Zadání absolventské práce

- Toto zadání je podepsané ředitelem školy a vedoucím práce,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě v archivu školy),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

Institut informatiky - Vyšší odborná škola, s.r.o.

Absolventská práce

Počítačová hra ve stylu Asteroids

Petr Bílek

Vedoucí práce: Ing. Jan Jirsa

Vzdělávací program: 26-47-N/11: Informační a komunikační systémy

Vzdělání: 26-47-N/.. Informační technologie

říjen 2012

Poděkování

Rád bych poděkoval Ing. Janu Jirsovi za vedení absolventské práce, Mgr., Ing. Janu Voříškovi za kontrolu statistických výpočtů v kapitolách o testování, své přítelkyni za její bezbřehou trpělivost s mými probdělými nocemi.

Prohlášení

Prohlašuji, že jsem svou absolventskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 30.10. 2012 ……..

Abstract

Computer Game in the style of Asteroids
This work describes design and implementation of computer game similar to legendary game Asteroids. The main focus of this work is in the implementation, which should test my knowledge of programming in Java language and other technologies used during the implementation.
This work should also be an introduction into developing (online) computer games (in 2D). Parts of this work will contain information about problems user can meet during developing games in an Android operating system.

Abstrakt

Tato práce se zabývá návrhem a implementací počítačové hry odvozené od legendární hry Asteroids. Hlavní těžiště práce spočívá v implementaci hry, která by měla prověřit získané znalosti v programování v jazyce Java a případně v dalších technologiích využitých během implementace. Práce by také měla sloužit jako úvod do programování (online) počítačových her (ve 2D). Součástí práce budou informace o problémech, se kterými se může vývojář setkat při implementaci her v systému Android.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

1.1 Výběr tématu absolventské práce

Studium na IIVOS je zaměřeno na programování v jazyce Java. Tato absolventská práce by mi měla poskytnout dostatečný prostor pro vyzkoušení a upevnění nabytých znalostí. Naprogramování počítačové hry dává velkou možnost vyzkoušet si mnoho dosažených znalostí v praxi.
Hra bude fungovat na principu klient-server, což mi umožní otestovat práci se sítí. Bude pracovat ve 2D a ověří tak znalosti práce s 2D grafickou knihovnou Javy. Při programování uživatelského rozhraní využiji znalostí knihovny Swing. Důležitým prvkem při vytváření hry bude také práce s vlákny. Výsledky hry (skóre), informace o hráčích nebo nastavení hry se mohou ukládat do XML souborů nebo databáze.

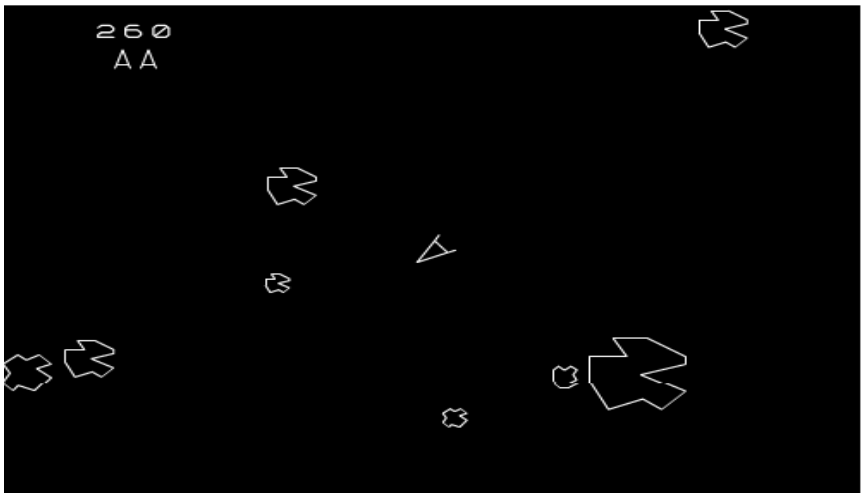
Je zřejmé, že návrh hry prověří mnoho technologií, a jak doufám, tak zábavnou formou. Během psaní práce budu předpokládat, že vážený čtenář má již základní znalosti o všech výše zmíněných technologiích. Hlavní těžiště toho to dokumentu bude v popisu návrhu hry. Popisem zdrojového kódu se v tomto dokumentu budu zabývat jen okrajově, pokud vůbec. Vlastní implementaci je možné nastudovat ze zdrojových kódů přiložených na CD.

Jinými slovy, po přečtení tohoto dokumentu by měl být čtenář schopen porozumět v první řadě důvodům **proč** byla hra naprogramována tak jak je. To **jak** byla hra naprogramována již uvidí ve zdrojových kódech. Ale i v tomto textu by měl čtenář najít základní informace, umožnující se rychle ve zdrojovém kódu zorientovat.

Vzhledem k zaměření této práce nelze očekávat, že bude výsledná hra vyšperkovaná všemožnými „featurami“. Měla by však být dostatečně dobře navržena na to, aby byla připravena na další rozšiřování (jak každý dobře ví, hry a software obecně se dá vylepšovat do nekonečna). Informace zde poskytnuté by měli být dostatečně obecné, aby je bylo možno využít při návrhu a implementaci zcela jiného typu počítačové hry, nejen v Javě.

1.2 O hře Asteroids

Jak již bylo řečeno (nebo spíše napsáno), budu se zabývat návrhem a implementací hry odvozené od hry Asteroids. První otázka, kterou je třeba si položit je, co je to vlastně za hru?



Obr. 1.2.1: *Screenshot ze hry Asteroids*

Nejlépe se se hrou seznámíte, když si ji zahrajete. Na adrese http://www.play.vg/games/4-Asteroids.html existuje jedna z mnoha jejích verzí. Na obrázku (1.2.1) můžete vidět screenshot z této hry. V levém horním rohu je vidět skóre (dosažený počet bodů) a počet zbývajících životů (dvě raketky ve tvaru velkého A). Zhruba uprostřed je raketka (znovu ve tvaru A), kterou může hráč ovládat šipkami na klávesnici a ze které může střílet (mezerníkem) na meteority (to jsou ty ostatní divné tvary na obrázku). Když hráč vystřelí a strefí se do meteoritu, získá za to body. Meteorit se rozbije na menší meteority, které se pohybují po obrazovce a při překročení okraje obrazovky se objeví na druhé straně. Nejmenší meteority se už nedělí a zmizí. Po rozbití všech meteoritů se objeví nové velké meteority.

Srážka s meteoritem ubere jeden život. Ve hře se i čas od času objeví UFO, za jehož sestřelení může hráč získat další extra body. Za určitý počet dosažených bodů dostane hráč nový život.

Účelem hry je samozřejmě získat co nejvíce bodů. Hra končí, když hráč přijde o všechny životy. Jde o velmi jednoduchý a přitom zábavný herní scénář.

První verze hry Asteroids byla vydána už v roce 1979 firmou Atari Inc. Jednalo se o její nejúspěšnější videohru (prodalo se jí přes 70 000 kusů)[1]. Hra používala pouze vektorovou grafiku[1], ale s rostoucí rychlostí výpočetní techniky se dočkala i rastrové grafiky[2]. Hra Asteroids byla implementována na mnoho počítačových platforem, Atari počínaje, mobilními telefony konče.

Původní hru Asteroids navrhl Lyle Rains a naprogramovali Dominic Walsh a Ed Logg.[1] Implementována byla na hardwaru vyvinutým Howardem Delmanem.

Hra Asteroids je populární dodnes. O tom svědčí i fakt, že 6. dubna 2010 byl překonán oficiální rekord dosaženého skóre Johnem McAllisterem v Portlandu, Oregonu.[4]

Obr. 1.2.2: *Port na Xbox Live Arcade*[3]

1 Grafické prvky jsou popsány geometrickými tvary, jako například bod, čárka, křivka atp.[2]
2 Grafické prvky jsou popsány pomocí pixelů, tj. každý bod na obrazovce je popsán svou barvu[2]

2. Návrh hry

Jak taková hra Asteroids vypadá jsem popsal v předchozí části *1.2 O hře Asteroids*. Ovšem hra, kterou se chystám naprogramovat, nebude mít úplně stejná pravidla. Už před začátkem programování by mělo být jasné, jak bude výsledná hra vypadat. Z toho bude vycházet analýza hry (návrh konkrétních datových modelů, rozhraní a objektů).

Je možné vymyslet nepřeberné množství variací této hry, ale účelem této práce není vymyslet co nejkrkolomnější pravidla, nýbrž navrhnout základní kostru hry, kterou bude možno do budoucna rozšiřovat. Nyní tedy popíšu to, co se bude nebo nebude implementovat.

KISS - Keep It Simple, Stupid (tj. „udržuj to jednoduché, hlupáku“), případně Keep it short and simple (tj. „udržuj to krátké a jednoduché“) je zkratka, která pochází z oblasti počítačů, kde shrnovala empirickou zkušenost, že problémy by se měly řešit jednoduchým způsobem, neboť přílišná komplikovanost přináší nežádoucí vedlejší účinky.[5] Tímto pravidlem se budu snažit během návrhu řídit. Prvním krokem tedy bude navrhnout co nejjednodušší funkční variantu hry, kterou bude možné v budoucnu rozšiřovat. Z této úvahy vychází následující návrh hry, ve kterém popíšu všechny aspekty hry z pohledu uživatele, hráče.

2.1 Výběr jména hry

Abych se nemusel na hru stále odkazovat jako na „hru“, pokusil jsem se vybrat hře nějaké vhodné jméno. Výběr jména je důležitý především z marketingového hlediska. Dobře zvolené jméno by mělo být snadno zapamatovatelné, jednoduše vyslovitelné a hláskovatelné a musí dobře ilustrovat, o co ve hře jde.[6] Pokud budete hru někde nabízet, její jméno je to první, co potencionálního hráče může nalákat nebo naopak odradit. Pokud by hra mohla být „světová“, není na škodu vymyslet název v angličtině, nebo si vymyslet kompletně nové jméno, které však bude jednoduše vyslovitelné pro anglicky mluvící národy. Jméno by také mělo být dostatečně originální, aby šlo snadno vyhledat v internetových vyhledávačích typu Google.

Aby jméno mé hry dobře ilustrovalo o co jde, mělo by obsahovat slovo Asteroids. To však není ani trochu originální[3]. Asteroid je vlastně starší a nepřesné, leč stále často používané označení těles, které se správně nazývají planetky, anglicky planetoids.[7] Planetoids je už originálnější název, ale stále ne dost unikátní.[4] Vzhledem k tomu, že se princip hry od původní hry Asteroids příliš lišit nebude, nepomůže nám její princip v hledání vhodného přívlastku ke slovu Planetoids. Uvažoval jsem o použití anglického přívlastku „juglar“ (zabijácký, vražedný). To už zní dostatečně unikátně[5], ale není to příliš dobře vyslovitelné [ˈdʒʌgjʊlə]. Nakonec jsem se rozhodl pro název jenž je dostatečně výstižný a unikátní[6]: **Deadloids** (česky **Dedloidy**). Jde o spojení slov dead [ded] (mrtví, neživý) a asteroids/planetoids. Nedává to sice takový smysl jako „Juglar planetoids“, ale zní to dobře!

2.2 Sprity ve hře

Sprite reprezentuje pohyblivý symbol ve hře. Sprite (někdy též sprajt) je označení používané v počítačové grafice pro většinou malý dvourozměrný obrázek (resp. animaci), který je integrován do větší scény.[8] Zde však nebude sprite označovat jen grafický prvek, ale objekt reprezentující jednu samostatnou entitu, která bude „žít“ v našem herním světě.

3 Google nalezne kolem 23 600 000 výsledků pro slovo asteroids a 15 900 000pro asteroids game
4 Google nalezne kolem 1 390 000 výsledků pro slovo planetoids a 3 440 000 pro planetoids game
5 Google nalezne kolem 8 650 výsledků pro slovní spojení jugular planetoids.
6 Google nalezne 4 výsledky pro slovo deadloids

Všechny sprity ve hře budou mít několik společných vlastností (atributů). Například **velikost** bude určovat kolizní kruh, jehož protnutí s kolizním kruhem jiného spritu vyvolá střet, na který sprity budou reagovat výbuchem, odrazem, nebo taky nijak (tj. jeden sprit přejede přes druhý, jako by tam ani nebyl). **Život** bude číselná hodnota, která bude určovat poškození spritu. Pokud sprit přijde o veškerý život, bude zničen. **Pozice** bude určovat souřadnice středu spritu vzhledem k zobrazovací ploše, na které se sprit nachází. **Vektor rychlosti** bude určovat, kterým směrem a jak rychle se sprit pohybuje.

Kromě těchto a několika dalších vlastností budou mít sprity také vlastní „**chování**“. Chování bude určovat to, jak a kam se sprit bude pohybovat po hrací ploše a jak bude reagovat na různé události ve hře (například na střet s jiným spritem). Pro většinu spritů bude chování znamenat pouze posun po hrací ploše jedním směrem, případně vyhýbání se překážkám atp.
Chování spritu je ovlivněno jeho vlastnostmi. Například přesun spritu po ploše je ovlivněn jeho maximální dovolenou rychlostí.

V následujících kapitolách budou popsány sprity, které se budou ve hře vyskytovat.

2.2.1 Asteroid

Asteroid bude v několika verzích (velikostech). **První verze** asteroidu bude mít nejvyšší hodnotu života i velikost. Pro jeho zničení bude potřeba asteroid zasáhnout 3x (standardní) raketou. Po zničení se rozpadne na 3 menší asteroidy s životností dvou zásahů raketou. Tato **druhá verze** asteroidu se po zničení rozpadne na 4 nejmenší asteroidy. Nejmenší, **třetí verze** asteroidů se zničí po jednom zásahu raketou.

Každá verze asteroidu se bude pohybovat v náhodném směru náhodně zvolenou rychlostí v daném rozmezí minimální a maximální rychlosti daného typu asteroidu. (Konkrétní hodnoty těchto rychlostí se rozhodnou až během testování hotového prototypu hry). Faktorem pro rozhodování bude samozřejmě co nejlepší hratelnost hry. Asteroid bude také rotovat kolem své osy. Rychlost a směr rotace bude další jeho vlastností náhodně zvolenou (při vytvoření a vložení asteroidu do hry) z daného rozmezí.

Zde se také nabízí, aby měl asteroid určenou výší skóre, které hráč dostane za jeho zničení. To však nebude určovat asteroids, ale „herní strategie“. Herní strategie bude třída, která bude určovat pravidla hry a bude popsána později. Ten kdo odmění hráče za zničení asteroidu nebude přece sám asteroid!

Asteroidy nebudou na střet s jiným tělesem nijak reagovat. Jeden sprit přeletí přes druhý, jako by ani neexistoval. Při střetu s jiným spritem nebude na událost střetu nijak reagovat (o reakci se postarají jiné sprity).

2.2.2 Spaceship

Spaceship (vesmírná loď) bude sprite ovládaný hráčem (hráčem by mohla být i umělá inteligence!). Spaceship se bude pohybovat směrem vpřed nebo vzad (bude mít omezenou maximální rychlost pro pohyb vpřed i vzad) a bude moci zatáčet vlevo a vpravo. Směr pohybu bude hráč ovládat pomocí šipek klávesnice. (Do budoucna by se mělo počítat s tím, že uživatel bude moci klávesy pro ovládání spritu měnit). Z vesmírné lodi bude možné vystřelit raketu, která může zničit nebo poškodit jinou vesmírnou loď, asteroid nebo UFO. Při srážce s jiným spritem Spaceship vybouchne a poškodí vše ve svém blízkém okolí silou tří raket, v širším okolí silou jedné rakety. Vhodná velikost okruhu síly výbuchu, tj. v jaké vzdálenosti od výbuchu rakety budou sprity poškozeny, se stanoví během testování hry.

O vytvoření nové rakety, odebrání života a případně ukončení hry se nebude starat raketa, ale „herní engine“.

Spaceship bude mít také barvu charakterizující, ke kterému týmu patří (pro případ hry více hráčů, viz dále). V případě hry jednoho hráče si bude moci vybrat barvu hráč dle vlastního vkusu.

2.2.3 Raketa

Raketa reprezentuje střelu vystřelenou ze Spaceship (nebo Ufa). Poletí rovně ve směru ve kterém bude natočen Spaceship v okamžiku výstřelu. Při srážce s předmětem vybouchne a poškodí vše ve svém okolí. Poškozením se myslí snížení života spritu (o kolik přesně se určí až během testování hry), viz *2.1 Sprity ve hře*. Raketa zmizí, pokud opustí hrací plochu. Spaceship bude mít omezený počet raket, které může mít vystřelené (na hrací ploše) v jeden okamžik a také rychlost, s jakou může střílet.

2.2.4 Ufo

Ufo je sprite, který se bude objevovat ve hře od druhého levelu v náhodném časovém intervalu a bude se pohybovat z levé strany k pravé, kde zase zmizí (nebo opačně). Během svého letu se bude snažit vyhýbat všem spritům, které bude mít v cestě. Při srážce se spritem vybouchne a poškodí sprite silou jedné rakety.

Život spritu bude ve výši síly výbuchu jedné rakety (jinak řečeno, jedna raketa zničí Ufo). Ufo je jakýmsi zpestřením hry, za jehož zničení bude mít uživatel extra body.

Ufo se bude pohybovat náhodným směrem. Rychlost Ufa bude obdobně jako v případě Asteroidu vybrán náhodně z daného rozmezí minimální a maximální rychlosti.

2.3 Herní strategie

Herní strategie (nebo též mód hry) popisuje pravidla hry. V jakém stavu hra začne, za jakých podmínek hra končí a jak bude herní engine - reprezentující herní strategii - reagovat na události ve hře (událostí je například střed spritů). Herní strategie může ovlivňovat všechny aspekty hry, například kolik životů bude mít hráč k dispozici, nebo za kolik bodů dostane další nový život. Bude zodpovědná za vytváření spritů.

Příklad hry s mnoha herními strategiemi může být hra Quake 3 Aréna. Jedná se o mulitplayerovou střílečku z pohledu první osoby, která obsahuje herní módy jako Free for all (všichni hrají proti všem),Team death match (TDM) (dva týmy hrají proti sobě a snaží se co nejčastěji zabít protivníka) nebo Capture the flag (dva týmy hrají proti sobě, cílem je sebrat nepříteli jeho vlajku a donést ji do své základny, ale přitom ubránit svou. Nerozhoduje tu počet zničení nepřítele, ale počet úspěšných obsazení vlajek).

Deadloids bude implementovat dvě herní strategie – hru jednoho hráče a hru více hráčů. Zásadní rozdíl v implementaci těchto herních strategií bude v tom, že hru jednoho hráče bude moci hrát uživatel s počítačem bez připojení k internetu, zatímco hru více hráčů bude možné hrát jen po internetu (nebo intranetu). Vytvoření dalších módů hry (např. týmová hra více hráčů) bude již rutinní záležitostí zkopírováním a modifikací jednoho z herních strojů implementujících tyto strategie.

2.3.1 Hra jednoho hráče

Herní stroj implementující herní strategii hry jednoho hráče při startu hry vloží Spaceship hráče do středu herní plochy a na náhodné místo (dostatečně vzdálené od vesmírné lodi) největší typ asteroidu. Hra odstartuje stiskem klávesy <Enter>. Hráč bude mít k dispozici tři životy. Ztrátou posledního života hra končí. Během hry bude v náhodném časovém intervalu vytvářet Ufo.

Po zničení všech Asteroidů vytvoří hra nový největší Asteroid. Asteroidům nastaví maximální možné rychlosti. S každým novým vložením největšího Asteroidu se budou maximální rychlosti Asteroidů zvětšovat (čímž bude stoupat náročnost hry).

Hře půjde nastavit (nebo spíše bude nutné nastavit) jméno hráče a barvu jeho lodi.

2.3.2 Hra více hráčů

Hra více hráčů bude obměnou hry jednoho hráče. Rozdíl bude jen v tom, že se do hry bude moci zapojit více hráčů (1 až 4) a to jak při startu hry, tak během hry.

Hráčova loď se při zapojení do hry zobrazí na náhodném místě na hrací ploše. Pak bude mít 5 vteřin na to, aby se přesunout na bezpečné místo (aby nebyl ve střetu s Asteroidem nebo jinou lodí). Během této doby nebude moci střílet.

Do hry se budou vkládat asteroidy, jejichž ničením budou hráči sbírat body, případně s jejich srážkou naopak ztrácet. Při zničení všech asteroidů se vloží nový největší asteroid se stejnou rychlostí jako předchozí (hra se nebude ztěžovat zrychlováním meteoritu).

Hráči budou mít neomezený počet životů. Hráč může sestřelit jiného hráče a získat za to extra body. Navíc se mu bude počítat skóre zničených Asteroidů. Smyslem hry je získat co největší počet bodů. Při dosažení maximálního skóre nebo maximálního počtu zničených protihráčů se hra zastaví, zobrazí žebříček podle počtu zničených protihráčů a nasbíraného skóre a po 10 vteřinách se restartuje (všem hráčům se vynuluje skóre a hra začíná od znova).

Hra začíná okamžikem svého vytvoření. Hra se bude vytvářet na serveru a odstartuje jejím založením (administrátorem serveru). V okamžik vytvoření hry se mohou hráči libovolně ke hře přihlašovat a odhlašovat. Hra nikdy nekončí, pouze administrátor serveru ji může zrušit.

2.4 Model server – klient

Hra Deadloids půjde spustit v režimu server nebo klient. **Server** bude vytvářet hry více hráčů (na požádání administrátora) i hry jednoho hráče (na požádání každého uživatele). Ke hrám se bude moci připojit uživatel/hráč pomocí **klientské verze** Deadloids.

Server a klient spolu budou komunikovat přes síťovou vrstvu (TCP/IP). Aby si mohl hráč zahrát hru pro jednoho hráče bez připojování k serveru, bude program schopný pracovat jako server a klient v jednom - serverová i klientská část budou schopny běžet naráz a komunikovat spolu bez síťové vrstvy (v takovém případě samozřejmě nebude serverová část programu vyžadovat autorizaci).

Program Deadloids bude načítat svou konfiguraci z XML souboru "*cesta k deadloids*/config.xml" (pokud nebude určeno jinak). Všechny parametry které bude možné programu předat jako argumenty příkazové řádky budou mít přednost před hodnotami parametrů z konfiguračního souboru (popis tohoto souboru viz dále).

2.4.1 Server

Pokud se hra Deadloids spustí z příkazové řádky s parametrem *--server*, bude program fungovat jako server (jak jinak). Server nebude mít žádné grafické rozhraní a ovládat se bude přes rozhraní klienta. Nepovinné parametry příkazové řádky pro server budou následující:

- --config *filename* Cesta ke konfiguračnímu souboru. Výchozí hodnota je "*cesta_k_deadloids*/config.xml".
- --port Port, na kterém server bude naslouchat (výchozí hodnota je 50505[7])
- --xml [*filename*] Jméno XML souboru, který bude sloužit pro uložení dat serveru (uživatelů). Pokud se jméno neuvede, bude se brát soubor "cesta_*k_deadloids*/database.xml".
- --database {mysql|sqlite} Typ databáze, který se bude používat pro ukládání dat serveru (uživatelů).
- --dbname *schema* Jméno databáze (má smysl pouze s volbou --database). V případě SQLite databáze jde o cestu k souboru s databází.
- --dbuser *username* Uživatelské jméno pro přístup k databázi.
- --dbpassword *passwd* Heslo pro přístup k databázi.

Je možné použít pouze jednu z voleb xml nebo database. Pokud není určena žádná (ani parametrem příkazové řádky, ani v konfiguračním souboru), použije se volba --xml.

Všechny tyto volby mají smysl jen v případě, že hra je spouštěna jako server. Klientská verze programu bez připojení k serveru nebude moci serverové úložiště nijak konfigurovat. Do budoucna ale nelze vyloučit, že by klient mohl využít informací z konfiguračního souboru a po startu pracovat přímo s databázovým úložištěm (tj. komunikovat přímo s vlastní serverovou částí bez síťové vrstvy). Přijde mi však z uživatelského hlediska jasnější, když se musí uživatel nejdříve přihlásit na server (byť localhost) a až poté může server konfigurovat. Nutnost přihlášení je i jistou bezpečnostní pojistkou proti neoprávněnému zásahu (ačkoliv poměrně chabou, protože se znalostí konfiguračního souboru bude každý schopný přistupovat k serverovému úložišti).

2.4.1.1 Uživatelské účty a role

K přístupu na server bude potřeba **uživatelský účet**. Uživatelský účet si bude moci vytvořit každý kdo se k serveru přihlašuje. Po připojení k serveru bude uživatel požádán o email (bude sloužit jako unikátní identifikátor) a heslo. Nový uživatel bude požádán o email, přezdívku (libovolné unikátní jméno, pod kterým se bude zobrazovat ostatním hráčům) a heslo (bude potřeba jej zadat pro kontrolu dvakrát). Pod jedním emailem může být zaregistrován jen jeden uživatel. Existence emailu by mohla být ověřována posláním emailu s aktivační adresou (dokud by tato adresa nebyla otevřena, nemohl by se uživatel zapojit do hry).

Role uživatelů v systému budou:

- **Superman** – může vše co administrátor a navíc měnit role uživatelů na administrátora nebo hráče, a nebo zcela smazat uživatelský účet. Také může zakázat vytvářet hry pro jednoho hráče (aby zbytečně nevytěžovali server. Hru jednoho hráče si může klient hrát na svém počítači bez připojování k serveru).
- **Administrator** – může zakládat a rušit hry, vyhazovat hráče ze hry ke které je aktuálně přidělen a blokovat uživatelské účty[8].

7 Port 50505 byl vybrán náhodně z rozsahu vyhrazeného pro dynamické přidělování a soukromé využití (49152 až 65535)[9]

8 Zablokovaný účet neumožní uživateli vytvořit nový účet se stejným e-mail nebude moci vytvořit nový účet.

- **Player** – nemá žádná speciální práva. Může se pouze připojit k serveru a zapojit se do některé spuštěné hry.

První zaregistrovaný uživatel dostane automaticky roli Superman. Všichni ostatní roli Player (kterou už bude moci superman měnit). Role Superman nepůjde smazat ani vytvořit nová, bude možné ji pouze předat někomu jinému (z původního Supermana se stane jen Administrátor).

2.4.1.2 Datové úložiště serveru

Server bude ve svém datovém úložišti ukládat **uživatele** s těmito **atributy**:

- Nickname – unikátní přezdívka uživatele pod kterou bude vystupovat na serveru. Přezdívku bude moci uživatel kdykoliv změnit.
- Email – unikátní identifikátor uživatele (nelze změnit). Do budoucna by mohl mít server ještě jeden email na uživatele pro zasílání zpráv, který by šel změnit.
- Password – heslo uživatele.
- Role – role uživatele (viz 2.3.1.1 Uživatelské účty a role).
- Blokace – UNIX TIME do kdy je uživatel blokován (0 znamená že není blokován, -1 znamená že je blokován navždy). V první verzi programu bude možné nastavovat jen hodnoty 0 a -1.
- Poslední přihlášení – DATETIME Datum a čas posledního přihlášení (zapisuje se jen 1x za den, kvůli statistice viz níže).
- Počet odehraných dnů – statistika počtu přihlášení uživatele (počítá se nejvýše 1 přihlášení za den).

Do budoucna se bude počítat s tím, že si server bude pamatovat nejlepší dosažené skóre hráče pro daný typ hry, nebo počet odehraných minut atp. Dalším rozšířením by mohlo být zaznamenávání a zobrazování nejlepších 10 hráčů ve hře pro jednoho hráče.

Datovým úložištěm může být databáze, nebo XML soubor. Pokud to bude XML soubor a nebude existovat, automaticky se vytvoří. Databáze už bude muset být vytvořena, ale program by měl umět automaticky vytvořit všechny potřebné tabulky.

2.4.2 Klient

Pokud se hra Deadloids spustí bez parametru --server, spustí se klientská verze. Zobrazí se grafické rozhraní, přes které bude uživatel moci provádět všechny požadované akce (start hry, připojení k serveru, konfigurace atd.). Pro více detailů o funkcích klienta viz *2.5 Grafické rozhraní klienta*.

Klientovi bude možné předat jen jednu nepovinnou volbu (argument příkazové řádky):

- --config *filename* Cesta ke konfiguračnímu souboru. Defaultně "*cesta_k_deadloids*/config.xml".

2.4.3 Konfigurační soubor

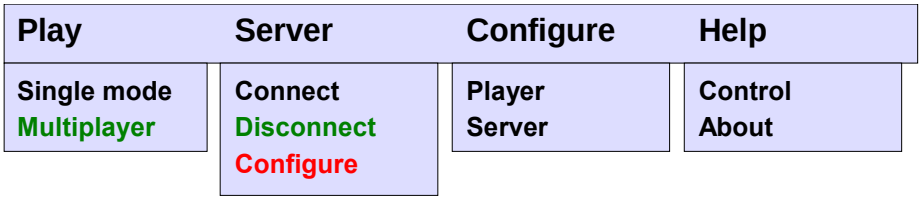
Jak již bylo napsáno dříve, program Deadloids bude načítat svou konfiguraci z XML[9] souboru "*cesta_k_deadloids*/config.xml" (pokud nebude určeno jinak argumentem příkazové řádky --config).

Serverová část programu si bude do konfiguračního souboru ukládat jen volby potřebné pro start serveru. Ostatní konfiguraci či data si bude ukládat do vlastního datového úložiště (viz *2.4.1.2 Datové úložiště serveru*).

Klient si do konfiguračního souboru bude ukládat naposledy použité přihlašovací údaje k serveru (IP adresu, port, email a šifrované heslo). Tyto údaje mu budou nabídnuty ve chvíli kdy se pokusí přihlásit na server, aby je nemusel stále vyplňovat ručně. Dále si bude v konfiguračním souboru udržovat informaci o naposledy zvolené barvě vesmírné lodě a jeho přezdívky pro hru jednoho hráče (tato přezdívka se bude obzvláště hodit, až si bude – do téhož souboru – ukládat nejlepší dosažená skóre).

2.5 Grafické rozhraní klienta

Grafické rozhraní klienta se bude skládat z menu (viz Obr. 2.5.1) a herní plochy. Herní plocha bude mít přesný rozměr (například 600x600 px).



Obr. 2.5.1: *Menu klienta – zeleně označené položky jsou dostupné po přihlášení k serveru. Červeně označené jsou přístupné jen při přihlášení k serveru pod administrátorským účtem*

Je třeba rozlišovat mezi rozměry herní plochy (a spritů) se kterými bude počítat program a velikostí herní plochy zobrazované hráči. Grafický výstup by měl počítat s tím, že se herní plocha bude zobrazovat například na obrazovce telefonu. Vnitřně sice může program beze změny počítat pohyby všech spritů po ploše 600x600 px, ale zobrazovat se už může na ploše menší (nebo větší).

Herní plocha bude po startu klienta zobrazovat deset nejlepších dosažených skóre. Na pozadí by mohla být běžící simulace hry Deadlodis s jedním hráčem. Při připojení k serveru budou na ploše zobrazeny informace o serveru včetně přehledu her běžících na serveru (typ hry, počet hráčů atp.). Na pozadí by mohla běžet simulace hry více hráčů.

9 Hlavní výhodou XML souboru je možnost snadno popsat stromovou hierarchii čitelnou pro stroj i člověka.

2.5.1 Menu Play

2.5.1.1 Single mode

Single mode spouští hru jednoho hráče (lokálně, tj. bez připojení přes internet). Před spuštěním se zobrazí dialog s možností výběru přezdívky hráče a barvy vesmírné lodě. Zadané údaje se uloží do konfiguračního souboru a při příštím spuštění budou v dialogu předvyplněné.

2.5.1.2 Multiplayer

Tato volba bude dostupná až po připojení k serveru. Po jejím výběru se zobrazí tabulka se všemi spuštěnými hrami na serveru (pro jednoho i více hráčů), která umožní hráči připojit se ke hře. Součástí tabulky budou i informace o počtu hráčů hrající hru (u her více hráčů). Řádek tabulky s hrou jednoho hráče bude informovat o tom, který typ hry jednoho hráče bude moci hráč spustit. Informace o běžících hrách jednoho hráče nebudou k dispozici.

2.5.2 Menu Server

2.5.2.1 Connect

Volba Connect zobrazí dialog pro přihlášení k serveru. Bude obsahovat pole pro IP adresu serveru, port serveru, přihlašovací jméno (tj. email) a heslo. Zadané údaje se uloží do konfiguračního souboru a při příštím spuštění budou v dialogu předvyplněné.

2.5.2.2 Disconnect

Tato volba bude dostupná až po připojení k serveru. Odpojí hráče od serveru.

2.5.2.3 Configure

Tato volba bude dostupná až po připojení k serveru a pouze pro administrátorské role (Superman, Administrator). Konfigurace serveru bude dialog s několika záložkami.

První záložka bude obsahovat volbou povolující/zakazující hru jednoho hráče a zobrazovat seznam běžících her pro jednoho hráče. Vypsané hry bude možné ukončit.

Druhá záložka umožní vytvořit hru pro více hráčů a zobrazovat seznam běžících her pro více hráčů. Vypsané hry bude možné ukončit. V některé z budoucích verzí bude možné zobrazit seznam všech hráčů ve hře (v novém dialogu) a vybrané hráče vyhodit ze hry.

Třetí záložka bude obsahovat přehled všech uživatelů a umožní měnit role uživatelů, blokovat jim přístup, nebo je mazat.

2.5.3 Menu Configure

2.5.3.1 Player

Volba Player zobrazí konfigurační dialog pro uživatele. Zde by mohlo být nastavení kláves pro ovládání hry. Další možnosti, které by zde pro hráče mohli být (jako přezdívka, barva lodi, přihlašovací údaje k serveru) se budou ukládat automatiky při jejich vyplnění před zahájením hry nebo připojením na server atp. V první verzi hry nebude možné nic nastavit, proto nebude ani volba Player v menu Configure zobrazena.

2.5.3.2 Server

Pod touto volbou v menu se zobrazí dialog na konfiguraci serveru. Jde o volby, které je možné zadávat z příkazové řádky, ale také jsou uloženy v konfiguračním souboru (nebo databázi). Volby zde editované budou uloženy právě v tomto konfiguračním souboru (databázi).

2.5.4 Menu Help

2.5.4.1 Control

V tomto menu bude zobrazena obrazovka s informacemi o přehledu ovládání hry (viz *2.6 Ovládání hry*).

2.5.4.2 About

V tomto menu budou zobrazeny informace o jménu a verzi programu, datu vytvoření, autoru programu a licencích (v programu můžou být použity zvuky a obrázky pod různou licencí).

2.6 Ovládání hry

Kromě výše popsaných dialogů (viz *2.5 Grafické rozhraní klienta*) zbývá ještě popsat jak bude hráč ovládat hru. Informace zde uvedené budou dostupné v menu pod *Help → Control*:

- Odstartování hry jednoho hráče: *Enter* (hra více hráčů odstartuje v okamžik vytvoření)
- Ovládání směru pohybu lodi: Šipky doleva a doprava
- Ovládání rychlosti lodi: Šipky nahoru (zrychlení) a dolů (zpomalení, zastavení)
- Vystřelení rakety: Mezerník
- Zobrazení skóre protihráčů: Tabulátor (pouze u her více hráčů)
- Zobrazení nejlepších skóre pro hru: Tabulátor (pouze u her jednoho hráče)
- Ukončení hry: Escape
- Pozastavení hry: Malé nebo velké písmeno P (pouze u her pro jednoho hráče)

Při implementaci je třeba brát v potaz, že v některé z budoucích verzí Deadloids si bude moci hráč klávesy měnit.

2.7 Ozvučení

Hra by měla být vhodným způsobem ozvučena. Například rozpad meteoritu nebo výbuch vesmírné lodi by měl být doprovázen vhodnou zvukovou kulisou. Hru by také mohla doprovázet melodie zvyšující u hráčů touhu po destrukci.

Ozvučení je určitě důležitou součástí počítačových her, která může zlepšit uživatelskou zkušenost.

2.8 Shrnutí

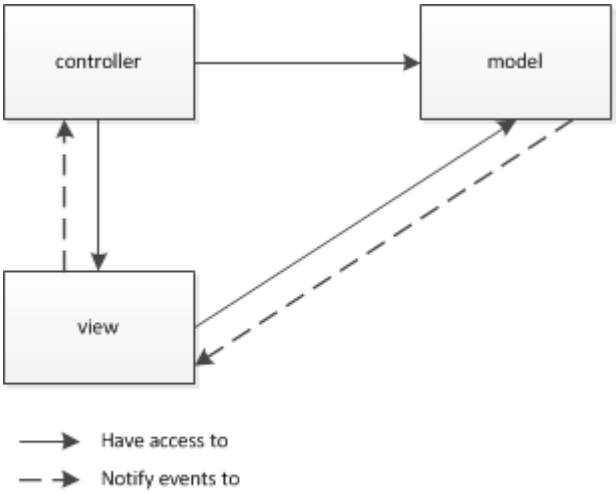
Nyní jste se již seznámili se zadáním a budoucí podobou hry Deadloids. Návrh hry počítá s možností rozšíření a vylepšení budoucích verzí (proto je například výhodné mít konfiguraci uloženou v XML – je to snadno rozšiřitelné). Jedno z mnoha možných rozšíření by například mohla být možnost během hry více hráčů komunikovat s ostatními spoluhráči jednoduchým chatem, nebo si ukládat různé nastavení přihlašovacích údajů pro více serverů v klientovi. Další užitečnou možností rozšíření může být zobrazení přehledu všemožných statistik hráčů. S těmito potenciálními rozšířeními by měl návrh designu aplikace počítat.

3. Analýza

Analýza slouží k popsání problémů a rozdělení složitějších problémů na jednodušší. V této části budou popsány nejdůležitější třídy a rozhraní, které budou potřeba pro úspěšnou implementaci programu.

Aplikace bude založena na **návrhovém vzoru MVC**. Hlavní komponenty tohoto známého návrhového vzoru jsou[12]:

1) **Model (model)**, což je doménově specifická reprezentace informací, s nimiž aplikace pracuje.
2) **View (pohled)**, který převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli.
3) **Controller (řadič)**, který reaguje na události (typicky pocházející od uživatele) a zajišťuje změny v modelu nebo v pohledu.



Obr. 3.0.1: *MVC[13]*

Hlavní třídou reprezentující model bude **GameWorld**. Tato třída bude obsahovat všechny sprity obsažené ve hře (Asteroidy, Spaceship, rakety atd.). **Každý** takový **sprit bude reprezentován vlastním modelem**.

Controller bude reprezentovat třída **GameStrategy**. Ta se bude starat o vkládání a odebírání spritů do/z třídy GameWorld (přes přístupové metody). Bude inicializovat GameWorld na začátku hry, přijímat události od hráčů a předávat je správným spritům (Spaceshipu hráče). Bude také počítat skóre hry a zjišťovat ukončení hry. **Každý sprit bude mít vlastní třídu controlleru, která se bude starat o jeho pohyb, reakci na události (např. srážku s jiným spritem) atp.**

Interface **WorldView** a **StrategyView** budou sloužit pro zobrazení GameWorld (hrací plocha) a informací z GameStrategy (například skóre). **Každý sprit bude mít vlastní třídu pro zobrazení sebe sama.**

Aplikace bude fungovat jako klient – server, model hry (GameWorld se sprity) se bude přenášet po síti. Proto bude potřeba ještě datový model pro GameStrategy – **StrategyModel**.

3.1 Návrh rozhraní GameWorld (Model)

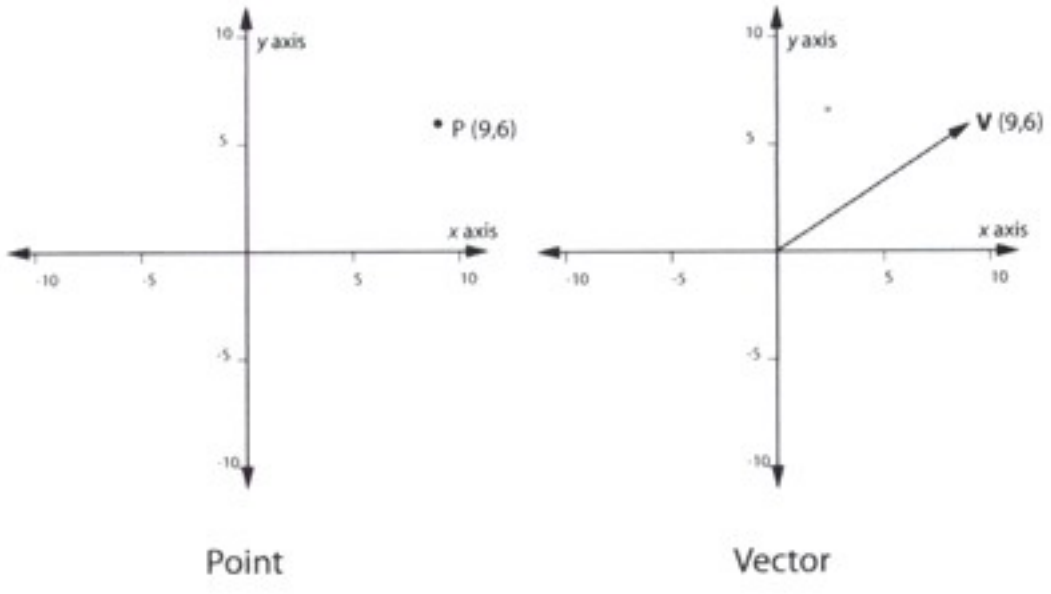
GameWorld bude obsahovat informace o hrací ploše (např. velikosti) a modely spritů. Dále bude obsahovat přístupové metody k těmto informacím, spritům, ale také metody na vyhledávání nejbližšího spritu atp.

GameWorld bude vytvářen a poběží na serveru a celý se bude posílat klientovi. Na klientské straně se tak vždy vytvoří celý GameWorld jako je na serveru jedna k jedné.

3.1.1 Popis polohy a pohybu spritů

Poloha spritů bude dána souřadnicemi X a Y (hra bude zobrazena ve 2D) a rychlostí danou vektorem rozděleným na X-ovou a Y-ovou složku. Tím bude reprezentována rychlost ve směrech souřadnic X a Y, a tak i směr pohybu. Jak polohu, tak rychlost bude reprezentovat třída **Vector2D**.

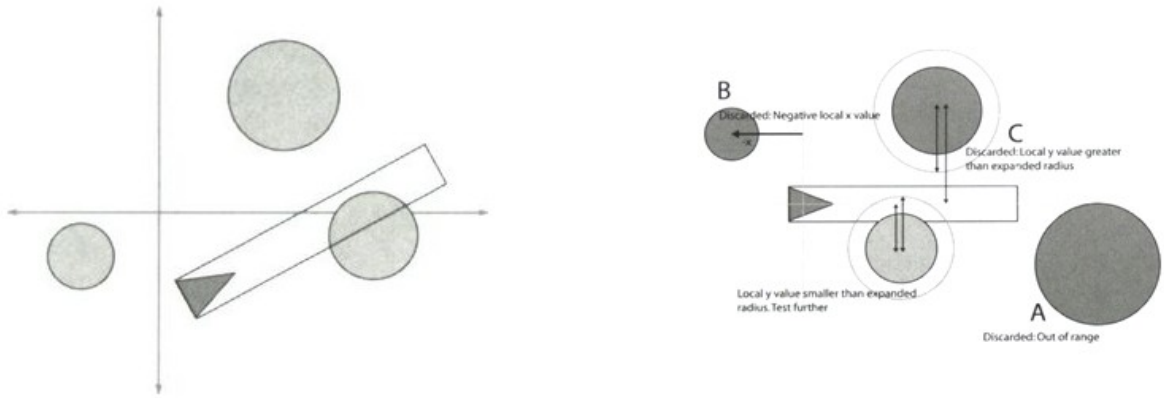
Sprit bude mít také vlastní natočení, které se může lišit od směru pohybu (směr, kam míří předek spritu). Například spaceship, která bude couvat bude mít opačný směr pohybu oproti směru natočení.



Obr. 3.1.1: Poloha (P) a vector (V) [14]

Třída Vector2D bude obsahovat několik užitečných výpočtů nad vektory, jako například zjištění vzdálenosti dvou vektorů (v případech kdy vektory reprezentují polohu [X,Y]), případně výpočet úhlu svírající dva vektory (bude se hodit při otáčení jednoho spritu směrem k druhému) a mnoho dalších.

Výpočty pohybu spritů budou ve třídě **SteeringBehavior**. Tato třída bude obsahovat metody pro výpočet rychlosti spritu v závislosti na zvoleném typu pohybu (nebo kombinaci několika různých pohybů). Například metoda *obstacleAvoidance() se* bude snažit pohybovat spritem tak, aby se při své cestě vyhýbal meteoritům a dalším spritům ve hře.



Obr. 3.1.2: Popis výpočtu změny rychlosti (směru pohybu)[14]

Rozměry a polohy spritů budou využity při výpočtu kolizí spritů. Z obrázku 3.1.2 je patrné, že pro výpočet kolizí budou tvary spritů zjednodušeny na kružnici. Každý sprit bude mít svojí velikost danou poloměrem kružnice. To má své výhody i nevýhody.

Výhodou je určitě jednodušší výpočet kolizí (stačí zjistit, zda jsou středy spritů od sebe ve větší nebo menší vzdálenosti než součet jejich poloměrů). Nevýhodou je, že může dojít ke střetu spritů, ačkoliv jejich grafická reprezentace se nestřetne.

Grafické zobrazení spritů nebude mít tvar kružnice, takže se může stát, že pro sprity menší než jejich velikost danou kružnicí dojde ke kolizi, aniž by se ve skutečnosti jejich grafické reprezentace střetly. Naopak, pokud budou kružnice menší než grafická reprezentace spritu, může dojít k tomu, že hráč uvidí sprity se střetnout, ale program to nezaregistruje. Naštěstí se ukazuje, že při vhodně zvolených tvarech spritů si toho hráč ani nevšimne.

Třídy Vector2D a SteeringBehavior jsem převzal z knihy Programming Game AI by Example[14]. Proto je zde nebudu detailně popisovat a jen doporučím váženému čtenáři, aby si z ní detaily o implementaci nastudoval. Zdrojové kódy jsou v knize v jazyku C++, nicméně jejich přepis do Javy jsem se snažil udělat tak, aby si byly zdrojové kódy co nejpodobnější. Neměl by tak být problém nalézt v mé Java implementaci zdrojových kódů části, na které se kniha odkazuje ve svých C++ kódech. (Nevýhodou je, že je kód v Javě napsán poněkud krkolomně, například je často zneužívána konstrukce „import static“).

3.1.2 Návrhy konkrétních spritů

Návrhy spritů jsou popsány již v kapitole 2.1.3 Sprity ve hře. Sprit bude rozdělen na MVC části následovně:

- Model – bude popisovat **velikost** (poloměr kružnice), **polohu**, **natočení** a **rychlost** spritu, viz *3.1.1 Popis polohy a pohybu spritů*. Dále bude obsahovat informaci o svém **životě** (číslo, které se snižuje například výbuchem rakety nebo srážkou s meteoritem). Model také může udržovat informaci o tom, k jakému týmu patří (ve hře kde by hrálo několik hráčů proti sobě). Model je to, co se bude přenášet po síti a musí tak obsahovat všechny relevantní údaje pro zobrazení spritů u klienta, ale nic víc. **Modely spritů bude udržovat třída GameWorld.**
- View – grafické zobrazení spritu. Půjde o třídu, která bude vykreslovat sprit ve třídě WorldView. Ve většině případů půjde o obrázek vykreslený dle informací o poloze a směru. **WorldView bude procházet všechny sprity z GameWorld a přiřazovat jim View dle jejich typu a stavu** (například Spaceship s nízkou hodnotou života může mít obrázek poničené lodi atp.)

- Controller – třída, která se bude starat o zpracovávání událostí poslaných spritům (viz *3.1.3 Popis událostí*) a řídit chování spritů (viz *3.1.4 Konečný automat (FSM)*).

Hlavní vlastnosti modelů konkrétních spritů budou:

- **Asteroid** nebude obsahovat žádné extra informace.
- **Spaceship** bude mít v sobě navíc informaci o jménu hráče, barvě, dosaženém skóre atp.
- **Raketa** bude obsahovat pouze informaci o tom, kdo ji vystřelil (aby bylo jasné, komu připast skóre za zničený asteroid atp).
- **UFO** nebude obsahovat žádné extra informace.

3.1.3 Popis událostí

Sprity budou dostávat informace o událostech ve hře pomocí zpráv (messages). Událostí může být například to, že došlo ke střetu s jiným spritem, nebo že na sprit někdo vystřelil raketu, zásah výbuchem rakety spolu se silou výbuchu atp. Tyto informace bude dostávat od jiných spritů, nebo z GameStrategy.

Typy událostí budou vypsány v třídě **MessageTypes**. Zprávy se budou posílat pomocí třídy **MessageDispatcher**, která bude přijímat zprávy ve formě popsané třídou **Telegram**. Telegram bude obsahovat informaci o tom kdo, komu a jakou zprávu posílá. Navíc bude mít i informaci o čase, kdy má zpráva dorazit (zpoždění s jakým má zpráva spritu dorazit).

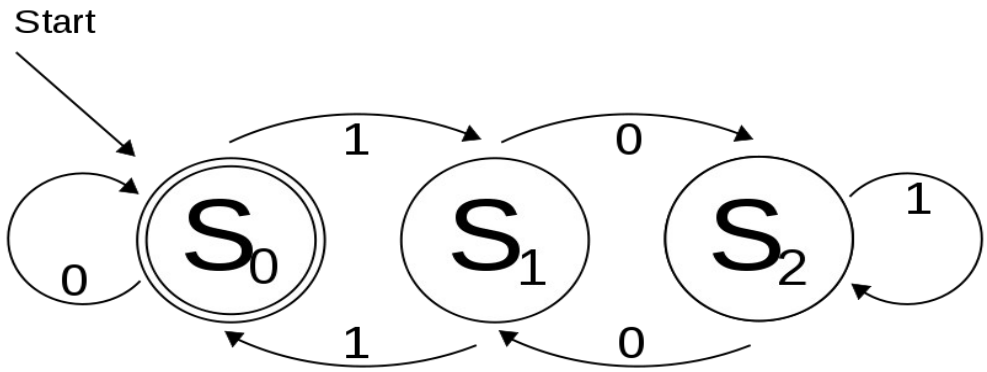
Uvedu příklad. Raketa vybouchne a chce oznámit všem spritům ve svém okolí, že byli zasaženi výbuchem. Bude tedy posílat zprávy. Vytvoří telegramy pro všechny sprity ve svém blízkém okolí a předá je třídě MessageDispatcher. Typ zprávy bude třeba MessageTypes.EXPLOSION, odesílatel zprávy bude raketa, příjemce sprit v okolí výbuchu a čas doručení (zpoždění) bude 0 (tj. doručí se ihned). Součástí zprávy může být informace o síle výbuchu. Pro vzdálenější sprity se vytvoří stejné zprávy, jen s vyšším časem doručení (než dorazí tlaková vlna výbuchu, tak to chvíli trvá) a s menší silou výbuchu.

Výhoda tohoto řešení událostí je, že sprit vyvolávající událost se už nemusí starat o to, jakým způsobem na zprávu ostatní sprity zareagují (zjednodušuje to a zpřehledňuje programování). Některé sprity totiž můžou události ignorovat, nebo je zpracovávat odlišně dle vlastního stavu.

Třídy MessageTypes, MessageDispatcher a Telegram jsou s mírnými modifikacemi převzaty z knihy Programming Game AI by Example[14]. Doporučuji váženému čtenáři, aby si z této knihy detaily implementace nastudoval.

3.1.4 Konečný automat (FSM)

Konečný automat (KA, též FSM z anglického finite state machine, či DFA z anglického deterministic finite automaton) je teoretický výpočetní model používaný v informatice pro studium vyčíslitelnosti a obecně formálních jazyků. Popisuje velice jednoduchý počítač, který může být v jednom z několika stavů, mezi kterými přechází na základě symbolů, které čte ze vstupu. Množina stavů je konečná (odtud název), konečný automat nemá žádnou další paměť kromě informace o aktuálním stavu[15]. Tolik definice z Wikipedie.



Obr. 3.1.3: Znázornění konečného automatu[15]

Konečný automat bude použit pro popis stavů spritů a herních strategií. Každý controller spritu se bude nacházet v nějakém stavu, který bude řídit chováni spritu, vytvářet události (viz *3.1.3 Popis událostí*) a přecházet do jiného stavu. Pokud bych se rozhodl naprogramovat umělou inteligenci řídící Spaceship, pak by se mohla tato UI (reprezentovaná controllerem spritu) nacházet ve stavu zaměřování asteroidu k sestřelení, ve stavu útěku před asteroidem, nebo čekání na zahájení hry. Asteroid zase může být ve stavu letu vesmírem, nebo ve stavu výbuchu a rozpadu na menší asteroidy.

Stav spritu bude reprezentován třídou **State**. Třída State bude mít metodu *OnMessage()* reagující na události (například změnou stavu) a metody *enter()*, *execute()* a *exit()*, které se budou volat při přechodu do stavu (*enter*), během každého cyklu aktualizace stavu (*execute*) a při jeho opuštění (*exit*).

Metody třídy State bude obsluhovat třída **StateMachine**. Ta bude udržovat informace nejen o aktuálním stavu spritu, ale i o předchozím stavu, pokud by se chtěl sprit někdy dostat z jednoho stavu do svého předešlého stavu. Pro jednoduchost se bude udržovat jen jeden předchozí stav, takže se půjde vrátit také jen o jeden stav zpět.

Sprit bude moci mít, kromě vlastního stavu, i tzv. globální stav. Půjde o stav, který bude společný všem spritům, nebo nějaké skupině spritů, který se bude volat před vlastním stavem spritu a který se nebude měnit při změně aktuálního stavu spritu.

3.2 Návrh rozhraní GameStrategy (Controller)

Třídy implementující **GameStrategy** budou mít na starost logiku zvolené hry. Deadloids bude mít v základu implementovány dvě varianty. Hru pro jednoho hráče a hru pro více hráčů. GameStrategy bude zodpovědné za vytvoření GameWorld, spritů a inicializaci jejich stavů a controllerů.

Po odstartování hry bude třída v nekonečném cyklu aktualizovat stav hry, procházet všechny controllery a volat jejich metody pro aktualizaci stavu (metoda *update()*). Hra by sice mohla vytvořit samostatné vlákno pro každý sprit ve hře a využívat tak efektivněji více procesorů počítače, ale kvůli náročnosti synchronizace je daleko jednodušší aktualizovat stavy spritů jeden po druhém.

Třídy implementující GameStrategy budou také **zodpovědné za zpracování reakcí na uživatelské vstupy** (stisky kláves, události myši) předávané z WorldView, případně je předávat (posílat jako zprávy událostí) spritu ovládaného hráčem (jeho Spaceship).

3.2.1 Pro hru jednoho hráče

Třída **SingleGameStrategy** bude reprezentovat hru jednoho hráče. V konstruktoru získá odkaz na GameWorld. Vloží do něj Spaceship hráče a největší Asteroid na startovní pozice a nastaví jejich chování (state).

SingleGameStrategy bude mít metodu reagující na vstup od uživatele. Stisknutím libovolné klávesy odstartuje hru (změní stav Asteroidu a Spaceshipu) a další události z klávesnice bude předávat controlleru spaceshipu.

Během každého cyklu bude kontrolovat, zda nebyl zničen spaceship. Pokud ano, sebere hráči život, chvíli počká a pak vloží novou Spaceship do GameWorld. Pokud hráči všechny životy dojdou, ukončí hru a zobrazí skóre. SingleGameStrategy bude také kontrolovat, zda se v GameWorld nachází ještě nějaké asteroidy. Pokud ne, přejde do „vyššího levelu“ - vloží nový největší Asteroid a zvětší jeho maximální rychlostí (čímž se hra ztíží).

Z výše popsaného vyplývá, že se SingleGameStrategy bude nacházet ve 3 stavech - před začátkem hry, vlastní hra a konec hry. Navíc přibude ještě čtvrtý stav, stav pro pozastavení hry (pause).

3.2.2 Pro hru více hráčů

Třída **MultiGameStrategy** bude reprezentovat hru více hráčů. V konstruktoru získá odkaz na GameWorld. Vloží do něj Asteroid a hru ihned spustí. Server bude moci třídu MultiGameStrategy požádat o vytvoření nového Spaceshipu (maximálně mohou být 4) pro připojeného hráče, případně požádat o odebrání Spaceshipu (pokud se hráč odpojí). Server bude do MultiGameStrategy předávat přes internet přenesené události z klávesnice.

Hra nikdy nekončí, administrátor serveru ji může pouze zrušit nebo restartovat. MultiGameStrategy hru automaticky restartuje když z ní odejdou všichni hráči. Pokud ve hře někdo dosáhne maximálního skóre, nebo uplyne nastavený maximální čas, hra se zastaví, zobrazí se skóre všech hráčů a po několika vteřinách se restartuje.

Během každého cyklu se bude kontrolovat, zda nebyl zničen některý Spaceship. Pokud ano, po několika vteřinách se vloží do hry nový (respektive se resetuje jeho život a znovu se zapojí do hry, aby uživatel zničením Spaceshipu nepřišel o své skóre).

MultiGameStrategy bude mít jen 2 stavy: hra a zobrazení skóre. Případně by mohla mít ještě stav pause.

3.3 Komunikace mezi serverem a klientem

Protože hra Deadloids bude fungovat jako multiplayer (hra více hráčů), musíme se postarat o komunikaci mezi serverovou částí hry a klientskou aplikací.

Komunikace mezi serverem a klientem se dá rozdělit na dvě hlavní části. Na přenos dat o stavu hry, který musí probíhat v reálném čase (real-time) a na to ostatní (přihlášení k serveru, administrace serveru atp.).

Komunikace mezi serverem a klientem bude řešena pomocí dnes nejběžnějších protokolů **TCP/IP**. Architektura TCP/IP je členěna do čtyř vrstev (na rozdíl od obdobného referenčního modelu OSI se sedmi vrstvami)[16]:

- aplikační vrstva (application layer) – reprezentována hrou Deadloids
- transportní vrstva (transport layer) – TCP nebo UDP, viz dále
- síťová vrstva (network layer) – zajišťuje adresaci, nejčastěji IP protokol
- vrstva síťového rozhraní (network interface) – nejčastěji Ethernet

Vrstvu síťového rozhraní samozřejmě není možné při vývoji hry ovlivnit. Jako síťovou vrstvu budu používat IP verze 4. Aplikační vrstvu reprezentuje samotný program Deadloid. Nejdůležitější je výběr transportní vrstvy. Máme na výběr mezi TCP a UDP.

TCP (Transmission Control Protocol) je tzv. spolehlivá transportní služba. TCP navazuje spojení, přenáší data a ukončuje spojení. Spolehlivost přenosu je zaručena potvrzováním doručení dat adresátem. Zaručuje tak, na rozdíl od UDP, že dorazí všechny data a navíc ve správném pořadí. Nevýhodou je obousměrný přenos dat (kvůli potvrzování doručení dat) a navazování a ukončování spojení, důsledkem čehož se zvětšuje objem přenesených dat a latence[10].

UDP (User Datagram Protocol) poskytuje nespolehlivou transportní službu. Nezaručuje, že data dorazí k adresátovi, ani že dorazila ve správném pořadí (packety mohou na síti putovat po různých trasách, například z důvodu vytížení přenosové linky se může router rozhodnout posílat packety střídavě dvěma různými cestami[17]).

Při běhu real-time hry (kterou Deadloids je), rozhoduje především rychlost. Pokud bychom použili TCP protokol, mohlo by čekání na přeposílání ztracených packetů příliš zdržovat. Zatímco by se TCP snažil doručit přenesená data a informace o úspěšném přenosu, server by již tou dobou mohl posílat nová aktuální data. **Důležitější je dostávat co nejrychleji poslední aktuální stav hry, než mít zaručeno, že dostaneme každý stav hry, který nám server poslal. Z tohoto důvodu bude použit** ve hře Deadloids **protokol UDP**. (Ze stejného důvodu je UDP používáno u většiny moderních real-time MMOG, například tribes, quake, unreal, starwars battlefront, battlefield 1942, halo, gears of war, team fortress 2, COD4).

Při řešení komunikace pomocí UDP se bude muset počítat s nevýhodami, které to přináší (nespolehlivost komunikace).

Komunikaci, která nebude muset probíhat v reálném čase, bude výhodné řešit pomoci TCP. Tato komunikace nebude probíhat tak často (v porovnání s přenosem stavu hry) a na její rychlosti také tolik nezáleží. TCP se využije například při přihlašování uživatele k serveru, nebo k administraci serveru.

10 Časová prodleva, v tomto případě způsobená navazováním a ukončováním spojení, posíláním informací o doručení nebo čekáním na ztracený packet

Poznámka k TCP a UDP na závěr. **Přenos dat pomocí TCP protokolu může mít negativní vliv na doručování (velkých) UDP packetů**[20]. Rozhodně tedy není dobrý nápad používat kombinaci TCP a UDP pro přenos stavu hry (například by se nabízelo posílat informace o dosažených bodech hráčů pomocí TCP - tyto informace nemusí dojít tak rychle, ale hlavně když jsou správně - a pozice spritů pomocí UDP).

3.3.1 Jak a jaká data se budou přenášet

Jak jsem psal v úvodu této kapitoly, komunikace která se nebude týkat stavu běžící hry se bude přenášet pomocí TCP. Půjde o přihlašování k serveru, informace o spuštěných hrách a administraci serveru.

Všechna data se pomocí TCP/IP přenášejí jako proudy bajtů. Jak všechny přenesené informace převést na bajty? Jedno z možných řešení je posílat serializované objekty. **Serializace** objektů sice není nejefektivnější co do objemu dat, to nás ale v tomto případě netrápí. Horší problémem je, že aby klient dokázal serializované objekty deserializovat, musí mít informaci o stejné verzi třídy serializovaného objektu jako server. Kdyby se třídy objektů posílaných od serveru ke klientovi změnily (po aktualizaci serveru), nebyl by klient schopný se serverem vůbec komunikovat. Dokonce by jsme mu nebyli schopni poslat ani informaci o tom, že je klient zastaralý a měl by se aktualizovat.
Z tohoto důvodu bude nejlepší **pro komunikaci která nemusí běžet v reálném čase používat XML**. XML přináší ještě větší nároky na objem přenesených dat než serializace, ale to u komunikace která nemusí probíhat v reálném čase není problém. Změna XML souboru (například přidání nových elementů) nezpůsobí klientovi žádný problém. Klient si z XML dokáže vybrat informace, kterým rozumí. XML může být v aplikaci uložen jako datový typ String, který se snadno pomocí standardních metod Javy převede na bajty a zpět na String.

Kódování XML dokumentu bude UTF-8.

Pro přenos stavu hry se XML ani jiná textová reprezentace nehodí. Dokonce i serializace datových objektů se může ukázat jako příliš náročná. Rozhodně platí, že čím méně dat se bude přenášet, tím lépe. V základní verzi multiplayerové hry Deadloids budou moci být ve hře až **4 hráči**. **Každý hráč** může vystřelit **až 5 raket** (toto číslo se může změnit). Meteorit se může rozpadnout až na **12 malých meteoritů** (nejdříve na 3 menší a každý pak na další 4 nejmenší). A do toho ještě může přiletět **1 Ufo**. Teoreticky se tak může ve hře pohybovat **až 37 spritů**. A pokud by se hra někdy rozšířila tak, že by se asteroidy mohli rozpadnout ještě jednou na další 4, bylo by to **až 73 spritů** (4 x Spaceship + 5 x 4 rakety + 1 x Ufo + 48 asteroidů).

Namísto serializace implementované Javou bych mohl implementovat vlastní, efektivnější serializaci. Každému objektu, který by se měl přenášet po síti by se mohla implementovat metoda pro serializaci (převedení všech dat, které se musí poslat po síti, do pole bytů) a deserializaci (vytvoření nové instance z pole bytů).

Při serializaci by se mohla využít i jednoduchá **ztrátová komprese**. Například poloha spritu se ukládá v třídě Vector2D, která používá pro souřadnice X,Y typ double, který je velký 64 bitů. Nicméně na obrazovce se sprit může pozicovat s přesností na pixely. Navíc bude mít obrazovka rozměr jen několik stovek pixelů. K tomu by bohatě stačil typ short (16 bitů). A najednou je potřeba přenést po síti 4x méně místa!

Ztrátová komprese sebou přináší i nevýhody. Především může mít neblahý vliv na predikci na straně klienta (viz *3.3.4 Návrh rozhraní klienta)*. Nebo kdybychom se rozhodli na monitoru s vysokým rozlišením vytvořit klienta s dvojnásobnou hrací plochou, pak by se sprity nepohybovaly

po jednom pixelu[11], ale po dvou pixelech, což by při pomalých pohybech spritu mohlo působit rušivým trhavým dojmem (tento problém částečně řeší predikce, viz dále).

Bezeztrátová komprese se může s výhodou použít pro přenos dat od klienta k serveru. Klient bude v zásadě posílat serveru jen seznam stisknutých kláves, kterým hráč ovládá hru. Pokud jich bude maximálně osm, není důvod posílat více než jeden bajt informací (každý bit indikuje stav jedné klávesy – stisknuto/nestisknuto).

V první verzi programu Deadloids bude použita standardní serializace objektů jazyka Java. Zjednoduší a zrychlí to programování a optimalizace se může kdykoliv provést ve chvíli, kdy bude hotový funkční prototyp programu.

3.3.2 Predikce

Návrhy rozhraní serveru a klienta jdou ruku v ruce. Co odesílá server, to přijímá klient a naopak.

Jako nejjednodušší varianta spolupráce serveru a klienta se nabízí varianta, kde server počítá veškerou logiku hry, spočte pozice spritů a události ve hře (například střety spritů) a pošle je klientovi, který je jen zobrazí. Server je tedy autoritativní ohledně pozice spritů – klient nerozhoduje o tom kde se sprity nachází, ani o své vlastní herní postavě (vesmírné lodi). Mohl by totiž snadno posílat serveru podvržená data a podvádět. Takový přístup má ale zřetelné nevýhody.

K tomu, aby se lidskému oku zdála animace plynulá, musí se překreslovat alespoň **rychlostí 25 obrázků za vteřinu**[22]. To znamená, že by server musel odeslat a klient přijmout stav hry nejméně každých 40 ms. Navíc by muselo být zaručeno, že data dorazí vždy ve stejném časovém intervalu. To ale u přenosu po TCP/IP síti zaručit nelze. Mohlo by se tak snadno stát, že hráč několik desítek milisekund nedostane žádnou aktualizaci od serveru (hra by se zastavila) a pak dostane naráz několik aktualizací (hra najednou z ničeho nic zrychlí).

Řešením je predikce stavu hry na straně klienta. Klient nebude jen hloupý terminál, který zobrazuje co mu přijde od serveru, ale bude se snažit predikovat a zobrazovat stav hry, dokud mu nedojde aktualizace ze serveru (maximálně 0.5 – 1 vteřinu) a pak podle ní synchronizuje svůj stav[21].

Sprit ovládaný hráčem by měl být na straně klienta řízen vstupem přímo od uživatele (a posílat vstup na serveru) a až dle odpovědí serveru korigovat svou pozici. Hráč tak získá dojem plynulejšího ovládání své postavy (Spaceshipu).

K docílení co nejpřesnější predikce by mělo být **chování světa deterministické**. To se ale nedá zaručit u hráčů. Navíc, nejen díky kompresi, se může predikce na straně klienta s výpočty na straně serveru mírně lišit. Ovšem i drobné odchylky mohou mít fatální následky. Například by se mohlo stát, že u jednoho hráče klient predikuje srážku soupeře s asteroidem, zatímco soupeř ještě stihne o kousek se svou vesmírnou lodí uhnout a srážce se vyhnout. Hráč by tak mohl pozorovat výbuch svého protihráče a asteroidu a několik milisekund poté jeho „znovuzrození“. Nejjednodušší obranou proti takovýmto nepříjemnostem bude, když se **události** (například srážka s asteroidem**) budou posílat od serveru ke klientům.** Klienti je nebudou predikovat ale pouze zobrazovat.

Díky predikci bude dostačovat, pokud bude docházet k synchronizaci mezi serverem a klientem zhruba pětkrát za sekundu. Tj, packet může cestovat od klienta k serveru 100 ms a od serveru ke klientovi dalších 100 ms.

11 Pixel je nejmenší bod na obrazovce, který dokáže monitor zobrazit. Čím vyšší rozlišení monitoru, tím více pixelů dokáže zobrazit pixelů a pixely jsou menší.

Predikce se týká pouze stavu GameWorld (respektive spritů, které GameWorld bude obsahovat). Stav GameStrategy (tj. například skóre hráčů) nebude nutné predikovat a bude se aktualizovat jen když přijdou nová data ze serveru.

3.3.3 Návrh rozhraní Serveru

MasterServer bude třída, která se bude starat o úvodní komunikaci s klientem. MasterServer bude přijímat a odesílat zprávy v XML. Komunikačním protokolem bude TCP. Bude se starat o autorizaci uživatele, správu serveru administrátorem atd. Pokud administrátor vytvoří novou hru, MasterServer spustí v novém vlákně **GameServer**, který se bude starat o běh hry a bude komunikovat s klienty přes UDP. Pokud hráč rozhodne připojit k GameServeru, MasterServer požádá GameServer o přijetí nového hráče a pokud to GameServer povolí, předá hráči spojení na GameServer (port, na kterém GameServer poslouchá).

GameServer bude udržovat instanci hry (GameStrategy), předávat informace MasterServeru o stavu hry, připojených hráčích atp. a starat se o komunikaci s klienty. Zatímco MasterServer bude jen jeden, GameServerů bude tolik, kolik bude běžet her.

GameServer musí klientovi posílat:

- GameWorld – pozici a rychlost spritů, jejich život atd.
- StrategyModel – informace o připojených hráčích, jejich skóre, restartu hry atp.
- Události – střety spritů

Události by měly přicházet ve stejný okamžik s GameWorld. K události dojde jen v jeden okamžik, ale posílat jí jen s jedním přenosem GameWorld by nebylo rozumné, protože by mohlo dojít k její ztrátě a klient by o ni přišel (neuviděl by výbuch). Jedním z řešení by mohlo být vkládat události do GameWorld jako jakési neviditelné sprity s omezenou dobou životnosti. V naší hře se události budou projevovat změnou stavu spritů (při střetu rakety s asteroidem sprity přejdou do stavu „výbuch“). Pokud sprity necháme v tomto stavu rozumně dlouhou dobu (např. jednu vteřinu), bude to dostatečný čas pro doručení události ke klientovi. Kvůli správné synchronizaci klienta a serveru by si měl sprit pamatovat kdy k změně stavu došlo.

StrategyModel se nemusí přenášet tak často jako GameWorld. Aby se zbytečně nezatěžovala síť, může se StrategyModel posílat 1x za vteřinu.

GameServer tedy bude posílat dva hlavní druhy zpráv – GameWorld a StrategyModel. Aby klient rozlišil, která zpráva mu došla, budou zprávy zapouzdřeny do obálky **GameMessage** s příznakem typu posílaných dat. GameServer bude posílat ještě další informace, například o ukončení hry – aby se hráč odpojil a nečekal zbytečně na spojení.

3.3.4 Návrh rozhraní klienta

Obdobně jako na straně serveru, bude i klient tvořen dvěma hlavními třídami.

MasterClient bude komunikovat přes TCP se MasterServerem a **GameClient** přes UDP s GameServerem.

Na straně klienta poběží hra obdobně jako na straně serveru, pouze při příchodu dat od serveru aktualizuje svůj stav a stav GameWorld. Hra na straně klienta nebude vytvářet žádné události, pouze je přejímat z updatů od serveru. Klient tak bude jen posouvat sprity na základě jejich pozice a rychlosti (viz *3.3.2 predikce*). K tomu se dá s výhodou použít globální stav spritů

(viz 3.1.4 *Konečný automat (FSM)*), který bude sprity pouze posouvat, zatímco vlastní stav spritů se vyřadí ze hry (nebude se jeho kód provádět). GameClient bude odesílat na server pouze vstupy od hráče (řídící pohyb jeho lodi).

Aby se eliminoval problém s nestejnoměrnou rychlostí přenosu dat, bude součástí každé zprávy „časové razítko“, které umožní lepší synchronizaci mezi serverem a klientem. Klientovi budou chodit zprávy staršího data, než ve kterém se klient sám bude nacházet (díky predikci). Klient tak bude muset aktualizovat svůj model, dopočítat posun podle rozdílu času mezi daty od serveru a vlastním časem a pokračovat v predikci. Klient si také bude pamatovat „nejvyšší“ časové razítko, které od serveru dostane. Pokud dorazí některá data se starším časem, bude ho klient ignorovat (UDP nezaručuje stejné pořadí příchozích dat jako odchozích).

3.3.5 Vztah mezi třídami GameServer, GameClient a GameStrategy

Server by měl umožnit hráči hrát nejen MultiGameStrategy, ale i SingleGameStrategy (a případně další implementace GameStrategy).

Třídy GameClient a GameServer budou pracovat s rozhraním GameStrategy, takže budou schopné přenášet po internetu jakoukoliv jejich implementaci.

GameClient bude implementovat rozhraní GameStrategy a tak jej bude možné použít všude stejně, jako by se jednalo o skutečnou strategii a ne jen internetového klienta.

GameStrategy se také bude používat pro posílání příkazů hráče z grafického zobrazení do GameStrategy. Příkazy hráče bude reprezentovat třída **PlayerCommands**.

3.4 Grafické zobrazení (View)

Hra se bude vykreslovat pomocí Java 2D API. Java 2D API je množina tříd pro vykreslování grafiky ve 2D, zpracovávání obrázků a textu. Zvládá vykreslovat obrázky s průhledným pozadím, otáčet je, nebo měnit jejich velikost[23].

Za snímkovou frekvenci pro plynulou grafiku (při které lidské oko nerozezná jednotlivé obrazy) se považuje 25 fps (frames per second). Zhruba 60 fps se bere jako naprosto dostatečné pro bezproblémový chod hry[22]. Díky predikci (viz 3.3.1 Predikce) by neměl být problém mít k dispozici posun spritů 60x za vteřinu. Otázkou zda Java 2D je dostatečně rychlé na zobrazení alespoň 25 fps se zabývá kapitola *4.3 Testování grafického výstupu*.

Herní plocha bude mít pevně daný rozměr, například 600x600 px (přesný rozměr se doladí během testování hry). Vykreslování by však mělo být naprogramováno tak, aby nebyl problém hru zobrazit v libovolném násobku základního rozměru (na monitorech s vysokým rozlišením by mohlo být okno příliš malé a naopak).

Díky MVC návrhu nebudou třídy reprezentující výstup nijak závislé na zbytku hry, neměl by tak být problém jeden grafický výstup jednoduše zaměnit za jiný (například za výstup pro display zařízení s androidem).

Grafické zobrazení se bude zapisovat do **double bufferu**, díky kterému se bude grafika překreslovat bez blikání[24]. Jako buffer bude použita třída java.awt.image.BufferedImage. Ve hře by bylo možné použít i triple buffering. Jeden buffer pro GameWorldView a druhý pro GameStrategyView (třetí buffer je vyrovnávací paměť obrazovky). Je však otázkou, zda by to přineslo výrazné zlepšení vykreslování. První verze Deadloids si určitě vystačí s double bufferem.

Třída, která se bude starat o inicializaci a zobrazování double bufferu se bude jmenovat **DeadloidView**, bude v balíčku pojmenovaném java2D javax.swing.JPanel. Tato třída také bude zachytávat vstup klávesnice pro ovládání hry. Bude implementovat rozhraní **GameView**, které bude poskytovat metody pro přístup k vstupu uživatele. Třída **Java2D** se bude starat o zobrazení a přístup k menu, GameView a všemu co souvisí se zobrazením rozhraní klienta. Java2D bude implementací rozhraní **AppView** . Třída Java2D bude potomkem třídy javax.swing.JFrame.

Ve hře budou zobrazeny různé animace, například animace výbuchu. Animace se bude zobrazovat pomocí série po sobě jdoucích obrázků. Který obrázek je zrovna na řadě bude počítat třída starající se o zobrazení (GameWorldView). Pokud nepůjde o nekonečnou animaci (jdoucí stále dokola), bude mít animace svůj začátek, dobu trvání a konec. To znamená, že v animaci trvající jednu vteřinu, která se bude skládat z dvaceti obrázků se každý obrázek zobrazí na dobu 0.05 vteřiny. Pro plynulý stav animace tak není potřeba, aby se na serveru vypočítával aktuální obrázek v animaci a klient na něj čekal. Klientovi postačí pro výpočet aktuálního obrázku v animaci znát její počátek.

3.4.1 Grafické zobrazení WorldView

Třída **WorldView** se bude starat o zobrazení hry. Na pozadí bude zobrazen obrázek vesmíru a v popředí sprity (Spaceshipy, Asteroidy atp.). Data pro zobrazení si bude brát z modelu GameWorld.

Třída WorldView bude pracovat s modelem, který bude získávat od GameStrategy vždy znovu před každým počátkem vykreslování. A to kvůli tomu, že při síťové hře bude klient sestavovat vždy nový GameWorld na základě dat získaných od serveru. Klient tedy bude mít dva modely: jeden, jehož stav predikuje a který se zobrazuje ve WorldView a druhý, který zrovna sestavuje z dat získaných od serveru a kterým po sestavení nahradí ten první. Po nahrazení starého modelu novým musí WorldView získat odkaz na tento nový model.

3.4.2 Grafické zobrazení StrategyView

Implementace rozhraní **StrategyView** se bude starat o zobrazení modelu StrategyModel. Bude zobrazovat skóre, počet zbývajících životů, případně hlášku o čekání na začátek hry.

Je jasné, že ty to informace se budou pro různé herní strategie lišit. Například hra více hráčů, oproti hře jednoho hráče, bude zobrazovat skóre až 4 hráčů, ale nebude zobrazovat informace o zbylých životech (neboť jich bude ve této hře neomezeně). Z tohoto důvodu bude muset existovat více implementací StrategyView, pro každou herní strategii alespoň jedna.

3.4.3 Ozvučení událostí

Pro lepší požitek ze hry bude hra ozvučena. Po odstartování hry poběží ve smyčce zvuková kulisa hry. Další zvuky budou doprovázet různé herní události, například výbuch rakety. O zvuky se budou starat třídy zodpovědné za View. Třída spustí zvuk při prvním zaregistrování nějaké události, zároveň by si měla zapamatovat že tuto událost už zaregistrovala.

3.5 Zachytávání a zpracování událostí myši a klávesnice

Hráč bude svou Spaceship ovládat pouze klávesnicí. Události myši budou ignorovány (čímž se nám to zjednodušuje). Hráč bude moci měnit směr a rychlost své lodi a střílet rakety.

Události klávesnice bude zachytávat třída zobrazující herní plochu (double buffer BufferedImage, viz *3.4 Grafické zobrazení (View)*). Zachycené klávesy převede na objekt reprezentující požadavky hráče, který předá třídě GameWorld. Nebude jí tak předávat informaci o tom, která klávesa byla stisknuta (například šipka vlevo), ale informaci, že hráč chce zatočit doleva. To v budoucnu umožní hráčům nastavit si klávesnice pro ovládání hry dle libosti.

Kromě ovládání své lodi, bude moci hráč ovládat GameStrategy. Například ve hře pro jednoho hráče bude moci klávesou P hru pozastavit. Tato informace bude znovu předána v podobě objektu reprezentujícím požadavek hráče, tentokrát pro rozhraní GameStrategy.

Informace o aktuálně stisknutých klávesách (respektive o požadavcích hráče) budou předávány po každém x-tém průchodu zobrazení hry a vždy při změně (když hráč stiskne nebo uvolní klávesnici). A to především z důvodu, jakým budou data doručována od klienta k serveru. Protokol UDP (viz *3.3 Komunikace mezi serverem a klientem)* totiž není spolehlivý a tak by se mohlo stát, že by server tuto změnu vůbec nezaregistroval.

GameWorld bude využívat událostí klávesnice k predikci svého stavu. Nebude čekat na odpověď od serveru, čímž se zaručí okamžitá reakce vesmírné lodi na hráčův požadavek.

V momentě, kdy GameWorld získá aktualizovaný stav hry od serveru, bude již několik kroků (díky predikci) napřed. GameWorld si spočte časový rozdíl mezi získaným stavem hry a jeho aktuálním predikovaným časem a tento rozdíl použije pro novou predikci tak, aby se dostal do stejného časového bodu, ve kterém se nachází.
V tento okamžik je však problém s pozicí hráčovi lodě. Tu nelze predikovat pouze na základě její pozice a rychlosti, musí se do úvahy vzít i požadavky na změny směru od uživatele, které se odeslaly na server, ale ještě nebyly serverem zpracovány.

Aby to bylo možné, měl by si klient pamatovat posledních X vstupů od uživatele a to včetně času, kdy je od View obdržel. Při aktualizaci GameWorld na základě dat získaných od serveru se podívá na poslední čas aktualizace, který server získal, smaže všechny aktualizace staršího a stejného data a všechny novějšího data znovu použije pro predikci. Na konci této predikce by měla loď zůstat ve více méně stejné pozici, jako před zahájením aktualizace.

4. Testování

Před začátkem programování aplikace je dobré zjistit, jaké možnosti poskytuje dnešní hardware. Nejdůležitějšími parametry pro programování hry Deadloid bude rychlost s jakou lze posílat data mezi klientem a serverem, rychlost vykreslování hry na obrazovku (FPS) a rychlost výpočtu stavu hry (pohybu spritů).

Vzhledem k výkonu dnešních počítačů lze předpokládat, že u hry jako je Deadloid by neměly mít dnešní počítače s výkonem problém. Bylo by však dobré otestovat, zda je použitá implementace Java2D dostatečně optimalizovaná na vysoké FPS.

Problém může být s rychlostí sítě. Že lze hrát hry po síti v reálném čase už je dnes také jasné, otázkou však je, kolik dat si můžeme dovolit přenášet, aby hra byla hratelná.

Čas potřebný pro výpočet stavu hry (vlastní logiky hry) by měl být nejméně nákladnou operací (Deadloid nebude obsahovat žádnou vyspělou a náročnou umělou inteligenci), proto jej nebudu testovat. Vlastně by to ani nebylo možné, protože dokud nebude hra hotová, není moc co testovat.

4.1 Specifikace hardwaru a softwaru

K testování použiji kancelářský notebook Lenovo ThinkPad Edge 14" černý 0199-67G. Jedná se o levný notebook, jehož prodejní cena se pohybovala na začátku roku 2011 kolem 11.000 Kč včetně DPH. Obsahuje dvoujádrový procesor AMD Athlon II P360 s frekvencí 2,3 GHz, 4 GB paměti DDR3 1066 MHz (2 x 2 GB) a grafickou kartu ATI Radeon HD3400. Pro komunikaci po síti může využít Ethernet LAN 10/100/1000 Mbit/s nebo Intel WiFi 1000 BGN 802.11b/g/n. V notebooku je pevný disk o velikosti 500 GB s rychlostí 7200 RPM a rozhraním Serial ATA.

Ze specifikace je vidět, že se nejedná o nikterak výkonný počítač. Lze tedy předpokládat, že v případě úspěšného výkonnostního testu hry na tomto počítači, bude hra fungovat na většině dnes prodávaných počítačových sestavách a noteboocích.

Kromě hardwaru má na výkon nezanedbatelný vliv také softwarová výbava počítače. Především pak na výkon grafické karty mají velký vliv ovladače. Ze zkušenosti mohu říct, že první verze ovladačů pro výše zmíněnou grafickou kartu v Linuxu byla pro vytvoření 2D hry nepoužitelná. Počítač jsem totiž koupil v době, kdy byla jeho grafická karta ještě poměrně nová a proprietární linuxové ovladače na ní nebyly připravené. Zhruba po půl roce od koupě počítače byla vydána novější aktualizace ovladačů, která výkon v Linuxu výrazně zlepšila.

Z tohoto důvodu jsem se rozhodl testovat grafický výstup na 2 operačních systémech. Na OS dodávaném s počítačem, Microsoft Windows 7 Home Premium 64bit, a na linuxové distribuci OpenSuSE 12.1 s proprietárními ovladači grafické karty (amd-driver-12.6-legacy.x86_64). V Linuxu budou testovány 2 grafická rozhraní, KDE a LXDE.

Pro testování v KDE a LXDE jsem se rozhodl z toho důvodu, že se jedná o zástupce nejpokročilejšího a jednoho z nejrychlejších desktopových prostředí v Linuxu.
LXDE (Lightweight X11 Desktop Environment) je rychlé a hardwarově nenáročné open source prostředí napsané v jazyku C s grafickým prostředím vytvořeným pomocí knihoven GTK+[25].
Zatímco LXDE nenabízí mnoho funkcí, KDE je pravým opakem. Současně vyvíjená verze KDE 4.x je založena na Qt 4.x, která je vydána pod licencí GPL[26]. KDE nabízí velké možnosti nastavení, grafických efektů atp., ale je také jedno z hardwarově nejnáročnějších linuxových desktopových prostředí.

Vliv softwaru na rychlost přenosu dat po síti není tak významný, proto jsem testoval přenos dat pouze v Linuxu. Jako server jsem použil vlastní počítač připojený do sítě přes Wifi k modemu UPC (10/1 MiB). Klienta jsem potom spustil (přes ssh) na virtualizovaném serveru v práci. (Kvůli na stavení firewallu jsem nemohl pracovní server použít jako server). Na přiloženém CD je v souboru /tests/traceroute.txt zaznamenán výstup z programu traceroute těsně před spuštěním testování.

4.2 Program R pro analýzu statistických dat

R je jazykem a prostředím pro statistické výpočty a grafiku. Jedná se o GNU projekt podobný jazyku a prostředí S vyvinutému v Bell Laboratories (dříve AT&T, nyní Lucent Technologies) Johnem Chambersem a kolegy.[27] R lze nainstalovat jak v OS Windows, tak Linux. R jsem použil pro zpracování naměřených dat.

RKWard je svobodné rozhraní pro statistický a programovací jazyk R. Vyvíjeno je pro prostředí KDE, ale lze ho používat i v dalších prostředích, existuje také port pro Windows.[28] RKWard není nutné pro použití programu R, ale usnadňuje jej.

K základní instalace programu R bylo potřeba ještě doinstalovat následující dodatečné balíčky:

- vsePackage
- MASS
- corpcor

Balíček vsePackage je k dostání ze stránek[31] autorů knih o statistice [29] a [30]. Z těchto knih jsem také čerpal při následující analýze naměřených dat.

Balíček vsePackage je pro Windows ke stažení v binární formě, v Linuxu se instaluje ze zdrojových kódů (stačí jedním příkazem *install.packages(repos=NULL, pkgs="vsePackage_2.0.tar.gz")*).

Balíček MASS lze stáhnotu ze stránek projektu cran[32] a nainstalovat příkazem *install.packages(repos=NULL, pkgs="MASS_7.3-14.tar.gz")*.

Balíček corpcor se nainstaluje z repozitářů příkazem *install.packages(corpcor)*.

4.3 Testování grafického výstupu

Projekt použitý pro testování grafického výstupu je na CD v adresáři /source/4.3TestGrafiky/. V projektu jsem vytvořil všechny třídy a rozhraní zmíněné v předchozím textu, ačkoliv jich mnoho z nich nebylo k ničemu použito a zůstali prázdné. Navíc jsem vytvořil třídy **GraphicsTestStrategy** a **GraphicsTestStrategyView**, které implementují speciální „herní strategii“ - test grafiky.

GraphicsTestStrategy lze spustit ve třech módech pojmenovaných random, deadloid a linear. Mód lze vybrat při spuštění pomocí argumentu příkazové řádky *--test*. Význam jednotlivých módů je popsán v následujících kapitolách.

Výsledky testu jsou zkresleny několika faktory, kvůli kterým by se měly brát pouze jako orientační. Například není známo, jaké všechny další programy běžely na počítači v době testu a ovlivňovaly tak jeho výkon.

4.3.1 Mód random

V tomto módu GraphicsTestStrategy generuje náhodný počet spritů od každého druhu v rozmezí od 0 do 150, které se pak vykreslují. Protože je ve hře 6 různých druhů spritů, může se vygenerovat a vykreslit v každém kroku 0 až 900 spritů.

Následující tabulka přibližuje rozdělení součtu náhodně vygenerovaných spritů v jednom kroku pro různé prostředí:

Prostředí	**Min.**	**1st. Qu.**	**Median**	**Mean**	**3rd Qu.**	**Max.**
Win	81	380	454	453.4	527	798
KDE	126	381	454	454.0	526	802
XFCE	81	377	452	451.9	525	802

Tabulka 1: Rozdělení součtu náhodně vygenerovaných spritů

Naměřená data jsou uložena na přiloženém CD v souboru /tests/graphicsRand.csv. Výpočty jsou uloženy v souboru /tests/graphicsRand.R.

Tento test by měl dát odpověď na otázku, jak rychle klesá FPS v závislosti na počtu spritů a zda existuje závislost změny rychlosti na typu spritu. Vykreslení velkého spritu určitě trvá déle než malého, ale jsou rozdíly mezi sprity ve hře natolik zásadní, aby byly nezanedbatelné?

Jak bylo popsáno v kapitole *3.4 Grafické zobrazení (View)*, ideální rychlost pro hry je přes 60 FPS, dostatečná je pak 25 FPS. Flashové animace hojně používané na webu si vystačí dokonce s 12 FPS.

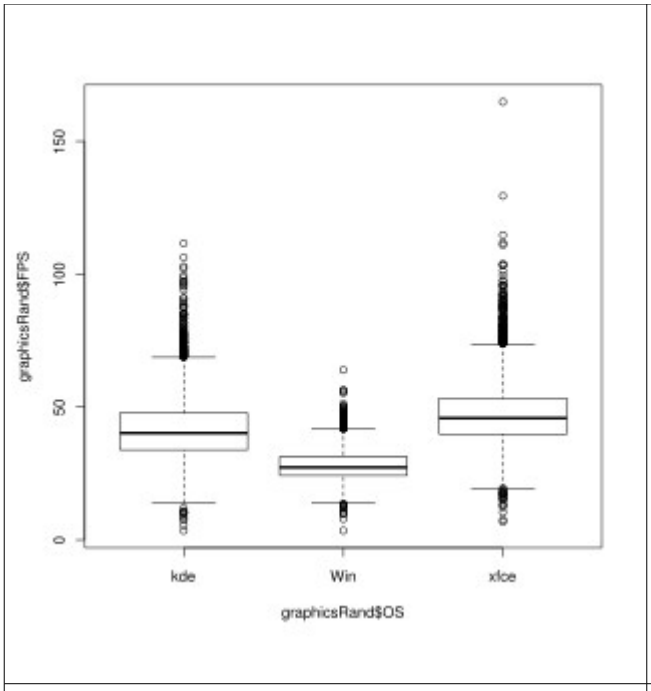
Obrázky 4.3.1 a 4.3.2 ukazují boxploty a histogramy naměřených FPS pro Windows, KDE a XFCE. Z obrázků je dobře vidět, že si nejlépe vede prostředí XFCE, po něm následuje KDE a nejhůře je na tom prostředí Windows 7. FPS jsou samozřejmě ovlivněny počtem náhodně generovaných spritů, takže z boxplotů nelze usuzovat na rychlost hry Deadloids (o tom nám více poví testování v módu deadloid, viz dále).

Pro zkoumání naměřených dat by bylo ideální, aby se rozdělení FPS řídila normálním rozdělením. Shapiro-Wilkův test normality však normalitu zamítl. Proto jsem pro formální posouzení rozdílů v rozděleních FPS napříč různými systémy použil Kolmogorův-Smirnovův test. Ten na 5% hladině významnosti prokázal, že se rozdělení FPS ve všech prostředích liší.

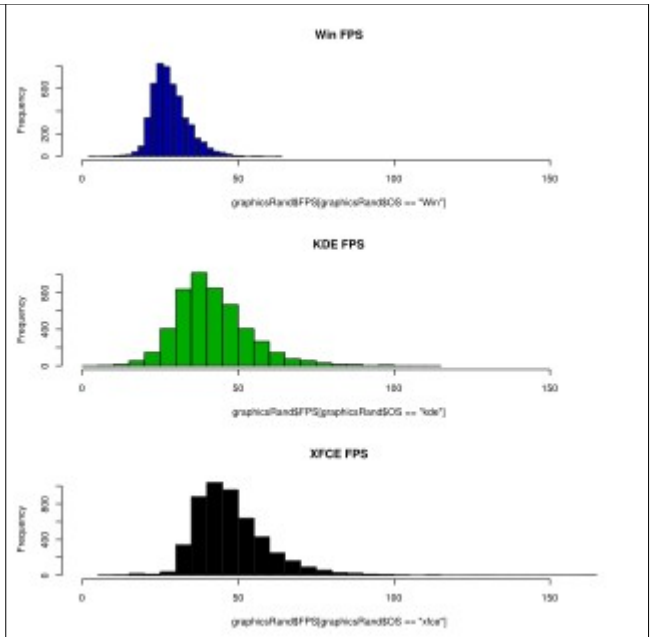
Následně jsem sestavil lineární model pro regresní analýzu a spočítal koeficienty pro jednotlivé druhy spritů a OS.

```
model <- lm(FPS ~ OS + Spaceship + UFO + Rocket + Asteroid1 + Asteroid2 + Asteroid3
+ OS:Spaceship + OS:UFO + OS:Rocket + OS:Asteroid1 + OS:Asteroid2 + OS:Asteroid3,
data=graphicsRand,x=T)                                    (1)
```

Výsledné odhady koeficientů jsou v tabuce 2: Odhadnuté koeficienty regresní funkce.



Obr.4.3.1: Boxploty FPS pro různé OS

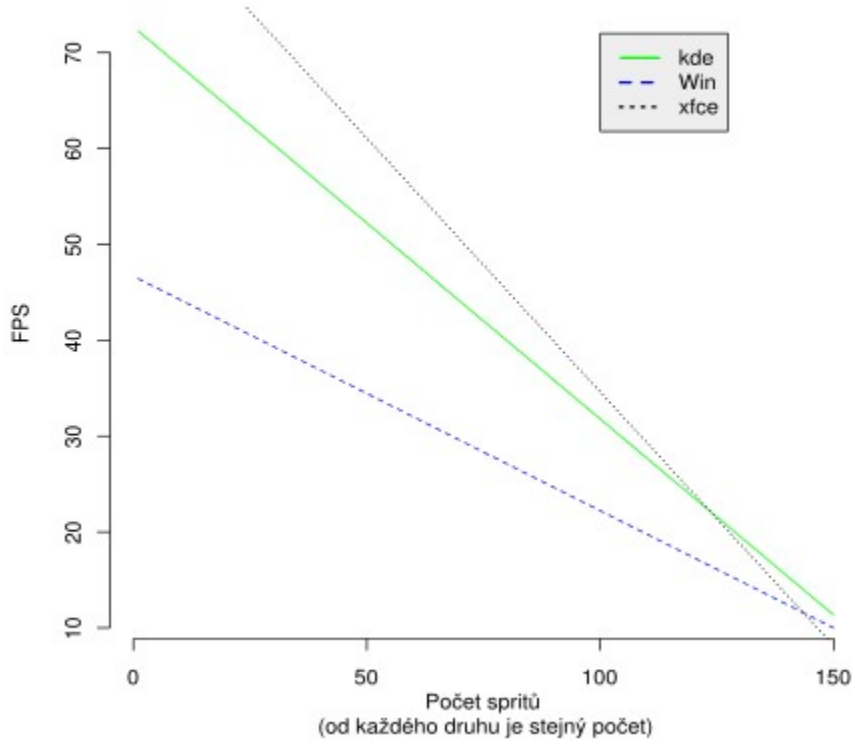


Obr. 4.3.2: Histogramy FPS pro různé OS

(Intercept)	72.679219	OSWin:Spaceship	0.035631
OSWin	-25.997915	OSxfce:Spaceship	-0.033323
OSxfce	14.873690	OSWin:UFO	0.023050
Spaceship	-0.112736	OSxfce:UFO	-0.018004
UFO	-0.058037	OSWin:Rocket	0.015146
Rocket	-0.032416	OSxfce:Rocket	-0.015341
Asteroid1	-0.103061	OSWin:Asteroid1	0.043058
Asteroid2	-0.060260	OSxfce:Asteroid1	-0.025615
Asteroid3	-0.042469	OSWin:Asteroid2	0.026043
		OSxfce:Asteroid2	-0.018639
		OSWin:Asteroid3	0.021429
		OSxfce:Asteroid3	-0.009611

Tabulka 2: Odhadnuté koeficienty regresní funkce

Z výsledků lze například vyčíst, že testování je ve Windows oproti KDE o cca 26 FPS pomalejší, zatímco XFCE je oproti KDE o cca 14.9 FPS rychlejší. Přidání jednoho Spaceship spritu pak zpomalí (nezávisle na prostředí) vykreslování o 0.112736 FPS. V prostředí Windows je pak zpomalení díky Spaceshipu o 0.035631 FPS pomalejší v porovnání s KDE atd.



Obr. 4.3.3: Predikce závislosti FPS a počtu spritů

Již není překvapením, že největší vliv na výsledek má prostředí, ve kterém test běžel. Význam jednotlivých spritů na FPS je poměrně malý, ale nezanedbatelný.

Další zajímavé grafy, ověřování předpokladů a výpočty lze najít v přiloženém souboru na CD /tests/graphicsRand.R.

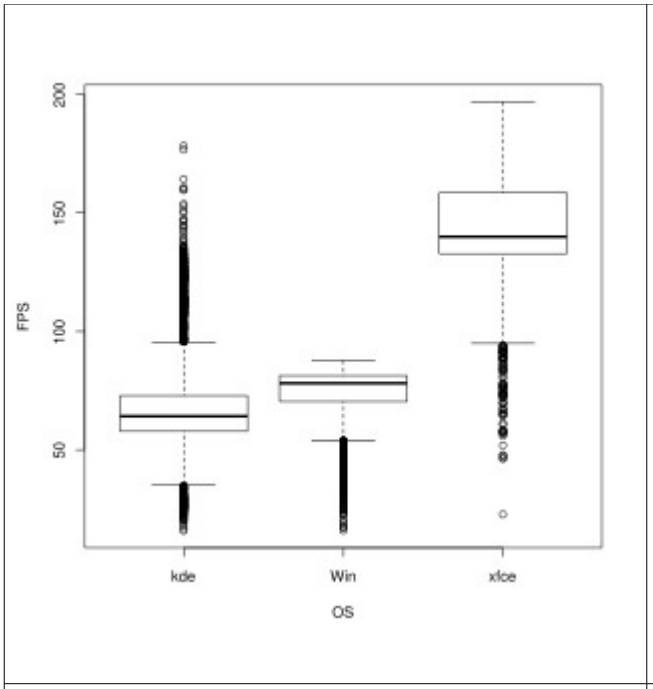
4.3.2 Mód deadloid

V kapitole *3.3.1 Jak a jaká data se budou přenášet* se odhadl maximální počet spritů na 37, respektive 73 (4 x Spaceship, 1 x Ufo, 20 x Rocket a 48 x Asteroid3). Z druhého odhadu pak vychází testovací mód deadloid.

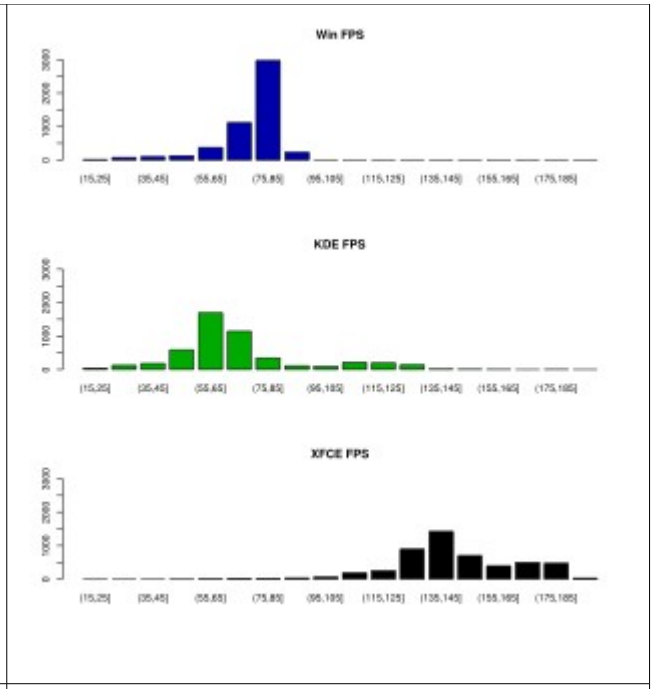
Naměřená data jsou uložena na přiloženém CD v souboru /tests/graphicsDeadloid.csv. Výpočty jsou uloženy v souboru /tests/graphicsDeadloid.R.

Tento test by měl dát odpověď na otázku, kolik FPS hra s tolika sprity zvládne.

Jak bylo popsáno v kapitole *3.4 Grafické zobrazení (View)*, ideální rychlost pro hry je přes 60 FPS, dostatečná je pak 25 FPS. Flashové animace hojně používané na webu si vystačí dokonce s 12 FPS.



Obr.4.3.4: Boxploty FPS pro různé OS



Obr.4.3.5: Histogramy FPS pro různé OS

Z obrázků 4.3.4 a 4.3.5 je patrné, že s rychlostí FPS je na tom výrazně nejlépe XFCE, následuje Windows 7 a těsně za ním prostředí KDE.

Prostředí	**Min.**	**1st. Qu.**	**Median**	**Mean**	**3rd Qu.**	**Max.**
Win	15.93	70.38	78.12	74.34	81.32	87.70
KDE	15.88	57.79	64.18	69.19	72.88	178.30
XFCE	22.77	132.80	139.90	144.10	158.50	196.90

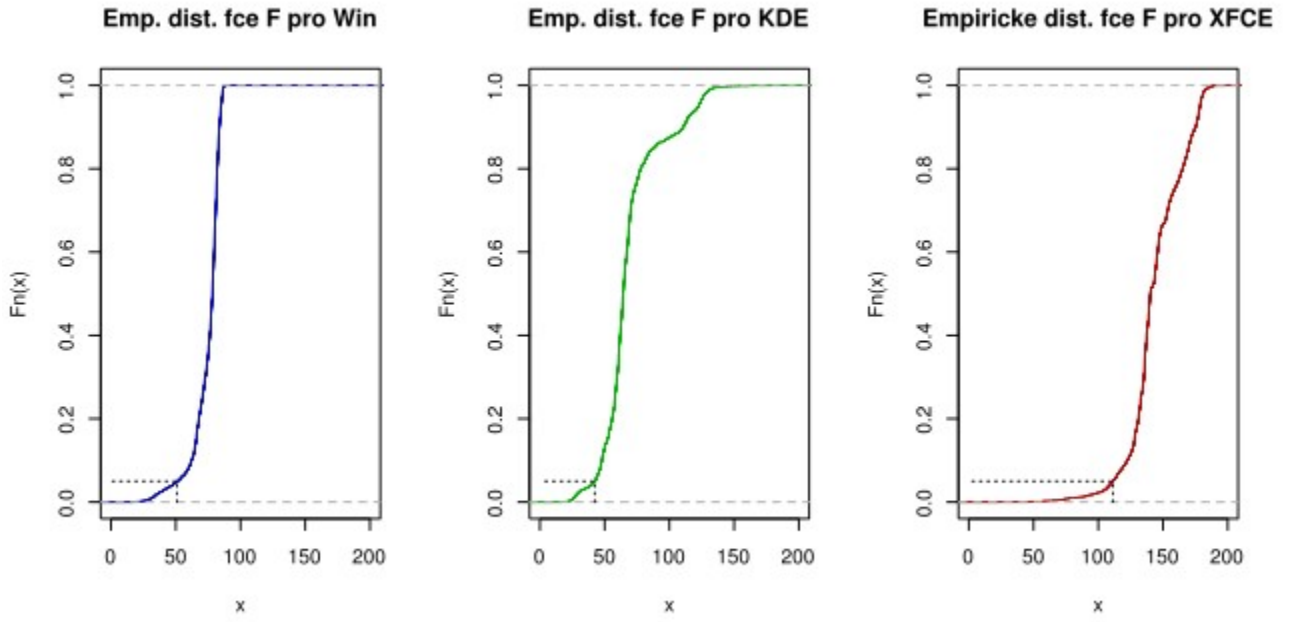
Tabulka 3: Rozdělení FPS dle prostředí

Prostředí	**95 % > FPS**
Win	51.03
KDE	42.34
XFCE	111.32

Tabulka 4: Hodnota, kterou překračuje 95 % měření

Pokud uvážíme, že dostatečná rychlost pro hry je 25 FPS, pak i nejpomalejší prostředí KDE v 95 % případů tuto hodnotu překračuje o 17.34 FPS.

Z dat lze také zjistit, že pouze v 0.72 % případů klesla rychlost FPS v prostředí KDE pod 25 FPS. U Windows je to pak 0.3 % a v XFCE 0.02 %. V žádném prostředí pak neklesla hodnota FPS pod 12 FPS (viz minimální hodnotu v tabulce rozdělení FPS dle prostředí výše).



Obr. 4.3.6: Empirické distribuční funkce s vyznačením 5% dolní hranice FPS

Prostředí	**Kvart. rozpětí**	**Směrodatná odchylka**
Win	15.1	22.6
KDE	10.9	11.0
XFCE	25.7	20.9

Tabulka 5: Tabulka kvartilových rozpětí a směrodatných odchylek

Z tabulky kvartilových rozpětí a směrodatných odchylek je vidět, že výrazně nejvyšší kvartilové rozpětí má prostředí XFCE. To však také dosahuje nejvyšších hodnot FPS, proto nám nejméně vadí. Směrodatná odchylka je nejhorší u KDE. To se projevuje i na boxplotu, kde má KDE nejvíce odlehlých hodnot na obou stranách. Spolu s tím, že má KDE nejnižší hodnoty FPS se ukazuje, že KDE je na hraní Deadloid nejméně vhodné (ale stále více než dostačující).

Pomocí pravostranného asymptotického testu o střední hodnotě se podařilo prokázat, že pro všechny prostředí je střední hodnota FPS větší než 60 (s 95% jistotou).

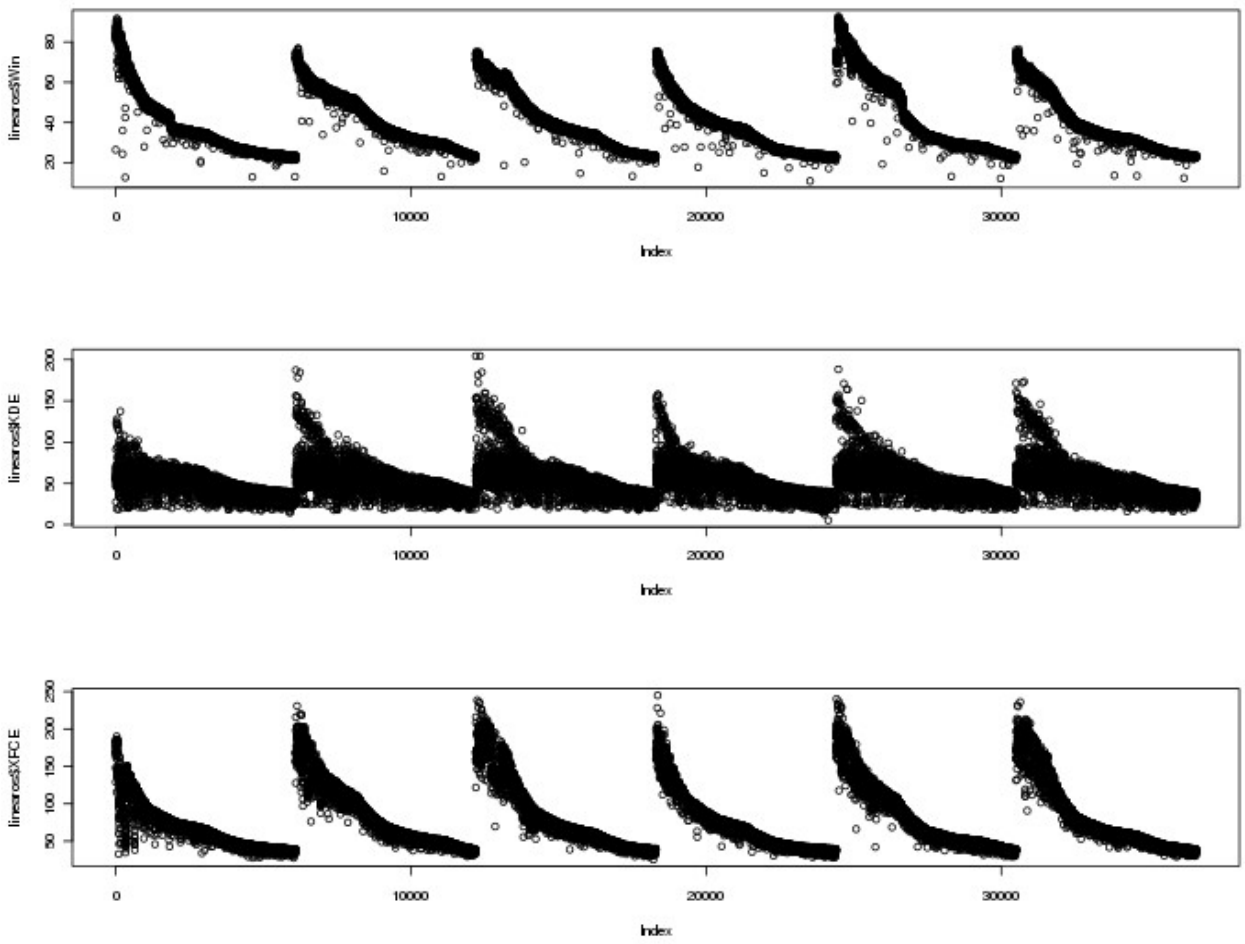
Prostředí	**Odhad FPS**
Win	73.98
KDE	68.43
XFCE	143.4

Tabulka 6: Odhady průměrné hodnoty FPS s 99% jistotou

Další zajímavé grafy, ověřování předpokladů a výpočty lze najít v přiloženém souboru na CD /tests/graphicsDeadloid.R.

4.3.3 Mód linear

V tomto módu počty spritu při testování stoupaly od 1 do 101 po 10 pro každý druh spritu, počínaje Spaceship a konče Asteroid3. Totéž se pak opakovalo pro UFO a dále pro další sprity až po Asteroid3.

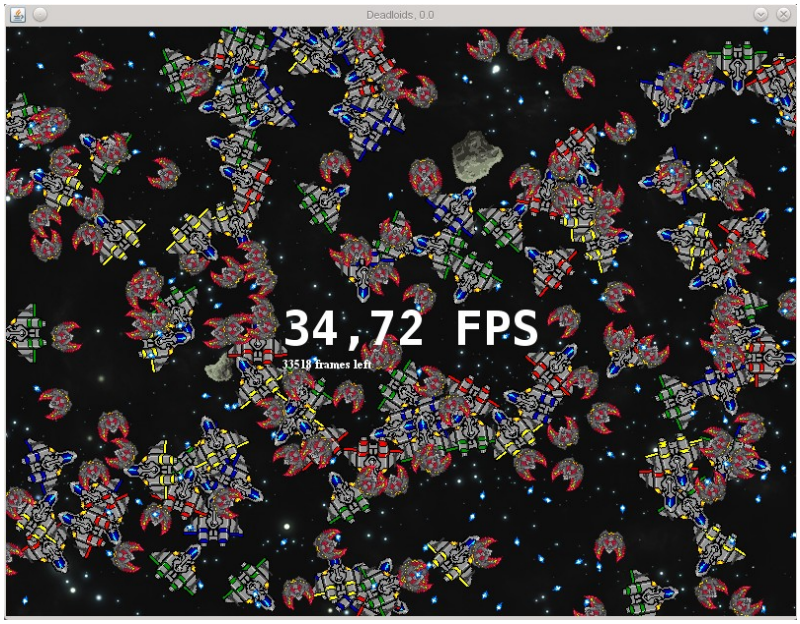


Obr. 4.3.7: Zobrazení FPS v čase

Na obrázku 4.3.7 je vidět, jak s přibývajícími sprity klesá FPS (postupně pro Windows, KDE a XFCE). Protože jsou počty spritů závislé na čase, není vhodné toto měření použít pro statistické výpočty (většina metod předpokládá nezávislost nasbíraných dat).

Test se spíše hodí pro vizuální pozorování klesající rychlosti FPS v závislosti na přibývajících spritech.

Naměřená data jsou uložena na přiloženém CD v souboru /tests/graphicsLienar.csv. Výpočty jsou uloženy v souboru /tests/graphicsLinear.R.



Obr. 4.3.8: Testování grafiky v módu deadloid

4.4 Testování přenosu dat po síti

Projekt použitý pro testování sítě je na CD v adresáři /source/4.4TestSite/.

Projekt navazuje rozšiřuje předchozí projekt 4.3TestGrafiky. Navíc jsem vytvořil (mimo jiné) třídy **NetTestStrategy a NetTestStrategyView**, které implementují speciální „herní strategii“ - test grafiky. Existují dvě verze StrategyView, jedna je implementovaná pomocí Java2D a druhá pouze jako textový výstup na obrazovku. To umožňuje testovat síť i na strojích bez grafického výstupu.

NetTestStrategy lze spustit ve dvou módech pojmenovaných randomServer a deadloidServer. Mód lze vybrat při spuštění pomocí argumentu příkazové řádky *--test*. Tím se spustí **klientská verze** programu. Význam jednotlivých módů je popsán v následujících kapitolách.

Před spuštěním klienta už musí běžet server na známém portu a IP adrese. Server se spustí pomocí argumentu příkazové řádky *--server*.

Výsledky testu jsou zkresleny několika faktory, kvůli kterým by se měli brát pouze jako orientační. Například není přesně známa cesta, kterou packety putovaly od klienta k serveru a zpět. Není ani známo, jaké všechny další programy běžely na počítači v době testu, komunikovaly po síti a ovlivňovaly výkon počítače.

Test probíhal mezi mým domácím počítačem a pracovním serverem. Cesta je zaznamenána v souboru /tests/traceroute.txt na přiloženém CD.

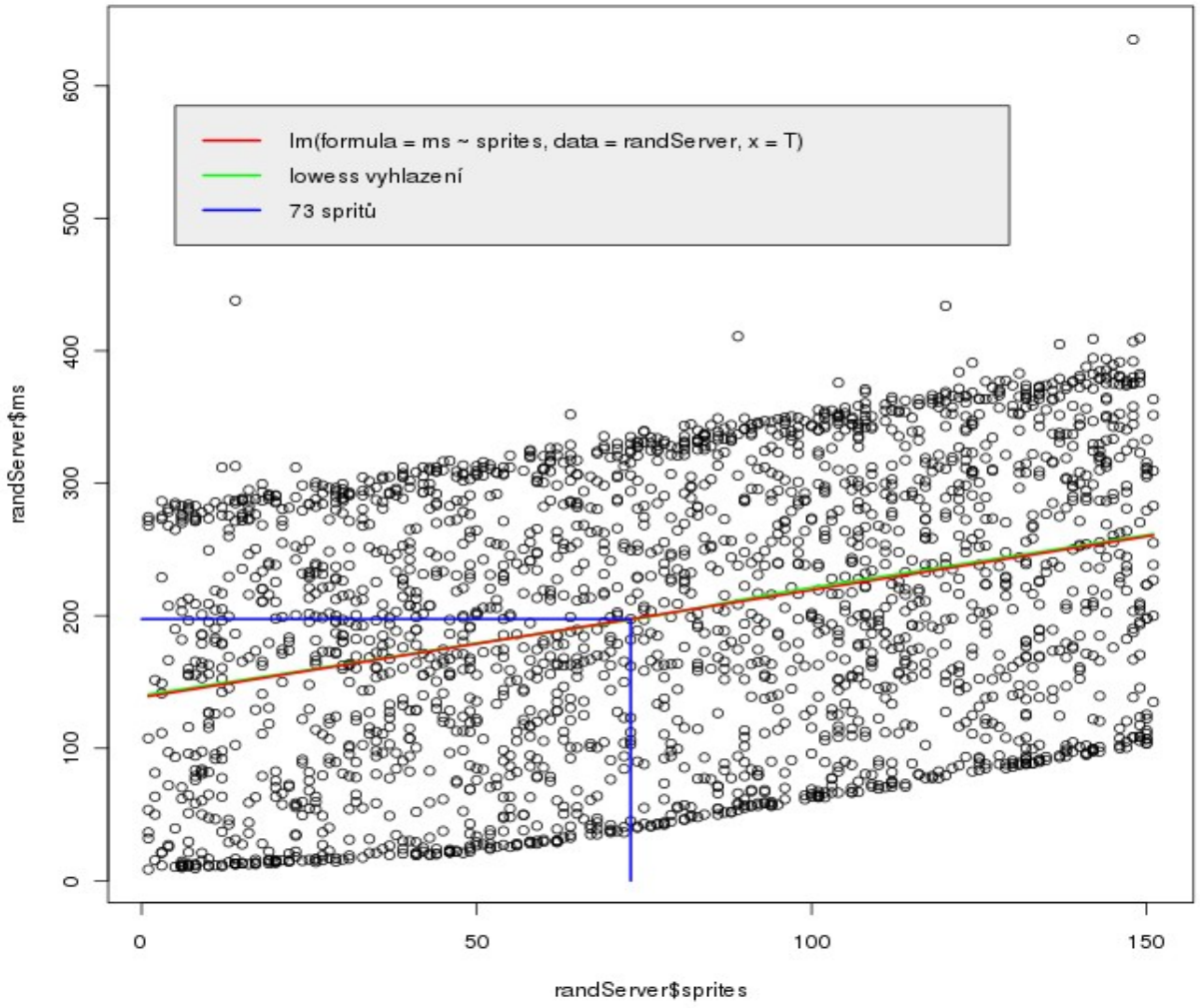
Aby se nemusel synchronizovat čas na milisekundy mezi mým počítačem a serverem (což jsem ani nemohl z technických důvodů udělat), počítal jsem čas cesty packetu od serveru k domácímu počítači a zpět (kvůli nastavení firewallu v práci jsem neměl na serveru k dispozici žádný veřejný port, ale na svém domácím počítači ano).

Naměřený výsledný čas jsem potom vydělil dvěma. To je ale další nepřesnost v měření. Doma jsem měl totiž linku připojenou přes poskytovatele internetového připojení UPC s rychlostí 10/1 MiB/s. Příchozí a odchozí rychlost se tak diametrálně liší. Ačkoliv jsem při testování ani zdaleka nepřenášel 1 MiB za sekundu, je to další důvod k tomu, proč brát testy sítě jen jako orientační.

4.4.1 Mód randomServer

V tomto módu se přenáší po síti náhodně vygenerovaný počet spritů. Na typu spritu nezáleží, protože všechny typy spritů přenášejí stejné množství informací (polohu, rychlost atd.). V tomto testu se hledá odpověď na otázku, jaký má vliv množství spritů na přenosovou rychlost.

Naměřená data jsou uložena na přiloženém CD v souboru /tests/serverRandom.csv. Výpočty jsou uloženy v souboru /tests/serverRandom.R.



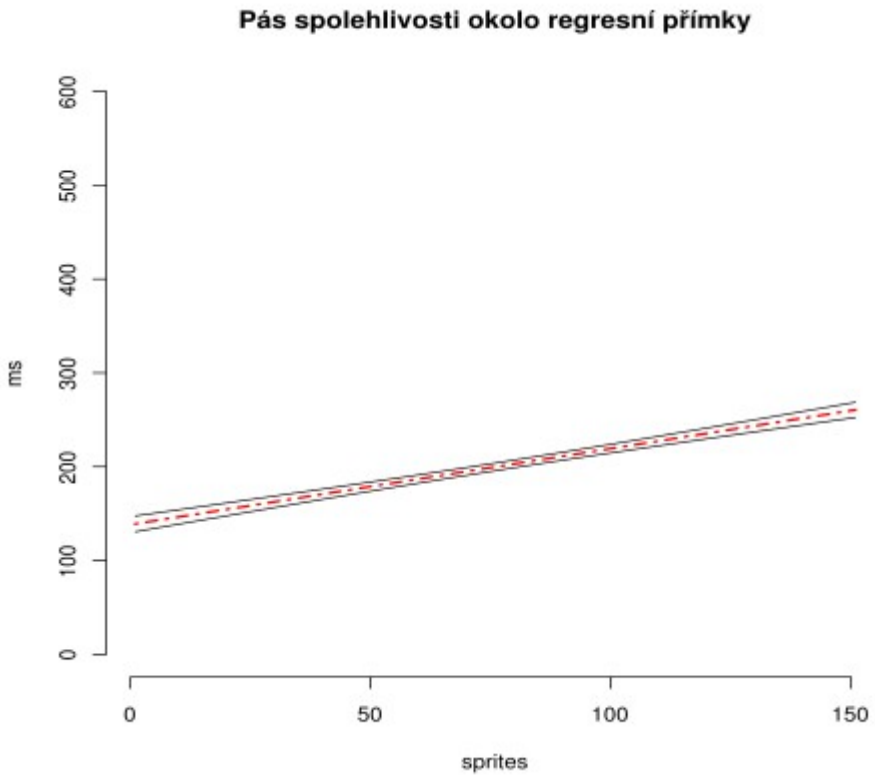
Obr. 4.4.1: Závislost rychlosti na počtu spritů

Na obrázku 4.4.1 jsou vidět naměřené rychlosti v milisekundách v závislosti na počtu přenášených spritů. Z obrázku je vidět velký časový rozptyl (směrodatná odchylka modelu vyšla 99.46 ms). Naměřené časy jsou rozprostřeny poměrně rovnoměrně, pouze v horní a dolní části jsou více nahuštěné. (Proč tomu tak je by stálo za samostatné prozkoumání). Z obrázku je také vidět jasná lineární závislost mezi počtem spritů a rychlostí.

Mezi počtem spritů a velikostí přenesených bajtů je přímá úměra. Každý sprit navíc znamenal přírůstek 168 bytů.

Regresní model závislosti ms na počtu spritů vypadá takto:

lm(ms ~ sprites, data=randServer,x=T) (2)



Na obrázku 4.4.2 je vidět pás spolehlivosti okolo regresní přímky. Pás je poměrně úzký z čehož je také vidět poměrně stabilní rozložení přenosových rychlostí.

Intercept modelu vyšel 138.38 ms, koeficient u počtu spritů 0.81 (tj. přidáním jednoho spritu se přenos zpomalil v průměru o 0.81 ms).

Jak je vidět, přidáním spritu se příliš čas přenosu nezpomalí. Za velký intercept může také to, že se během testování přenášel celý GameWorld a i GameStrategyModel.

Rychlost by měla být ideálně vždy pod 100 ms. Ovšem odhad průměru pro 73 spritů vyšel 197.5 ms. A to je jen průměr. Spolu se směrodatnou odchylkou se blíží horní hranice intervalu rychlosti přenosu k 300 ms. Z toho je zřejmé, že rychlost přenosu není pro hraní dostatečná a bude nutné snížit datový tok kompresí dat, případně rozdělením přenosu GameWorld a GameStrategyModel do různých packetů (čímž by se měl snížit intercept).

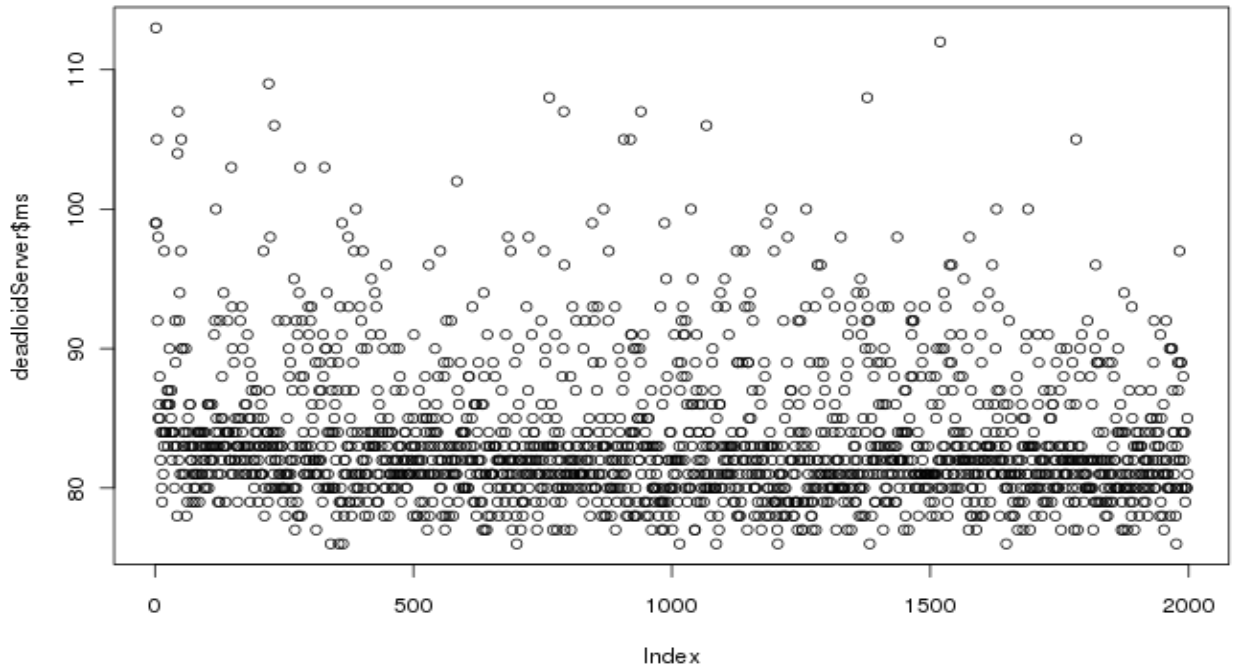
Taktéž velikost spritu není nezanedbatelná. Pro 73 spritů vychází zpomalení 59.13 ms (73*0.81). Otázka je, zda není tak velký počet spritů na hru po internetu už příliš. V kapitole *3.3.1 Jak a jaká data se budou přenášet* byl odhad počtu spritů roven v jednodušší variantě hry 37 a až v náročnější 73. Pro 37 spritů vychází zpomalení 29.97 ms, což je mnohem příznivější (navíc 37 je pouze teoretické množství spritů které by se mohlo na obrazovce objevit, prakticky jich tam tolik zřejmě nikdy nebude – účelem hry je meteority ničit, ne si jich nechávat na obrazovce co nejvíce).

4.4.2 Mód deadloidServer

V tomto testu se přenáší 73 spritů, což je maximální teoretické množství spritů, které se mohou objevit ve hře v jeden okamžik. Testování proběhlo za stejných podmínek jako test popsaný v kapitole *4.4.1 Mód ranodmServer*.

Naměřená data jsou uložena na přiloženém CD v souboru /tests/serverDeadloid.csv. Výpočty jsou uloženy v souboru /tests/serverDeadloid.R.

Hledá se odpověď na otázku, kolik ms trvá přenost GameWorld a GameStrategy. Ideální přenosová rychlost je pod 50 ms, maximální přípustná rychlost pak 100 ms.



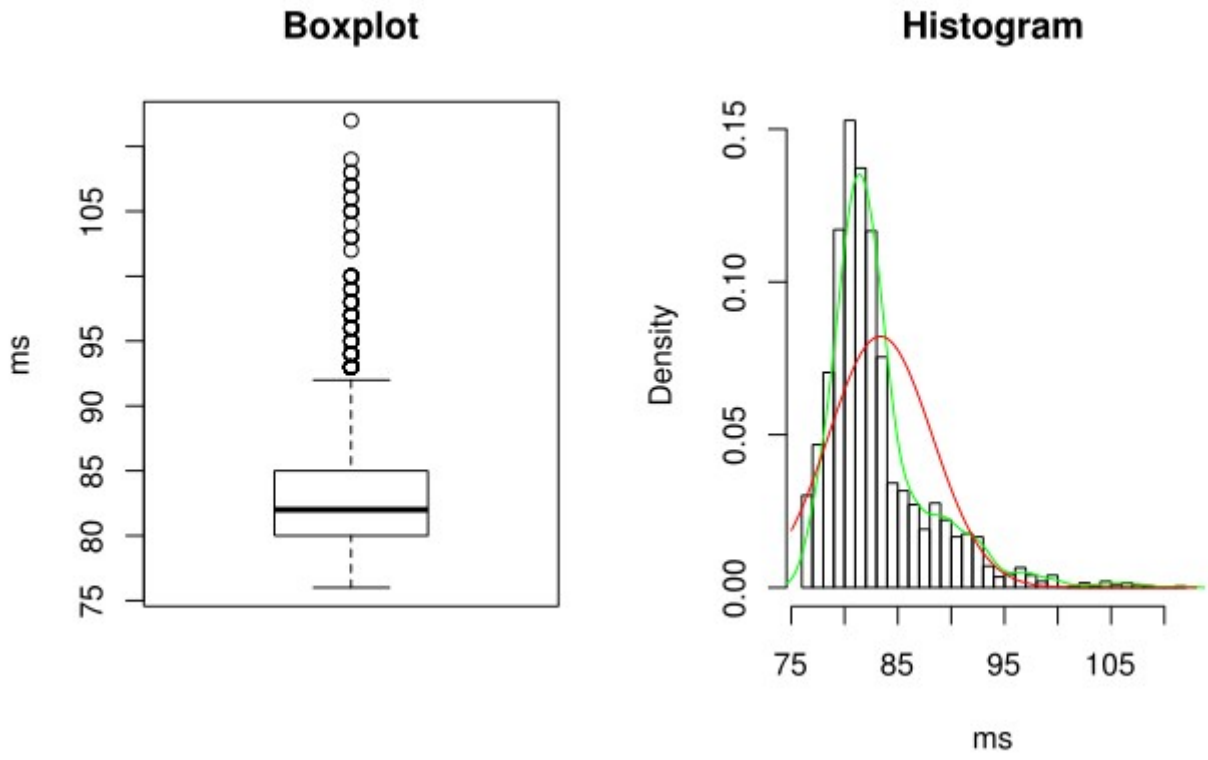
Obr. 4.4.3: Naměřené hodnoty rychlosti v ms

Protože se počet přenášených spritů neměnil, přenášelo se vždy 13.26 KiB. Z obrázku je vidět, že rychlosti přenosu vycházejí mnohem příznivěji, než v předchozím testu popsaném v *4.4.1 Mód ranodmServer.* Možné vysvětlení je, že větší objem přenášených dat měl za následek zpomalení ostatních packetů putujících po síti. Zde z prvního pohledu na obrázek je patrné, že čas potřebný k přenosu je většinou pod 100 ms!

Min.	1st. Qu.	Median	Mean	3rd Qu.	Max.
76	80	82	83	85	112

Tabulka 7: Popis rozdělení v ms

Z tabulky popisu rozdělení (v ms) je vidět, že se naměřené rychlosti (pro 2000 měření) pohybují v intervalu 76 až 112 ms. Medián (menší než průměr) je jen 82 ms. Navíc lze spočítat, že 95 % naměřených hodnot je pod 93 ms a 99 % hodnot pod 100 ms! Směrodatná odchylka vyšla pouhých 4.85 ms. Vizuální představu o rozdělení naměřených rychlostí podává obrázek *4.4.4 Boxplot a histogram.*



Obr. 4.4.4 Boxplot a histogram

Už z těchto výsledků lze usoudit, že přenosová rychlost není ideální (pod 50 ms), ale minimálně pro naprogramování a testování hry zcela dostačující. (Vzhledem k tomu, že hru budu programovat a testovat na lokální síti, nemusím se v tomto případě rychlostí vůbec znepokojovat).

Další zajímavé výpočty, grafy a ověřování předpokladů je v souboru /tests/serverDeadloid.R.

5. Implementace

Implementace programu (zdrojové kódy) lze nalézt na CD v adresáři /source/. Funkce první implementace **4.3TestGrafiky** je popsána v kapitole *4.3 Testování grafického výstupu*. Funkce druhé implementace, **4.4TestSite**, je popsána v kapitole *4.4 Testování přenosu dat po síti*.

V těchto implementacích jsou vytvořeny všechny třídy popsané v analýze, ale implementovány jsou jen ty části, které byly potřeba.

Zdrojové kódy jsou rozděleny do následujících balíčků:

- **com.beust.jcommander**
  Tento balíček slouží k parsování argumentů příkazové řádky. Jedná se o jednoduchý balíček vytvořený Cédricem Beustem[33]. Vybral jsem jej, protože má dobrou dokumentaci a snadno se používá. Jeho jednoduchost ale může být i jeho nevýhodou (například neumožňuje definovat závislosti mezi použitými volbami příkazové řádky).
- **deadloids.cmdParser**
  V tomto balíčku jsem vytvořil třídy využívající com.beust.jcommander definující volby příkazové řádky.
- **deadloids.common**
  Tento balíček obsahuje třídy, které jsem vytvořil na základě zdrojových kódů ke knize Programming Game AI by Example[14]. Jde v podstatě o přepis zdrojových kódů z jazyka C++ do jazyka Java. Protože jsem se snažil o co nejbližší podobu zdrojových kódů, nadměrně jsem využíval konstrukce „import static“, vkládal zbytečné *finalize*() metody (tam kde se v C++ používal destruktor) atp. Důvodem bylo, aby si čtenáři knihy mohli snadno najít citované pasáže v C++ kódu. Kompletní Java kódy lze najít na adrese http://www.sallyx.org/game-ai/.
- **deadloids**
  V tomto balíčku je ještě jeden převzatý soubor z knihy Programming Game AI by Example[14], a to třída SteeringBehavior.java. Její význam je popsán v kapitole *3.1.1 Popis polohy a pohybu spritů*. Používají se především tři „chování“ pro pohyb Ufa: wander pro náhodný pohyb po obrazovce, pursuit pro pronásledování hráče a separation pro vyhýbání se asteroidům a raketám.
- **deadloids.net**
  Zde jsou všechny třídy související s přenosem hry po síti.
- **deadloids.sprites**
  Zde jsou definovány sprity. Třída Controller byla nakonec nahrazena třídami implementujícími stavový automat (z balíčku **deadloids.common.FSM**) popsaný v kapitole *3.1.4 Konečný automat (FSM)* (od verze 5.1Deadloid).
- **deadloids.strategies**
  Obsahuje třídy reprezentující herní strategie. Ve verzi 4.2TestGrafiky je implementována strategie GraphicsTestStrategy a ve verzi 4.4TestSite NetTestStrategy.
- **deadloids.view**
  Obsahuje třídy grafického rozhraní (UI). Jsou implementovány 2 verze. **Java2D** je implementována pomocí Java Swing Frameworku, **Textová** verze je implementována pomocí standardního textového výstupu. Textová verze se hodí na testování sítě v případě, že na testovaném serveru/klientovi není možné spouštět grafické prostředí.

Hlavní cyklus aplikace, který se stará o běh hry a vykreslování, je ve třídě Main (mimo balíčky), ve které se také zpracovávají parametry příkazové řádky.

5.1 DeadloidSingle

V adresáři /source/5.1DeadloidSingle/ je první implementace hry Deadloid, strategie SingleGameStrategy.

Tato verze programu umí spustit strategii jednoho hráče. Při spuštění je hráč dotázán na svůj nick a na typ lodě, se kterou chce hrát (barvu Spaceshipu). Tyto informace se pak uloží do konfiguračního souboru. Hra po zničení všech spritů přechází do vyššího levelu, kde se sprity pohybují rychleji. Od druhého levelu se ve hře objevuje Ufo, které pronásleduje hráče (jeho sprit) a v náhodných intervalech střílí, dokud není zničeno, nebo není zničen hráč. Po skončení hry se uloží skóre do souboru config.xml.

5.1.1 Hlavní změny

Oproti předchozím verzím se hlavní cyklus aplikace přesunul do třídy **deadloids.Deadlois.java**.

Nový balíček **deadloids.sound** se stará o ozvučení hry.

Byly zprovozněny události klávesnice, viz **deadloids.events**.

Strategie je implementována ve třídě **deadloids.strategies.SingleGameStrategy.java**.

Balíček **deadloids.facilty** obsahuje třídy, které se starají o „všechno okolo“. V této verzi je tím myšleno načítání a ukládání základní konfigurace (jméno hráče a barva lodi) a skóre.

Implementace MessageDispatcheru obsahovala chybu, kvůli kterému se zprávy uložené v dispatcheru při jejich doručení nemazaly. Kvůli opravě a otestování funkčnosti jsem vytvořil **JUnit testy** pro třídy MessageDispatcher a Telegram. Během dalšího programování jsem postupně doplňoval další JUnit testy pro problémové třídy.

5.2 DeadloidNet

V adresáři /source/5.2DeadloidNet/ je druhá implementace hry Deadloid. Tato verze umožňuje spouštět hru jednoho hráče přes internet.

Předchozí verze implementovala spoustu nových vlastností spritů. Kvůli tomu narostla velikost serializovaných objektů. GameWorld se tak už nevejde do jednoho datagramu a nefunguje posílání po síti. Proto bylo nakonec nutné vytvořit vlastní způsob serializace objektů.

Všechny serializovatelné objekty implementují nové rozhraní **deadloids.net.Serialize**. To obsahuje metodu *serialize()*, jejímž účelem je převést důležitá data do pole bajtů. K serializaci a deserializaci jsem vytvořil pomocné třídy **ByteReader** a **ByteWritter**. K deserializaci se většinou používá konstruktor s argumentem typu ByteReader. Tam, kde je to výhodné, se pro deserializaci používá statická metoda (především u výčtových typů).
Nakonec se mi díky serializaci podařilo zmenšit objem přenášených dat téměr desetkrát. V JUnit testu serializace třídy GameMessage se kontroluje velikost přenášených dat oproti 1500 bytům. Tím by mělo být zaručeno, že při další změně dat přenášených v GameMessage se velikost přenášeného datagramu vejde do této velikosti. 1500 bytů je běžná maximální velikost datagramu, který se při přenosu po síti nerozděluje na menší, tzv. MTU[34].

Ve hře vznikají a zanikají sprity poměrně často. Tím dochází k rychlému nárůstu sekvence jejich ID. Kdyby neexistoval způsob jak řadu ID čísel restartovat, brzo by na serveru při spuštění několika her došlo k vyčerpání všech možných čísel (typu integer). Proto se o přidělování ID spritům stará nově třída **GameWorld**. Každý GameWorld udržuje vlastní řadu ID, díky čemuž ID nedojdou tak rychle.

Třída **EntityManager** slouží k rychlejšímu vyhledávání spritů podle ID. EntityManager fungoval jako singleton, tj. existovala jedna jediná instance této třídy. Protože na serveru poběží více instancí her (a každá hra má vlastní řadu ID číslování), bylo potřeba vytvořit i více instancí EntityManageru (pro každou hru jeden).
Totéž bylo třeba udělat s třídou **MessageDispatcher**, která se stará o doručování Telegramů spritům.

5.3 DeadloidOnline

V adresáři /source/5.3DeadloidOnline/ je další verze hry Deadloid. Tato verze umožňuje připojení klienta k serveru pomocí **MasterServer** a **MasterClient** tříd a hrát po síti hru jednoho hráče.

Prvním krokem bylo přejmenování tříd GameClient a GameServer na TestGameClient a TestGameServer a vytvoření nových tříd GameClient a Game server. Testovací klient i server totiž posílají trochu jiný typ dat, než je potřeba pro hraní her. Ačkoliv by se jistě daly třídy upravit tak, aby dokázaly ovládat jak hraní hry tak testování, z časových důvodů jsem zvolil tento postup.

Dalším krokem bylo vytvoření základní komunikace mezi MasterServerem a MasterClientem pomocí XML. MasterServer zatím neověřuje totožnost klienta, ale komunikace je od začátku připravena k tomu, aby to bylo možné snadno dopracovat.

Dále byla vytvořena třída **Predictor**, která slouží k predikci stavu hry na straně klienta. Prediktor posouvá sprity dle jejich směru a rychlosti přijatých od serveru, ale také posouvá sprit hráče dle příkazů přijatých z view. Díky tomu je hra plynulejší i když packety přicházejí s velkým časovým odstupem.
Lze to vyzkoušet například přenastavením konstanty Constants.MIN_GAME_SERVER_LAG na nějakou velkou hodnotu (například 200 ms). Bez prediktoru se sprity nepohybují plynule, ale jakoby poskakují z jedné pozice na další (vždy když přijde nový packet od serveru).

Další změnou je posílání State u Spaceship spritu, který je důležitý pro jeho zobrazení a predikci.

Poslední významnou změnou je vytvoření tříd **SoundTranslator** a **SoundEvent**. SoundTranslator zachytává zprávy posílané z GameStrategy a vytváří vhodné zvukové události. Hlavním důvodem pro toto řešení je možnost posílat zvukové události ze serveru klientovi.

5.4 DeadloidFinal

V adresáři /source/5.4DeadloidFinal/ je poslední verze hry Deadloid. Tato verze umožňuje hrát hru více hráčů po internetu.

První změnou bylo přejmenování třídy deadloids.net.Player na **PlayerConnection** a vytvoření třídy deadloids.Player. PlayerConnection, která reprezentuje připojení hráče a pomáhá identifikovat příchozí packety. Naproti tomu Player jednoznačně identifikuje hráče. Protože ještě nebylo implementováno ukládání hráčů na straně serveru, používá se jako jednoznačný identifikátor pouze nickname hráče. MasterServer se stará alespoň o to, aby přihlášení hráči do hry měli nickname unikátní.

Po přihlášení se údaje o hráči posílají klientovi, aby mohl identifikovat svůj Spaceship. Nickname se také dočasně vytváří z loginu hráče.

Většinu logiky hry více hráčů implementuje třída **MultiGameStrategy**. Pro přenos dat po síti využívá třídu **MultiGameStrategyModel**. Protože na serveru běží několik herních strategií najednou, musela být přepracována třída **FrameCounter** tak, aby každá strategie měla vlastní instanci (podobně jako tomu bylo u EntitiManageru a MessageDispatcheru).

V balíčku deadloids.sound vznikla nová třída **AePlayWave**[35], která se stará o přehrávání velkých WAV souborů. Třída java.sound.sampled.Clip totiž nedokáže přehrávat velké soubory v OS Windows (ačkoliv v Linuxu to dokázala).

Mezi další drobné změny patří například to, že hráč může ovládat svůj sprit i když je ve stavu ReturnToGame, aby se mohl přesunout na bezpečné místo (nemůže samozřejmě střílet). GameClient zachytává PlayerCommand.SCORE, aby se tento příkaz nemusel přenášet po síti. Bylo opraveno několik chyb, včetně znovu zprovoznění NetTestStrategy.

5.5 TODO

Z časových důvodů se mi nepodařilo implementovat všechny funkce, které bych si přál a o kterých jsem se zmiňoval v analýze. Mezi nejdůležitější patří:

- Ukládání uživatelů na straně serveru do XML (database.xml) nebo databáze.
- Registrace hráče (nastavení uživatelského jména (nickname), změna emailu atp).
- Nabízení vlastní instance hry pro jednoho hráče každému připojenému hráči (teď běží na serveru jen jedna hra jednoho hráče ke které se může připojit jen jeden hráč).
- Možnost konfigurace serveru z aplikace Deadloids (spouštění, restartování a ukončování her více hráčů atp.)
- Implementace uživatelských rolí.

6. Android

Tato část by měla být zamyšlením nad tím, jaké by byly potřeba udělat změny v aplikaci Deadloids pro zprovoznění v prostředí Android.

6.1. O systému Android

Android je nejpopulárnější mobilní platforma na světě. Google Play[12], online obchod pro Android, obsahuje přes 600 000 aplikací a her, milióny písní a knih a tisíce filmů[36]. Android je dodáván především na mobilních telefonech a tabletech a je jednou z nejdůležitějších platforem dnešní doby.

Android je rozsáhlá open source platforma, která byla vytvořena společností Android Inc. Založené v roce 2003 v Kalifornii a odkoupené v srpnu roku 2005 společností Google Inc.[37]

Verze	Kódové jméno	Datum představení
1.5	Cupcake	30. dubna 2009
1.6	Donut	15. září 2009
2.0/2.1	Eclair	26. října 2009
2.2	Froyo	20. května 2010
2.3/2.4	Gingerbread	6. prosince 2010
3.0/3.1/3.2	Honeycomb	22. února 2011
4.0-4.0.4	Ice Cream Sandwich	19. října 2011
4.1	Jelly Bean	9. července 2012

Tabulka 8: Historie verzí Androidu[37]

6.2. Aplikační rozhraní

Architektura operačního systému Android je rozdělena do pěti vrstev, od jádra operačního systému, přes knihovny napsané v C/C++, Android Runtime vrstvu obsahující virtuální stroj zvaný Dalvik, **aplikační rozhraní** až po aplikace, které využívají běžní uživatelé.

Pro vývojáře je nejdůležitější vrstva aplikačního rozhraní (application framework). Hlavní těžiště práce při implementaci hry Deadloids pro Android by bylo právě v přizpůsobení hry aplikační vrstvě.

Vývoj aplikací pro Android je oficiálně podporováno ve vývojovém prostředí Eclipse. Do něj lze nainstalovat plugin pro snazší práci s andoid pluginem. Nástroje pro vývoj aplikací jsou pak součástí Android Software Development Kitu (SDK), který je volně ke stažení pro operační systémy GNU/Linux, Windows i Mac OS. Součástí SDK je i emulátor Androidu, který umožňuje testovat aplikace i bez fyzického zařízení.

12 https://play.google.com/

6.3. Potřebné změny hry pro OS Android

Android je používán na mnoha různých zařízeních, lišící se svými hardwarovými specifikacemi, jako například rozlišením displaye, výkonem, dostupností paměti atd. Aplikační rozhraní s tímto počítá a umožňuje aplikacím se přizpůsobit prostředí, ve kterém běží. Například je možné dodat obrázky v různých velikostech a Android pak vybere nejlepší možné rozlišení dle velikosti displaye.

Aplikace v prostředí Android jsou svázány různými právy. Není například možné bez udělení práv přistupovat k internetu, nebo kamkoliv zapisovat na disk. V Deadloids by se tak musela například přepracovat třída Config, která čte a zapisuje data na disk v aktuálním adresáři.

Pro hraní více hráčů by mohla aplikace využívat nejen připojení k internetu pomocí TCP/IP, ale také technologie bluetooth.

Grafické rozhraní nemůže v Androidu využívat knihovny javax.swing. Naštěstí podporuje vlastní podobné rozhraní pro vykreslování obrázků ve 2D. To by znamenalo odstranit z projektu Deadloids view Java2D vytvořit nové view pro Android. Toto nové view by muselo brát v potaz skutečnou velikost displaye, případně i jeho natočení.

V potaz se musí brát i výkon zařízení, na kterých by byla hra provozována. Dnešní chytré telefony mají dostatečně silný hardware pro hraní her, ale nelze je srovnávat s výkonem notebooků nebo stolních počítačů.

Hra Deadloids se ovládá pomocí kláves. Některé telefony nebo tablety klávesnice nemají a ovládají se pouze přes dotykovou obrazovku. Hra by tak musela umožňovat ovládání jak přes dotykovou obrazovku, tak přes hardwarovou klávesnici telefonu.

Taktéž by pravděpodobně bylo potřeba vytvořit nové implementace tříd starajících se o ozvučení hry.

Vlastní herní logika by pravděpodobně mnoho změn neprodělala. A to především díky oddělení tříd pro práci se soubory, zvuky, ovládání a zobrazování.

Obr. 6.3.1: Ukázka hry z emulátoru Androidu

7. Závěr

Během vytváření této absolventské práce jsem si procvičil znalosti programování v Javě nabyté během studia a naučil se spoustu nových. Díky této práci jsem se dostal k mnoha zajímavým poznatkům ze světa vývoje počítačových her, především pak umělé inteligence.

Zatímco osobním přínosem této práce spatřuji ve vlastním vzdělávání, celosvětový dopad této práce leží především v překladu zdrojových kódů knihy Programming Game AI by Example[14] z jazyka C++ do Javy. Po domluvě s autorem jsem zdrojové kódy poskytl ke stažení na stránce http://www.sallyx.org/sally/game-ai/, přičemž se na ni autor odkazuje ze svých stránek[13] a zpřístupňuje ji tak širokému publiku.

Věřím, že tato absolventská práce obsahuje dostatečné množství informací pro případné zájemce o vytváření her v prostředí Java, vytváření online her a další studium umělé inteligence.

13 http://www.ai-junkie.com/books/toc_pgaibe.html

Seznam literatury

[1] Asteroids (video game). Wikipedia [online]. 02:26, 14 September 2012. 2012 [cit. 2012-09-15]. Dostupné z: http://en.wikipedia.org/wiki/Asteroids_(video_game)
[2] Vektorová grafika. Wikipedia [online]. 19. 6. 2012, 12:46. 2012 [cit. 2012-09-15]. Dostupné z: http://cs.wikipedia.org/wiki/Vektorová_grafika
[3] File:Asteroids 360.jpg. Wikipedia [online]. 19:32, 26 November 2007. 2007 [cit. 2012-09-15]. Dostupné z: http://en.wikipedia.org/wiki/File:Asteroids_360.jpg
[4] Asteroids Player Smashes 27-Year-Old High Score. WIRED [online]. 04.05.10 5:40 PM. 2010 [cit. 2012-09-15]. Dostupné z: http://www.wired.com/gamelife/2010/04/asteroids-record/
[5] KISS. Wikipedia [online]. 25. 8. 2012, 16:40. 2010 [cit. 2012-09-15]. Dostupné z: http://cs.wikipedia.org/wiki/KISS
[6] Jak správně zvolit název firmy?. Marketing Journal [online]. 1.4.2010. 2010 [cit. 2012-09-15]. Dostupné z: http://www.m-journal.cz/cs/praxe/pripadove-studie/jak-spravne-zvolit-nazev-firmy__s354x6482.html
[7] Asteroid. Wikipedia [online]. 9. 7. 2012, 15:41. 2012 [cit. 2012-09-15]. Dostupné z: http://cs.wikipedia.org/wiki/Asteroid
[8] Sprite (počítačová grafika). Wikipedia [online]. 11. 5. 2012, 18:17. 2012 [cit. 2012-09-15]. Dostupné z: http://cs.wikipedia.org/wiki/Sprite_(počítačová_grafika)
[9] Seznam čísel portů TCP a UDP. Wikipedia [online]. 8. 9. 2012, 18:13. 2012 [cit. 2012-09-15]. Dostupné z: http://cs.wikipedia.org/wiki/Seznam_čísel_portů_TCP_a_UDP
[10] Unix time. Wikipedia [online]. 11:12, 15 September 2012. 2012 [cit. 2012-09-15]. Dostupné z: http://en.wikipedia.org/wiki/Unix_time
[11] Extensible Markup Language. Wikipedia [online]. 12. 8. 2012, 12:24. 2012 [cit. 2012-09-15]. Dostupné z: http://cs.wikipedia.org/wiki/XML
[12] Model-view-controller. Wikipedia [online]. 29. 7. 2012, 07:48. 2012 [cit. 2012-09-15]. Dostupné z: http://cs.wikipedia.org/wiki/Model-view-controller
[13] MVC vs MVP – Part 1. ByDavy [online]. . 2011 [cit. 2012-09-15]. Dostupné z: http://www.bydavy.com/2011/02/mvc-vs-mvp/
[14] BUCKLAND, Mat. Programming Game AI by Example: Kindle Edition [online]. 2010 [cit. 2012-09-16]. ISBN 1556220782. Dostupné z: http://www.amazon.com/Programming-Game-AI-Example-ebook/dp/B0029LCJXE/ref=cm_cr_pr_orig_subj
[15] Konečný automat. Wikipedia [online]. 7. 8. 2012, 08:36. 2012 [cit. 2012-09-16]. Dostupné z: http://cs.wikipedia.org/wiki/Konečný_automat
[16] TCP/IP. Wikipedia [online]. 10. 6. 2012, 07:56. 2012 [cit. 2012-09-16]. Dostupné z: http://cs.wikipedia.org/wiki/TCP/IP
[17] How Does Load Balancing Work?. CISCO [online]. . 2005, Aug 10, 2005 [cit. 2012-09-16]. Dostupné z: http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094820.shtml
[18] Pixel. Wikipedia [online]. 6. 8. 2012, 12:48. 2012 [cit. 2012-09-16]. Dostupné z: http://cs.wikipedia.org/wiki/Pixel
[19] UDP vs. TCP: Introduction. Glenn Fiedler's Game Development Articles and Tutorials [online]. . 2008 [cit. 2012-09-16]. Dostupné z: http://gafferongames.com/networking-for-game-programmers/udp-vs-tcp/
[20] Characteristics of UDP Packet Loss: Effect of TCP Traffic. Internet Society [online]. . [1997] [cit. 2012-09-16]. Dostupné z: http://www.isoc.org/INET97/proceedings/F3/F3_1.HTM
[21] Networked Physics: Introduction. Glenn Fiedler's Game Development Articles and Tutorials [online]. . 2006 [cit. 2012-09-16]. Dostupné z: http://gafferongames.com/game-physics/networked-physics/

[22] Snímková frekvence. Wikipedie [online]. 6. 9. 2012, 13:24. 2012 [cit. 2012-09-16]. Dostupné z: http://cs.wikipedia.org/wiki/Snímková_frekvence
[23] Java 2D API. Oracle [online]. . 2012 [cit. 2012-09-16]. Dostupné z: http://java.sun.com/products/java-media/2D/
[24] HARBOUR, Jonathan S. Beginning Java Game Programming [online]. 2nd Editiion. 2008 [cit. 2012-09-16]. ISBN 1598634763. Dostupné z: http://www.amazon.com/Beginning-Java-Programming-Second-Edition/dp/1598634763/ref=pd_sxp_f_pt
[25] LXDE. Wikipedie [online]. 8. 6. 2012, 07:18. 2012 [cit. 2012-09-16]. Dostupné z: http://cs.wikipedia.org/wiki/LXDE
[26] KDE. Wikipedie [online]. 8. 6. 2012, 07:18. 2012 [cit. 2012-09-16]. Dostupné z: http://cs.wikipedia.org/wiki/KDE
[27] Úvod do R. R projekt pro statistické výpočty v České republice [online]. 8. 6. 2012, 07:18. [2012] [cit. 2012-09-16]. Dostupné z: http://www.r-project.cz/about.html
[28] RKWard. Wikipedia [online]. 10. 2. 2012, 21:19. [2012] [cit. 2012-09-16]. Dostupné z: http://cs.wikipedia.org/wiki/RKWard
[29] KOMÁRKOVÁ, Lenka a Arnošt KOMÁREK. Základy analýzy dat a statistického úsudku s příklady v R. Praha: Oeconomica, 2007. ISBN 9788024512.
[30] KOMÁRKOVÁ, Lenka a Arnošt KOMÁREK. Statistická analýza závosilostí s příklady v R. Praha: Oeconomica, 2007. ISBN 9788024512266.
[31] R. Software Arnošt Komárek [online]. © 2007-2011 [cit. 2012-09-16]. Dostupné z: http://www.karlin.mff.cuni.cz/~komarek/software.html
[32] MASS: Support Functions and Datasets for Venables and Ripley's MASS. The Comprehensive R Archive Network [online]. [2002] [cit. 2012-09-16]. Dostupné z: http://cran.r-project.org/web/packages/MASS/index.html
[33] JCommander. JCommander [online]. 2010, July 31st, 2012 [cit. 2012-09-16]. Dostupné z: http://cran.r-project.org/web/packages/MASS/index.html
[34] Maximum transmission unit. Wikipedia [online]. 23. 5. 2012, 22:37. 2012 [cit. 2012-09-16]. Dostupné z: http://cs.wikipedia.org/wiki/Maximum_transmission_unit
[35] Java play WAV sound file. AnyExample [online]. © 2010 [cit. 2012-09-16]. Dostupné z: http://www.anyexample.com/programming/java/java_play_wav_sound_file.xml
[36] Android: Discover Android. GOOGLE INC. Android [online]. [2012] [cit. 2012-10-17]. Dostupné z: http://www.android.com/about/
[37] Android: operační systém. In: Wikipedia [online]. 12. 12. 2012, 09:05 [cit. 2012-10-17]. Dostupné z: http://cs.wikipedia.org/wiki/Android_(operační_systém)

Dodatek

A. Seznam použitých zkratek

API	Application Programming Interface
FPS	Frames per second
FSM	Finite State Machine (konečný automat)
ID	Identifikátor (unikátní označení objektu)
IP	Internet Protocol
JAR	Archivní soubor pro programovací jazyk Java (založen na ZIP kompresi)
MMOG	Massively multiplayer online game
MTU	Maximum transmission unit[34]
MVC	Model View Controller
OS	Operační systém
PX	Viz PIXEL
PIXEL	Zkrácení anglických slov picture element, obrazový prvek, zkracováno na PX[18]
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
UI	User Interface (grafické rozhraní)
UNIX TIME	Systémový čas popsaný jako počet vteřin od půlnoci 1.1.1970 UTC[10]
XML	eXtensible Marckup Language[11]

B. Uživatelská příručka

Toto je uživatelská příručka k programu Deadloids verze 5.4. Na přiloženém CD lze tento program najít v podobě JAR archivu *5.4DeadloidFinal.jar* v adresáři */dist/*.

Deadloids umožňuje hrát hru jednoho hráče offline i online a hru pro více hráčů online. Program lze spouštět jako server i klienta. Pomocí programu je také možné testovat výkon grafiky a rychlost přenosu po síti. Všechny tyto funkce jsou popsány v následujících kapitolách.

Instalace a spuštění

Program Deadloids se nemusí nijak instalovat. Ke svému běhu potřebuje pouze Javu JRE verze 1.7 nebo vyšší a právo čtení a zápisu do konfiguračního souboru. (Konfigurační soubor se vytváří v aktuálním adresáři, pokud není uvedeno jinak).

Program se spustí z příkazové řádky následovně:

> java -jar *cesta_k_adresari_programu*/5.4DeadloidFinal.jar

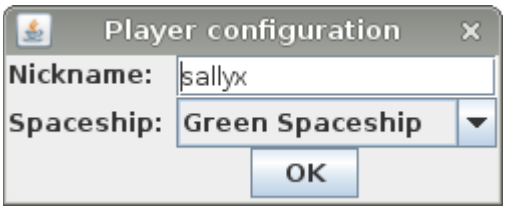
Program podporuje několik argumentů příkazové řády. Vypsat je lze následujícím způsobem:

> java -jar 5.4DeadloidFinal.jar --help

Hra pro jednoho hráče

Při spuštění programu Deadloids se otevře okno, ve kterém je zobrazeno top 10 nejlepších skóre (čtou se z konfiguračního souboru). Na pozadí běží simulace hry jednoho hráče.

Hra se zahájí z menu *Player → Single Mode*. Otevře se dialog, kam je možné zapsat libovolné jméno (bude se ukládat v top 10 nejlepších skóre) a barvu lodi. Hra se odstartuje klávesou *Enter*.



Obr. B.1: Player configuration

Ovládání hry je popsáno v dialogu z menu *Help → Control*.

Účelem hry je zničit všechny Asteroidy, Ufa, dostat se do co nejvyššího levelu a nasbírat co nejvíce bodů. Za každé zničené Ufo dostává hráč 1000 bodů, za zničenou raketu Ufa 500 bodů a za poškození asteroidu je přičteno tolik bodů, kolik je odebráno asteroidu (maximálně 100).

Hráč se musí vyhýbat asteroidům a střelám Ufa, které je mohou zničit. Hráč má k dispozici tři životy. Hráč může mít vystřelené v jeden okamžik nanejvýše čtyři rakety.

Spuštění serveru

Hru lze spustit jako server. Server vytvoří jednu instanci hry pro jednoho hráče a jednu instanci hry pro více hráčů. Hráči se pak mohou na server přihlásit z menu *Server → Connect*.

Spuštění serveru:

> java -jar 5.4DeadloidFinal.jar --server --port *<číslo portu>*

Server bude očekávat příchozí spojení na portu *<číslo portu>*. Port není nutné na příkazové řádce uvádět, přednastavená hodnota je 50505.

Po spuštění server vypíše tuto hlášku:

Server started at IP 0.0.0.0 port 50505. Waiting for connections …

Poté se mohou začít hráči připojovat k serveru.

Server dále vypisuje informace o připojení/odpojení hráče k serveru a připojení a odpojení se od hry. Například následující řádka se zobrazí po připojení hráče k serveru a podává informaci o tom, odkud se hráč připojil (server a port hráče):

Spojení přijato od: localhost:50395

Po připojení hráče ke hře se zobrazí informační řádka podobná této:

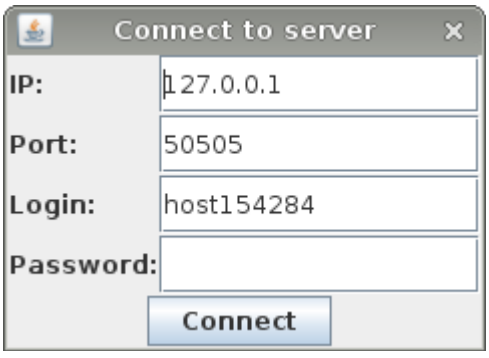
Hráč PlayerConnect(localhost/127.0.0.1, 50395, 47872, [deadloids.Player@809b2773]) se připojil ke hře 48492

V řádce je zobrazena informace o připojeném hráči (adresa serveru, port pro TCP komunikaci (viz port informační řádky výše), port hráče pro UDP komunikaci s hrou a port na kterém přijímá UDP packety hra.

Server se ukončí klávesovou kombinací CTRL+C.

Hra pro více hráčů

Pro hraní her více hráčů je nutné se nejdříve připojit k serveru. To se provede z menu *Server → Connect*. Objeví se dialog, kam je nutné zapsat IP adresu počítače, kde běží server, číslo portu na kterém server poslouchá a login.



Obr. B.2: Connect to server

Login by měl být email, ale vzhledem k tomu že server zatím nepodporuje registraci a autorizaci, lze vyplnit libovolný text. Server pouze kontroluje, zda je login unikátní (nikdo jiný se pod stejným loginem ještě nepřihlásil). Ze stejného důvodu není třeba vyplňovat heslo (server jej nekontroluje).

Po připojení k serveru se stane aktivní volba v menu *Player → Multiplayer.* Po jejím vybrání se otevře dialog se seznamem všech běžících her na serveru. Poklepáním na řádek se hrou se program pokusí připojit ke hře. Podmínkou úspěšného připojení je, že hra není obsazená (každá hra má omezený maximální počet připojených hráčů).

Účelem hry je nasbírat co nejvíce bodů zabíjením protihráčů, Ufa a ničením asteroidů. Bodování je stejné jako ve hře jednoho hráče, pouze za zničení lodi se odebírá 500 bodů. Hráč má ve hře nekončený počet životů a hra nikdy nekončí (v příštích verzích by měla hra skončit po dosažení daného množství bodů jedním hráčem nebo po časové prodlevě).

Hráč může vystřelit jednu raketu za 3 vteřiny. Po výstřelu se kolem jeho rakety objeví čárkované kolečko, které ubývá do chvíle než může hráč znovu vystřelit.

Testování grafiky

Testování grafiky lze spustit ve třech módech: deadloid, linear a random. Jednotlivé módy jsou popsány v kapitolách *4.3.1 Mód random*, *4.3.2 Mód deadloid* a *4.3.3 Mód linear* absolventské práce.

Spuštění testování:

> java -jar 5.4DeadloidFinal.jar --test <*test*> --operating-system <*název OS*>

Kde <*test*> je jedna z následujících možností: deadloid, linear, random a <*název OS*> je libovolný název operačního systému, který se vypisuje s naměřenými hodnotami.

Po spuštění testu začíná 5vteřinové odpočítávání, po kterém začne testování. Výsledky se vypisují na standardní výstup.

Sloupce v módu deadloid znamenají (postupně zleva): Název operačního systému, FPS.

Sloupce v módu linear a random znamenají (postupně zleva): Název operačního systému, FPS, počet Spaceship, počet Ufo, počet Raket, počet největších Asteroidů, počet středních asteroidů a počet nejmenších asteroidů.

Testování sítě

Testování sítě lze spustit ve dvou módech: randomServer a deadloidServer. Jednotlivé módy jsou popsány v kapitole *4.4 Testování přenosu dat po síti* absolventské práce.

Pro testování je třeba zapnout nejdříve server a pak klienta. Server se spustí následujícím příkazem:

> java -jar 5.4DeadloidFinal.jar --test [*randomServer* | *deadloidServer*]

Po spuštění serveru se vypíše hláška „*Max sprites per packet = 48*“ a až poté je možné spustit klienta.

Klient se spustí příkazem:

> java -jar dist/5.4DeadloidFinal.jar --test net --server-address <*IP*> –text-view

Klient se pokusí připojit k serveru na zadané IP adrese. Volba --text-view je nepovinná a způsobí, že se program nepokusí otevřít grafické rozhraní. (To může být užitečné, pokud se klient spouští na vzdálené pracovní stanici bez grafického rozhraní).

Po úspěšném spojení začne server vypisovat statistická data.

Sloupce v módu randomServer i deadloidServer znamenají (postupně z leva): Doba přenosu dat od serveru ke klientovi a zpět v milisekundách, počet spritů v přenášených datech a počet přenesených bytů.

Server i klient se ukončí klávesovou zkratkou CTRL+C.

Server přijímá pouze jedno spojení, na které pak odesílá packety. Po ukončení klienta už server žádné nové spojení nenaváže a je nutné jej ukončit a znovu spustit.

C. Obsah přiloženého CD