niekoľko rôznych variant, ako *quick game*, *training* a *multiplayer*. Následne si vyberie mapu a zvolí si s akými súpermi chce súťažiť. V prípade *training* módu je dôležité, aby si vybral toho *avatara*, ktorého chce trénovať.

Ďalšou možnosťou v menu sú *online games*. Tu posiela používateľ svojich natrénovaných *avatarov* na *online zápasy*. V tomto móde *avatari* zbierajú body pre používateľa. Taktiež môže používateľ pozorovať, či je jeho *avatar* natrénovaný dostatočne alebo potrebuje pretrénovať.

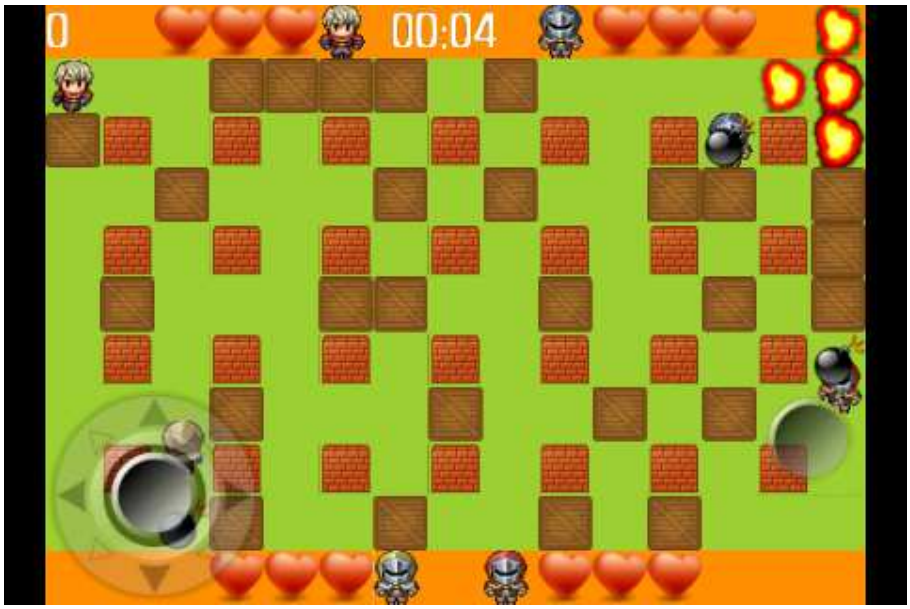
V menu je taktiež možné editovať nastavenia hry. Používateľ si môže nastaviť svojho základného *avatara*, uložiť si automatické prihlasovanie a pod.

Obr. 30 - Hlavné menu hry Smart Bomber

14.8 Herné prostredie

Základné prostredie, kde sa súboje medzi hráčmi odohrávanú je bojisko. Mapa bojiska má tvar mriežky s rozmermi *M x N*. Základné objekty, ktoré sa na mape môžu nachádzať, sú stena a škatuľa. Tieto komponenty poskytujú jednotlivým mapám rôznorodosť. Stena nie posúvateľná a ani zničiteľná. Naopak, škatuľu je možné aj posunúť aj zničiť. Avšak nie je možné posunúť viac ako jednu škatuľu súčasne. Ďalším komponentom je bomba. Bombu môže položiť akýkoľvek hráč, či už sa jedná o používateľa, *avatara* alebo *bota*. Bomba ničí všetky komponenty v dosahu (okrem steny). V prípade zasiahnutia výbuchom bomby je hráč porazený.

Ďalšími komponentmi sú už spomínaný hráči. Nachádzajú sa tu tri rôzne druhy hráčov: *human*, *avatar* a *bot*. Hráči a ostatné komponenty sa na mape môžu pohybovať štyrmi základnými smermi: hore, hole, doprava a doľava, pričom sa vždy musia nachádzať v konkrétnom poli mriežky. V jednom poli sa môže nachádzať v danom čase iba jeden komponent (výnimkou je situácia, keď na danom mieste sa nachádza jeden hráč a jedna bomba) zároveň presun z jedného miesta na druhé je postupné, čiže preskakovanie komponentov (prekážok) nie je umožnené.



Obr. 31 – Herné prostredie

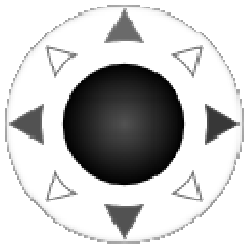
- *Ako už bolo spomínané, je na výber viacero máp. Tieto mapy sú načítavané zo súboru. Výber spočíva na hráčovi. Štartovacia pozícia jednotlivých hráčov sa môže meniť v závislosti od mapy. Každá mapa má však označené štyri miesta, kde môže byť umiestnený hráč.*

  Jednotlivé mapy sú reprezentované v podobe textového súboru. Každý súbor obsahuje základné informácie o mape, ako napr. rozmery, názov mapy a pod. Ďalej súbor obsahuje „mriežku" tvorenú znakmi, ktoré reprezentujú počiatočný stav mapy na začiatku hry. Počiatočný stav obsahuje pozíciu všetkých stien a škatúľ, zároveň štartovacie pozície pre maximálny možný počet hráčov. Pri načítaní mapy sa na nej umiestnia všetky steny a škatule, následne sa postupne obsadzujú štartovacie miesta hráčov.

14.9 Ovládanie

Ovládanie v hre sa používa pri hráčovi typu *human,* ktorý je riadený používateľom. Na toto slúžia dva riadiace prvky: *joystick* a tlačidlo na položenie bomby. Obidva prvky sú umiestnené na displeji zariadenia a reagujú na dotyk. *Joystick* sa nachádza v ľavom dolnom rohu obrazovky. Funguje ako riadiaca páka, slúži na posúvanie príslušného hráča. Používateľ položí prst do stredu ovládača a posunutím do zvoleného smeru (naklonením páky) posúva hráča. Tlačidlo na položenie bomby je

v pravom dolnom rohu oproti ovládaču. Pri jeho stlačení príslušný hráč položí bombu na jeho aktuálne miesto.



Obr. 32 a obr. 33 – Joystick a tlačidlo na položenie bomby

- *Ovládanie hry podporuje multitouch, t.j. dokáže spracovať viac vstupov v danej chvíli. Počas pohybu môže používateľ položiť bomby bez toho, aby prestal posúvať hráča.*

14.10 Herné módy

Používateľ má možnosť si zahrať rôzne typy hier. Aplikácia ponúka herné módy určené pre humánneho hráča, pre *avatara*, aj pre oboch. Úspešnosť používateľa, resp. *avatara* v rôznych herných módoch je odmenená rôznym počtom bodov. Všetky typy hier sa odohrávajú na mape opísanej vyššie. Cieľom hráčov je zneškodniť ostatných hráčov. Výherca je ten, ktorý prežije.

14.10.1 Quick game

Tento typ hry je určený najmä pre nových používateľov, alebo pre tých, ktorí si chcú zahrať hru bez nutnosti ďalšieho potrebného nastavovania. *Quick game* umožňuje používateľovi vyskúšanie základnej hrateľnosti, vďaka čomu sa môže oboznámiť so základným herným prostredím a naučiť sa princíp ovládania hráčov. Hra sa odohráva v klasickom hernom prostredí, t.j. na bojisku a zúčastňuje sa jej jeden hráč typu *human* riadený používateľom a ľubovoľný počet súperov, ktorí sú v tomto prípade hráči typu *bot*. Pred spustením hry si používateľ môže nastaviť počet súperov a ich úroveň náročnosti. Taktiež si môže zvoliť ktorúkoľvek mapu, na ktorej si chce vyskúšať hru.

14.10.2 Training

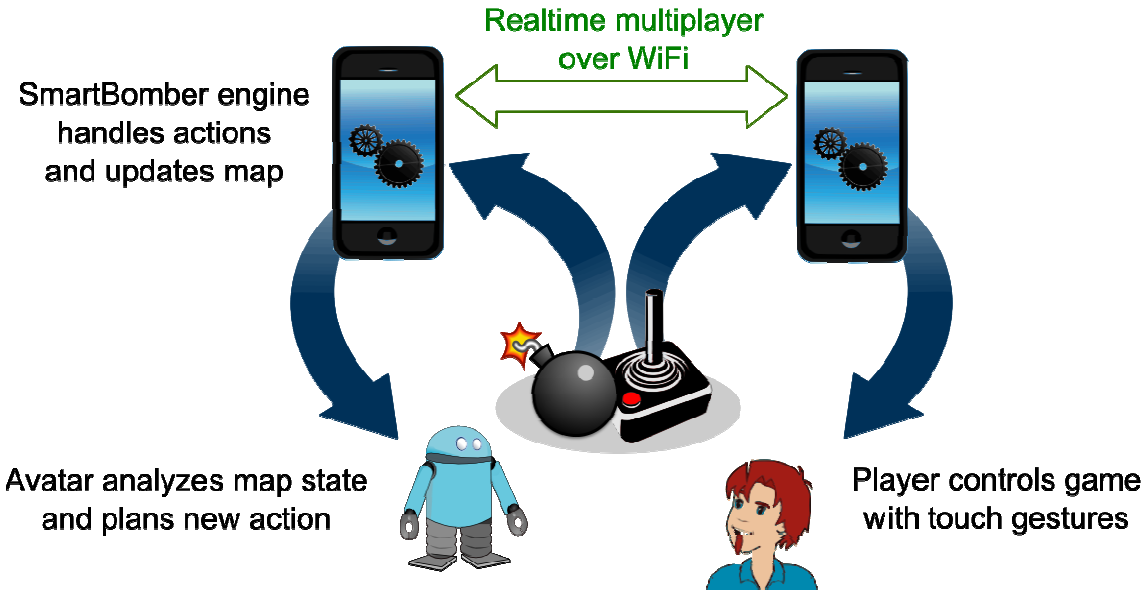
Tento herný mód, ako prezradzuje aj jeho názov, si používateľ zahrá za účelom trénovania svojho *avatara*. Úspech používateľa v konečnom dôsledku spočíva v úspechu jeho *avatara*, preto je dôležité, aby ho natrénoval a naučil spôsob, ako zvíťaziť v súboji.

Tento typ hry sa naoko nelíši od predchádzajúceho. Podobne sa odohráva na bojisku (na mriežke), kde sa nachádzajú tie isté typy predmetov, činnosť hráčov je taktiež totožná. Rozdiel spočíva v zúčastnených hráčoch a v hlavnom cieli hry. Tréningu sa vždy zúčastní používateľ, ktorého reprezentuje hráč typu *human*, ľubovoľný počet hráčov typu *bot* s voliteľnou úrovňou náročnosti a *avatar* používateľa. Postup hry je podobný, ako v ostatných prípadoch: každý sa snaží zneškodniť druhého; pritom však hlavným cieľom používateľa je dbať na správnu stratégiu, pretože *avatar* sleduje kroky svojho učiteľa. Pri trénovaní si zúčastnený *avatar* zapamätá postupnosť vykonaných akcií používateľa a tým preberá jeho herný štýl. Po danom čase sa *avatar* naučí „rozmýšľať“, ako jeho učiteľ. Výsledok úspešného trénovania sa preukáže v situáciách, keď *avatar* na daný stav hry reaguje podobným spôsobom, ako to robil používateľ počas trénovania. Hru tohto typu je taktiež možné zahrať na ľubovoľnej mape zvolenej používateľom pred spustením hry.

14.10.3 Multiplayer

Hra *multiplayer* sa odohráva vo zvyčajnom hernom prostredí, pričom sa na mape nachádzajú iba hráči typu *human*. Ide o priame súťaženie dvoch alebo viacerých používateľov. Pre vytvorenie *multiplayer* hry musí mať každý zúčastnený používateľ k dispozícii vlastné mobilné zariadenie, zároveň všetky zariadenia musia byť pripojené do jednej siete *WiFi*. Hru musí vytvoriť jeden používateľ, ostatní sa pridajú do existujúcej hry. Používateľ, ktorý hru vytvára, má možnosť vybrať ľubovoľnú mapu, na ktorej si spoločne zahrajú.

- *Pri hre typu multiplayer sa medzi mobilnými zariadeniami nadväzuje Peer-to-Peer komunikácia, čiže všetky zariadenia obsluhujú ostatné a zároveň od nich čerpajú dáta. Sekvencia operácií je znázornená na Obr. 34. Na zariadení, ktoré hru vytvorilo beží časovač. Po každom uplynutí časového intervalu (20 milisekúnd) sa zozbierajú všetky požadované akcie hráčov a pošlú na druhé zariadenie. Tým sa inicializuje výmena a spracovanie akcií. Na druhom zariadení sa taktiež zozbierajú akcie od hráčov, následne sa pridajú do zoznamu prijatých akcií a vykonajú sa na mape. Na konci aj druhé zariadenie pošle akcie požadované lokálnymi hráčmi na inicializujúce zariadenie, aby aj to mohlo aktualizovať mapu.*



Obr. 34 – Multiplayer

14.10.4 Online tournament

Ako sme už viackrát spomínali, základný cieľ hráča v našej hre je vychovať úspešného *avatara*. Pri *multiplayer* hre si porovnávajú silu iba humánni hráči, preto sme navrhli posledný typ hry, ktorý je určený len pre *avatarov*. Tento typ sa nazýva „*Online Tournament*“, ktorý jeho meno dostal o mieste konania. *Online* hry sa hoci odohrávajú vo zvyčajnom hernom prostredí, ale nie priamo na mobilných zariadeniach. Ak sa používateľ rozhodne, že je jeho avatar dostatočne pripravený, má možnosť ho poslať do súboja. Súboje sa uskutočňujú na vzdialenom serveri, kde sa nachádzajú dáta všetkých existujúcich používateľov a ich avatarov (viac v kapitole „Vzdialený server“). *Avatary*, ktoré boli ich učiteľmi poslané do boja neustále súťažia. Do *online súbojov* už používatelia nemôžu priamo zasahovať, *avatary* sa musia uplatniť pomocou schopností, ktoré sa naučili počas trénovania. Používateľ má možnosť svojho *avatara* kedykoľvek zavolať naspäť. Informácie o výsledkoch súbojov sú dostupné v podobe herných štatistík, na základe ktorých si používateľ môže usúdiť, či je jeho *avatar* dostatočne pripravený, prípadne ho ešte musí dotrénovať.

14.10.5 Bodovanie

Určité činnosti hráčov počas hry sú odmenené rôznymi počtami bodov. Body za dané činnosti sa líšia pre rôzne typy hier, taktiež závisia od typu hráča (*human* alebo *avatar*). Získané body sa

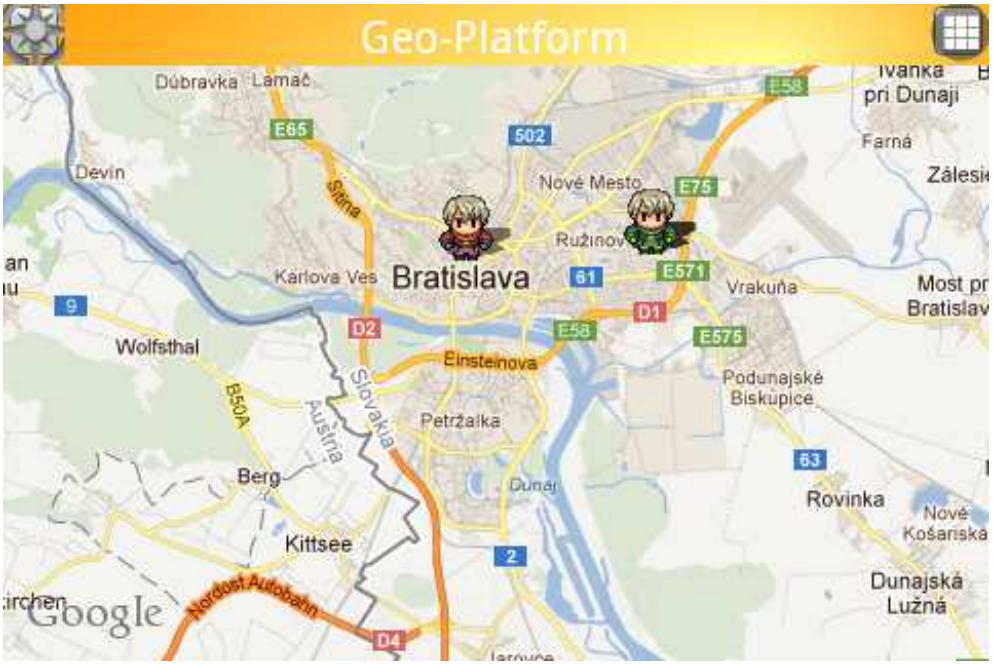
uchovávajú a zarátavajú do globálnych výsledkov hráčov i avatarov. Presný rozpis odmeňovacieho systému je uvedený v Tab. 47. Každý hráč má na začiatku hry tri životy, t.j. 3 krát treba daného hráča zraniť, aby bol zničený.

	Počet bodov	**Koeficient**			
		Quick Game	**Training**	**Multiplayer**	**Online Tournament**
Zničenie škatule	5	Human: 1x	Human: 1x Avatar: 2x	Human: 3x	Avatar: 1x
Zničenie hráča	25				
Víťazstvo v hre	200				

Tab. 47 – Odmeňovací systém

14.11 Geo-platforma

Ako bolo na začiatku spomenuté, našou snahou bolo vytvoriť mobilnú hru, ktorá motivuje ľudí v tom, aby sa zoznamovali a častejšie stretávali. Tento sociálny charakter aplikácie spočíva v spôsobe hľadania súperov pre multiplayer hru. Ak si z menu zvolíme záložku „Multiplayer“, pred spustením hry si môžeme prejsť do režimu „Geo-platform“. V tomto režime sa otvorí mapa miesta, kde sa práve nachádzame. Zistenie polohy sa realizuje prostredníctvom GPS zariadenia, resp. pomocou GSM siete. Na mape za označia polohy všetkých používateľov aplikácie Smart Bomber, ktorí sa nachádzajú v blízkom okolí, alebo na zobrazenej časti mapy. Toto používateľom umožňuje, aby sa navzájom našli, aj keď predtým jeden o druhom ani nevedeli. Pravidelné stretávanie sa používateľom prináša väčšie množstvo odohraných multiplayer hier, čo je pre používateľov (*human* hráčov) najlepšia príležitosť na získanie vzácnych bodov (viď. Tab. 47 – Odmeňovací systém).



Obr. 35 – Geo-platforma

- *Tento prototyp aplikácie je určený pre platformu Android. Vďaka tomu pri návrhu Geo-platformy sme mali možnosť využiť technológiu Google API a knižnicu Google Maps, ktoré zabezpečili zobrazenie mapy. Označenie polôh používateľov je realizovaný prostredníctvom vzdialeného servera, kde sa neustále aktualizujú údaje používateľov vrátane súradníc polohy. O centrálnom úložisku dát viac čítane v kapitole „Vzdialený server".*

14.12 Nastavenia hry

Posledná možnosť v hlavnom menu aplikácie sú nastavenia hry. Tu sa nachádzajú všetky informácie a nastavenia, ktoré má používateľ možnosť prezerať a meniť. Táto záložka je rozdelená na 4 časti: manažment avatarov, nastavenia hry, nastavenia prihlasovania a herné štatistiky.

Prvá časť obsahuje nastavenia avatarov používateľa. Poskytuje možnosť vytvorenia nových avatarov, výveru preddefinovaného avatara (s ktorým sa hrá) a mazania avatarov. Druhá časť obsahuje nastavenia hry, ako viditeľnosť polohy používateľa na Geo-platforme a zvuky hry. V treťom časti si môže používateľ zvoliť možnosť automatického prihlasovania do aplikácie, resp. zmeniť jeho prihlasovacie údaje.

Posledná sekcia nastavení ponúka informácie o herných štatistikách. Ako sme sa z predchádzajúcich častí dozvedeli, používateľ súťaží priamo proti ostatným používateľom, zároveň

posiela do boja svojich avatarov, ktorých úspech alebo neúspech tiež patrí používateľovi. Herné štatistiky ponúkajú podrobný rozpis o odohraných hrách pre používateľa (humánny hráč) a zároveň pre jeho preddefinovaného avatara. Na obrazovke sa zobrazí počet odohraných hier a počet výhier používateľa a zvlášť preddefinovaného avatara pre všetky typy hry, t.j. hodnoty:

- počet hier Quick Game používateľa
- počet výhier v Quick Game používateľa
- počet hier Training používateľa
- počet hier Training avatara
- počet výhier v Training používateľa
- počet výhier v Training avatara
- počet hier Multiplayer používateľa
- počet výhier v Multiplayer používateľa
- počet hier Online Tournament avatara
- počet výhier v Online Tournament avatara
- percento úspešnosti používateľa
- percento úspešnosti avatara

Obr. 36 – Herné štatistiky

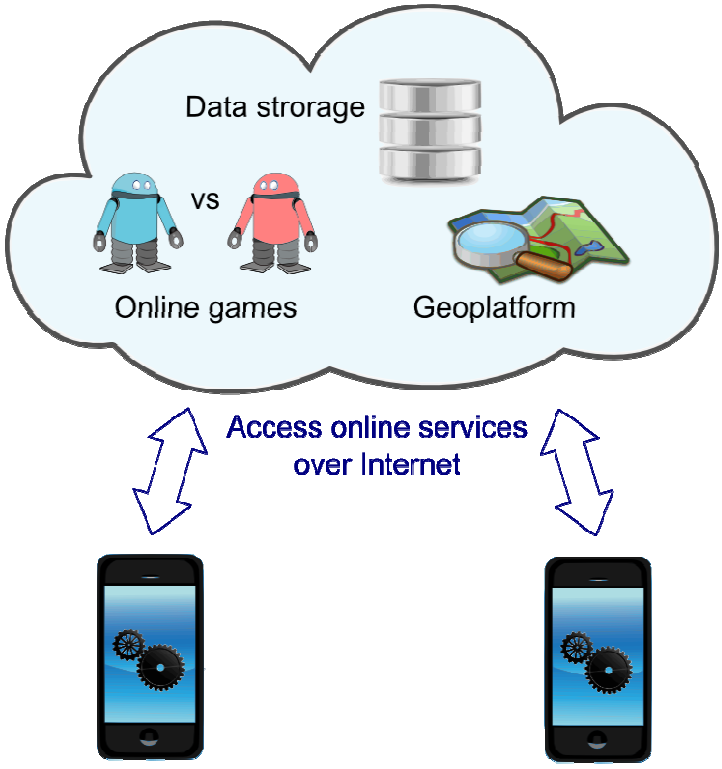
14.13 Vzdialený server

Nakoľko hra Smart Bomber ponúka funkcionality, ako súťaženie avatarov, lokalizáciu používateľov a pod., určité činnosti sú podporované vzdialeným serverom, s ktorým jednotlivé

zariadenia komunikujú prostredníctvom internetu. Bez sieťového pripojenia tieto činnosti nie sú možné vykonať. Vzdialený server má tri hlavné komponenty: centrálne úložisko dát, Geo-platforma a online súťaženie avatarov.

V úložisku sa udržiavajú všetky údaje registrovaných používateľov, ako osobné a prihlasovacie údaje, herné nastavenia, poloha, avatary používateľa a pod. Tieto údaje sa pravidelne aktualizujú podľa potreby. Online súboje neustále bežia medzi avatarmi, ktoré sú poslané do boja. Výsledky súbojov sa pravidelne aktualizujú v databáze. Na lokálnych zariadeniach sú dočasne uložené len vybrané údaje, ako napr. aktuálne prihlásený používateľ a pod. Všetky ostatné informácie zobrazené na mobilnom zariadení sa získavajú zo servera. Pri určitých činnostiach používateľa sa zmeny relevantných dát automaticky aktualizujú na serveri.

- *Pre umožnenie vhodného spôsobu komunikácie sme identifikovali činnosti používateľa, ktoré vyžadujú internetové pripojenie, následne sme ich zoskupovali a špecifikovali na nižšej úrovni tak, aby boli realizovateľné v podobe webovej služby. Tieto služby sú poskytované vzdialeným severom a využívané mobilnými zariadeniami. Pomocou webových služieb sa uskutočňuje prenos dát používateľov, avatarov a herných štatistík z mobilných zariadení na server a naopak. Architektúra vzdialeného servera a komunikácia s mobilnými zariadeniami je znázornená na Obr. 37.*



Obr. 37 – Komunikácia servera s mobilnými zariadeniami

14.14 Zhrnutie

Smart Bomber je hra určená pre mobilné platformy, založená na princípe klasickej počítačovej hry Bomberman. Autori sa pri tvorbe zameriavali na dve inovatívne myšlienky: nepriame súťaženie používateľov pomocou učenia a súťaženia avatarov a socializácia používateľov.

Základné herné prostredie je mapa v podobe mriežky, na ktorej sa nachádzajú rôzne predmety, resp. samotní hráči. Hráči môžu byť troch typov: humánny hráč, avatar a bot. *Humánny hráč* je riadený používateľom, *avatar* je „žiak“ používateľa, ktorý je schopný sa naučiť herný štýl používateľa pomocou umelej inteligencie a následne hrať samostatne proti iným používateľom, resp. ich avatarom, *bot* je súper ovládaný vopred naprogramovanou logikou.

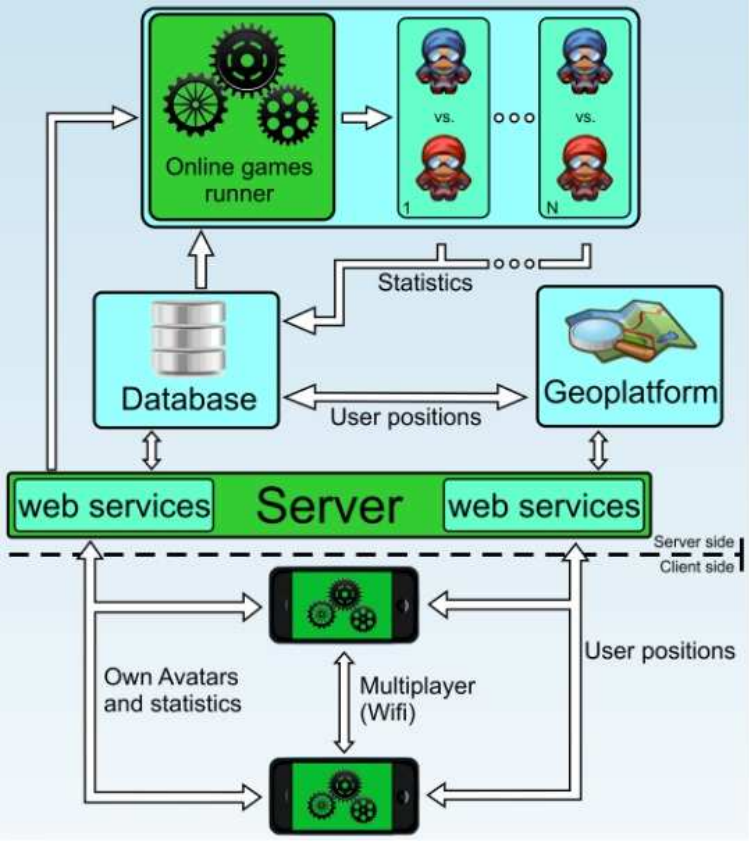
Hra podporuje štyri herné módy. *Quick game* je jednoduchý typ hry určený pre vyskúšanie hry. V *training* móde hrá proti sebe humánny hráč a jeho avatar, kde používateľ trénuje svojho žiaka. *Multiplayer* poskytuje možnosť dvom používateľom, aby si zahrali proti sebe prostredníctvom WiFi

siete. *Online game* sa uskutočňuje na vzdialenom hernom serveri, kde súťažia avatari používateľov. Výsledky sú k dispozícii v podobe herných štatistík.

Používatelia sa pred spustením aplikácie musia zaregistrovať a následne prihlásiť. Prihlasovacie údaje sú spolu s dátami avatarov udržiavané na serveri, preto je pre spustenie hry potrebné mať pripojenie na internet. Podobne je na serveri uložená informácia o lokalite jednotlivých používateľov. Toto sa využíva pri funkcionalite Geo-platforma, ktorá zobrazením mapy poskytuje informácie používateľom o polohe ostatných používateľov.

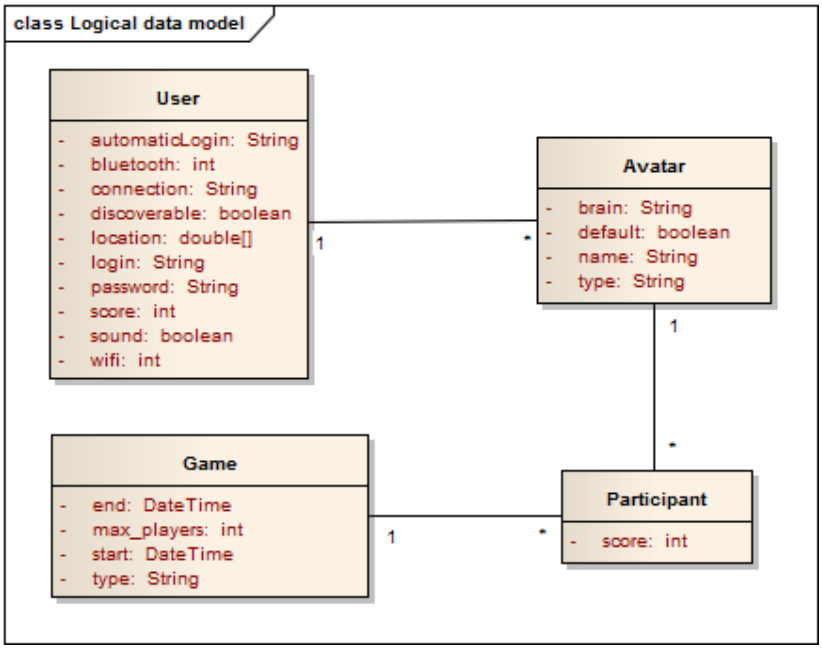
14.14.1 Globálna architektúra

Globálny kontext interakcie podsystémov je možné pozorovať na Obr. 38. Celý systém hry je rozdelený na dva hlavné podsystémy: klientska aplikácia (samotná hra na mobilnom zariadení) a server. V prípade multiplayer hry komunikácia medzi dvomi mobilnými zariadeniami sa uskutočňuje prostredníctvom WiFi siete. Medzi zariadeniami sa vytvorí Peer-to-Peer komunikácia, t.j. posielajú si medzi sebou dáta o aktuálnych vykonaných akciách, čím sa zabezpečuje rovnaká činnosť na obidvoch stranách. Základná sekvencia činnosti jadra hry pozostáva zo zberu požiadaviek (požadovaných akcií) prítomných hráčov na mape, následne ich spracovanie a vykonávania na mape (zmena stavu mapy). Pri multiplayer hre pred spracovaním akcií, každé zariadenie pošle zoznam akcií na druhé zariadenie, aby na obidvoch stranách bola vykonaná tá istá situácia – tým sa zariadenia synchronizujú.



Obr. 38 – Globálna architektúra aplikácie

Po ukončení akéhokoľvek typu hry, registrácii, prihlásení alebo nastavení údajov avatara sa zmeny, ako napr. herné štatistiky, získané body, zmena stavu mozgu avatara a pod., posielajú na vzdialený server, kde sa ukladajú do centrálneho úložiska. Z úložiska sa čerpajú dáta o polohe používateľov, ktoré sú poskytované pre mobilné zariadenia prostredníctvom Geo-platformy, ktorá zároveň tieto hodnoty získava a aktualizuje v databáze. Na serveri neustále bežia turnaje avatarov prihlásených do online hier. Tento komponent postupne pridáva avatrov do hry a generuje reálne výsledky, ktoré sú uložené do databázy a následne poskytnuté pre používateľov. Dátová štruktúra, v ktorej sú údaje na serveri organizované, je uvedená na Obr. 39.



Obr. 39 – Dátový model na serveri

14.14.2 Vnútorná architektúra aplikácie

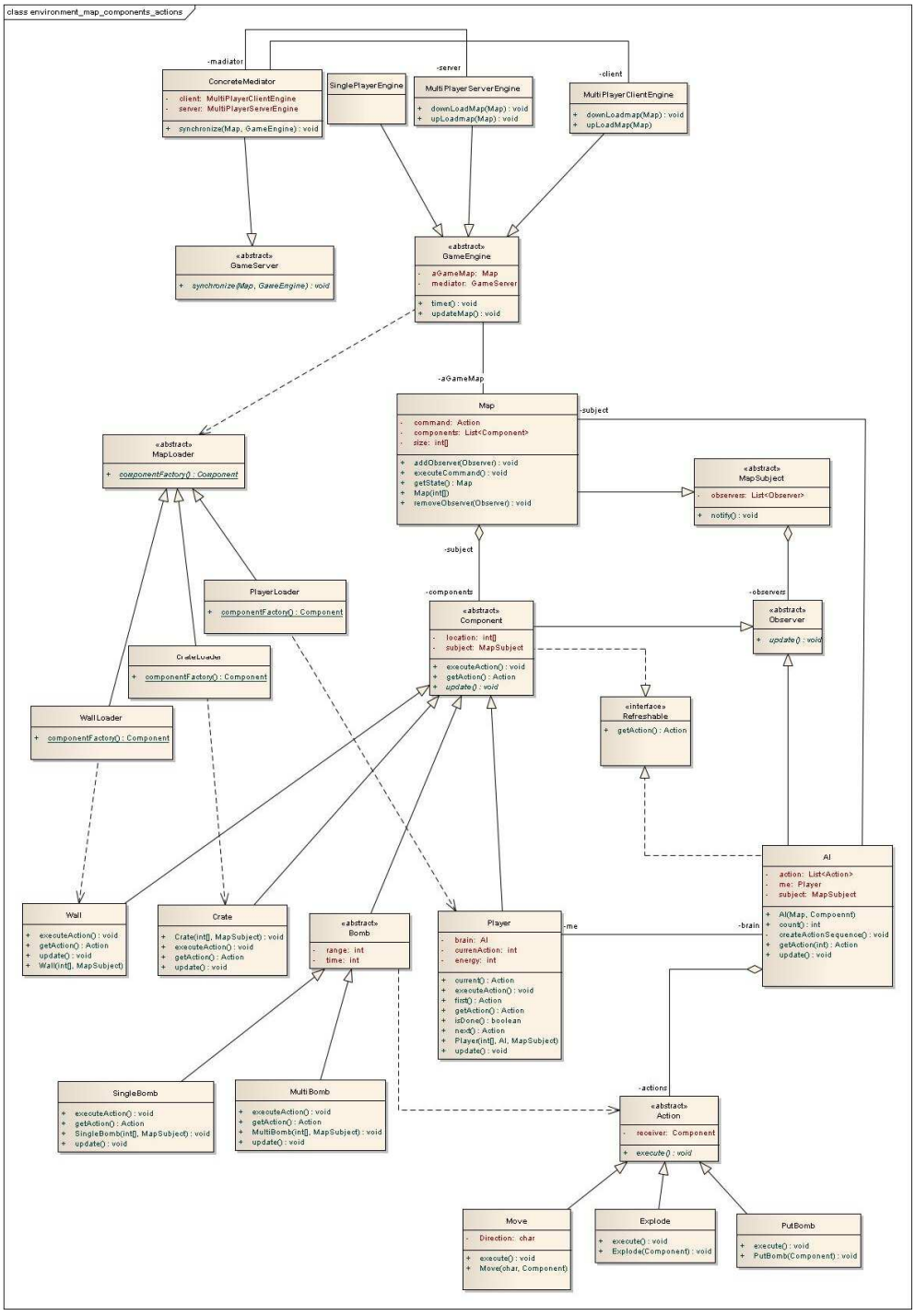
Vnútorná architektúra hry (aplikácia na mobilnom zariadení) je znázornená na Obr. 40. Najdôležitejšie komponenty systému sú opísané nižšie.

Jadro hry tvorí objekt *GameEngine*. Udržiava mapu a je zodpovedný za jej aktualizáciu, v prípade multiplayer hry za synchronizáciu s druhým zariadením.

Map je objekt reprezentujúci samotnú hernú mapu, na ktorej sa hra uskutočňuje. Agreguje skupinu abstraktných komponentov, ktoré sú všetky predmety, resp. hráči, ktorí sa na mape nachádzajú. Komponent môže byť objekt typu stena, krabica, bomba alebo hráč.

Hráč je špecifický typ komponentu z pohľadu riadenia. Správanie sa ostatných komponentov je definované v rámci daného objektu, k hráčovi je však priradený jeden typ ovládača, ktorým môže byť humánny ovládač (používateľ), alebo umelý „mozog“ (mozog bota alebo avatara s umelou inteligenciu - AI).

Akcie, ktoré komponenty môžu vykonávať (posun, položenie bomby a výbuch) sú definované v abstraktnom objekte *Action*. Je to objekt obsahujúci akciu, ktorú je jadro hry schopné vykonať na mape.



Obr. 40 – Vnútorná architektúra aplikácie

Interakcia mapy s komponentmi je obojsmerná – komponenty (alebo ich ovládače, ako napr. AI) sledujú mapu a získavanú jej aktuálny stav. Táto situácia je vyriešená pomocou architektonického vzoru *Observer*, pričom objekt *Map* tvorí predmet vzoru, komponenty a AI sú sledovači. Druhá časť komunikácie je reakcia komponentov na stav mapy. Reakcia spočíva vo vygenerovaní a odoslaní požadovanej akcie. Každý komponent, resp. AI si vytvorí inštanciu objektu *Action* požadovaného typu s ľubovoľnými parametrami. Táto inštancia sa posiela mape, kde sa následne vykoná. Generovanie, prenos a vykonávanie akcií sú navrhnuté na základe vzoru *Command*.