VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hra Simulace světa v prostředí Internetu

Game World Simulation over Internet

2015

Jan Michálek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jan Michálek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Hra Simulace světa v prostředí Internetu
Game World Simulation over Internet

Zásady pro vypracování:

Cílem práce je vytvořit hru Simulace světa v 2D (z horního pohledu) v jazyce Java. Hra bude umožňovat hru více hráčů v prostředí Internetu. Mapu světa bude možno vytvořit v editoru. Svět bude obsahovat různé typy budov a povolání.

Hra bude umožňovat:
1. Editor mapy světa (budovy, územní plán, území gangů).
2. Zapojení jednoduché umělé inteligence pro všechny typy postav.
3. Volbu povolání a jejich změnu v průběhu hry (taxikář, politik, gangster, ...).
4. Postavit různé typy budov (škola, banka, činžovní domy, vily, ...)
5. Sdružovat hráče do gangů a firem.
6. Hru více hráčů.
7. Program umožní připojení do současně vyvíjeného portálů her v jazyce Java.

Práce bude obsahovat:
1. Implementaci výše popsané hry.
2. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.
3. Uživatelskou dokumentaci aplikace.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

Dále podle pokynů vedoucího práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 ..... Jan Milášek .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, zejména mému vedoucímu Ing. David Ježek, Ph.D., který si na mě vždy našel čas a poradil mi, když jsem to potřeboval.

Abstrakt

Bakalářská práce se zabývá návrhem počítačové hry a je naprogramována v jazyku JAVA, v 2D grafice a z horního pohledu. Hra pracuje přes internetovou komunikaci, hráči se připojují pomocí clienta na server. Hra má odrážet reálný svět. Hráči si budou moct vybrat z několika povolání, které mohou střídat. Za provedené úkony povolání se jim zvyšuje hodnost. Hráči se mohou sdružovat ve firmách, ale jenom v určitých povolání, a nebo v gangu. Pro každé povolání je vytvořena jednoduchá umělá inteligence.
Další částí bakalářské práce je vytvoření jednoduchého editoru mapy. Kromě vytvoření samotné mapy se dají vložit i umělá inteligence a auta.

Klíčová slova: JAVA, Reálný svět, Simulace

Abstract

Bachelor thesis deals with computer games and is programmed in Java 2D graphics and top view. The game works through Internet communication, the players connected via a client in server. The game is meant to reflect the real world. Players will be able to choose from several professions that can rotate. For acts performed profession they rank increases. Players may form themselves into companies, but only in certain professions, or in a gang. For every job is created a simple artificial intelligence.
Another part of the thesis is to create a simple map editor. In addition to the creation of the map itself can also insert artificial intelligence and cars.

Keywords: JAVA, Real world, Simulation

Seznam použitých zkratek a symbolů

AI	–	Artificial Intelligence
API	–	Application Programming Interface
FPS	–	Frames Per Second
DTO	–	Data Transfer Object
IP	–	Internet Protocol
JAR	–	Java ARchive
LWJGL	–	LightWeight Java Game Library
OpenGL	–	Open Graphics Library
VRAM	–	Video Random-Access Memory
UI	–	Umělá inteligence

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

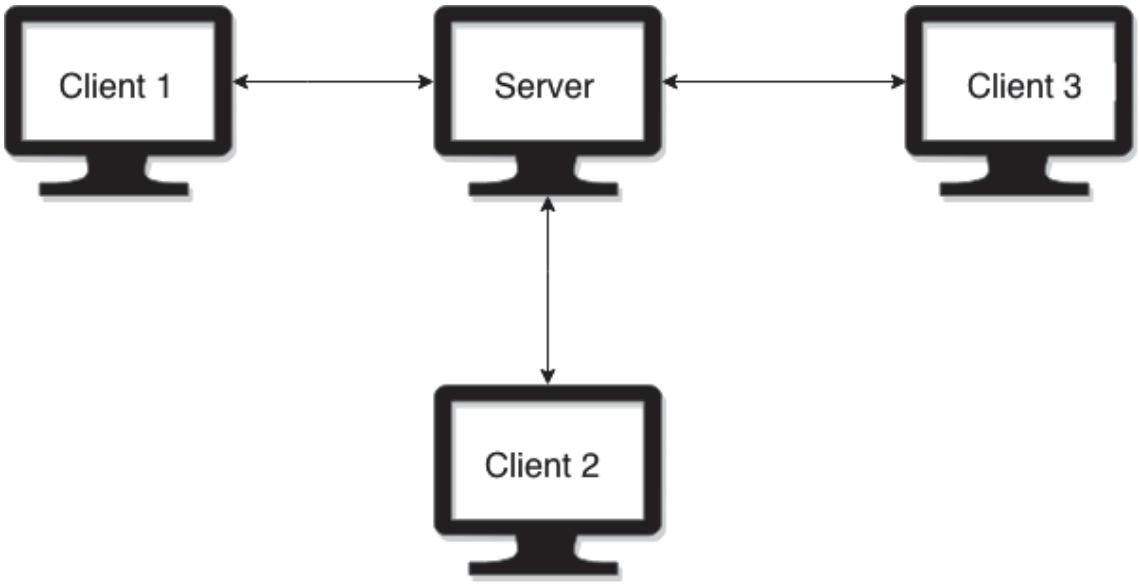
1 Úvod

Existuje několik her zabývající se obdobnou tématikou, ale většinou se zaobírají nějakou specifikací z reálného života. Například série „The Sims“, která se zaobírá stavbou bytu. Ale samotné chození do práce je už v pozadí. Dále existuje mód pro hru „Grand Theft Auto: San Andreas“ pro více hráčů. Můžete si zde zvolit z několika povolání, obchodovat s nemovitostmi a nebo jezdil kdejaké závody. U téhle hry jsem našel inspiraci, jelikož jsem u ní strávil několik desítek hodin. Jelikož je to jenom mód, tak to není úplně ono a najde se mnoho chyb, tak proč nevytvořit jednoduchou hru na téma simulace reálného života.

Komplexní simulace světa je rozsáhlá úloha, překračující rámec rozsahu bakalářské práce, proto bylo zadání omezeno jenom na některé funkcionality. Hlavním záměrem bylo simulovat několik základních profesí, mezi ně patří policista, záchranář, právník, řidič taxíku a nebo řidič autobusu a dále jednoduchou umělou inteligenci.

2 Architektura

Celá hra funguje na principu server-client. Na nějakém počítači běží server a hráči se na něj připojují skrz klienta. Server zpracovává úkony hráče a následně odešle aktualizované informace ostatním hráčům. Umělá inteligence je zakomponovaná na serveru. Jelikož je na straně serveru, nemusí získávat informace o světu speciální cestou, získává si data přímo. Zpracovává se při každé aktualizaci serveru Komunikace mezi serverem a klientem probíhá pomocí protokolu TCP/IP. Vykreslování se používá 2D grafika, z horního pohledu.



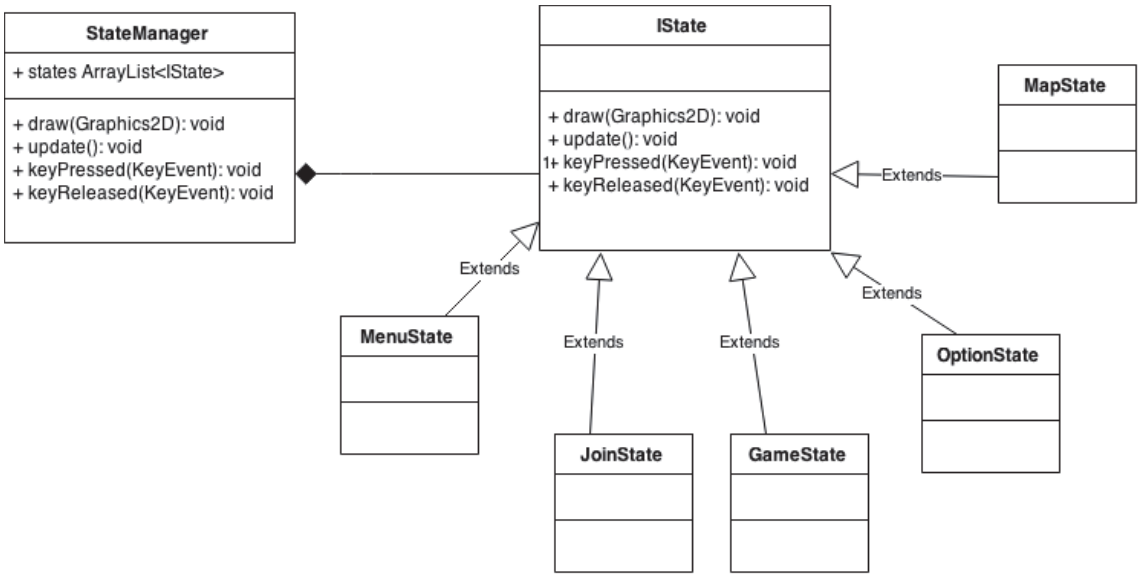
Obrázek 1: Schéma Server-Client

3 Návrh a implementace

Na úplném začátku bylo nejprve vytvořit vykreslování a pohybování hráče. Po vytvoření základního jádra hry, přišel na řadu server. Část jádra hry a to pohybování hráče je přesunuto na server. U vytváření serveru je naimplemetována komunikace mezi klientem a serverem a ošetření některých pádů hry během hry. Následně se vytváří už samotná simulace světa. To jsou budovy, zprávy mezi uživateli a obsluhu speciálních zpráv, jako například zavolání taxíku. Další řešenou částí jsou povolání a implementace ke každému povolání umělou inteligenci. Dále implementace samotné umělé inteligence, její náhodné chození, řízení auta a vyhledání nejkratší cesty k cíli. Poslední částí je vytvoření jednoduchého editoru mapy.

3.1 Přechod mezi stavy

Jak se pohybovat mezi jednotlivými stavy hry existuje mnoho technik. Nebo-li jak se pohybovat mezi menu, samotnou hru, nastavení a podobně.



Obrázek 2: Třídní diagram - Stavový správce

3.1.1 Stavový správce

Pro přepínání mezi jednotlivými stavy hry byl vytvořen takzvaný „State Manager“[1]. Tento manager se stará o přepínání mezi jednotlivými stavy. V kolekci má uložené nynější a předešlé stavy. Každý stav má ukazatel na tento manager, pomocí kterého mění aktuální stav. Manager má uložený aktuální index na kolekci. Když se zavolá metoda update, draw a nebo jiné metody, vytáhne si z kolekce aktuální stav a na něj zavolá příslušná metoda.

3.1.2 State interface

Pro přepínání mezi stavy byl použit interface který pak jednotlivé stavy implementují. Manager tento interface využívá pro generickou kolekci.

3.1.3 Jednotlivé stavy

Jednotlivé stavy se starají o vykreslování obrazu, obsluhu tlačítek a o aktualizaci údajů. Jelikož samotná hra se vypočítává na straně serveru, bude vysvětleno později, tak se zde skoro žádné aktualizace neprovádějí.

Následující kapitoly se zabývají řešením jednotlivých problematik při vývoji.

- *MenuState* - Hlavní menu
- *JoinState* - Pro připojování na server
- *OptionState* - Nastavení tlačítek, jazyka a jména
- *GameState* - Obsluha hry
- *MapState* - Zobrazení celé mapy, určení místa k navigování

3.2 Server

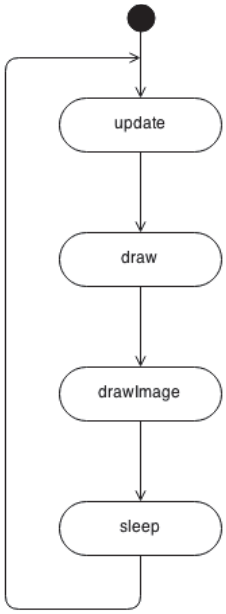
Server se nijak neliší od klienta 4, kromě vykreslování, které není zapotřebí na straně serveru. Samotný server se stará jenom o obsluhu klientů a vypočítávání jejich nových souřadnic a dalších dat, které následně odešle klientům. Klienti tyto údaje převedou do grafické podoby. Veškeré výpočty byly převedeny na server z toho důvodu, že klienty by si mohli hráči upravit pro svůj prospěch, třeba zvýšit rychlost pohybu.

4 Vykreslování scény

Pro vykreslování a aktualizací scény byl vybrán algoritmus 3. Základem tohoto algoritmu je vypočítání doby potřebné pro vykreslení scény a aktualizací dat. Aby tento čas byl dostačující, bylo použito 40 FPS. To znamená, že bude vykresleno 40 snímků za sekundu. Pomocí jednoduchého vzorečku vypočítáme potřebný čas pro jeden snímek. $t = 1000/FPS$. Za dosazení FPS nám výjde 25ms. Abychom tohohle dosáhli, uložíme si čas začátku vykreslování a aktualizaci údajů. Až všechny úkony skončí zjistíme si čas trvání, který použijeme pro výpočet, jak dlouho bude vlákno v nečinnosti.

Na začátku se zavolá metoda update, která aktualizuje některá data. Potom se vykresluje scéna pomocí takzvané metody „Double Buffering“[4] [**?**]. Jde o to, že scéna se prvně vykreslí do obrázku a až pak se tenhle obrázek vykreslí na samotnou scénu. Tímhle předejdeme nějakým blikáním obrazu. Do samotného obrázku se všechno vykreslí po zavolání metody `draw` a následně se zavolá metoda `drawImage`, která obrázek vykreslí na scénu. Následně se zavolá metoda na aktuální vlákno `sleep`, které předáme parametr, jak dlouho má být v nečinnosti. Klient jenom odesílá události o stisknutých tlačítek. Pro vykreslování se využívá speciální bitmapa `VolatileImage`. Jde o bitmapu, která je uložena v VRAM grafické kartě. Práce s ní je, ale daleko těžší, jelikož `VolatileImage` může mít několik stavů. První stav je „IMAGE_OK“, který indikuje, že se obsah nezměnil. Druhým stavem je „IMAGE_INCOMPATIBLE“, který indikuje, že se pravděpodobně změnil grafický režim. A třetím stavem je „IMAGE_RESTORED“, který indikuje, že je nutno překreslit bitmapu. Stav zjistíme zavoláním metody `validate`. Před vykreslením do bitmapy se vždycky musí zkontrolovat stav bitmapy.

Pro vykreslování se používá knihovna Swing. Jelikož hra nevyžaduje vysoký výkon pro vykreslování, stačilo využívat tuhle knihovnu, která ale není určena primárně pro hry, ale stačí pro jednoduché hry. V budoucnu by se mohla využívat knihovna LWJGL, která zpřístupňuje využívání OpenGL.



Obrázek 3: Digram překreslování scény

4.1 Vykreslování mapy

Mapa je postavena na takzvaném „Tile engine“. Nebo-li mapa je rozdělena na jednodušší obrázky, které se následně poskládají. V tohle případě je mapa uložena jako dvourozměrné pole ve kterém jsou uchovány číslice. Každá číslice reprezentuje typ jednotlivého obrázků. Hlavní výhodou je menší náročnost na velikost mapy v paměti, ale větší náročnost na vykreslování mapy, jelikož se mapa při každém vykreslení musí poskládat. Dá se tomu zamezit, pokud se mapa často nemění, vykreslit si část mapy do speciálního obrázku a pak následně tenhle obrázek vykreslovat.

Mapa se vykresluje s pohybem hráče. Jelikož velikost vykreslovacího okna se může v důsledků možnosti mít hru v celo-obrazovém módu měnit, bylo obtížnější najít vzorec, který bude dynamicky a správě vykreslovat mapu. Mapa se nachází v několika stavech jak se bude vykreslovat:

1. hráč se nachází v levém horním rohu, mapa se nepohybuje, dokud hráč není uprostřed okna
2. hráč se nachází v pravém dolním rohu, mapa se nepohybuje, pokud hráč není uprostřed okna
3. hráč se nachází mezi levém horním rohu a pravém dolním rohu, mapa se pohybuje s pohybem hráče

Každý stav se dále vztahuje na pozici X a pozici Y, ale vzorečky jsou podobné.

```
//Prvni pripad - X
if (x <= ((int)xE)) {
    bXG = 0;
    eXG = Settings.WIDTH/48;
    offX = 0;
}
//Prvni pripad - Y
if (y <= ((int)yE)) {
    bYG = 0;
    eYG = Settings.HEIGHT/48;
    offY = 0;
}
//Druhy pripad - X
if (x > ((int)xE) && x < ((world.getSizeGridX()*64)-( (Settings.WIDTH/48.0) - ( Math.floor( (
      Settings.WIDTH/48.0) / 2.0 ) ) )*64)) {
    bXG = xG-((int)Math.floor((Settings.WIDTH/48.0)/2.0));
    eXG = xG+((int)Math.ceil((Settings.WIDTH/48.0)/2.0))+1;
    offX = ((x-(xG*64))*3/4);
}
//Druhy pripad - Y
if ( (y > ((int)yE)) && (y < ((world.getSizeGridY()*64)-( (Settings.HEIGHT/48.0) - ( Math.floor( (
      Settings.HEIGHT/48.0) / 2.0 ) ) )*64) )) {
    bYG = yG-((int)Math.floor((Settings.HEIGHT/48.0)/2.0));
    eYG = yG+((int)Math.ceil((Settings.HEIGHT/48.0)/2.0))+1;
    offY = ((y-(yG*64))*3/4);
}
// Treti pripad - X
if ( y >= ((world.getSizeGridY()*64)-(( (Settings.HEIGHT/48.0) - ( Math.floor( (Settings.HEIGHT
      /48.0) / 2.0 ) ) )*64))) {
    bYG = world.getSizeGridY()-(int)Math.ceil(Settings.HEIGHT/48.0);
    eYG = world.getSizeGridY()-1;
    offY = (int)(Math.ceil(Settings.HEIGHT/48.0)*48-(Settings.HEIGHT/48.0)*48);
}
// Treti pripad - Y
if ( x >= ((world.getSizeGridX()*64)-((Settings.WIDTH/48.0) - ( Math.floor( (Settings.WIDTH
      /48.0) / 2.0 ) ) )*64)) {
    bXG = world.getSizeGridX()-(int)Math.ceil(Settings.WIDTH/48.0);
    eXG = world.getSizeGridX()-1;
    offX = (int)(Math.ceil(Settings.WIDTH/48.0)*48-(Settings.WIDTH/48.0)*48);
}
g.drawImage(grassImage, (48*(i-bXG))-offX, (48*(z-bYG))-offY, 48,48,null);
```

Výpis 1: Vykreslení jednotlivého obrázku

Vysvětlení jednotlivých proměnných:

- `bXG` - začátek pozice X v mapě
- `eXG` - konec pozice X v mapě
- `offX` - o kolik bodů v ose X se celá mapa posune
- `bYG` - začátek pozice Y v mapě

- `eYG` - konec pozice Y v mapě
- `offY` - o kolik bodů v ose Y se celá mapa posune
- `g` - instance grafiky, do které se vykresluje obrázek
- `grassImage` - obrázek který se bude vykreslovat
- `x` - přesná pozice hráče ve světě
- `y` - přesná pozice hráče ve světě
- `xG` - pozice v poli mapy
- `yG` - pozice v poli mapy

Pro první stav ve kterém se může mapa naházet bylo jednoduché vytvořit mapu, jelikož se mapa vykresluje od začátku, nebylo potřeba vypočítávat `offX` a `offY`.Bylo potřeba jenom vypočítat kolik se vleze jednotlivých obrázků do okna. To samé bylo i u druhé osy.

V druhém stavu se už musela vypočítat proměnná `offX` a `offY`, pro které byl vytvořen následující vzorec:

$$offX = ((x - (xG * 64)) * 3/4);$$

Pro třetí stav mapy byl vytvořen vzorec:

$$offY = (int)(Math.ceil(Settings.HEIGHT/48.0)*48-(Settings.HEIGHT/48.0)*48);$$

Matematická funkce ceil zaokrouhlí číslo vždy nahoru.

4.2 Vykreslování hráče

Pro vykreslování hráče a nebo auta se využívají podobné rovnice jak u vykreslování mapy (viz. kap. 4.1).

Při otáčení hráče se využívá jenom jeden obrázek, který se před vykreslením otočí na požadovaný úhel. K tomu se využívá třída `AffineTranform` a `AffineTranformOp`. Do první zmíněné třídy vloží údaje o otočení. Prvním parametrem je úhel otočení, druhým a třetím je bod, podle kterého se obrázek bude otáčet. Do druhé třídy se vloží instance první třídy a typ otáčení. Pro otočení obrázku využijeme u třídy `AffineTranformOp` metodu filter, které předáme zdrojový obrázek. Metoda vrátí výsledný otočený obrázek a nebo do druhého parametru vloží výsledný obrázek.

```
AffineTransform tx = AffineTransform.getRotateInstance(Math.toRadians(dir), locationX, locationY);
AffineTransformOp op = new AffineTransformOp(tx, AffineTransformOp.TYPE_BILINEAR);
pG.drawImage(op.filter(bImage, null), 0, 0,null);
```

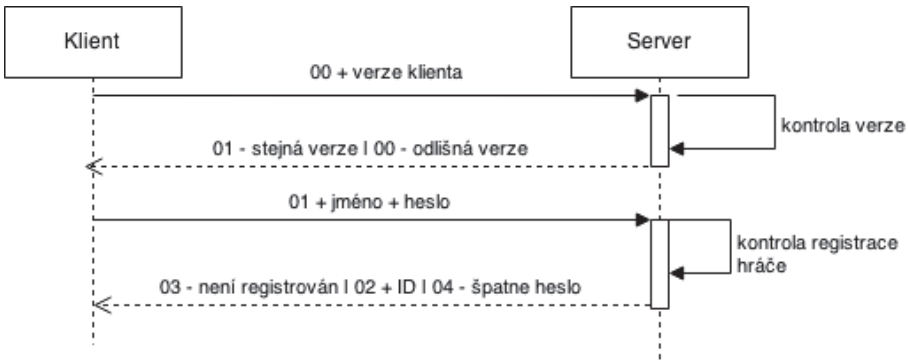
Výpis 2: Otáčení obrázku

5 Komunikace

Pro komunikaci se využívá třída *Socket*, které stačí zadat jenom v případě serveru jenom port na kterém bude poslouchat a v případně klienta i IP adresu na kterou se má připojit. Po úspěšném propojení se ze *Socket*u vytáhnout streamy, pomocí kterých se dále komunikuje. Pro komunikaci mezi klientem a serverem se používají dva druhy komunikace. Pro jednoduché příkazy se využívá textová komunikace a pro posílání složitějších dat se posílají objekty.

5.1 Textová komunikace

Pro jednoduché příkazy se odesílá prostý text ve kterém jsou vloženy jednoduché data. První dvě čísla v textu označují kód zprávy, které se pak mohou dále dělit do dalších typů zpráv. Pro dělení dat se využívá znak „;“. Pro jednoduché rozdělení dat se využívá metoda split. Správným příkladem může být počáteční inicializace, kdy se ověřuje verze klienta, jestli je uživatel registrován a podobně. Na začátku klient odešle kód ve tvaru 01 plus verzi. Server následně porovná verze a jestli se shodují, odešle kód 01 a pokud se neshodují 00. Pokud se neshodují klient uzavře spojení. Jestliže se shodují, odešle kód 01 plus uživatelské jméno a heslo. Server zkontroluje jestli dané uživatelské jméno existuje a případně jestli je heslo shodné. Při neexistujícím uživatele se odešle kód 03, při špatném heslu se odešle kód 04 a pokud vše proběhlo úspěšně odešle se 02 s dvě identifikátory, první je chat a další je pro informace, které je vztahují k uživatelovi, které se musí zachovat i po vypnutí serveru. Dále se při inicializace využívá objektová komunikace pro odesílání mapy a dalších dat.



Obrázek 4: Sekvenční diagram - počáteční inicializace

5.2 Objektová komunikace

Pro složitější odesílání dat se odesílají objekty, jelikož v textové podobně by to bylo složitější dešifrovat a dále kvůli zvýšené datové velikosti oproti objektu. Pomocí objektů se odesílají data o poloze, stavu zdraví a podobně, které se odesílají v jediném objektu.

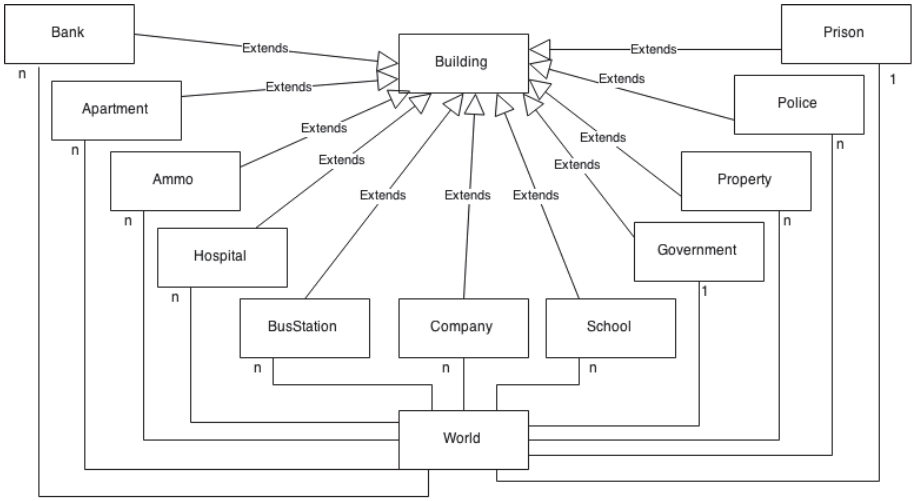
6 Simulační jádro

Veškeré informace o aktuálním světě se nacházejí ve třídě `World`. U klienta a u serveru se třída jmenuje stejně. Obsahuje veškeré algoritmy pro výpočtu detekce kolize, výpočet nové pozice hráče. Veškerá data se ukládají pomocí serializace třídy `DataMap`.

6.1 Budovy

Všechny budovy dědí ze třídy `Building` která obsahuje informace o poloze, směru a identifikátoru. Většina budov může být ve světě kolik chce, kromě budov „Government“ a „Prison“. Seznam a popis jednotlivých budov:

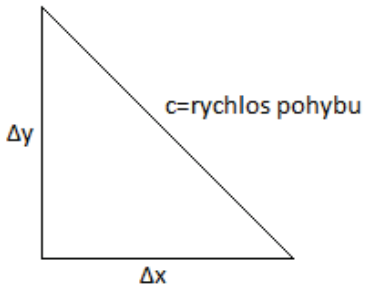
- Prison - Uvěznění hráčů, kteří byly zatknuti policistou
- Government - Hlasování o daních, kandidovat na politika
- Bank - Vkládání a vybírání peněz z banky
- Apartment - Ukládání auta a vkládání a vybírání peněz
- Ammo - Nákup zbraní a nábojů
- Hospital - Vyléčit se a vzít práci zdravotníka
- BusStation - Jednotlivé autobusové zastávky pro autobusového řidiče
- Company - Hlavní budovat gangu a nebo firmy
- School - Nákup licencí pro zbraň, auta
- Property - Nákup nemovitostí na určitou dobu
- Police - Vzít práci policisty



Obrázek 5: Třídní diagram - Budovy

6.2 Výpočet dalšího pohybu

Pro výpočet nové pozice hráče se využívají goniometrické funkce. Přesněji sinus a cosinus, které najdeme ve třídě `Math` jako statické metody. Goniometrické funkce v javě pracují jenom s radiány, proto se musí úhel, kam směřuje hráč, převést ze stupňů na radiány.



Obrázek 6: Výpočet nové souřadnice hráče

Abychom zjistili o kolik se hráč posunul ve směru y, využijeme funkci sinus. Vzorec byl upraven ze vzorce $sin(uhel) = a/c$. Za přeponu se dosadí rychlost hráče, která je 3. Ve směru x použijeme funkci cosinus a vzorec byl upraven ze vzorce $cos(uhel) = b/c$. Pro výpočet nové souřadnice auta se využívají stejné rovnice, kromě rychlosti.

```
ys = (int) (speed * Math.sin(Math.toRadians(myDir+270)));
xs = (int) (speed * Math.cos(Math.toRadians(myDir+270)));
```

Výpis 3: Výpočet nové souřadnice hráče

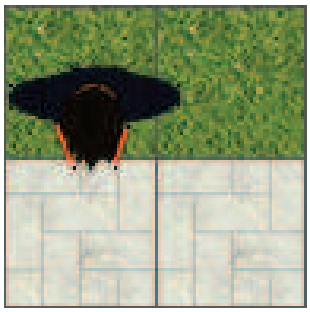
6.3 Detekce kolize

6.3.1 Kolize hráče

Pro detekci kolizí se zdmi bylo na postavě vytvořeno několik bodů. Jelikož postava nemá jednoduchý tvar, bylo vytvořeno deset přímek ze kterých se vytvoří přímky. Přímky se následně porovnávají z jednotlivými body na mapě. Konstantní proměnné které se využívají v detekci, byly vypočítány z obrázku hráče. Jedny určují vzdálenost od středu a další jsou úhly od rovnoběžky s vertikální přímkou. Nejprve se vypočítají dva body a pak se vytvoří objekt `Line2D`, který reprezentuje přímku mezi dva body. Třída má metodu `intersects`, která porovná dvě přímky jestli se protínají. Druhá přímka je jedna ze stran bodu mapy. Pokud se protínají, byla detekována kolize a vypočteny nové souřadnice se nevyužijí. Algoritmus pro detekci kolize jedné přímky je vypsán níže (viz. kód 4).

Obrázek 7: Body pro detekci kolize hráče se zdí

Na obrázku číslo 13, je názorné zobrazené detekce kolize. Červené přímky znázorňují detekci kolizi se zdí.



Obrázek 8: Přímky které detekovaly kolizi

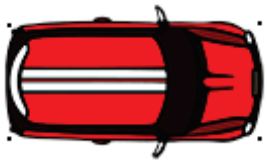
```
ys1 = (int) (a * Math.sin(Math.toRadians(dir + al)));
xs1 = (int) (a* Math.cos(Math.toRadians(dir + al)));
ys2 = (int) (b * Math.sin(Math.toRadians(dir + be)));
xs2 = (int) (b * Math.cos(Math.toRadians(dir + be)));
line = new Line2D.Double(20+xs1+x, 20+ys1+y,20+xs2+x, 20+ys2+y);
```

```
if (line.intersects(rec)) {
    returnN = true;
}
```

Výpis 4: Detekce kolize hrače se zdí

6.3.2 Kolize auta

U kolize auta se využívá stejný postup jako u hráče(6.3.1). Jelikož auto je jednoduššího tvaru, byly vytvořeny 4 body.



Obrázek 9: Body pro detekci kolize auta se zdí

6.3.3 Kolize střel

Střela má tvar přímky, ale pro detekci se používá její střed a může se srazit s hráčem, autem a se zdí. Pro pohyb střel se využívají stejné rovnice jako u hráče(viz. kapitola 6.3.1) a nebo auta(viz. kapitola 6.3.2). Každá střela má jinou účinnost. Uchovává si informaci ze které zbraně byla vystřelena.

Pro jednoduchou kolizi se hráčem, byl vytvořen ze hráče jednodušší tvar, obdélník. Využívá se zde třída `Rectangle`, která obsahuje metodu `contains` která porovná jestli zadaný bod je uvnitř obdélníku. Pokud střela zasáhla hráče, tak se mu sníží jeho zdraví. O kolik se sníží jeho zdraví závisí na typu střely. Pokud zdraví hráče je menší než 0, tak se mrtvý hráč objeví u jedné ze všech nemocnic. Střela má dále informaci, jestli daná střela vyšla od nějakého gangstera. Pokud ano a mrtvý hráč je taky z gangu, tak se hráči který střelu vystřelil nezvýší jeho hledanost. Pokud tohle neplatí zvýší se hledanost(takzvaná hvězdička) o jedna.

```
Rectangle rec = new Rectangle((int)ct.getX()+14,(int)ct.getY()+14,36,36);
if (rec.contains(bullet.getX()+32, bullet.getY()+32)) {
}
```

Výpis 5: Detekce kolize střely s hráčem

Jelikož auto má tvar obdélníku využívá se zde taky třída `Rectangle` a metoda `constains`, která porovná jestli daný bod je uvnitř obdélníku. Každé auto má životnost která se snižuje při střelení do něj. I zde závisí na typu střely. Při výbuchu auta se auto objeví na

jeho původní pozici. Jelikož auto se může uložit u apartmánu, tak při uložení se původní pozice změní a objeví se u apartmánu.

U detekce kolize se zdí se již nepoužívá metoda `constains` ani střed střely. U střely se vypočítají čtyři body, ze kterých se následně porovná, jestli daný bod je ve zdi. Bod se vypočítá jednoduchým vydělím 64, jelikož každá část mapy má velikost 64 pixelů.

```
tmpX = xB + 25;
tmpY = yB + 25;
if (detCollGrid(mapGrid[(int) tmpX/64][(int)  tmpY/64])) {
    return true;
}
tmpX = xB + 25;
tmpY = yB + 39;
if (detCollGrid(mapGrid[(int) tmpX/64][(int)  tmpY/64])) {
    return true;
}
tmpX = xB + 39;
tmpY = yB + 25;
if (detCollGrid(mapGrid[(int) tmpX/64][(int)  tmpY/64])) {
    return true;
}
tmpX = xB + 39;
tmpY = yB + 39;
if (detCollGrid(mapGrid[(int) tmpX/64][(int)  tmpY/64])) {
    return true;
}
return false;
```

Výpis 6: Detekce kolize střely se zdí

6.4 Zprávy

Třída `TextChat` obsahuje:

- Samotný text
- Typ textu
- Od koho byl text napsán, jméno a identifikátor
- Pro kterého hráč je text určen, jméno a identifikátor
- Pro kterou firmu/gang je text určen, identifikátor

Jednotlivé typy textu:

- 0 - Obyčejný text
- 1 - Hlasovací text, hlasování na nového politika
- 3 - Zobrazení příkazů

- 4 - Vlastník firmy a seznam zaměstnanců
- 5 - Přivolání taxíka

Jednotlivé texty chatu se uchovávají v `ArrayList`. Jednotlivému hráči se odesílají posledních 10 zpráv. Hráči se odesílá ta zpráva, která je určena pro všechny, pro něj(podle identifikátoru) a nebo ta, která je určena pro jeho firmu.

Chat se vykresluje pomocí grafiky. Nejprve se vypočítá délka textu a následně se vykreslí text do `BufferedImage`. Počet řádků se vypočítá z jednoduché rovnice, kdy se sečtou délky od koho je text a samotného textu a vydělí se to číslem 260. Je to délka jednoho řádku. V listu jsou všechny vykreslené texty v `BufferedImage`. Potom se prochází jednotlivé `BufferedImage` a vykreslí se to do speciálního obrázku, který se následně vykresluje na scénu. Jelikož teto část kódu je trošku náročná, vypočítávání se děje, jenom když se změní chat.

```
g.getFontMetrics().getStringBounds(textChat.getText(), g).getWidth();
```

Výpis 7: Výpočet grafické délky textu

6.5 Povolání

Každý hráč má proměnnou, která uchovává typ povoláním, kterým je hráč. Každé povolání má i specifickou barvu. Každý hráč si uchovává v poli, pro jednotlivé povolání, počet dovedností. Identifikátory jednotlivých prací:

- -1 - hráč je bez práce
- 0 - Policista
- 1 - Zdravotník
- 2 - Politik
- 3 - Gangster
- 4 - Řidič taxíku
- 5 - Řidič autobusu
- 6 - Právník

6.5.1 Policista

Pro práci policisty musí vlastnit hráč licenci na zbraň, kterou koupí v jedné ze škol. Jenom na policejní stanici se může hráč stát policistou. Vydělává tím, že zatkne hráče, který má víc jak jednu hvězdu (hledanost, dále jenom hvězda). Pro zatýkání se používá tlačítko „Práce“. Výchozím tlačítkem je „Q“, které se dá změnit v nastavení tlačítek. Po zatknutí hráče se hráč objeví ve vězení a policistovi se zvýší dovednost. Aby hráče zatknul, musí být být velmi blízko něj, v průměru 32 bodů. Zatknutému hráči se dál odeberou všechny zbraně. Na délce pobytu ve vězení závisí na počtu hvězd které má.

6.5.2 Zdravotník

Pro práci zdravotníka musí hráč vlastnit zdravotnický průkaz, kterou koupí v jedné ze škol. Jenom v nemocnici se může hráč stát zdravotníkem. Vydělává tím, že uzdraví hráče, který ale musí mít míň jak 100% zdraví. Pro uzdravování se používá tlačítko „Práce“, které se zdá změnit v nastavení. Po uzdravení hráče dostane zdravotník peníze a zvýší se mu dovednost, která zvyšuje vydělané peníze. Aby hráče uzdravil musí být vedle něj velmi blízko, v průměru 32 bodů.

6.5.3 Politik

Aby se hráč stal politikem, musí být zvolen ve volbách do kterých se může přihlásit v budově „Vláda“. Je zvolen jenom na dobu 5 týdnů. Pokud politiků není 5, je volen do té doby než je zvoleno 5 politiků a dále pokud nikdo není zvolen v některých následujících volbách. Pokud je zvolen, hráč který je nejdéle zvolen, již už nebude politikem. Politik má právo měnit daně. Pro to aby byly daně schváleny musí hlasovat víc jak 3 politici. Pro změnu daní se používá budova „Vláda“. Pro hlasování na změnu daní se hlasuje v budově „Vláda“.

Pří každé aktualizaci světa se kontroluje čas data voleb a následně čas ukončení voleb. Jelikož v nejnovější verzi Javy se již nesmí už využívat částečně třída `Date`, tak pro uložení konci návrhu nových daní se využívá třída `Calendar`.

6.5.4 Gangster

Aby se hráč stal gansterem, musí ve světe existovat alespoň jeden gang. Pro to by do gangu vstoupil, musí hlava gangu mu poslat pozvánku. Ve světe jsou definovány zony, které mohou gangy zabírat. Pokud je hráč v gangu, všechny zóny se mu objeví na mapě, kterou najde v menu hry. Jsou odděleny barvou. Každé zabírání trvá minimálně 3 minuty. Pokud hráč z gangu, který zónu vlastní, vejde do zóny, tak se odpočítávání zastaví. Aby odpočítávání pokračovalo, musí hráč z konkurečního gangu zabít. Pokud v zabírané zóně není ani jeden hráč, odpočítávání přestane a zabírání se vypne. Po úspěšném zabrání zóny dostanou ti hráči, kteří jsou v zóně dovednosti a peníze.

6.5.5 Řidič taxíku

Pro práci jako řidič taxíku potřebuje hráč licenci na auto. Jsou dva způsoby jak se stát řidičem taxíku. První je nasednout do auta a stisknout tlačítko „Práce“. Druhým je se přidat do firmy. Pro zavolání taxíku se píše příkaz do chatu. Následně se na straně serveru, vyhledá nějaký taxík, který není plný, každý taxikář si uchovává proměnnou `taxiEngaged`, která značí jestli už vozí nějakého zákazníka, a pošle se mu pozice hráče, který na něj čeká. Cena taxíku se odvíjí od času ztráveného v taxíku. Po vystoupení zákazníka se řidiči zvýší dovednost. Aby se zákazníkovi měnila jeho pozice s taxíkem, tak si každé auto uchovává seznam spolujezdců. Při aktualizace pozice řidiči se zkontroluje jestli v listu spolujezdců někdo není. Pokud ano, aktualizují se mu souřadnice.

6.5.6 Řidič autobusu

Pro prvotní vydělávání hráče nepotřebuje pro řízení autobusu licenci na auto. Hráč se může stát řidičem autobusu pokud nastoupí do autobusu a zmáčkne tlačítko „Práce“. Pokud ve světě existují autobusové zastávky, hráč jezdí dokola a mezi nimi. Pro to aby si vydělával nemusí mít zákazníky. Každou projetou zastávkou se mu zvýší dovednost a dostane peníze. Pro aktualizaci hráčů, kteří jsou v autobuse, se využívá list spolujezdců v autě. Při zastavení na zastávce se auto zablokuje a hráči mohou nastupovat do autobusu. Zastavení trvá 5 sekund. Cesta na další zastávku se mu ukáže na mini mapě. Využívá se zde algoritmus pro vyhledání nejkratší cesty.

6.5.7 Právník

Pro práci jako právník musí mít hráč právnický průkaz, který se dá zakoupit ve škole. Aby mohl být hráč právníkem musí být ve firmě. Právník snižuje hledanost hráčům. Pokud zmáčkne právník tlačítko „Práce“, hledanému hráči se sníží hledanost a právníkovi se zvýší dovednost a dostane peníze.

```
taxiDriver.getArrayOfSkills()[4] = taxiDriver.getArrayOfSkills()[4] + 1;
int money = taxiDriver.getMoney();
if (taxiDriver.getIdOfCompany() == -1) {
    money += (1 + (taxiDriver.getArrayOfSkills()[4]) * 0.05)*700;
}
else {
    Company company = getMapOfCompanies().get(taxiDriver.getIdOfCompany());
    money += (1 + (taxiDriver.getArrayOfSkills()[5]) * 0.05)*(700*company.getMultiplyCash());
}
taxiDriver.setTaxiEngaged(false);
Calendar now = Calendar.getInstance();
long timenow = now.getTimeInMillis();
long remain = (timenow - taxiDriver.getStartMesTaxi())/1000;
pass.setMoney(pass.getMoney() - safeLongToInt(remain));

taxiDriver.setMoney(money);
if (pass instanceof ClientThread) {
    ((ClientThread) pass).updatePlData();
}
if (taxiDriver instanceof ClientThread) {
    ((ClientThread) taxiDriver).updatePlData();
}
```

Výpis 8: Příklad pracovní akce

6.6 Firmy a gangy

Hráči se mohou sdružovat v podobě firem a nebo gangů. Pokud je hráč ve firmě/gangu zvyšuje se mu výdělek. Založení firmy a nebo gangu může buď zakoupení již už postavené budovy a nebo zakoupení pozemku a následně postavení budovy při které může

zvolit úroveň budovy. Úroveň budovy hraje roli, kolik může být maximálně zaměstnanců ve firmě.

6.7 Aktualizace objektů v okolí

Při změně pozice hráče se následně hned aktualizují všechny objekty v okolí.

6.7.1 Aktualizace hráčů

Pro snížení počtu odeslaných informací přes internet, se odesílají hráči jenom ve vymezeném okruhu okolo hráče. Každý hráč si uchovává list hráčů, kteří jsou blízko něj. U některých povolání se prochází list hráčů a ti kteří jsou vedle něj se provede nějaká akce(například ho vyléčí). Proto se využívá list hráčů, kteří jsou v okruhu vedle něj, jelikož procházení všech hráčů zabere nějaký ten čas.

Pro aktualizace hráčů v okolí se volá rekurzivně metoda `getListOfPlayerInCoor`. Hráč kterému se již odeslali hráči, které má ve svém okolí, se přidá jeho identifikátor do listu, ve kterém jsou identifikátory těch hráčů, kterým byly poslány nové informace. U každého hráče se prochází všechni hráči ve světě a jestli jsou v určitém poloměru od hráče, přidají se jeho informace o poloze, zdraví a hledanosti do listu, který se potom odešle hráči. Po přidání informací se zjistí, jestli daný hráč se už aktualizovat a pokud ne zavolá se znova ta samá metoda a dále aktualizují i auta které má ve svém okolí. Rekurze končí tehdy, pokud všichni v okolí byli už zaktualizováni. Jeden z nedostatků tohoto algoritmu je to, že se zaktualizuje i ten hráč, který není v dohledu hráče u kterého se tahle metoda volala. Data o hráčích se odesílají pomocí třídy `Enemy`.

```
for(IEntityThread ct : listOfClient ) {
    double pX = ct.getX(), pY = ct.getY();
    if (pX > (x − 10∗64) && pX < (x + 10∗64) && pY > (y − 10∗64) && pY < (y + 10∗64) && ct.
        getPlayerData().getId() != id) {
        Enemy enemy = new Enemy(pX, pY, ct.getID(), ct.getLogin(), ct.getDir(), ct.isInCar() ,ct.
            getHealth(), ct.getWanted());
        enemy.setIndexAnim(ct.getIndexOfAnim());
        enemy.setType(ct.getJobType());
        listOfPl .add(enemy);
        list2 .add(ct);
        if (!listOfUpdatedPl.contains(ct.getPlayerData().getId())) {
            ct.sendObject(getListOfPlayerInCoor(ct));
            ct.sendObject(getListOfCarInCoor(ct));
        }
    }
}
```

Výpis 9: Odesílání hráčů v okolí

6.7.2 Aktualizace aut

Pro odeslání aut v okolí se volá metoda `getListOfCarInCoor`. Stejně jako u aktualizací hráčů v okolí(viz. kapitola 6.7.1) se i zde odesílají jenom auta, které má hráč ve svém

okolí. I zde se využívá zmenšený list aut a to u nasedání do auta, které se provádí na straně klienta. Data o autech se odesílají zakódované v textu pomocí oddělovače „;“ jelikož informací o autu není mnoho. Jelikož třída `ArrayList` s texty se využívá vícekrát, tak se na začátek přidává indikátor o jaké data se jedná. V případně o autech se na začátek přidává číslo 1.

```
ArrayList<String> list = new ArrayList<String>();
int x = (int) me.getX();
int y = (int) me.getY();
list.add("1");
for(Car car : listOfCars) {
    double cX = car.getX(), cY = car.getY();
    if (cX > (x − 10∗64) && cX < (x + 10∗64) && cY > (y − 10∗64) && cY < (y + 10∗64)) {
        String carText = cX + ";" + cY + ";" + car.getDir() + ";" + car.getId() + ";" + car.getpID()
                + ";" + car.getHealth() + ";" + car.isLocked() + ";" + car.getType();
        list.add(carText);
    }
}
return list;
```

Výpis 10: Odesílání aut v okolí

7 Umělá inteligence

Součástí bakalářské práce bylo naimplementovat jednoduchou umělou inteligenci(Dále jenom UI), která dokáže ovládat všechny povolání. Pro vkládání UI se využívá editor mapy. Při vložení UI se jí nastaví pozice, jméno a typ povolání. Každá UI může být jenom jednoho povolání a nemůže jí změnit. Ukládá se do stejné třídy jako informace o mapě. Při zapnutí serveru se jednotlivá UI převedou na třídu `AIThread`, která implementuje rozhraní `IEntityThread`. Mají podobné proměnné jako obyčejný hráč. Dále mají své proměnné, které zajišťují jejich pohyb po světě. UI dokáže chodit i jezdit s autem.

7.1 Ovládání hráče

Pro vypočítání nové souřadnice hráče jsou využity stejné algoritmy jako u obyčejného hráče. S tím rozdílem, že využívá návratovou hodnotu těchto algoritmů. Je to proto, že pokud UI narazí do zdi, může podniknout kroky, které jí otočí a podobně. Pokud UI nemá povolání, pohybuje se po světě zcela náhodně. Využívá se k tomu třída `Random`, která poskytuje metodu `nextInt`, které jako parametr můžeme předat maximální číslo, které se může vygenerovat. Při každém aktualizování UI se kontroluje poslední změna směru. Pokud uběhla jistá doba od změny směru, tak se volá metoda, pokud nemá povolání, `randomMove`. Podle náhodné číslice se vykoná jiný úkon. Aby bylo chození UI co nejvěrohodnější, tak pro pohyb do předu byl udělená největší pravděpodobnost. Pokud by pro každý pohyb byla stejná pravděpodobnost, UI by se neustále točila skoro na tom samém místě. Všechny algoritmy pracují s proměnnými, které značí kam UI má směřovat. Stejné proměnné se využívají u obyčejného hráče, které značí, jaké tlačítka pro pohyb byly zmáčknuty. U jedné z náhodných čísel se kontroluje jestli neprobíhají volby a jestli už daná UI nehlasovala. Pokud nehlasovala, náhodně hlasuje pro jednoho z kandidátů.

```
Random r = new Random();
int  randomNumber = r.nextInt(10);
if (randomNumber == 0) {
    keyUp = false;
    keyDown = false;
    keyLeft  = true;
    keyRight = false;
     targetDir  = r.nextInt(359);
}
if (randomNumber == 1) {
    keyUp = false;
    keyDown = false;
    keyLeft  = false;
    keyRight = true;
     targetDir  = r.nextInt(359);
}
if (randomNumber > 1 && randomNumber < 6) {
    keyUp = true;
    keyDown = false;
    keyLeft  = false;
    keyRight = false;
     targetDir  = -1;
```

```
}
if (randomNumber >= 6) {
    keyUp = false;
    keyDown = false;
    keyLeft = false;
    keyRight = false;
    targetDir = -1;
    if (world.isElection() && !world.getListOfPlayersVote().contains(id))      {
        Set<Integer> list = world.getElection().keySet();
        if ( list.size() > 1) {
            int [] ids = new int[ list.size() ];
            int i = 0;
            for(int id : list ) {
                ids[i] = id;
                i++;
            }
            int rand = new Random().nextInt(list.size() );
            synchronized (World.class) {
                int count = world.getElection().get(ids[rand]);
                count++;
                world.getElection().put(ids[rand], count);
            }
        }

    }
}
```

Výpis 11: Náhodný pohyb umělé inteligence

7.1.1 Ovládání povolání

U ovládání povolání závisí na typu povolání. Je to proto, že každé povolání se ovládá jiným způsobem až na pár výjimek. Povolání zdravotník, policista a právník využívají stejný postup. Vyhledají ve svém okolí hráče a následně za ním jdou. Odlišné je to jakého hráče vyhledají. Zde se využívá zmenšeného listu hráčů ve svém okolí ze kterého se vyhledává hráč pro akci povolání. Indikací, že UI jde k hráčovi je to, že v listu `listOfPath` je víc jak jedna položka(zatáčka). Při nalezení hráče se vypočítá trasa(viz. kapitola Nalezení cesty 7.3) a pak se pohybuje podle vypočítané trasy. V listu cesty jsou jenom zatáčky kde se má UI otočit a dále konec. Pokud dosáhne cíle využívají se metody pro pracovní akci, které jsou ve třídě `World`. Stejné metody se využívají i u obyčejného hráče.

Jiného typu povolání jako je řidič taxíku se UI chová jinak, je to proto, že řídí auto, které se chová jinak. Druhý typ jiného chování je gangster. Kontroluje jestli jeho gang nezabírá nějakou zónu a nebo někdo jiný zabírá jeho zónu, tak se přesune na danou zónu. Všechny povolání využívají třídu `Finder` pro vyhledání cesty až na pár uprav, kvůli pohybu (viz. kapitola Nalezení cesty 7.3).

7.2 Ovládání auta

Ovládání auta se využívá jenom v případně, že je UI povoláním řidič taxíku a nebo autobusu. I zde se využívá nalezení cesty pomocí třídy `Finder`, ale nalezená cesta je dále optimalizovaná pro cestu autem(viz. dále Nalezení cesty 7.3.

Po nalezení cesty se nejprve kontroluje jestli dané auto je otočené směrem k první zatáčce. Pokud není, UI se otáčí do té doby než jej má před sebou. Využívá se i zde indikace nárazu auta do zdi. Pokud auto při počátečním otáčení narazí, UI začne jet dozadu. Po správném otočení auta se začne jezdit po vyhledané cestě. Při každém aktualizování UI se vypočítává brzdná dráha auta pomocí rovnice:

$$breakDist = ((currSpeed - 3.0)/breaksSpeed) * ((currSpeed + 3)/2.0) + 10;$$

Vysvětlení jednotlivých proměnných:

- `breakDist` - Dráha potřebná pro brzdění před zatáčkou
- `currSpeed` - Aktuální rychlost vozidla
- `breaksSpeed` - O kolik se rychlost snižuje při brzdění

Od aktuální rychlosti vozidla se odečítá 3, jelikož v zatáčce auto pojede rychlostí 3. Při dosažené brzdné dráhy, auto začne brzdit a přestane se vypočítávat brzdná dráha. Rychlost auta na 3 je udržována uměle. To znamená, že se nastavuje manuálně pro jednoduchost. Dále se kontroluje vzdálenost kdy má auto začít se otáčet. Tato vzdálenost je určena podle toho na jakou stranu auto se otáčí. Pro náročnost vypočítávání směru zatáčení, se tohle prování už při nalezení cesty. Při otáčení se pak prování jestli auto je natočené přímo na další bod. Pokud ono, první bod se vymaže z listu. Při dosažení cíle auto brzdí a následně se vymaže celý list. Který indikuje, že auto dojelo do cíle.

```
double distX2 = (nextPoint.getX()*64+32) - (x+75);
double distY2 = (nextPoint.getY()*64+32) - (y+75);
double xs = 0, ys = 0;
double dist2 = Math.sqrt(distX2 * distX2 + distY2 * distY2);
ys = (int) (dist2 * Math.sin(Math.toRadians(dir)));
xs = (int) (dist2 * Math.cos(Math.toRadians(dir)));
xs += x + 75;
ys += y + 75;
if (nextPoint.getX()*64 < xs && nextPoint.getX()*64 + 64 > xs && nextPoint.getY()*64 < ys &&
    nextPoint.getY()*64+64 > ys) {
  listOfPath.remove(0);
  keyRight = false;
  keyLeft = false;
  keyUp = true;
  isCalBreaking = false;
}
```

Výpis 12: Kontrola směru jízdy

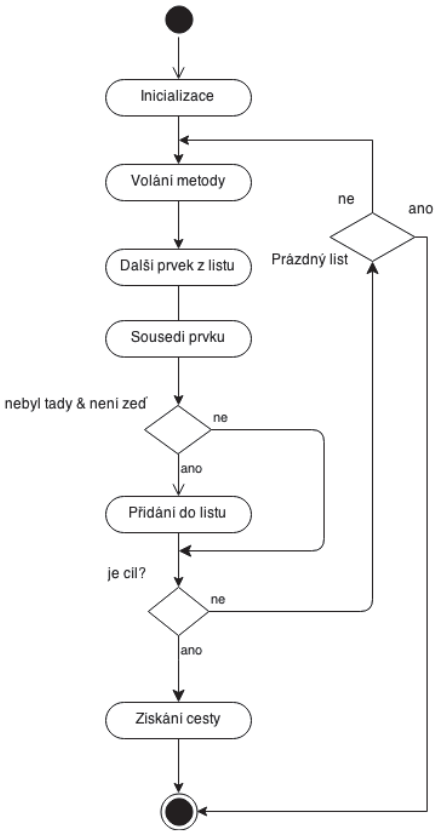
7.3 Nalezení cesty

Na vyhledání nejkratší cesty se využívá třída `Finder`[6]. Pro hledání se využívá algoritmus podobný Dijkstrovu algoritmu. Při inicializaci se nastaví který režim se bude využívat, vytvoří se dvourozměrné pole, které je stejně velké jako mapa. Jako body se využívají jednotlivé body mapy. Při každém iteraci se procházejí sousedi jestli daný bod není cíl. Pokud ne, tyhle body se přidají do listu a taky do dvourozměrného pole, který se používá pro kontrolu jestli se tento bod už nekontroloval. Tento list se při další iteraci prochází. Dále se zjišťuje jestli na daném místě není zeď. Hledání cesty si uchovává i ten bod, který je nejblíže k cíly. Pokud se hledání cíle nezdaří, vezme se bod který je nejbližší a vytvoří se cesta podle tohoto bodu. Každý bod cesty je tvořen třídou `Cell`. Uchovává si předešlý bod, pozici v mapě a počet bodů do nejbližší zatáčky. To poslední se využívá u třetího režimu.

Hledání cesty má tři režimy. Prvním je pro UI která jezdi autem. Je to proto, že UI autem jezdí jenom po silnici, takže při zjišťování, jestli na daném bodu není zeď, se provádí i například pro trávu. Druhým režime je pro UI, která chodí pěšky, takže se se kontrolují jenom zdi. Třetím režimem je jenom pro obyčejného uživatele. Tento režim se nachází jenom na straně klienta, kterému se objeví nejbližší cesta k cíli(například pro jízdu autobusem). I zde se jenom kontrolují zdi. Třetí režim je nejjednodušší, jelikož se uživatelovi zobrazují všechny body na mapě. U ostatních se nechají jenom ty body ve kterých se UI má obrátit.

Vytvoření cesty pro první režim je nejtěžší. Je to proto, že aby UI dokázala jet z jednoho místa ke druhému musí se optimalizovat cesta. Už při hledání cesty se kontroluje jestli nejsou hned za sebou zatáčky, jelikož UI by asi pravděpodobně to nedokázala uřídit. U tohohle se využívá proměnná `countToChangeDirCell`. Pokud je tato proměnná větší nebo se rovná 2, cesta se může otočit. Pokud ne, tahle trasa není možná a proto se do pole vloží `null`, který značí, že tenhle bod ještě nebyl použit. Druhá fáze je odstranění rovinek a ponechání jenom zatáček. Dále se zde jednotlivé body předělávají na jinou třídu a to na `TurnCell`. Tahle třída uchovává dále i na jakou stranu se bude zatáčet a z jaké strany se auto/hráč blíží. Tohle se využívá u navigování UI, jelikož by při každé aktualizace se tohle musel počítat a stálo by to nějaký procesorový čas. Je to jednoduchý algoritmu. Porovnávají se dva body a podle jejich pozice se určí jestli jde o zatáčku a nebo rovinku. Třetí fáze je nejtěžší. Využívá se jenom pro jízdu autem. Aby řízení auta bylo jednodušší, je cesta optimalizována tak, aby auto jezdilo jenom v pravém pruhu. Zde se naráží na jeden problém a to je, že cesty musí být konstruovány s jistými pravidly. Nesmějí být šikmé, kolem cesty pokládat něco jiného než cestu a cesta musí mít na šířku 3 body. V tomhle kroku se dál do jednotlivých bodů uchovává z jakého směru UI jde a na jakou stranu se bude otáčet. Počáteční úprava je základě pozorování, jak se daná cesta vypočítává a jaké situace mohou nastav. Následně se procházejí jednotlivé zatáčky a upravují se tak, aby auto jezdilo po pravé straně. Právě u téhle fáze se využívají rohy křižovatek. Pokud by v rohu křižovatky byla i silnice, algoritmus nedokáže rozeznat a posunout zatáčku na správné místo. Pokud se zpracovává první zatáčka, kontroluje se zde, jestli je opravdu bod na pravé straně ve směru jízdy. Prochází se celá silnice v zatáčce a kontroluje se dokud netrefí roh zatáčky, pokud ano zatáčka se přemístí na tohle

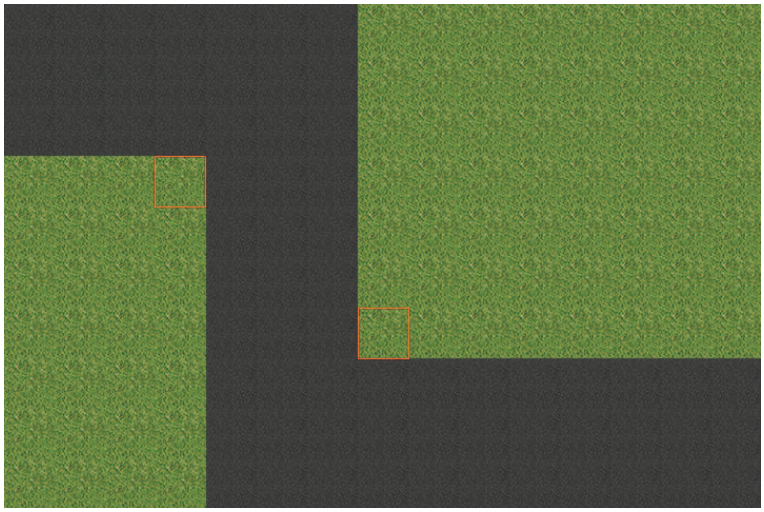
místo. Pro druhý režim se používá metoda `convertPath`, která předělá jednotlivé body z `Cell` na `TurnCell`.



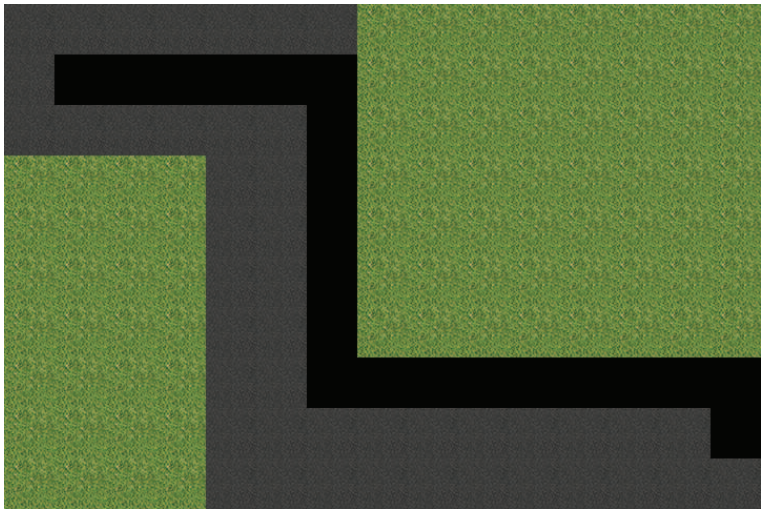
Obrázek 10: Diagram aktivit - Nalezení cesty k cíli

Na následujících obrázcích(obrázek č. 11 a 12), je znázornění nalezené cesty před veškerou optimalizací a po optimalizaci pro jezdění autem. Zatáčka, která je zobrazená modře značí, že další zatáčka je napravo a červená značí nalevo. U optimalizace pro auto je vložen i cílový bod. U první režimu už tak není.

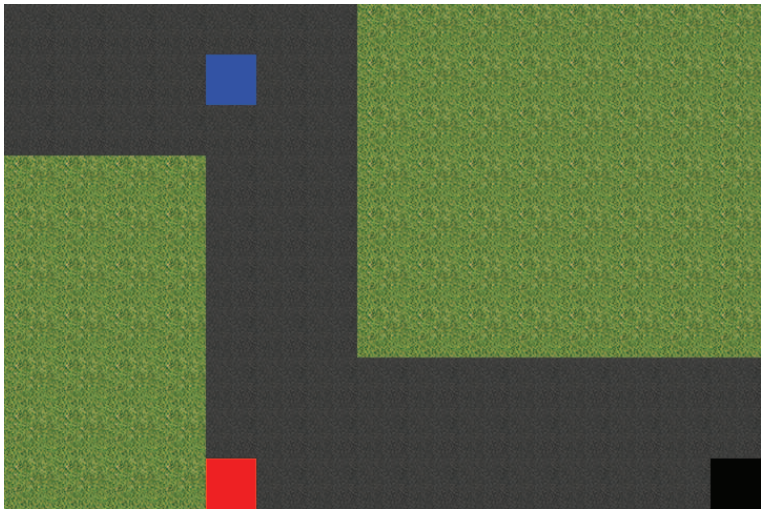
Obrázky byly pořízeny v editoru mapy, který sloužil pro testování.



Obrázek 11: Vyžadován tvar silnice



Obrázek 12: Nalezená cesta



Obrázek 13: Optimalizovaná cesta pro auto

8 Editor mapy

Třetí částí bakalářské práce bylo vytvořit editor mapy. Pohybování po mapě se využívají pohybové tlačítka na klávesnici. Pro jednoduchost se mapa pohybuje po jednotlivých částech mapy. Jelikož se zde využívají další komponenty, tak při kliknutí na některou z nich by se následně při klikání šipek nepohybovala mapa, muselo se na každou komponentu zakázat „focusable“.

Jelikož se scéna nemění, pokud se nepohybuje myší po mapě, vykresluje se mapa jenom při pohybu myší. Pro samotné vykreslování se používá knihovna Swing. Jelikož do `JComboBox` není jednoduché přidávat k textům i ikony, byl na internetu nalez jednoduchý návod[3], jak přidat ikony. Využívá se zde rozhraní `ListCellRenderer`, které vykreslí každou buňku zvlášť.

I zde se nachází třída `World`, která obsahuje veškeré informace světu. Editor umožňuje tyhle informace uložit na pevný disk jako serializaci třídy `DataMap`, kterou využívá i server a dále umožňuje tyhle informace i otevřít a případně editovat. Pro jednoduché přidávání UI bylo přidávání zahrnuto do editoru mapy. Zde můžeme nastavit její polohu, jméno a povolání. Tyto informace se ukládají do stejné třídy jako informace o hráčích, ale nejsou součástí stejného souboru. Je to proto, aby při editace mapy nebyly nechtěně smazány informace o hráčích.

9 Závěr

Všechny požadavky na bakalářskou práci byly splněny. Většina funkcí reálného života je naimplementována tak, jak bylo v představě na začátku návrhu práce. Největší důraz byl kladen na povolání a umělou inteligenci. U umělé inteligenci bylo nejtěžší jí naučit řídit automobil. Nakonec se to povedlo a UI dokáže jezdit z jednoho místo na druhé.

Program se stále může vylepšovat. Ze začátku by bylo potřeba optimalizovat aktualizace objektů, které má hráč ve svém okolí. Zde dochází k zbytečným, redundantním, datům, které se odesílají hráčovi. Dále by se mohl vyměnit „Swing“ na knihovnou LWJGL, která obaluje API „OpenGL“. Zvýšilo by to výkon v zobrazování scény. Dalším vylepšením by mohlo být zlepšení pohybu UI autem a dále by UI mohla měnit své povolání. Zatím má předem určené, které se zadá v editoru mapy. Nové povolání by se mohlo přidat a stávající povolání politik rozšířit o další možnosti volby.

Jan Michálek

10 Reference

[1] Java 2D Game Programming Platformer Tutorial - Part 1 - The Game State Manager. FOREIGNGUYMIKE. *YouTube* [online]. 2013 [cit. 2015-04-27]. Dostupné z: https://www.youtube.com/watch?v=9dzhgsVaiSo

[2] REYNOLDS, Craig. Steering Behaviors For Autonomous Characters. REYNOLDS, Lisa. Reynolds *Engineering & Design* [online]. 2004 [cit. 2015-04-27]. Dostupné z: http://www.red3d.com/cwr/steer/

[3] How to Use Combo Boxes. *The Java™ Tutorials* [online]. 2004 [cit. 2015-04-27]. Dostupné z: https://docs.oracle.com/javase/tutorial/uiswing/components/ combobox.html

[4] Java2D a dokonalé vykreslování. FUGICZEKS. *Itnetwork.cz: Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další.* [online]. 2012 [cit. 2015-04-27]. Dostupné z: http://www.itnetwork.cz/java-2d-dokonale-vykreslovani

[5] KRČMÁŘ, Petr. Pohled pod kapotu JVM – bitmapy typu BufferedImage a VolatileImage v celoobrazovkových režimech. TIŠNOVSKÝ, Pavel. *Root.cz: informace nejen ze světa Linuxu* [online]. 2013 [cit. 2015-04-27]. Dostupné z: http://www.root.cz/clanky/pohled-pod-kapotu-jvm-bitmapy-typu-bufferedimage-a-volatileimage-v-celoobrazovkovych-rezimech/

[6] Path Finding Algorithms (DFS, BFS, A*, Greedy, Dijkstra) in Java. KANARGIAS, Nikos. *YouTube* [online]. 2013 [cit. 2015-04-27]. Dostupné z: https://www.youtube.com/watch?v=CLbqqb53DLA

A Obsah DVD

Příloha na CD/DVD.
Obsah CD/DVD:

- *Eclipse projekt/Re-City* - Spustitelný jar soubor klienta plus data
- *Eclipse projekt/Re-City_Server* - Spustitelný jar soubor serveru plus data
- *Eclipse projekt/REdit1.0* - Spustitelný jar soubor editoru mapy
- *Eclipse projekt/DTO* - DTO soubory
- *Spustitelné projekty/Re-City* - Eclipse projekt clienta
- *Spustitelné projekty/Re-City_Server* - Eclipse projekt serveru
- *Spustitelné projekty/REdit* - Eclipse projekt editoru
- *Uživatelský_manuál* - Uživatelský manuál