Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Problematika multiplayerových her na platformě Android

Plzeň 2015 Matěj Sutr

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 25. června 2015

Matěj Sutr

Poděkování

Rád bych tímto poděkoval vedoucímu práce Ing. Ladislavu Pešičkovi za odborné vedení a čas věnovaný konzultacím. Dále děkuji svým rodičům a přítelkyni za podporu.

Abstract

The problematics of multiplayer games on Android platform
The aim of this thesis is to explore possibilities for multiplayer games on the Android platform. The theoretical part describes the basic attributes of these games, the differences compared to PC games, and the ability to transfer game data. The following part is devoted to the communication protocols used in multiplayer games and the possibilities of implementing physics library. The outcome of this work is the application FavRacer - top down view real-time action racing game. The applicability is analyzed and measured in various conditions of the network connection at the end of the work.

Problematika multiplayerových her na platformě Android
Cílem této práce je prozkoumání možností multiplayerových her na platformě Android. Teoretická část se zabývá popisem základních atributů takových her, jejich odlišností oproti hrám na PC a možnostem pro přenos herních dat. Následující část práce se věnuje komunikačním protokolům používaným v multiplayerových hrách a možnostem implementace knihoven fyzikálního světa. Výstupem práce je aplikace FavRacer - akční realtimová závodní hra z pohledu shora. Na té je v závěru práce analyzována a měřena použitelnost v různých podmínkách síťového připojení.

Obsah

1 Úvod

Tématem práce jsou multiplayerové hry na platformě Android, která má v současnosti nejvyšší zastoupení na trhu mobilních zařízení. V obchodu Google Play s aplikacemi pro OS Android existuje nepřeberné množství her. Zatímco na tradičních herních platformách (PC, herní konzole) jsou síťové hry více hráčů velmi populární a mají širokou podporu uživatelů, v nabídce her platformy Android stále převládají jednoduché hry pro jednoho hráče.

Na hardwarové i systémové rozdíly v možnostech platformy Android ve srovnání s tradičními herními platformami se zaměří první část této práce. V další části textu budou prozkoumány běžně dostupné síťové technologie pro přenos dat, které se dají použít pro vytvoření multiplayerové hry. Zároveň se bude práce soustředit na analýzu možností pro implementaci fyzikálních knihoven, které jsou v multiplayerových hrách často problematickým prvkem kvůli náročnosti zavedení dalších mechanismů potřebných pro udržování referenčního stavu herního světa u všech hráčů.

Praktická část práce popisuje konkrétní řešení, kterým je vytvořená mobilní aplikace FavRacer a její desktopová verze. Jedná se o multiplayerovou závodní hru s možností střelby po soupeřích. Aplikace využívá fyzikální knihovnu (konkrétně JBox2D) umožňující přirozenější reakce a interakce herních entit. Pomocí obou vytvořených verzí aplikace budou otestovány různé možnosti síťového připojení. Ty zahrnují připojení v lokální síti Wi-Fi, připojení ke vzdálenému serveru prostřednictvím Wi-Fi i prostřednictvím mobilního datového spojení různých technologií.

2 Síťové hry na platformě Android

Android je otevřená platforma pro mobilní zařízení vyvíjená společností Google. Hlavní složkou platformy je operační systém založený na upraveném jádru Linux poskytovaný výrobcům mobilních zařízení. Platforma také zahrnuje řadu nástrojů jako například Android SDK pro vývoj aplikací. Představen byl v roce 2008 a jeho zastoupení na trhu s mobilními zařízeními v současné době výrazně překonává konkurenční systémy. Základním programovacím prostředkem pro aplikace na platformě Android je vysokoúrovňový jazyk Java.

Poslední uvedenou verzí platformy je Android 5.0 Lollipop, jehož podpora a dostupnost ale záleží na výrobcích konkrétních modelů mobilních zařízení. Výrobci elektroniky dnes používají Android i v jiných zařízeních, než v mobilních telefonech a tabletech. Trendem současnosti je nositelná elektronika (Wearables) s OS Android, kam se řadí například hodinky a náramky. Dalším uplatněním systému jsou multimediální prvky v automobilech, fotoaparáty, televizory nebo různé domácí spotřebiče. Tato práce se soustředí na tradiční prostředí mobilních telefonů a tabletů, které je více univerzální.

Následující podkapitoly vysvětlují základní pojmy spojené se síťovými hrami a popisuje odlišnosti v jejich návrhu a implementaci od dalších tradičních platforem.

2.1 Multiplayerová hra

Multiplayerová hra je hra, která umožňuje současnou interakci více hráčů. Této interakce může být dosaženo na jednom zařízení o které se hráči střídají (Pass-and-Play) nebo které využívají společně (například s využitím dělené obrazovky - Split Screen). Odlišný přístup uvažuje hru na více propojených zařízeních. Základem takových her je síťová komunikace, která umožňuje výměnu a sdílení dat mezi hráči. Díky ní je možné zajistit shodný stav herního světa na všech připojených zařízeních, nebo se tomuto stavu bude alespoň blížit. Tato práce se zaměřuje především na výrazně komplikovanější přístup založený na síťové komunikaci.

2.2 Rozdíly oproti hrám pro PC

Ve srovnání se hrami vytvářenými pro herní konzole nebo platformu PC se hry pro mobilní platformy liší v množství zásadních aspektů. Ty nejdůležitější rozdíly jsou popsány v následujících podkapitolách.

2.2.1 Ovládání

Hráči u mobilního zařízení nemají až na výjimky k dispozici periferie typu klávesnice, myš nebo gamepad a primárním ovladačem se stává dotyková obrazovka zařízení. Množství současně registrovaných doteků záleží na konkrétním zařízení. Zařízení ASUS MeMO Pad, které je v této práci použito jako testovací zařízení pro měření výsledků, registruje až deset doteků současně. Hráči na hladké obrazovce zařízení mají jen minimální hmatovou odezvu. Mobilní telefony obvykle umožňují využívat vibrace, tablety ale většinou ani tuto možnost nemají. Z toho důvodu musí být hry přizpůsobeny co nejjednoduššímu způsobu ovládání. Mezi nejúspěšnějšími herními tituly převažují jednoduché hry, které se ovládají jedním prstem (Angry Birds [1], Candy Crush Saga [2], Fruit Ninja [3]).

Oproti počítačům mobilní zařízení obvykle umožňují využívat množství senzorů [4], které mohou sloužit i k ovládání hry. Nejčastěji využívaným senzorem je akcelerometr (někdy označovaný jako gravitační čip), který měří pohybové zrychlení přístroje. Akcelerometr je často využíván v kombinaci s gyroskopem, který určuje polohu (natočení) zařízení v prostoru. Oba senzory umožňují ovládat hry určenými pohyby zařízení. Ve hře Asphalt 7 Heat [5] se například závodní vůz ovládá pohybem zařízením simulující pohyb reálného volantu vozu.

Dalším senzorem mohou být například magnetický senzor (kompas), senzor okolního světla nebo senzor přiblížení (proximity senzor). Jejich využití ve hrách je ale ojedinělé a většinou jako experimentální nebo doplňková funkce.

Kromě senzorů je možné využít i další hardwarové výbavy mobilních zařízení. Často aplikace pracují s daty z GPS navigace. Příkladem může být aplikace Ingress [6], kde dva tábory hráčů svým pohybem ve vybrané lokalitě bojují o území (musí se se zapnutou aplikací fyzicky dopravit na místo). Výjimečně aplikace pracují i s daty z mikrofonu zařízení (například ve hře Dumb Ways To Die [7] je nutné ve správnou chvíli fouknout do mikrofonu).

Seznam dostupných senzorů jednotlivých zařízení je různý a liší se i konkrétní API, které zavádí jejich podporu. Proto je nutné v aplikaci jejich přítomnost detekovat a ovládání konkrétnímu zařízení adekvátně přizpůsobit. Popis senzorů, způsob jejich detekce a tabulka API je dostupná na webových stránkách [8].

Rozdíly platforem v ovládání je nutné zohlednit při návrhu konkrétní hry. Z hlediska multiplayeru není v práci se vstupy nijak zásadní rozdíl. Liší se maximálně typ dat, které udávají konfiguraci ovládacího prvku.

2.2.2 Velikost obrazovky

Velikost obrazovky mobilních zařízení je výrazně menší než u standardních monitorů, přesto její rozlišení může být stejné nebo dokonce vyšší. Přehled aktuálně využívaných rozlišení uvádí [9]. Protože se ale zařízení ovládá dotykovou obrazovkou, musí být ovládací prvky dostatečně veliké, aby se daly pohodlně používat a současně je nutné počítat se zakrýváním obrazovky obsluhujícími prsty. Uživatelské rozhraní tak musí být navrženo zcela odlišně než u klasických her pro stolní počítače a existuje pouze mizivé množství aplikací, které mohly být přímo portovány na mobilní zařízení bez změny uživatelského rozhraní. Při návrhu mobilní aplikace je kromě aspektů ohledně uživatelského rozhraní nutné počítat i s faktem, že velikost obrazovky i poměr stran se u různých zařízení může lišit (od 240 x 320px až po vysoké rozlišení 1440 x 2560px u zařízení Samsung G920F Galaxy S6).

2.2.3 Omezená výkonnost zařízení

Na trhu se vyskytuje nepřeberné množství zařízení na platformě Android s diametrálně odlišnou hardwarovou výbavou. Nedá se říct, že zařízení s vyšším API bude mít vyšší výkon než zařízení starší platformy. Počet jader CPU, jejich konfigurace, architektura, frekvence, velikost operační paměti, rozlišení displaye nebo podporovaná API se model od modelu liší.

Pro ilustraci může sloužit porovnání jednoho z prvních zařízení platformy Android Dev Phone 1 (prosinec 2008) s moderním zařízením HTC Nexus 9 v tabulce 2.1.

Zařízení	**Android Dev Phone 1**	**HTC Nexus 9**
API	Android 1.0	Android 5.0
Typ procesoru	Qualcomm MSM7201A	NVIDIA Denver 1.0
Frekvence	528 MHz	2300 MHz
Počet jader	1	2 (64-bit)
Operační paměť	192 MB	2048 MB
Display	3.2", 320 x 480px	8.9", 2048 x 1536px
Kapacita úložiště	256 MB	16 GB flash

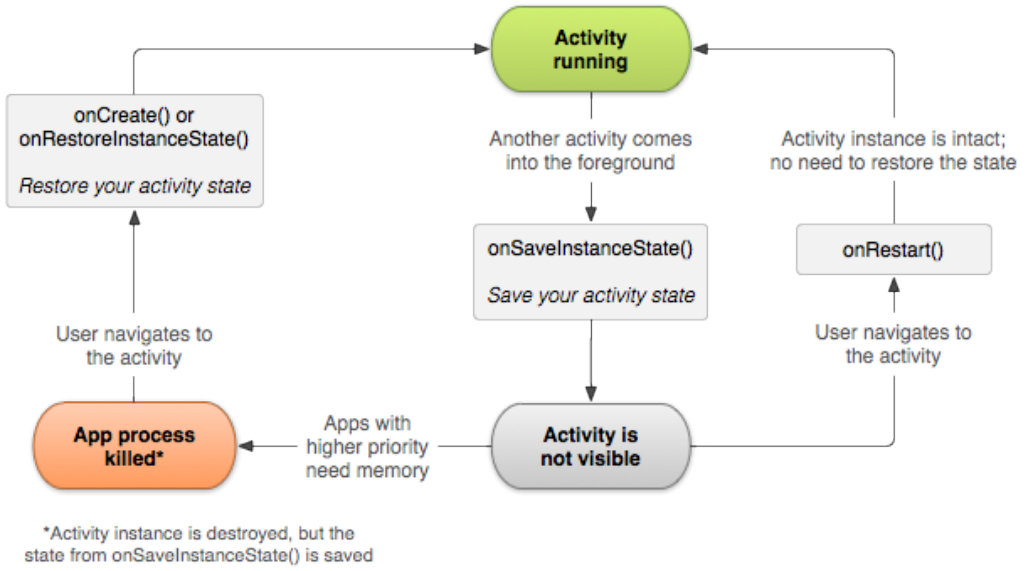
Tabulka 2.1: Porovnání extrémně odlišných zařízení platformy Android

V porovnání s platformou PC jsou obecně zařízení s operačním systémem Android ale pořád výrazně pomalejší. S tím je potřeba počítat při návrhu hry, především při návrhu rozdělené logické vrstvy mezi server a klienty, jak je popsáno v kapitole 2.5. Je také vhodné stanovit si minimální konfiguraci, pro kterou bude vytvořená aplikace poskytovaná. Některé parametry lze nastavit v manifestu aplikace další parametry je možné nastavit například při publikaci aplikace na obchod Google Play.

2.2.4 Proměnné prostředí

Na rozdíl od platformy PC existuje v operačním systému Android celá řada faktorů, které mohou přerušit nebo dokonce zcela ukončit spuštěnou aplikaci. Kromě zřejmých přerušení aplikace při událostech vyvolaných hardware (vybitá baterie, ztráta signálu, vyčerpání paměti RAM, zaplnění úložného prostoru) může totiž docházet i k přerušením jinou aktivitou. Typickým příkladem u mobilních telefonů je přerušení aplikací pro příjem telefoního hovoru, přijetí SMS nebo aktivace budíku. V takové situaci dojde k pozastavení spuštěné aplikace nebo dokonce jejímu ukončení.

Na následujícím obrázku 2.1 je znázorněno schéma přepínání aktivit v operačním systému Android. Při spuštění jiné aktivity dochází k přenosu aktivity do pozadí. Pokud má spuštěná aktivita vyšší prioritu a vyžaduje paměť, kterou původní aktivita používala, dojde k jejímu ukončení. Pokud chceme umožnit uživatelům návrat do aplikace ve stavu před jejím přenesením do pozadí, je potřeba uložit data pro její rekonstrukci do persistentního úložiště (k tomu slouží metoda *onSaveInstanceState()*). při návratu do původní aktivity je zavolána metoda *onRestoreInstanceState()*, která pak musí obsahovat kód pro načtení a aplikaci uložených dat.



Obrázek 2.1: Přepínání aktivit při přerušení, zdroj [11]

2.3 Počet hráčů

Při návrhu hry je nutné stanovit vhodný počet hráčů. Tento počet je určen především scénářem a typem hry. Při vysokém počtu hráčů je potom nutné zohlednit vyšší nároky na komunikační i grafickou složku hry, což může mít výrazný negativní dopad na herní prožitek. Například hra World of Tanks [10] má uživatelskou základnu s více než 90 miliony hráčů (z toho více než 10 milionů na platformě Android), ale umožňuje souboje maximlně 7 proti 7, protože s vyšším počtem hráčů ve hře už by nebylo možné zajistit dostatečný komfort s ohledem na různě výkonná zařízení hráčů.

2.4 Technologie pro přenos dat

Mobilní zařízení na platformě Android standardně umožňují pro bezdrátový přenos dat využít celou řadu technologií. Rozdíl mezi nimi je v energetické náročnosti, přenosové rychlosti i v podporovaných komunikačních protokolech a způsobu jejich využití. Z toho plyne vhodnost jednotlivých přenosových technologiích vzhledem k účelu funkce v konkrétní aplikaci. Výhody a nedostatky jednotlivých technologií budou popsány v následujících podkapitolách.

Pro jednoduchou výměnu dat mezi více klienty je vhodné použít Bluetooth [13] nebo Wi-Fi Direct [14] u zařízení s Androidem verze 4.0 nebo novějším. Obě tyto technologie poskytují hotové API s potřebnými metodami pro navází spojení, ověření uživatele, odeslání souboru nebo například detekci výpadku. Obsahují také základní mechanismy pro zabezpečení přenosu. Při jejich využívání není potřeba dalších síťových prvků a jejich využití není nijak zpoplatněno ani omezeno.

Pokud se mobilní zařízení nachází v dosahu lokální bezdrátové WLAN (Wireless Local Area Network) sítě, je možné připojit se pomocí Wi-Fi k přístupovému bodu. Konfigurace síťových prvků pak může připojeným zařízením navíc umožnit přístup do internetu. Tento datový přenos obvykle není zpoplatněn a jeho omezením mohou být pravidla a zákazy v místní síti nastavená jejím správcem.

Většina zařízení platformy Android může také sloužit jako přístupový bod (Wi-Fi hotspot) ke kterému se ostatní klienti mohou připojovat. To se často využívá v kombinaci s funkcí tethering (viz [15]), která sdílí klientům hotspotu mobilní internet.

Mimo dosah WLAN je možné pro přístup k internetu využívat mobilní připojení. V současné době je mobilním internetem pokryta většina České republiky, ale síla signálu a dostupná technologie se rapidně liší v závislosti na lokalitě. Tento typ datového přenosu je zpoplatněn telefonními operátory a účtuje se na základě objemu přenesených dat (obvykle jako předplacená služba s nastaveným měsíčním limitem čerpání). Je také třeba myslet na to, že ne všechna zařízení jsou vůbec vybavena slotem na SIM kartu a že mají uživatelé předplacený datový tarif. Mobilní telefony kromě vzácných výjimek kartu SIM obsahují (minimálně pro účely volání), ale u tabletů se stále jedná o nadstandardní výbavu.

Pro účely této práce je nutný přístup k serverové aplikaci na vzdáleném serveru. Z toho důvodu byly zvoleny dostupné technologie s možností přístupu do internetu, tedy Wi-Fi a mobilní internet různých typů.

2.4.1 Bluetooth

Bluetooth je technologie se specifickým hardwarovým vybavením a softwarovým řešením, které existuje pro nespočet platforem a elektronických zařízení včetně domácích spotřebičů nebo hraček. Pro Bluetooth je typická symet-

rická komunikace (Peer-to-Peer, P2P), kdy připojená zařízení nepřistupují k jednomu hlavnímu prvku, ale komunikují v rovnocenných rolích.

Podle [12] je v poslední verzi přidána podpora Low-power IP, která slibuje zpřístupnění jeho funkcí pro účely v současnosti marketingově významného IoT (Internet of Things - propojení prvků domácnosti).

Základní scénář využití (odeslání/příjem souboru) se při implementaci většinou omezuje na připravené API a základní metody. Další využití Bluetooth je v zařízeních jiných platforem, která je možné díky speciálnímu protokolu snadno připojovat. Jejich připojení nijak neomezuje přístup zařízení k Wi-Fi nebo mobilnímu internetu. Typickým příkladem mohou být sluchátka, reproduktory, herní periferie, klávesnice, myši, hodinky nebo například tiskárny.

Bluetooth existuje na mobilních zařízeních v různých verzích, které jsou pevně svázány s hardwarovým prvkem zařízení a záleží tak na výrobci, kterou verzi použije. Nejstarší oficiální verze je v1.1 a v současnosti nejnovější je v4.2, která byla vydána v prosinci 2014. Rozdíl mezi verzemi je v rychlosti přenosu dat (až 24 Mbps), maximálním dosahu (až 100m), optimalizaci přenosu a zabezpečení.

2.4.2 Wi-Fi Direct

Wi-Fi Direct je technologie používaná na platformě Android 4.0 (API level 14) a novější přes API Wi-Fi Peer-to-Peer (P2P), které popisuje [16]. To stejně jako API pro Bluetooth umožňuje jednoduché párování více zařízení a komunikaci se zařízeními v dosahu sítě poskytované jedním ze zařízení. Využívá standardních HW prvků pro přístup k Wi-Fi a díky tomu poskytuje výrazně vyšší přenosové rychlosti než Bluetooth (podle [17] až 250Mbps).

API Wi-Fi P2P je z hlediska multiplayerových her často využíváno. Jeho nevýhodou z pohledu této práce je odstínění od základních metod a omezení adaptéru Wi-Fi na jedno spojení, které brání současnému připojení do místní sítě a do hry. Nebylo by tedy možné použít stejný program pro Peer-to-Peer spojení a pro spojení server-klienti, které je použito pro test mobilního připojení.

2.4.3 Wi-Fi

Wi-Fi označuje sadu standardů IEEE 802.11 pro komunikaci v bezdrátové lokální síti (WLAN). Existuje několik verzí, které se liší kmitočtem používaného rádiového pásma (od 2.4GHz po 60GHz) i přenosovou rychlostí (od 2Mbps až po 1800Mbps).

V prostředí WLAN je možné vytvářet jak spojení Peer-to-Peer, tak spojení server-klienti. Je také možné jejím prostřednictvím komunikovat přes internet se vzdáleným serverem a testovat tak rozdíly oproti mobilní síti. Všechny vyjmenované možnosti Wi-Fi jsou využívány v implementované aplikaci FavRacer.

2.4.4 Mobilní internetové připojení

Mimo dostupnou lokální síť je možné používat placený mobilní internet (pokud zařízení obsahuje SIMkartu s předplaceným datovým tarifem). Na území České republiky poskytují telefonní operátoři řadu technologií, které se výrazně liší svými vlastnostmi. Popis těchto technologií uvedený v následujících odstavcích čerpá z článku [18] (2G) a z jeho pokračování [19] (3G) a [20] (4G).

Pro základní orientaci v jednotlivých typech internetového spojení poslouží přehled těch nejrozšířenějších. Začlenění do generací se v různých pramenech rozchází, ale pokud vycházíme z [18] a [21], pak můžeme pro přehled použít následující tabulku 2.2.

Generace	**Typ sítě**
1G	NMT
2G	GSM, HSCSD, CDMA
2.5G	GPRS, EDGE
3G	UMTS, CDMA
3.5G	HSDPA, HSUPA, LTE, WiMax
4G	LTE Advanced, WiMax 2.0

Tabulka 2.2: Rozdělení technologií pro mobilní internet podle generací

Obecně lze konstatovat, že každá generace přináší výrazný nárůst přenosové rychlosti. Dále obecně platí, že používání novějších generací přináší vyšší spotřebu energie koncových zařízení.

NMT

NMT (Nordic Mobile Telephone) byla používána v devadesátých letech pro přenos hlasu. Přestože se jedná o přenos dat, nelze tuto síť použít k mobilnímu internetu. Data byla přenášela analogově.

GSM, HSCSD

I tyto technologie byly používány primárně k přenosu hlasu, ale už využívaly digitální přenos. GSM (Global System for Mobile Communications) pracuje ještě s přepojováním okruhů.Pro přenos dat se u GSM používá protokol HSCSD (High-Speed Circuit-Switched Data), který má přenosovou rychlost 36kbps až 43kbps. Datové přenosy v těchto sítích jsou zpoplatněny podle času, ne podle objemu přenesených dat.

GPRS

GRPS (General Packet Radio Service) už lze zařadit do generace 2.5G. Při přenosu dat u něj dochází k přepínání paketů, které umožňuje sdílet jeden přenosový kanál více uživatelům. Maximální přenosová rychlost dosahuje 128Kbps.

EDGE

Technologie EDGE (Enhanced Data rates for GSM Evolution) přináší další vylepšení v podobě zavedení 8PSK šifrování. To díky nahrazení každé trojice bitů jediným symbolem umožňuje lepší modulaci přenosu a maximální přenosová rychlost tak dosahuje až 256Kbps.

UMTS

UMTS (Universal Mobile Telecommunication System) je zástupcem sítě 3G, který může dosahovat přenosové rychlosti až 2Mbps. Obsahuje QoS (Quality of Services) se čtyřmi třídami služeb pro řízení přenosu: konverzace, jednosměrné vysílání, obousměrné vysílání a základní přenos.

CDMA

Dalším zástupcem 3G sítě je CDMA (Code Division Multiple Access). Ten pokrývá množství standardů s různými přenosovými rychlostmi od 144kbps po 9,3 Mbps pro stahování (download) a 5,4 Mbps pro odesílání (upload).

HSDPA

HSDPA (Evolved High-Speed Downlink Packet Access) oproti CDMA obsahuje vyčleněný kanál pro uživatelská data, což mu umožňuje efektivněji řídit přenos a dosahovat rychlosti až 10Mbps v základní verzi. Novější verze sítě umožňují přenos rychlostí až 21Mbps a HSDPA+ dokonce až 42Mbps pro download a 11Mbps pro upload.

HSUPA

Technologie HSUPA (Evolved High-Speed Uplink Packet Access) vychází z HSDPA a obsahuje navíc posílený kanál pro odesílání dat (EUL - Enhanced UpLink). Jeho nevýhodou je nižší přenosová rychlost okolo 14Mbps pro download (5Mbps upload) a nutnost posílení výkonu vysílacích stanic pro její zajištění.

LTE

LTE (Long Term Evolution) se svou specifikací blíží parametrům 4G sítí a v České republice se pro něj u mobilních operátorů označení 4G běžně používá. Nabízí přenosovou rychlost 100Mbps pro download, 50Mbps pro upload a při použití více antén je její teoretická rychlost přenosu až 326,4Mbps pro download a 172,8Mbps pro upload. Operátoři také využívají různé šířky pásma, čímž mohou vlastnosti sítě dále profilovat.

Jedná se o čistě paketovou síť s IP adresováním. To, že vychází z GSM a UMTS, jí dává výhodu zpětné kompatibility a větší podporu u poskytovatelů mobilního internetu i výrobců mobilních zařízení.

WiMax

Síť WiMax (Worldwide Interoperability for Microwave Access) sice umožňuje snadnější pokrytí signálem než LTE (signál má větší dosah), ale není kompatibilní s GSM a W-CDMA a je vhodná spíše pro stacionární stanice. Její rychlosti se pohybují okolo 128Mbps pro download a 56Mbps pro upload.

LTE Advanced

Jak už název napovídá, síť LTE Advanced vychází z LTE a posunuje její specifikace na úroveď 4G podle definice ITU (International Telecommunication Union), jak zmiňuje [22]. Hlavním přínosem má být opět nárůst přenosové rychlosti. Ta se u stacionárních přenosů má pohybovat okolo 1Gbps pro download a 500Mbps pro upload. U rychle se pohybujících zařízení (například při jízdě ve vlaku) má přenosová rychlost stále dosahovat rychlostí LTE. V současnosti (červen 2015) čeští mobilní operátoři především pokrývají území sítí LTE, přičemž pokrytí sítí LTE Advanced zatím není běžné.

WiMax 2.0

WiMax 2.0 je 4G varianta sítě WiMax, která stejně jako LTE Advanced navyšuje oproti původní implementaci přenosové rychlosti. Na rozdíl od LTE s tímto typem sítě čeští mobilní operátoři pravděpodobně do budoucna nepočítají vzhledem k uvedeným nevýhodám sítě WiMax.

2.5 Model komunikace

Vzhledem k typu vyvíjené multiplayerové hry je nutné zvolit optimální model komunikace. Podle [23] se typicky využívá jeden ze dvou typů spojení: **Klient-Server** nebo **Peer-to-Peer**. Uvedený článek potom popisuje specální přístup typický pro akční realtimové hry typu FPS (First Person Shooter - střílečka z pohledu první osoby). Tímto přístupem je **predikce na straně klienta** (Client-side prediction) a podrobněji se mu věnuje kapitola 4.2.

2.5.1 Klient-Server

Za základní komunikační schéma můžeme považovat spojení na bázi Klient-Server. V této konfiguraci existuje server, ke kterému jsou připojeni všichni účastníci hry a který řídí celou hru. Klienti mezi sebou nijak nekomunikují ani nemusí znát IP adresy ostatních účastníků. Server se nemusí nacházet v lokální síti, často se jedná o vzdálený server připojený do internetu, který je schopný obsluhovat všechny připojené klienty. Důležitým požadavkem na serverovou část bývá její škálovatelnost výkonu, tedy aby byla schopná obsloužit simultánně všechny připojené klienty a současně nevyžadovala nadměrné zdroje fyzického stroje, na kterém se nachází.

Pro multiplayerové hry je model Klient-Server idální už z důvodu výrazně lepších možností pro zabezpečení hry před podvody hráčů. Pro určení množství programového kódu logické vrstvy na straně klientů se v literatuře běžně označuje termíny tenký klient (thin client) a tlustý klient (fat client).

Tenký klient

Pro tenkého klienta je typické, že obsahuje minimum z logické vrstvy a slouží převážně jako terminál, který pouze zobrazuje obsah určovaný serverem. Tento přístup je rozhodně výhodný v případě multiplatformních aplikací, protože výrazně omezuje množství kódu specifického pro cílovou platformu. Další výhodou je možnost spravovat, sledovat, řídit a upravovat kód serverové části bez nutnosti přeinstalací na straně klientů. Kód serveru je také lépe chráněn (není součástí distribuovaných klienstkých aplikací). Nevýhodou tenkého klienta je závislost na serveru, bez kterého nemůže plně pracovat a jehož aktuální výkonnost má přímý dopad na rychlost odezvy klientské aplikace.

Tlustý klient

Pokud se na straně klientů vyskytuje větší část logické vrstvy aplikace, hovoříme o tzv. tlustém klientovi. Jeho výhodou je lepší odezva, protože je schopný zobrazovaný obsah počítat lokálně a nemusí čekat na zprávy ze serveru. Server pak může být výrazně méně vytížen, protože řídí pouze komunikaci mezi klienty a základní správu her. Protože je provoz serverů placený obvykle na základě přidělených prostředků, může být tlustý klient zvolen už kvůli snížení nákladů na provoz.

To jaká část logické vrstvy bude svěřena serveru a jaká bude přenechána klientům má samozřejmě přímý dopad na kvalitu hry a možnosti zabezpečení korektního stavu herního světa. V implementované aplikaci FavRacer je na server distribuována kompletní aplikace, ale její použití se liší v závislosti na roli. Pokud je spuštěna na serveru, pak se logická vrstva (řízení hry, počítání fyziky, atp.) počítá tam a připojení klienti pouze zobrazují stav určený serverem. V případě hry jednoho hráče nebo v roli serveru při Peer-to-Peer ale klientská aplikace obsahuje kompletní kód, takže není potřeba žádné další části na jiném serveru.

2.5.2 Peer-to-Peer

Pro specifické případy může být lepší volbou Peer-to-Peer. V tomto modelu jsou všichni klienti spojeni mezi sebou (nebo alespoň s částí klientů, se kterými komunikují). Jeho hlavní výhodou je rovnoměrné rozložení zátěže mezi připojená zařízení a možnost provozu aplikace bez jakékoliv serverové části. Tento model se často používá pro distribuované výpočty nebo sdílení souborů. V oblasti multiplayerových her se objevuje u těch nejjednodušších (není potřeba server) a převážně z historických důvodů i v žánru RTS her (Real-Time Strategy - strategie v reálném čase).

V případě multiplayerových her je vždy nutné zvolit správce hry, který udržuje referenční stav herního světa. Tato role se může přesouvat na další hráče například při změně síťových podmínek, ale vždy je za hru někdo zodpovědný. Článek [23] zmiňuje, že sdílení herního světa výrazně komplikuje možnosti zabezpečit hru před podvody i zabezpečení identického stavu herního světa pro všechny připojené klienty. Podle [24] je pak další nevýhodou předpoklad vyšších latencí a větší závislost na výkonu všech připojených zařízení (na serveru je obvykle možné zajistit škálování výkonu, u koncových zařízení klientů je výkon obvykle omezený). Více se také projevuje kvalita spojení jednotlivých klientů, což je faktor v modelu Server-klient zcela odstíněný.

3 Přenos herních dat

Tato kapitola popisuje komunikační protokoly, které jsou běžně používané v multiplayerových hrách k přenosu herních dat sítí. V podkapitole 3.2 potom uvádí obvyklé typy zpráv, které se sítí typicky přenáší v různých fázích hry.

3.1 Protokol

Základními prostředky pro přenos dat mezi hráči jsou dva typy socketů: TCP sockety a UDP sockety. Oba jsou postaveny nad protokolem IP a každý z nich má jisté výhody i nevýhody. Kromě nich existuje řada protokolů, které je možné využít pro výměnu dat mezi serverem a klienty. Za zmínku určitě stojí protokol SCTP [26], ENet [27] nebo RUDP založený na RDP [25], i přesto že se nestal standardem.

Další možností je využít některé z dostupných frameworků a knihoven, které komunikaci řeší interně a uživateli poskytují API s univerzálními funkcemi pro přenos dat. Jejich nevýhodou může být v případě komerčních produktů cena za licenci, uzavřenost, závislost na dalších službách nebo přílišné odstínění od komunikační vrstvy.

Tato práce se zaměří na standardizované protokoly TCP a UDP, které se pro komunikaci v multiplayerových hrách volí nejčastěji. Základním kritériem při výběru protokolu je maximální přípustná doba mezi odesláním zprávy a přijetím odpovědi druhé strany a potřeba bezpečného přenosu.

Výčet atributů podle [28] shrnuje v následujících podkapitolách vlastnosti obou protokolů.

3.1.1 TCP

- Udržuje spojení klientů (přerušení je ihned detekováno)
- Zaručuje spolehlivý a uspořádaný přenos dat
- Rozděluje odesílaná data do paketů podle potřeby

- Zajišťuje plynulost přenosu (flow control) - data nejsou odesílána rychleji, než je používaná síť schopna přenést
- Snadné použití: čtení a zápis dat probíhá jako při práci se souborem

3.1.2 UDP

- Klienti nejsou trvale připojeni (virtuální spojení musí být spravováno externě)
- Není zaručen spolehlivý přenos, pakety mohou být duplikovány, ztraceny nebo ve špatném pořadí
- Data větší než paket musí být rozdělena a při příjmu opět spojena
- Není zaručena plynulost přenosu (při příliš rychlém odesílání budou pakety ztráceny)
- Detekce ani opakované doručení ztracených paketů není nijak zajištěno

TCP protokol se z uvedeného popisu sice zdá ve všech směrech výhodnější, problém ale představuje jeho schopnost přenést data co nejrychleji. Tento protokol je navržen tak, aby data přenášel především bezpečně. Toho je docíleno tím, že v případě ztracení paketu nebo jeho poškození během přenosu se TCP zastaví a vyžádá si opětovný přenos paketu od odesílatele. Tato vlastnost je pro bezpečnost samozřejmě podstatná, ale její vliv na schopnost přijímat co nejrychleji doručované zprávy a udržovat tak stav světa aktuální vzhledem k reálnému času je pro většinu síťových her důvodem pro volbu UDP.

Pro ilustraci si představme následující dvojici síťových her pro dva hráče: V prvním případě se jedná o hru, kde proti sobě může dvojice připojených hráčů hrát pexeso. V druhém případě se jedná o síťovou hru, kde může dvojice hráčů střílet na sdílené terče. Zatímco u pexesa nezáleží na rychlosti přenosu (otočení kartiček může být zpožděno o několik milisekund bez dopadu na hru), pro druhý scénář je rychlost přenosu kritická (o výstřelu musíme být informováni co nejdříve, aby bylo možné určit, který z hráčů vystřelil dříve a zasáhl terč).

Hlavní rozdíl mezi oběma popisovanými hrami je v tom, že druhý scénář má požadavek na doručení paketu v reálném čase: Nezajímá nás, co se stalo před vteřinou, zajímá nás aktuální stav herního světa.

Další nepříjemnou vlastností TCP pro rychlost doručení zpráv je využití Nagleho algoritmu [29] (Nagle's algorithm) v kombinaci s potvrzovacími zprávami. Tento algoritmus má za úkol spojovat více krátkých paketů, které čekají na odeslání. Většinou se dá zakázat nastavením vlajky TCP_NODELAY u konkrétní implementace TCP. Efekt vypnutí Nagleho algoritmu při odesílání krátkých zpáv je popsán v [32]. V [33] je uveden další příklad, ale také upozornění, že tento algoritmus byl do TCP implementován i z důvodu ochrany internetu před zahlcováním krátkými potvrzovanými zprávami. Jejich vypnutí může způsobovat zahlcování sítě, které se projevilo i u implementované aplikace FavRacer. Další informace o tomto algoritmu lze získat například v RFC [30] a [31].

3.1.3 Rozšíření UDP

Pokud nám protokol TCP nevyhovuje z důvodu potřeby doručení paketu co nejrychleji, je možné vytvořit nadstavbu protokolu UDP, která zajistí potřebné vlastnosti, které samotný protokol UDP neposkytuje. Tento přístup je u realtimových her nejčastější, jelikož neobsahuje negativa TCP a jejich pozitiva přináší do protokolu UDP.

Nad rámec UDP jsou obvykle u multiplayerových her požadovány následující vlastnosti:

1. Správa připojených klientů a evidence ztráty spojení
2. Spolehlivý a uspořádaný přenos dat pro klíčové zprávy
3. Dělení odesílaných dat do paketů podle potřeby
4. Zajištěná plynulost přenosu (flow control)

Správa připojených klientů

UDP protokol na rozdíl od TCP nepracuje s navázanými spojeními, pouze přijímá pakety na otevřeném kanálu. Pro potřeby multiplayerových her je obvykle nutné přidat vrstvu, která zajistí výměnu přihlašovacích údajů mezi hráči a jejich autentizaci. Po připojení všech hráčů do hry je možné v této vrstvě filtrovat pakety přicházející z jiných adres než přihlášených, případně

zajistit aktualizaci uložených údajů, pokud se adresa klienta od přihlášení změnila.

Současně je nutné sledovat, zda je klient stále dostupný. U TCP se výpadek spojení okamžitě projevuje ukončením navázaného spojení a je možné na něj přímo reagovat. Případný výpadek spojení se v UDP projeví pouze tím, že od klienta přestanou chodit zprávy. U každého klienta můžeme evidovat dobu od poslední přijaté zprávy. Pokud je tato doba větší než povolené maximum, můžeme předpokládat výpadek spojení mezi klientem a serverem.

Reakce na takovou situaci pak záleží na konkrétní implementaci, můžeme například pouze odeslat na klientovu adresu požadavek na zaslání ověřovací zprávy a až po uplynutí stanovené doby klienta odpojíme, pokud odpověď nedorazí. U UDP je tento postup trochu ošidný především v tom, že není zaručeno bezpečné doručení zprávy s požadavkem ani příslušná odpověď. Z tohoto důvodu by měla být komunikace ideálně kontinuální, bez delších přestávek některé ze stran, aby byl případný výpadek spojení snadno identifikovatelný. Během síťové hry předpokládáme ze strany serveru v pravidelných intervalech informační zprávu o aktuálním stavu světa a ze strany klientů zase očekáváme pravidelnou informaci o příkazech na základě jeho konfigurace ovládacích prvků. Už tento základ komunikace zajistí pravidelný provoz, jehož výpadek může být snadno odhalen.

Spolehlivý a uspořádaný přenos dat

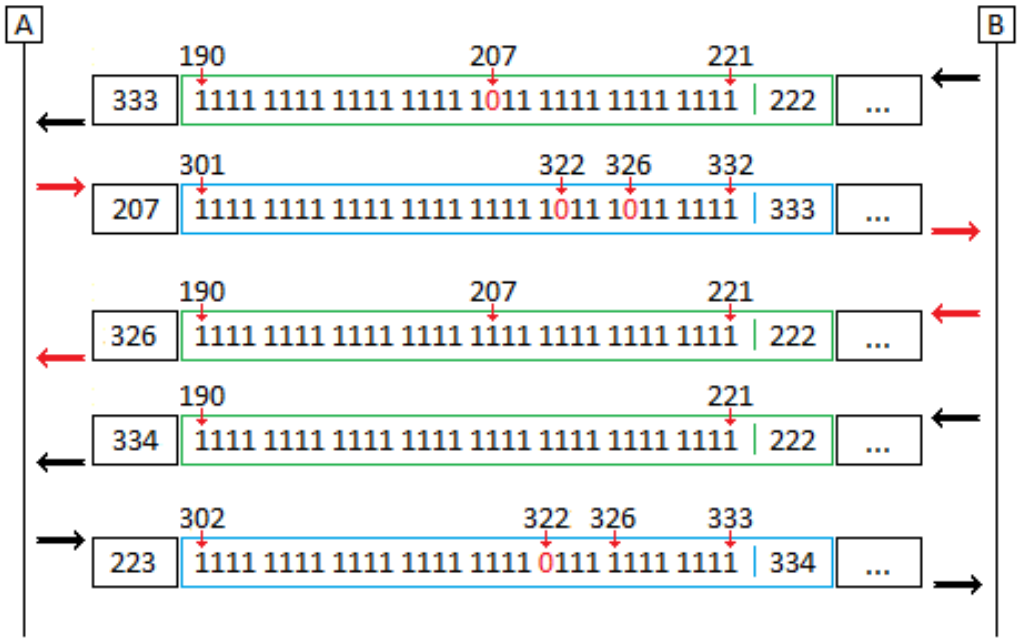
Abychom byli schopni určit správné pořadí přijatých zpráv, je nutné znát jejich pořadí při odesílání. Odesílané zprávy mohou být očíslované (do datové části paketu je uloženo číslo zprávy, tzv. sequencing). Na straně příjemce je potom možné kontrolovat, zda jsou obdržené zprávy ve správném pořadí a jestli některé zprávy v pořadí zcela nechybí.

Tento krok ale samostatně stále neumožňuje ani jedné straně identifikovat, která zpráva nebyla doručena a zda má být poslána opakovaně.

Obvyklá reakce na zjištěné nedoručení některé zprávy je vyžádání jejího opětovného odeslání. Samozřejmě musíme počítat i s tím, že každá zpráva se může cestou ztratit, tedy i žádost o přeposlání. Jak tedy zajistit, že obě komunikující strany budou vědět, které z jejich zpráv druhá strana obdržela a které ne? Do datové oblasti paketu může být přidáváno číslo ACK (acknowledged), které druhou stranu informuje o potvrzení přijetí s daným

číslem. Tato informace může být opět přibalena do datové oblasti paketu. Příjemce může snadno porovnat ACK číslo z přijaté zprávy s číslem poslední jím odesílané zprávy a v případě neshody odešle všechny zprávy, které má stále smysl odesílat vzhledem k současnému stavu světa a zatím nebyly potvrzeny. Toto ověření samozřejmě musí probíhat na obou stranách spojení (zprávy se mohou ztrácet v obou směrech a v obou požadujeme spolehlivý přenos).

Efektivnější řešení nabízí [34]. Základní myšlenkou je využití bitového pole pro uložení informace do každé odesílané zprávy společně s posledním ACK. Každý bit tohoto pole představuje předcházející zprávu posledního ACK. S využitím jediného 32 bitového integeru, který slouží jako bitové pole, je tak možné předat druhé straně informaci o 32 ACK (1) nebo NACK (0) předcházejících přijatých zpráv vzhledem k uvedenému poslednímu ACK (samostatný integer). Příjemce je tak schopný určit, které z jeho posledních 33 zpráv (32 + číslo poslední přijaté zprávy) úspěšně dorazily na druhou strahu spojení a které byly ztraceny. Opět může snadno určit, které zprávy je nutné poslat znovu. Ukázka komunikace je zobrazena na obrázku 3.1.



Obrázek 3.1: Ukázka potvrzování pomocí bitového pole

V této situaci posílá strana B straně A zprávu číslo 333 a kromě dat obsahuje ACK poslední přijaté zprávy číslo 222 od A a bitové pole pro potvrzení 32 předcházejících zpráv. Zvýrazněná nula v bitovém poli příjemci

signalizuje, že odesílatel neobdržel zprávu číslo 207. Ta byla určitě z A do B odeslána (číslo zprávy je inkrementováno s odesláním), ale pravděpodobně do B dorazila poškozená nebo se cestou zcela ztratila.

Reakce strany A je taková, že opětovně odešle požadovanou zprávu (stále má ještě smysl vzhledem ke stavu herního světa) a do datové části paketu opět přibalí ACK poslední obdržené zprávy od B (333) a bitové pole předcházejících zpráv. Protože zatím strana A ještě neobdržela zprávu číslo 322 a 326, je v bitovém poli na příslušných indexech nula (NACK).

Vzhledem ke stavu světa už zpráva 322 pro A nemá smysl, ale 326 je stále důležitá. Strana B tedy opětovně odesílá zprávu číslo 326. Číslo ACK se nezměnilo (poslední zpráva má číslo 207, což je méně než 222, kterou strana B již potvrdila v první zobrazené zprávě). Bitové pole nyní obsahuje samé ACK, protože jediná chybějící zpráva číslo 207 už je potvrzena.

Následuje zpráva z B do A číslo 334 a zpráva 223 z A do B. NACK v bitovém poli poslední uvedené zprávy zůstává (strana B o NACK ví, ale nijak nereaguje).

Konfigurace spolehlivého přenosu dat

Díky tomu, že je postup zajištění spolehlivého přenosu aplikován na obou stranách spojení, klient tak má přehled o zprávách, které od něj úspěšně obdržel server a server má přehled o zprávách, které úspěšně dorazily ke klientovi. Současně mají oba informaci o tom, které zprávy byly úspěšně doručeny a které zatím nebyly potvrzeny nebo které už není potřeba posílat vzhledem k současnému stavu světa.

Velikost datového zásobníku posílaných paketů je podle [35] v IPv4 možné bez rizika fragmentace a následného zpomalení přenosu volit do velikosti 576 bytů. Tento prostor může být ještě omezen hlavičkou paketu, která je určena na základě konkrétní konfigurace sítě. Jako bezpečná maximální délka datové části, která nebude mít problém během přenosu sítí, se dá považovat 512 bytů, přičemž rezerva pro IP hlavičku zbývá dostatečných 64 bytů. Velikost datového zásobníku se tedy typicky volí právě 512 bytů.

Prostor pro číslo zprávy, číslo potvrzení a bitové pole tedy při použití 32 bitových integerů v součtu tvoří 12 bytů, což není mnoho a pro data tak stále zbývá poměrně velký prostor. Samozřejmě je možné použít větší nebo menší

bitové pole. Doba pro označení výpadku spojení by měla odpovídat celkové době pro čekání na potvrzení všech zpráv potvrzovaných v bitovém poli. Jeho velikost tedy záleží na intervalu mezi zasílanými zprávami a maximální dobou čekání při výpadku spojení.

Pokud například očekáváme pravidelné zasílání 100 zpráv z A do B za sekundu a pro potvrzení použijeme pouze 32 bitové pole, může se stát, že bude potřeba opakovaně zaslat zprávu z minulosti, pro kterou už nebude v poli dostupný index. Taková hranice je v tomto případě 33/100 sekundy, tedy čas, po který jsou obě strany schopné přesně určit, které z jejich zpráv byly potvrzeny. Kdyby se během tohoto intervalu například nepodařilo opětovné zaslání důležité zprávy, nebylo by už možné ani na její nedoručení upozornit. Tento interval tedy také stanovuje dobu, po kterou je možné pokusy opakovat, než bude muset být taková zpráva bez potvrzení označena za nedoručenou a signalizuje tak nutnost opětovného odeslání původní zprávy s novým číslem (původní číslo už není jak potvrzovat).

Dělení odesílaných dat do paketů podle potřeby

Při dělení dat do více paketů je nutné uložit informaci o celkovém počtu paketů a do každého vytvořeného paketu potom vkládat informaci o pořadovém čísle ukládané části dat. Každý z dílčích paketů musí samozřejmě obsahovat hlavičku původní zprávy s jejím sekvenčním číslem, celkovým počtem paketů a pořadovým číslem konkrétní části.

Na straně příjemce pak stačí kontrolovat, zda přijatá zpráva obsahuje na místě pro informaci o počtu částí číslo větší než jedna. Pokud ano, bude spojena podle uvedeného čísla části s dalšími příchozími. Až s přijetím poslední části (není zaručeno, že pakety dorazí ve správném pořadí) můžeme zprávu považovat za přijatou a předat ji k dalšímu zpracování.

Datová část tak navíc bude obsahovat dvojici čísel: celkový počet součástí a pořadové číslo této části dat. Každé z těchto čísel může být v datové části uložena například jako datový typ byte.

Zajištěná plynulost přenosu

Základem pro zajištění plynulosti je nezahlcená síť. Pokud budeme do sítě posílat příliš velké množství zpráv, je možné, že se prvky sítě mezi odesí-

latelem a příjemcem zahltí a přestanou pakety stíhat odbavovat. Pro multiplayerové hry je samozřejmě žádoucí, aby obě strany měly co nejčerstvější informace a tak je cílem provádět výměnu zpráv mezi klienty a serverem co nejčastěji. Velké množství odesílaných zpráv ale vytvoří situaci, kdy zahlcení reálně hrozí a je proto na místě zavést mechanizmus, který mu bude předcházet.

V první řadě musíme být schopni zhoršenou odezvu sítě vůbec detekovat. Jako základní ukazatel může sloužit RTT (round trip time), který určuje dobu mezi odesláním zprávy a přijetím odpovědi. Při použití potvrzovacího mechanizmu je možné tuto dobu určit jako čas, který uplynul od odeslání zprávy po přijetí jejího potvrzení. Čím nižší RTT máme, tím rychlejší je celá komunikace. S vysokou nebo rychle narůstající hodnotou RTT můžeme předpokládat zahlcení sítě a následné problémy s přenosem.

Jediným možným řešením zahlcení sítě je omezení komunikace na nutné minimum. V případě, že hodnoty RTT jsou po určenou dobu dobré (v nastavených optimálních mezích), je možné frekvenci komunikace opět navýšit a zajistit tak kvalitnější zážitek ze hry. Aplikace ale musí být schopná na tyto skoky správně reagovat a přizpůsobovat se aktuální situaci. Určitě také není vhodné povolit časté změny frekvence zasílání zpráv (neustálé přeskakování mezi více režimy) vzhledem k režii a nárazově kolísajícímu oběmu dat v síti.

3.1.4 Implementované řešení

Do výsedné aplikace popsané podrobně v kapitole 5 byl implementován protokol UDP doplněný o správu připojených klientů a zajištění spolehlivého a uspořádaného přenosu zpráv přesně podle uvedeného popisu. Konkrétní detaily implementace jsou potom vysvětleny v kapitole 5.5.3 a 5.8.

Pro porovnání byl ale v aplikaci FavRacer použit i protokol TCP, u kterého můžeme předpokládat horší odezvy v realtimové hře. Výsledky porovnání obou protokolů jsou uvedeny v kapitole 6.

3.2 Typ přenášených dat

Tato kapitola popisuje základní typy dat, která je v multiplayerových hrách potřeba přenášet mezi serverem a klienty.

3.2.1 Data pro správu spojení

Při využití protokolu TCP stačí po připojení serveru oznámit verzi používaného protokolu (pro ověření shodnosti s protokolem serveru) a předat mu nastavení připojovaného hráče, které má význam pro ostatní hráče (zadané jméno, ID ve sdílené tabulce výsledků, vybraný vzhled herních prvků, atp.). Samotné spojení je vytvořeno v okamžiku otevření společného kanálu a až do odpojení hráče nebo ukončení hry může zůstat otevřené.

Při využití UDP je připojování o něco komplikovanější. Mezi hráčem a serverem není vytvořeno žádné virtuální spojení, navíc není u žádné ze zasílaných zpráv jistota doručení. Z toho důvodu je nutné nejprve informovat server, že adresa, ze které mu zpráva přišla, je adresa klienta, který se chce připojit k založené hře. Pokud ve stanovené době klient nedostane odpověď, musí zprávu odeslat znovu. V tuto chvíli se klient stal teprve kandidátem na připojení a server mu může zaslat žádost o potvrzení záměru. Tato zpráva se samozřejmě vzhledem k protokolu může cestou ztratit, proto by ji server měl odesílat ve stanoveném intervalu znovu, dokud neuplyne stanovená doba nebo nedostane odpověď. Když server obdrží klientovo potvrzení připojení, zná už jeho adresu a má potvrzeno, že i klient ví o svém přidání mezi připojené hráče. V následující komunikaci už může server i klient využít mechanismu pro zajištění spolehlivého připojení a bezpečně přenášet data uvedená u varianty TCP (protokol, nastavení, atp.).

Při změně adresy klienta (přechod mezi sítěmi) je nutné serveru opět prokázat svou identitu z nové adresy. Z toho důvodu je vhodné uvést už při připojování unikátní klientský ověřovací řetězec. Ten může být například po spuštění aplikace generován metodou *java.util.UUID.randomUUID()*. Pokud klient zjistí svou změnu adresy, musí serveru odeslat znovu žádost o připojení. Při filtrování zpráv, které server přijal z jiných adres než z klientských pak může snadno zjistit, že zpráva o připojení obsahující již zaregistrovaný unikátní řetězec přišla z jiné adresy. Následně může server data příslušného klienta aktualizovat a pokračovat bez přerušení hry nebo odpojení klienta.

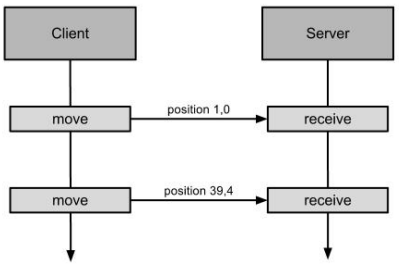
3.2.2 Herní data

V okamžiku připojení posledního zbývajícího hráče je možné zahájit hru. Typ herních dat záleží na typu hry, struktuře komunikační sítě a navrženém komunikačním protokolu. Podle [38] existují dva obecné přístupy, které jsou rozvedeny v následujících podkapitolách. Prvním z nich je řízení stavu herního světa podle konkrétních dat herního světa, druhý využívá událostí a jejich interpretace na straně serveru.

Aktuální data herního světa

V tomto případě klienti posílají serveru přímo aktuální stav svého herního světa. V herních zprávách jsou tak uloženy například konkrétní souřadnice herních prvků v daném čase, jejich rychlost a další atributy. Na straně serveru pak dochází pouze ke korekci souřadnic vzhledem ke stavům světa ostatních hráčů.

Tento přístup má hlavní nevýhodu v tom, že server musí být schopný odhalit případné podvody hráčů. Pokud bude hráč totiž podsouvat serveru souřadnice, které neodpovídají povoleným tahům (překročí maximální rychlost, výšku výskoku, atp.), může narušit stav hry pro všechny hráče. Obrázek 3.2 znázorňuje zjednodušenou komunikaci.

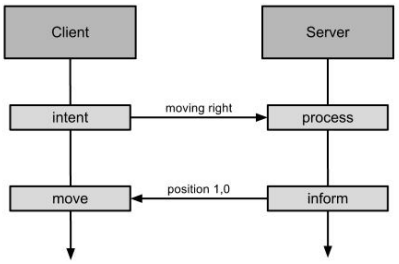


Obrázek 3.2: Odesílání aktuálních herních dat, zdroj [38]

Příkazy

Komunikační sítí proudí zprávy, které obsahují popis požadované akce (skok, střelba vlevo, pohyb nahoru, atp.). Na straně serveru je možné tyto zprávy

snadno interpretovat v kontextu herního světa. Nevýhodou je různé zacházení se zprávami na obou stranách spojení (viz obrázek 3.3).



Obrázek 3.3: Odesílání příkazů, zdroj [38]

4 Reprezentace fyzikálního světa

Pokud má chování objektů ve hře působit reálně, měl by být jejich pohyb řízen podle reálného světa. Existuje celá řada fyzikálních knihoven, které právě takové chování umožňují. Taková knihovna ve smyčce aktualizuje souřadnice zaregistrovaných objektů a na základě jejich definovné hustoty, pružnosti, tvaru, tření a dalších atributů určuje aktuální stav celého fyzikálního světa v závislosti na uplynulém čase. Pokud chceme, aby výsledek působil reálně, neměli bychom během simulace upravovat konkrétní hodnoty objektů (to by měl engine řešit sám). Na objekty je ale možné například aplikovat impulz síly nebo působit na jejich úhlovou rychlost.

Pro co nejplynulejší hru je nutné provádět výpočet aktuálního stavu světa co nejčastěji. Tato operace je ale náročná na výpočetní čas. Z toho důvodu je potřeba najít optimum, které představuje dostatečně plynulý fyzikální svět (pohyb objektů působí dostatečně plynule a reakce na kolize jsou dostatečně rychlé) při výkonu, který zvládne konkrétní zařízení poskytnout. Snížení zátěže je možné dosáhnout pomocí snížení frekvence přepočítávání fyzikálního světa. Zmenšením počtu stavů světa za sekundu je také možné snížit potřebnou komunikaci (není potřeba tak často informovat o novém platném stavu světa).

Fyzikální engine by měl pracovat na zcela odděleném vlákně, aby případné zpomalení při výpočtu nebrzdilo překreslování herní scény.

4.1 Fyzikální svět na serveru

Nejjednodušším přístupem pro přidání fyzikálního enginu do hry je přenechat řízení zcela na serveru, ke kterému jsou hráči připojeni. Na straně klientů pak stačí tenký klient, který serveru například odesílá informaci o poloze ovladače a přijímá od něj souřadnice, na kterých mají být herní objekty zobrazeny. Klientská aplikace tak nemá celkový přehled o fyzikálním světě, ale je schopná vykreslovat scénu a předávat požadavky serveru.

Tento scénář je vhodný pro takový typ her, kde nedochází k časté interakci objektů řízených různými hráči. Problém totiž nastává v situaci, kdy například dojde ke kolizi objektů dvojice hráčů. V tu chvíli každý z hráčů

posílá serveru žádost pohybovat svým objektem v určeném směru, ale odpověď od serveru dorazí až po přepočítání správného stavu světa. Ani jeden z hráčů nemá od doby odeslání požadavku po doručení nového stavu herní scény co vykreslovat a dochází tak k přerušování hry (zasekávání). Pokud k interakcím nedochází, pak tento jev není tak patrný, i když zpoždění mezi stiskem ovladače a reakcí herní scény bude vždy závislé na rychlosti odezvy serveru (RTT a režie pro výpočet stavu).

Pro akční hry, ve kterých navíc figuruje střelba, je tento přístup obvykle nevhodný. Prvním problémem je prodleva mezi střelbou a viditelnou reakcí, druhým problémem je zasekávání herní scény a nejzávažnějším problémem je otázka, ve kterou chvíli v herním čase vlastně bylo vystřeleno a který cíl tedy bude zasažen. Snadno totiž dojde k tomu, že hráč vidí na své obrazovce soupeře a vystřelí, přičemž vzhledem k jeho herní scéně má následovat úspěšný zásah. V herním světě na straně serveru ale ještě před obdržením požadavku ke střelbě došlo k obdržení žádosti soupeře o pohyb vpřed, čímž se dostává mimo osu střely. Výsledkem je nekonzistence mezi světy obou hráčů a server musí určit, který z nich bude tímto faktem poškozen. Je zřejmé, že k tomuto problému dochází z důvodu časové prodlevy mezi odesláním zprávy a obdržením odpovědi.

4.2 Predikce na straně klienta

Pro akční hry s častou interakcí herních objektů existuje model , který pomáhá řešit problémy popsané v předchozí kapitole. Podle [23] byl poprvé představen v počítačové hře QuakeWorld z roku 1996 a nazývá se predikce na straně klienta (Client-side prediction). Základní myšlenka spočívá v tom, že fyzikální svět existuje se všemi funkcemi nejen na serveru, ale i na zařízeních každého klienta. Fyzikální svět na straně klientů je schopný odvozovat stavy i kousek do budoucnosti vzhledem k poslednímu stavu světa ze serveru.

Predikce poskytuje klientům přístup k aktuálnímu stavu světa i ve chvíli, kdy čekají na odpověď serveru (podstatné pro plynulost herní scény). V případě, že dojde k různému stavu fyzikálního světa mezi hráči, rozesílá server referenční stav, do kterého se pak světy zasažených hráčů plynule přenesou (interpolovaný pohyb namísto skoku na správnou pozici).

Pro tento model je důležité udržovat referenční čas hry, ke kterému se pak dané stavy vztahují a měřit RTT, aby bylo možné určit, který stav je

vlastně serverem opravován a odkud musí být provedena nová predikce.

Největší výhoda modelu je patrná u reakční doby akcí klientů, kdy není třeba po každém stisku klávesy čekat na přenesení zprávy na server a zpět před požadovanou reakcí. Klient si může herní svět na svojí straně rovnou aktualizovat a server pak jeho stav pouze potvrdí.

Konkrétní implementaci predikce na straně kienta ve hře QuakeWorld popisuje [52] a [53].

Výhody predikce na straně klienta

1. Herní scéna je plynulá, interakce okamžitá (klient nemusí čekat na odpověď od serveru)
2. Herní svět je konzistentní pro všechny hráče (udržovaný na serveru)
3. V případě kolizí mezi světy více hráčů dochází k plynulému vyrovnání

Nevýhody predikce na straně klienta

1. Výrazně náročnější implementace
2. Paměťové a výpočetní nároky především na straně klientské aplikace

Implementované řešení

Ve finální aplikaci byl použit základní model s udržováním fyzikálního světa na serveru, protože navržený scénář není tak citlivý na odezvu uživatelského rozhraní.

5 Implementace hry FavRacer

Pro účely této práce byla vytvořena multiplayerová hra FavRacer na které jsou otestovány běžné postupy a techologie popsané v předchozích kapitolách. Následující část textu je věnována popisu této hry a obsahuje popis programu, jeho prvků a základních funkcí.

5.1 Návrh hry

Aby bylo pokryto co nejširší spektrum z oblasti multiplayerových her, byla pro scénář hry zvolena realtimová akční závodní hra s použitím fyziky na straně serveru. Návrh je inspirován počítačovou hrou Death Rally [39] firmy Remedy z roku 1996.

Obrázek 5.1: Ukázka z původní hry Death Rally

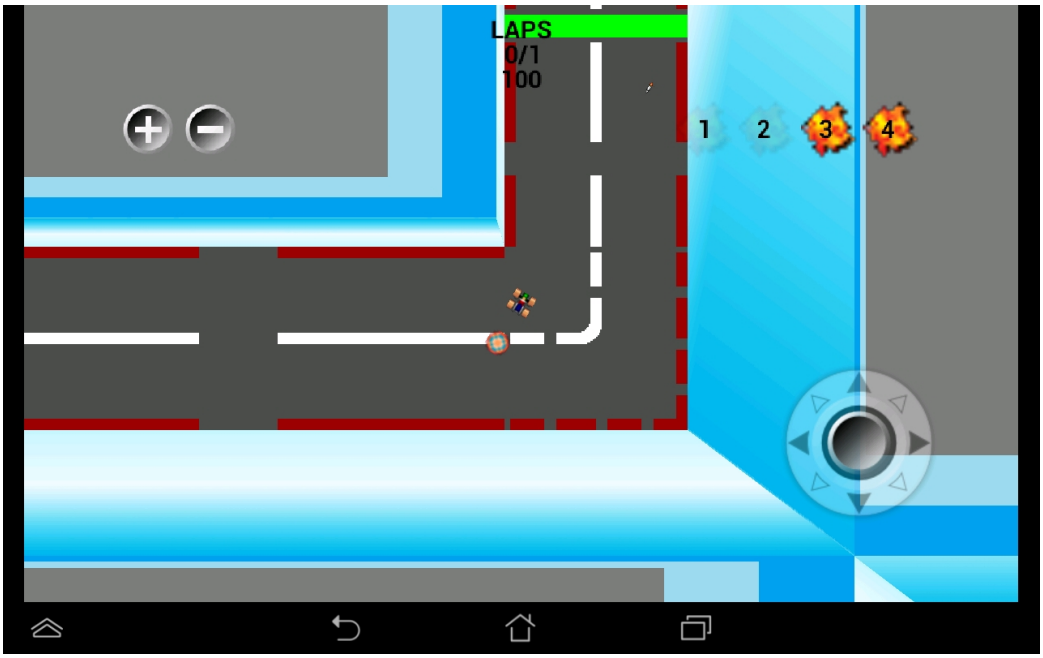
Jedná se o závodní hru z pohledu shora (Top-Down Perspective), ve které je navíc možnost střelby po soupeřích. Původní hra byla singleplayerová. V jednotlivých závodech tak hráč nastupoval proti skupině soupeřů řízených

počítačem. Tím se FavRacer zásadně liší, protože umožňuje síťovou hru více hráčů a naopak neumožňuje závodit proti umělé inteligenci.

Pro otestování implementace fyzikální knihovny byl použit základní fyzikální model, který umožňuje interakci s budovami a ostatními hráči ve hře.

Ve hře je také použita vlastní verze projekce objektů herní scény do 3D, díky které je herní svět výrazně živější.

Na následujícím obrázku 5.2 je zachycen snímek obrazovky finální mobilní aplikace FavRacer.



Obrázek 5.2: Mobilní aplikace FavRacer - snímek z průběhu hry

Mobilní aplikace primárně slouží k porovnání jednotlivých možností síťového připojení. Za tím účelem není potřeba ve hře ukládat výsledky odehraných závodů nebo udržovat žebříčky, které by pro různé konfigurace her nebyly příliš objektivní. Po každé dokončené hře je ale zobrazeno pořadí hráčů, čas jejich projetí cílem a ukazatel zdraví.

Podrobněji jsou jednotlivé části implementace popsány v následujících kapitolách.

5.2 Použité prostředky

Cílové Android API mobilní aplikace je nastaveno na Android 5.0 Lollipop (API level 21) a minimální Android 2.3 Ginger Bread (API level 9). Programovací jazyk Java je používán ve verzi 8.

Pro implementaci herní smyčky mobilní aplikace byl využit volně dostupný engine AndEngine [40] ve verzi GLES2-Anchor Center. Tento engine poskytuje základní prostředky pro vytváření herních scén pomocí OpenGL ES 2.0 [54], které podle [55] podporují zařízení s Android API 2.2 (API level 8) a vyšším. Kromě toho obsahuje metody pro správu a přepínání scén, animaci objektů, řízení zvuků, a množství dalších užitečných funkcí. Uživatelská základna je poměrně aktivní a řada dostupných návodů [44] výrazně zkracuje dobu potřebnou pro naučení potřebných postupů.

K uvedenému enginu existuje několik doplňkových rozšíření. *AndEnginePhysicsBox2DExtension* například obsahuje kompletní JNI wrapper k fyzikální knihovně Box2D [56] napsané v jazyce C++ včetně napojení do herní smyčky enginu.

Dalším zajímavým rozšířením je pak *AndEngineMultiplayerExtension*, které obsahuje připravené třídy pro zajištění komunikace v multiplayerových hrách. Zatímco první uvedený doplněk nakonec využit nebyl (viz 5.10), druhý je v aplikaci využíván jako základ pro odesílání a příjem veškerých zpráv.

5.3 Dvě verze aplikace

Pro potřeby testování různých scénářů a situací bylo nutné vytvořit kromě mobilní aplikace i její serverovou verzi pro platformu PC. Ta je stejně jako mobilní aplikace napsána v jazyce Java a může být spouštěna na serveru. To nám umožňuje porovnat rozdíly mezi připojením k místní síti pomocí Wi-Fi a připojením ke vzdálenému serveru pomocí mobilního připojení (např. 3G nebo 4G). Rozdílů v programech pro platformu Android a PC je celá řada, ale i s využitím frameworku vytvořeného výhradně pro platformu Android bylo možné používat v obou verzích shodné programové jádro.

5.3.1 Serverová aplikace

Základem je verze pro Android, serverová varianta potom navíc obsahuje sadu mockup tříd pro statické metody a definice tříd požívaných frameworkem, i když nejsou v serverové verzi používány a slouží pouze k umožnění kompilace celku. Funkce specifické pro danou platformu jsou navíc doplněny nad rámec jádra v obou verzích a vyčleněny do tříd označených ve svém názvu heslem **Android** nebo **Desktop**.

Jako příklad může posloužit dvojice tříd *DesktopUniqueIDGenerator* a *AndroidUniqueIDGenerator*, které poskytují aplikaci unikátní identifikátor, který zůstává uložen pro další opakovaná spuštění. Zatímco na platformě Android je pro tento účel vhodné podle [57] využívat úložiště **SharedPreferences**, na PC je ke stejnému účelu používán vytvořený soubor *FavRacer.properties*. Obě uvedené třídy implementují rozhraní *IUniqueIDGenerator*, takže v programu je s nimi možné pracovat stejně a rozdíl je pouze v místě jejich inicializace.

5.3.2 Rozdíly serverové a mobilní aplikace

Protože obě verze aplikace používají shodné třídy pro komunikační vrstvu i pro logickou vrstvu pro řízení hry, je zaručeno dodržení stejného průběhu hry a reakcí na obdržené zprávy. Uživatelské scénáře předpokládají vždy ovládání hry z role klienta na mobilním zařízení. Z toho důvodu jenom pro účely testování není potřeba používat v serverové aplikaci klientskou část aplikace a vykreslovat herní scénu. Hlavní rozdíl serverové aplikace je tedy v tom, že se jedná o konzolovou aplikaci bez grafického rozhraní. Další rozdíl je v jejím použití. Zatímco na mobilním zařízení probíhá v jeden okamžik vždy maximálně jedna hra, serverová verze umožňuje simultánní řízení více her najednou.

5.4 Možnosti připojení ke hře

Pro připojení ke hře je možné využít následující definované možnosti:

- **lokální hra** (přímé předávání zpráv / UDP)
- **hra v místní síti** (UDP / TCP)

- **hra na vzdáleném serveru** (UDP / TCP)
- **Peer-to-Peer** (UDP / TCP)

Desktopová aplikace může spravovat více her s různými počty hráčů a možnostmi připojení současně. Není ale možné kombinovat komunikační protokoly v rámci jedné hry. To znamená, že ve hře spuštěné pro protokol UDP budou všichni připojení klienti využívat UDP.

Díky architektuře vytvořených aplikací je teoreticky možné vytvářet hry pro libovolný počet hráčů. Reálně je tento počet omezen počtem startovních míst konkrétní mapě a především pak výkonností zařízení, která mobilním zařízení při vysokém počtu hráčů neumožní kvalitní herní zážitek. Z uvedených důvodů byl počet omezen na 6 hráčů.

5.5 Rozdělení do vrstev

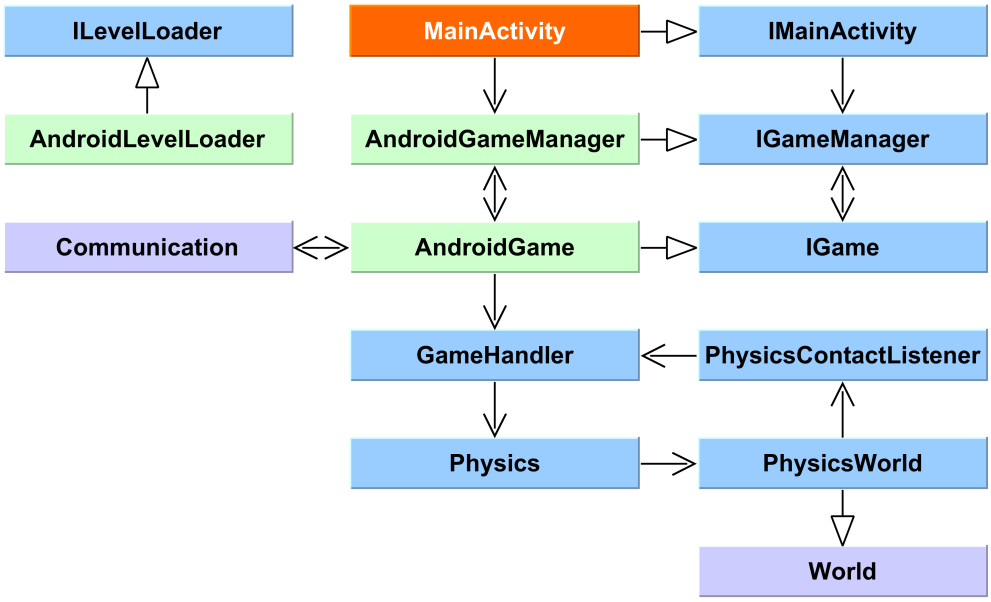
V předchozí kapitole už bylo zmíněno využití společného základu, který používá jak serverová, tak mobilní aplikace. Tento základ lze rozdělit do tří základních vrstev:

- **Logická** (reakce na zprávy, řízení průběhu hry, fyzika)
- **Grafická** (scény, 3D)
- **Komunikační** (správa klientů, odesílání a příjem zpráv)

5.5.1 Logická vrstva

V logické vrstvě jsou vytvářeny instance her a řízena fyzika hry. Zatímco řízení fyzikální knihovny je u mobilní a serverové aplikace identická, správa her je značně odlišná. Serverová aplikace umí spravovat současně více her, u mobilní aplikace tato funkce není implementována (omezeno na jednu hru). Hlavním důvodem je nedostatečná výkonnost mobilních zařízení.

Na následujícím obrázku 5.3 jsou zobrazeny hlavní třídy logické vrstvy mobilní aplikace FavRacer.



Obrázek 5.3: Vybrané třídy logické vrstvy mobilní aplikace

Hlavní třída aplikace *MainActivity* je společně se třídami vybarvenými zeleně specifická pro platformu (liší se v mobilní a serverové verzi). Každá z těchto tříd ale implementuje univerzální sdílené rozhraní, se kterým se pak pracuje ve zbytku aplikace.

Třída *Communication* je zde představuje napojení na komunikační vrstvu a třída *World* je objekt z fyzikální knihovny JBox2D.

Role

V obou verzích aplikace (mobilní a desktopové) se používá sdílená část logické vrstvy následující role:

- **Server** (desktopová aplikace)
- **Klient** (mobilní aplikace)
- **Server a klient** (mobilní aplikace)

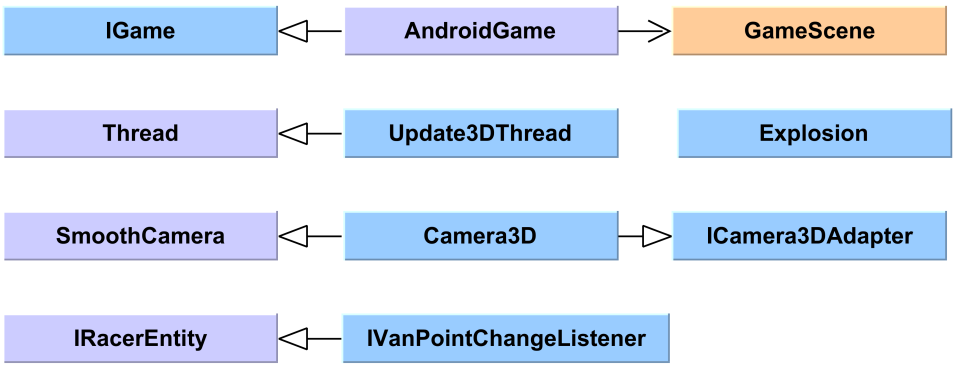
Přiřazená role má vliv například na počet otevřených komunikačních kanálů a přiřazenou sadu odesílaných zpráv.

Role **server a klient** je v mobilní aplikaci použita v případě, že uživatel vytvořil lokální hru, nebo v případě Peer-to-Peer spojení, kdy se zakládající hráč stane správcem hry (server) a klientem současně. Pro další hráče tak vystupuje jako server, ale na svém zařízení vidí obrazovku klienta.

5.5.2 Grafická vrstva

Zatímco mobilní aplikace v grafické vrstvě obsahuje kód pro správu grafického rozhraní a řízení renderování trojrozměrného zobrazení, serverová aplikace v této vrstvě udržuje pouze seznam objektů, které jsou součástí aktuálního herního světa.

Na obrázku 5.4 je zobrazen diagram nejzajímavějších tříd grafické vrstvy.



Obrázek 5.4: Vybrané třídy grafické vrstvy aplikace

Třída *GameScene* a *AndroidGame* je odlišná u mobilní a serverové aplikace. Do této vrstvy lze kromě tříd pro scény mobilní aplikace zařadit i třídu pro grafické efekty (*Explosion*) a třídy, které provádí projekci objektů do 3D.

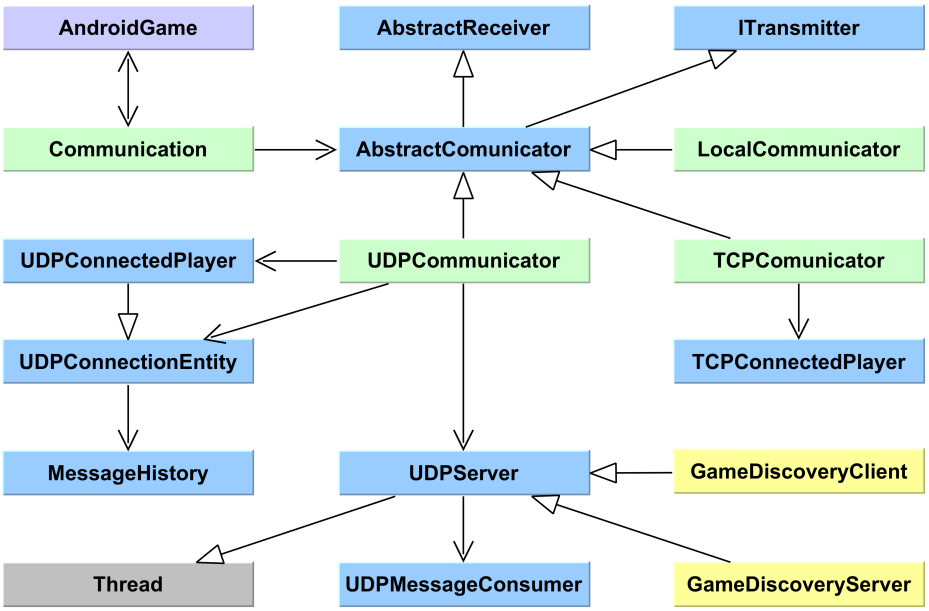
5.5.3 Komunikační vrstva

Aby bylo možné ověřit výhody a nedostatky komunikačních protokolů popisovaných v kapitole 3, byla implementována trojice komunikačních protokolů:

- **TCP/IP**
- **Rozšířené UDP/IP** (viz 3.1.3)
- **Lokální hra jednoho hráče**

Každý protokol může být použit pro danou hru pro všechny zúčastněné hráče (nelze je kombinovat). Používané zprávy jsou pro všechny stejné a to samé platí pro reakce na jejich přijetí. Všechny třídy z komunikační vrstvy jsou shodné pro mobilní a serverovou aplikaci.

Na následujícím obrázku 5.5 je zachyceno schéma vybraných tříd tříd komunikační vrstvy.



Obrázek 5.5: Vybrané třídy z komunikační vrstvy

Třídy označené fialovou barvou jsou objekty napojení komunikační vrstvy, přes které aplikace ke komunikační vrstvě přistupuje. Zeleně vybarvené třídy

jsou implementované protokoly, které jsou odvozené ze stejné abstraktní třídy *AbstractCommunicator*. Žluté třídy slouží k vyhledávání serverů s dostupnými hrami a k jejich spouštění.

TCP

Komunikační protokol, který obsahuje vrstvu zajišťující opakované přeposílání zpráv v případě jejich poškození nebo ztráty. Sockety pro přenos dat jsou vytvářeny s vlajkou TCP_NODELAY, která zakazuje aplikaci Nagleho algoritmu (viz 3.1 a 6.6).

Rozšířené UDP

Třídy používané v komunikační vrstvě pro přenos zpráv pomocí nespolehlivého UDP protokolu jsou doplněny o zajištění bezpečného přenosu důležitých zpráv podle popisu v kapitole 3.1.3. Kromě přeposílání ztracených zpráv spravuje i seznamy připojených klientů a dokáže detekovat případná přerušení spojení.

Použití protokolu UDP je možné jak pro všechny typy definovaných spojení (lokální hra, hra v místní síti, hra na vzdáleném serveru i Peer-to-Peer). Tento protokol je pro komunikaci brán jako referenční z důvodů popsaných v kapitole 3, a protože RTT při jeho použití vykazuje výrazně méně výkyvů než TCP.

Lokální hra jednoho hráče

Tato alternativa byla původně vytvořena čistě pro testovací účely klientské mobilní aplikace. Používá zprávy stejným způsobem jako dříve uvedené protokoly, ale zprávy nejsou serializovány do socketu. Namísto toho je rovnou předává z metody pro odeslání do metody pro příjem. To výrazně omezuje dobu potřebnou na předání zprávy mezi serverem a klientem (hráč má v tomto režimu obě role). Při porovnání se hrou jednoho hráče při použití UDP tak například můžeme zjistit, kolik času přibližně zabere přenos zprávy komunikačním kanálem od odesílatele k příjemci a jaký má tento čas dopad na kvalitu hry.

```
/*
 * Metoda pro odesilani zprav pri primem predavani zprav
 */
@Override
public void sendMessageToAllPlayers(IRacerMessage pMessage) {
  receive(pMessage);
}
```

Peer-to-Peer připojení

Jak už bylo uvedeno v přechozích kapitolách, serverová verze aplikace sdílí s mobilní aplikací celou komunikační vrstvu. Tím je mimo jiné umožněno komunikovat v roli serveru i z mobilní aplikace. Tato funkce umožňuje Peer-to-Peer spojení více mobilních zařízení v rámci lokální sítě zcela bez potřeby serverové aplikace. V takovém případě potom zařízení hráče, který hru vytvořil a ke kterému se další klienti připojili, vystupuje jak v roli klienta, tak v roli serveru (stejný případ jako při hře jednoho hráče).

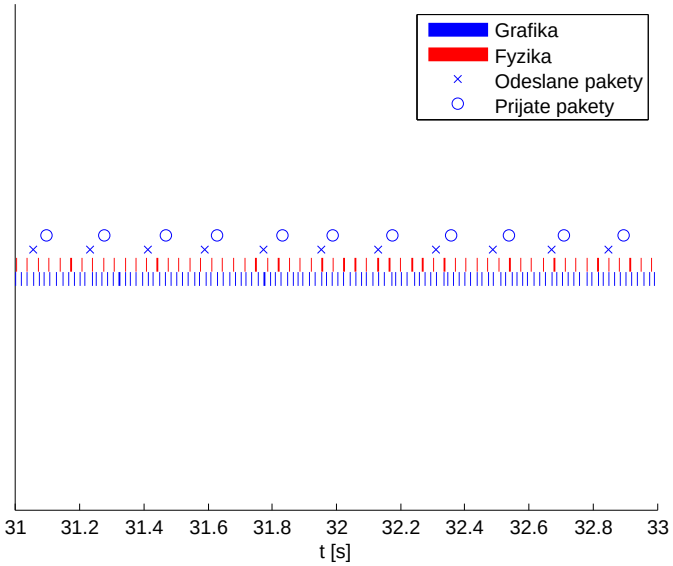
5.6 Hlavní vlákna

Jádro aplikace používá čtyři hlavní vlákna, která vykonávají svoji část programového kódu asynchronně. V následujících podkapitolách bude uveden jejich stručný popis a vysvětlení důvodu pro jejich vyčlenění do samostaných vláken.

- herní smyčka
- vlákno fyziky
- vlákno pro příjem zpráv (UDP)
- vlákno 3D projekce

Následující obrázek 5.6 zachycuje krátký naměřený časový úsek hry a časové záznamy dob vykonávání činosti herního vlákna (modré linky) a vlákna fyziky (červeně). Z obrázku je hezky vidět kolik času jsou vlákna uspaná mezi

svými kroky, jak dlouho jejich kroky trvají a že frekvence jejich krokování odpovídá nastavenému snímkování. Zobrazené odeslané a přijaté pakety označují čas odeslání měřící zprávy (křížky) a čas přijetí jejího potvrzení. Tyto zprávy jsou podrobněji popsány v kapitole 5.8.2.



Obrázek 5.6: Krokování vláken aplikace

5.6.1 Herní smyčka

Hlavní smyčka hry je řízena herním enginem. V pravidelných krocích každých $\frac{1}{60}s$ umožňuje logické vrstvě reagovat na příchozí zprávy. V každé iteraci pak překresluje objekty herní scény podle jejich aktuálních souřadnic.

5.6.2 Vlákno fyziky

Vlákno fyziky se stará o krokování fyzikálního světa každých $\frac{1}{30}s$. Výhoda oddělení výpočtu nových stavů fyzikálních objektů do vlastního vlákna spočívá především v minimalizaci prodlev herní smyčky v případě nečekaně dlouhého výpočtu nového kroku fyziky. Asynchronní výpočet umožňuje provádět obě operace (krokování fyziky a krokování herního světa) ve stejný okamžik, což má v důsledku pozitivní vliv na plynulost celé aplikace.

5.6.3 Vlákno pro příjem zpráv (UDP)

U implementovaného protokolu UDP je pro příjem zpráv používána konstrukce producent-konzument (viz [41]). Socket přijímající pakety (producent) zařazuje přijaté zprávy ihned po přijetí do fronty, která je tímto vláknem (konzument) vyprázdňována (zprávy jsou použity). To zabarňuje přerušení příjmu ve chvíli delšího výkonu akce zprávy (socket je připraven na příjem další zprávy ihned po vložení té přijaté do fronty ke zpracování).

5.6.4 Vlákno 3D Projekce

Vlákno 3D projekce je vlákno spouštěné pouze v mobilní aplikaci a jeho jediným úkolem je z posledních zaznamenaných souřadnic určit aktuální projekci objektu herní scény do 3D. Projekce se určuje vůči aktuální poloze kamery. Princip projekce popisuje kapitola 5.11.

5.7 Souřadnice entit

Aplikace pracuje s množstvím objektů (entit), kdy každý objekt obsahuje souřadnice polohy s několika různými významy a řízenými různými vlákny. Jsou to souřadnice objektu v zobrazované herní scéně, souřadnice objektu ve fyzikální reprezentaci herního světa, souřadnice objektu před začátkem výpočtu projekce do 3D, souřadnice objektu pro serializaci do sítě nebo naopak poloha, kterou by objekt měl zaujmout podle zprávy ze sítě. Tento výčet ještě není úplný (nezohledňuje například historické souřadnice, které mají význam pro interpolaci polohy ve scéně), přesto je zřejmé, že je nutné tato data oddělovat a nastavit proces jejich správy co nejbezpečněji vzhledem ke kolizím jednotlivých vláken. Dalším požadavkem je možnost využívat stejné principy v roli serveru, klienta nebo kombinace těchto rolí.

Následující podkapitoly vysvětlují správu souřadnic jednoho objektu na zjednodušených schématech. Ve skutečnosti je ve všech případech vykonáván stejný programový kód, ale některé kroky jsou podle příslušné role vynechané. Popisovaný model správy je vlastní řešení, které plní požadavky na univerzální použití a umožňuje ve všech rolích synchronizaci potřebných souřadnic.

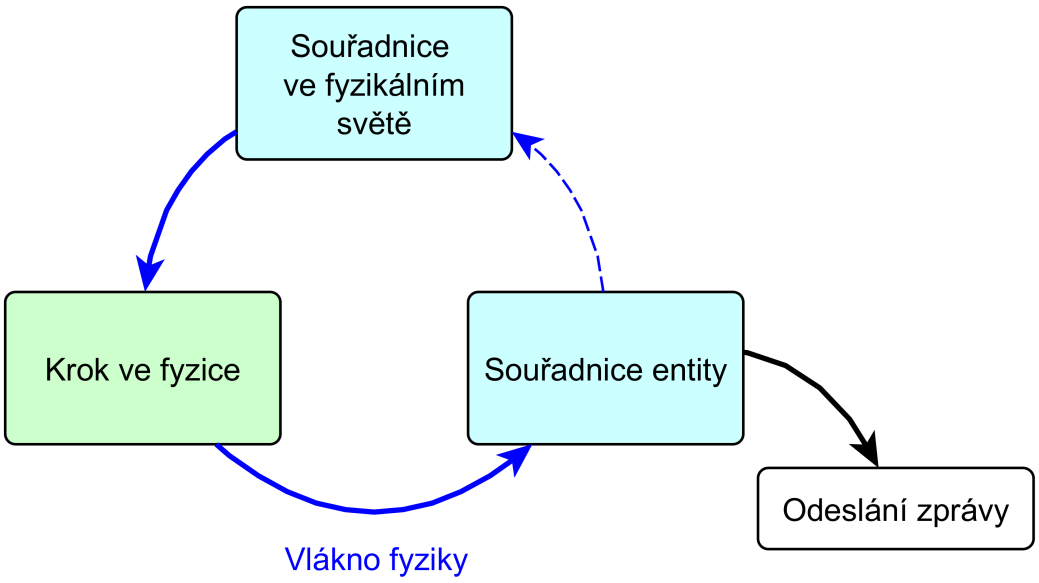
Pro všechny role platí, že hlavním vláknem řídícím jakoukoliv změnu sou-

řadnic objektu je vlákno uchovávající fyzikální reprezentaci světa.

Poslední část kapitoly vysvětluje důvody, výhody a použití interpolace pro souřadnice herní scény.

5.7.1 Souřadnice v roli server

Na následujícím obrázku 5.7 je zachycena základní smyčka, která nastavuje souřadnice v roli serveru.



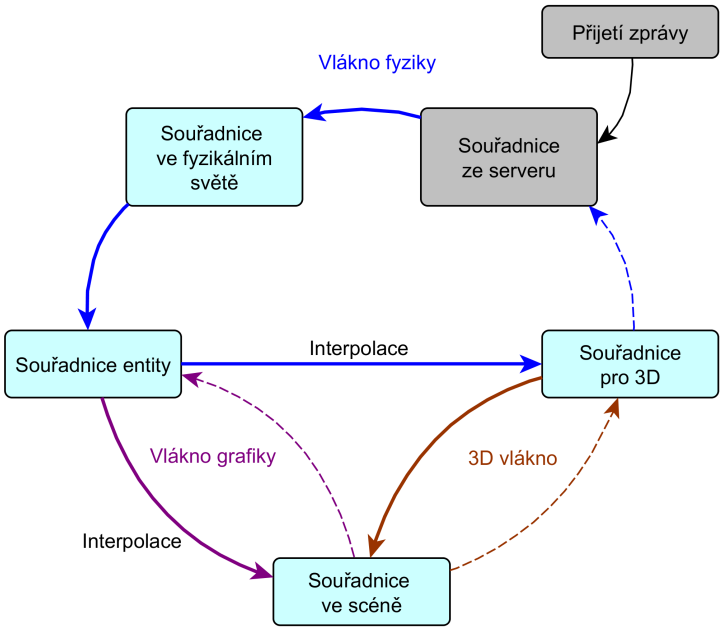
Obrázek 5.7: Správa souřadnic v roli serveru

Souřadnice tělesa fyzikálního světa jsou po provedení kroku uloženy do souřadnic entity (*RacerEntity*) a následně jsou společně se souřadnicemi ostatních těles rozeslány připojeným klientům jako nový stav herního světa.

V roli serveru není nutné zobrazovat herní scénu ani počítat k souřadnicím těles 3D projekce. Protože je server autoritativní, není potřeba ani upravovat souřadnice ve fyzikálním světě jinak než jeho krokováním (klient nemůže zasahovat do souřadnic ve fyzikálním světě serveru). V této roli je zcela vynechána smyčka vlákna pro aktualizaci herní scény a vlákna pro výpočet 3D projekce (server scénu nezobrazuje).

5.7.2 Souřadnice v roli klient

Situace se komplikuje v roli klienta, která je zobrazena na obrázku 5.8.



Obrázek 5.8: Správa souřadnic v roli klienta

Oproti obrázku 5.7 musela objektu nutně přibýt možnost nastavit souřadnice podle zprávy obdržené ze sítě. Klient neprovádí krokování fyzikálního světa ani nerozesílá ostatním klientům jeho stav. Oproti tomu ale udržuje grafickou reprezentaci herního světa, která cyklicky nastavuje polohu zobrazovaných těles podle poslední uložené hodnoty.

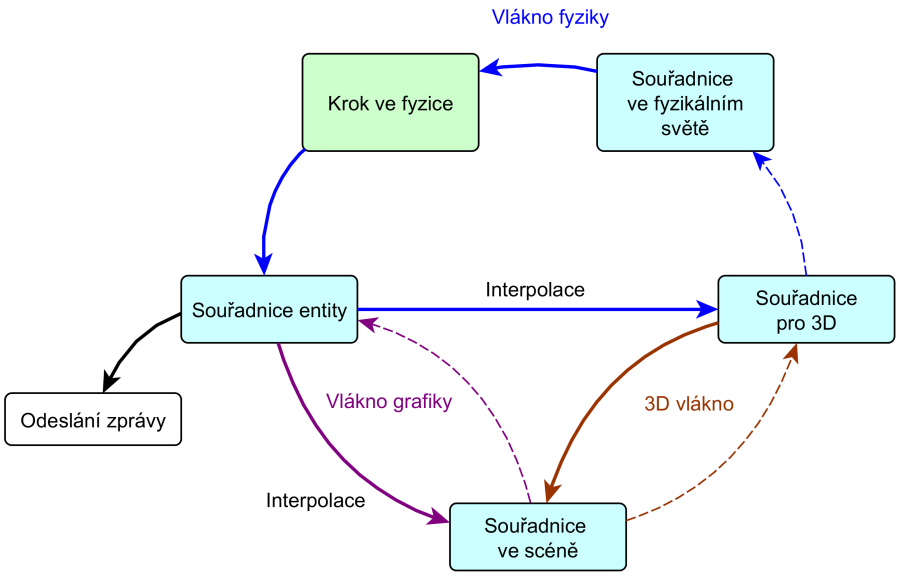
Vlákno pro aktualizaci 3D projekce nemění souřadnice základny tělesa, takže i v případě delšího výpočtu bude těleso zobrazeno na poslední (správné) pozici (podle fyzikálního světa). Může se stát, že jím nastavená 3D projekce bude vypočtená ze starší souřadnice, než na které se těleso nachází ve chvíli aplikace 3D efektu. Rozdíl ale bude těžko postřehnutelný a nemá žádný vliv na hru. 3D zobrazení se dá vypnout a správa souřadnic se nijak nezmění.

Jako hlavní vlákno pro nastavení souřadnic zůstalo vlákno fyzikálního světa, aby zůstal přístup konzistentní pro všechny typy rolí.

5.7.3 Souřadnice v roli server a klient

Pokud je zařízení současně v roli serveru i klienta, je nutné udržovat grafické a 3D vlákno. Současně je nutné krokovat ve fyzikálním světě a rozesílat výsledný stav ostatním klientům. Instance fyzikálního světa je pro klienta i server společná, proto není možné v jeho případě reagovat na příjem nového stavu světa, i když zpráva mu doručena je (server klienty nerozlišuje). Nesmí totiž nikdy dojít k přepsání stavu referenčního fyzikálního světa starším stavem.

Obrázek 5.9 zachycuje správu souřadnic v této roli.



Obrázek 5.9: Správa souřadnic v roli serveru i klienta

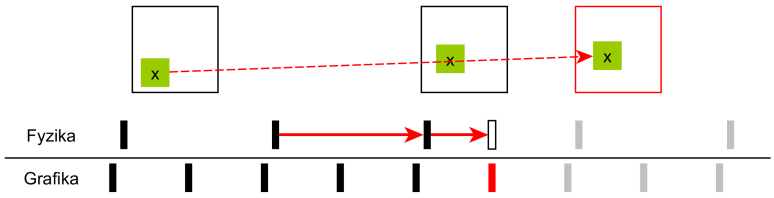
5.7.4 Interpolace

Protože všechna vlákna krokují souřadnice v pevných intervalech, působí pohyb objektů na obrazovce trhaně. Takový objekt totiž skáče v každém kroku na nové souřadnice, což samozřejmě těžko vytvoří iluzi plynulého pohybu. U souřadnic fyzikálního světa tyto skoky ničemu nevadí, fyzikální knihovna s nimi počítá a tělesa fyzikálního světa nejsou nikde přímo zobrazována.

U entit herní scény je ale situace jiná. V předchozích kapitolách byl popsán způsob, kterým jsou herní scéně předány aktuální souřadnice, na kterých mají být jednotlivé objekty vykresleny. Zavedením interpolace těchto souřadnic na základě doby uplynulé od posledního vykreslení umožní vyhladit pohyb objektů a minimalizovat počet viditelných skoků.

Vlákno fyziky krokuje každých $\frac{1}{30}s$ a herní vlákno spravující grafiku (engine AndEngine) krokuje po $\frac{1}{60}s$, což jsou hodnoty obecně považované za optimální vzhledem k požadavkům na kvalitu herního světa a výkonnost mobilních zařízení. To znamená, že během každého kroku fyziky dochází přibližně dvakrát k překreslení herní scény. Tento počet je přibližný proto, že krokování obou vláken je nezávislé a ve výjimečných případech se některé z vláken může během svého výkonu zdržet.

Interpolace je použita tak, že v okamžiku překreslování herní scény nejsou pro pohyblivé objekty použity jejich poslední fyzikou nastavené souřadnice, ale projekce posledních dvou pozic do aktuálního času překreslení. Obrázek 5.10 zobrazuje průběh interpolace.



Obrázek 5.10: Interpolace souřadnic v čase

Objekt scény (zelený čtverec) má v čase překreslení grafiky (červená značka) určenou dvojici souřadnic v reprezentaci fyzikálního světa. Místo toho, aby grafika použila pro vykreslení objektu poslední známou souřadnici, provede interpolaci. Pohyb tak působí plynuleji a souřadnice ve fyzikálním světě zůstávají nezměněné. Následující úsek kódu ukazuje způsob získání interpolované souřadnice:

```
/*
 * Metoda pro interpolaci hodnot pouzivana pro souradnice.
 *
 * @param pPrevValue hodnota v case t0
 * @param pNextValue hodnota v case t1
 * @param pInterpolation doba od t0 jako nasobek rozdilu (t1 - t0)
```

```
 *
 * @return interpolovana hodnota
 */
public static float getValue(float pPrevValue, float pNextValue,
    float pInterpolation){

  return (((1 - pInterpolation) * pPrevValue) + (pInterpolation *
      pNextValue));
}
```

5.8 Navržený protokol komunikace

Tato kapitola se věnuje zprávám odesílaným během hry sítí mezi serverem a klienty. V první části je popsán způsob inicializace objektů pro zprávy, v dalších pak pokračuje popis konkrétních objektů používaných ve hře.

5.8.1 Recyklace instancí zpráv

Komunikace v realtimové hře vždy generuje velký počet zpráv, které je nutné posílat sítí. Tyto zprávy jsou z pohledu programu objekty s poměrně krátkou životností. Obvykle je takový objekt alokován, pak jsou do něj uložena posílaná data a v okamžiku potvrzení příjmu zprávy druhou stranou může být z paměti odstraněn. Protože ale existuje vysoká pravděpodobnost, že v budoucnu bude stejný typ objektu opakovaně vytvářen a následně odstraňován, nabízí se možnost používat v takovém případě stále stejný objekt nebo omezenou množinu objektů. Nejspíš si nevystačíme s jedním objektem, ale určit který objekt může být recyklován rozhodnout dokážeme. Je tedy možné udržovat seznam již zrecyklovaných objektů, které mohou být poskytovány k dalšímu použití a až v případě jeho vyčerpání (nebo pokud je zatím prázdný) alokovat novou instanci. Tomuto postupu se říká pooling (vychází z návrhového vzoru pool) a podle [42] se pro něj v českém jazyce používá označení fond. Jeho využití umožňuje výrazně snížit režii spojenou s alokací paměti a hlavně jejího uvolňování.

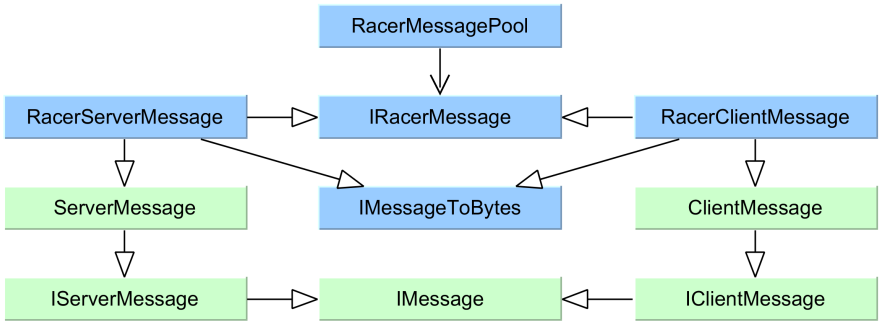
V programu se pool používá ve třídě *RacerMessagePool*, která udržuje statický zásobník alokovaných instancí používaných zpráv. Poskytuje další

dostupné instance požadované třídy zpráv a zároveň umožňuje jejich recyklaci v okamžiku, kdy už alokovaná instance nebude dál používána.

Aby mohl pool plnit svou funkci, je nutné skutečně důsledně recyklovat již nepoužívané instance. V opačném případě totiž hrozí, že bude poskytovat stále další a další nové objekty, ale reference na ty již nepoužívané bude stále existovat. Z toho důvodu by nebylo možné paměť uvolnit a její objem by neustále narůstal. Opačné rizikoje předčasná recyklace instance, kdy objekt poskytneme k recyklaci, ale některý objekt na něj stále drží referenci a dál ho používá i přes to, že byl objekt mezitím poskytnut k naplnění zcela odlišnými daty.

5.8.2 Používané zprávy

Obrázek 5.11 znázorňuje dědičnost a rozhraní využívané pro třídy jednotlivých zpráv. Podle implementace v rozšíření *AndEngineMultiplayerExtension* (zeleně) jsou striktně odděleny zprávy od klienta / serveru a současně jsou vytvořeny třídy s předponou Racer, které rozšiřují původní implementaci o další funkce (modře).



Obrázek 5.11: UML schéma tříd pro zprávy

- *RacerMessagePool*

  Význam a použití třídy *RacerMessagePool* už byl vysvětlen v předchozí kapitole. Pool je používán pro zprávy od serveru i pro zprávy od klienta (obě větve implementují rozhraní používané v poolu - *IRacerMessage*), jak znázorňuje uvedený obrázek 5.11.

- *IRacerMessage*

  Toto rozhraní poskytuje řadu metod pro ukládání a získávání dalších dat zprávy potřebných pro zajištění komunikace. Každý objekt implementující toto rozhraní tak musí nabízet metodu pro uložení a přečtení následujících proměnných: číslo zprávy, číslo poslední přijaté zprávy a bitové pole 32 ACK nebo NACK jí předcházejících zpráv (viz 3.1.3 část Spolehlivý a uspořádaný přenos dat), IP adresu a port odesílatele, čas přijetí zprávy a důležitost. Důležitost je definována hodnotou z množiny výčtového typu vlastní třídy *Significance* z následující tabulky 5.1.

Significance	Maximální doba přeposílání
CRUCIAL	až do ztráty spojení
RELEVANT_FOR_1_SEC	1 sekunda
RELEVANT_FOR_5_SEC	5 sekund
DISPENSABLE	není přeposílána

Tabulka 5.1: Tabulka výčtových hodnot typu Significance

  Kromě výše uvedených metod poskytuje rozhraní ještě metodu *copyAttributesInto(IRacerMessage pStoreMessage)* pro klonování dat zprávy. Ta se využívá při rozesílání stejné zprávy více klientům, kdy je nutné u každého z nich zprávu uchovávat až do jejího potvrzení nebo ztráty významu po uplynutí doby definované její důležitostí.

- *RacerServerMessage*

  Tato abstraktní třída je základem všech zpráv odesílaných ze serveru. Jak je patrné z výše uvedeného obrázku 5.11, je odděděná od třídy *ServerMessage* z rozšíření *AndEngineMultiplayerExtension*. Každý objekt zprávy, který je odděděný od této třídy tak musí poskytovat metody pro serializaci a deserializaci svých dat do datových proudů. K tomu slouží *onReadTransmissionData(DataInputStream pDataInputStream)* a *onWriteTransmissionData(DataOutputStream pDataOutputStream)*.

  Další abstraktní metodou, která musí být překryta, je metoda *getFlag()*. Ta vrací identifikátor typu short, který musí být unikátní mezi všemi typy rozesílaných zpráv. Slouží totiž k rozlišování zpráv při jejich přijetí. Tento způsob není zcela optimální (je nutné zajistit jedinečnost identifikátoru), ale je to doporučený postup používaný uvnitř rozšíření. Pro lepší čitelnost byl v aplikaci definován výčtový typ *Flag*, který obsahuje všechny definované typy zpráv. Hodnota, kterou objekty zpráv vrací potom představuje index příslušného typu ve výčtu *Flag*.

- *RacerClientMessage*

  Abstraktní třída *RacerClientMessage* je analogií k předchozí uvedené třídě s tím rozdílem, že jsou od ní odděděny objekty zpráv, které posílá klient.

Zprávy serveru

Následující výčet obsahuje seznam zpráv odesílaných servrem připojeným klientům.

- *DiscoveredServerMessage*

  Reakce na vyhledávání serverů, která je zaslána klientovi po zachycení jeho výzvy. V datové části je uložena jedna otevřená hra (spuštěná a neobsazená), která je na tomto serveru dostupná. Každá hra je určena IP adresou a portem, kde se k ní dá připojit a názvem. Kromě toho obsahuje informace o své konfiguraci, tedy počet hráčů, identifikátor herní mapy a počet kol závodu.

- *ConnectionRacerServerMessage*

  První potvrzení žádosti o spojení.

- *ConnectionEstablishedServerMessage*

  Povrzení, že byl klient přijat do hry. Zpráva nese identifikátor hráče (index vozidla).

- *ConnectionReestablishedServerMessage*

  Zpráva pro znovu připojeného klienta. Obsahuje všechna data pro možnost okamžitého připojení do hry, tedy identifikátor hráče (index vozidla) a seznam jmen ostatních hráčů.

- *ConnectionRejectedProtocolMissmatchServerMessage*

  Odmítnutí klienta z důvodu nekompatibilní verze protokolu. Data obsahují číslo verze protokolu serveru.

- *AllSlotsTakenServerMessage*

  Zamítací zpráva, která nenese žádná data. Klientovi je zaslána v případě, že už pro něj není ve hře místo.

- *ListPlayersServerMessage*

  Tato zpráva obsahuje pole jmen všech hráčů, kteří byli úspěšně připojeni ke hře.

- *CountDownServerMessage*

  Před začátkem hry se nejdřív spustí odpočet, aby se hráči stihli připravit. Tato zpráva nese ve svých datech hodnotu, kterou má ukazatel odpočítávání zobrazovat.

- *CurrentWorldStateServerMessage*

  Zpráva obsahující pole pozic a identifikátorů všech objektů ve hře, jejichž pozice není statická. Dá se z ní tedy kompletně určit stav herního světa, který server posílá klientům jako aktuální. Také obsahuje pole identifikátorů kontrolních bodů, které mají hráči projet jako další. To umožňuje klientům udržovat přehled o průběhu hry a je možné na základě této informace zvýraznit následující kontrolní bod bez nutnosti udržovat a synchronizovat informace o průběhu i na straně klienta.

- *ConnectionAliveServerMessage*

  Zpráva označující stále aktivní spojení, která nenese žádná data. Klientovi je zaslána v případě, že uplynula stanovená doba, ve které mu nebyla odeslána žádná jiná zpráva a hrozí tak z jeho strany signalizace výpadku.

- *GameOverServerMessage*

  Zpráva o ukončení hry, která obsahuje seznam hráčů a jejich pořadí, včetně času dokončení, úrovně zdraví a identifikátoru posledního projetého kontrolního bodu.

- *PongServerMessage*

  Tato zpráva slouží ke sledování RTT během hry. Když server obdrží zprávu typu *PingClientMessage*, zkopíruje její obsah do *PongServerMessage* a obratem ji pošle zpět. Klient tak má možnost zjistit, jak dlouho trval přenos zprávy (ping) sítí a jejího potvrzení (pong) zpět. V datech zprávy *PongServerMessage* je pouze časová značka získaná ze zprávy *PingClientMessage*.

Zprávy klienta

Zprávy uvedené v následujícím textu jsou odesílané na server připojenými klienty.

- *DiscoverServersClientMessage*

  Zpráva vyslaná do sítě s účelem získat seznam dostupných her, ke kterým se klient může připojit. V datové části je uloženo číslo verze protokolu, který klientská aplikace používá. To umožní současný provoz serverů s různým číslem protokolu v jedné síti.

- *ConnectionRacerClientMessage*

  Tato žádost o připojení do hry je odesílána na IP adresu a port hry a nenese žádná další data.

- *ConnectionEstablishClientMessage*

  Tato zpráva obsahuje v datové části jméno hráče a značí potvrzení klienta připojení ke hře.

- *GamePreparedClientMessage*

  Po úspěšné inicializaci herní scény odesílá klient tuto zprávu bez dalších dat.

- *ControlsStateClientMessage*

  Tento typ zpráv je periodicky odesílán serveru během hry. V datové části nese konfiguraci jeho ovladače.

- *ShootBulletClientMessage*

  Když chce klient vystřelit, odešle serveru tento typ zprávy. V datové části je uveden identifikátor typu zbraně.

- *ConnectionAliveClientMessage*

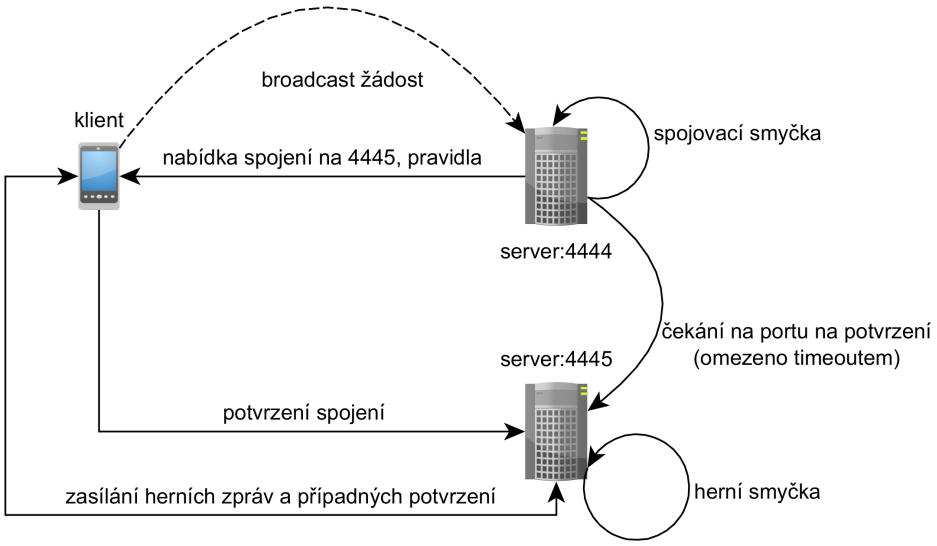
  Zpráva označující stále aktivní spojení, která nenese žádná data. Serveru je zaslána v případě, že uplynula stanovená doba, ve které mu nebyla odeslána žádná jiná zpráva a hrozí tak z jeho strany signalizace výpadku.

- *PingClientMessage* Kvalitu spojení testují klienti odesíláním této zprávy v pravidelných intervalech. V reakci na ní ostávají odpověď *PongServerMessage*, do které uložil čas přijaté *PingClientMessage*. Klient při příjmu porovnáním s aktuálním časem zjistí RTT.

5.8.3 Vyhledávání herních serverů

Aby bylo možné připojit se ke hře, musí klient znát adresu a port, na kterém server hru připravil. Protože může být v jeden okamžik na jedné IP adrese spuštěno větší množství serverů, obsahuje implementace mechanismus, který klientovi poskytuje seznam všech dostupných her na zadané adrese nebo v celé lokální síti WLAN. Tím je zajištěno, že klient získá potřebné informace o dostupných hrách bez nutné znalosti jejich konkrétních parametrů.

Protože TCP protokol pracuje s vytvořenými spojeními, není ho možné využívat pro broadcast, který by odhalil všechny servery v lokální síti. Z toho důvodu je pro mechanismus hledání volných her použit protokol UDP. Stejný mechanismusm může být použit i pro získání seznamu her ze známé adresy (pouze se nebude jednat o broadcast). Následující obrázek 5.12 zachycuje průběh komunikace při hledání volných her v lokální síti.



Obrázek 5.12: Schéma komunikace při hledání volné hry

Na levé straně je zobrazen klient, který prohledává místní síť (rozešle do ní broadcastem zprávu *DiscoverServersClientMessage* na určený port). Serverová aplikace zobrazená na pravé straně tento broadcast zachytí a odešle zpět seznam her, které jsou na serveru dostupné (zprávou *DiscoveredServerMessage*). V té době už je hra připravená k příjmu nového klienta (na ilustraci hra přijímá spojení na portu 4445). Číslo tohoto portu klient získal v odpovědi na broadcast, stejně jako parametry hry. Těmi jsou celkový

počet hráčů, počet již připojených hráčů, ID herní mapy a typ připojení (TCP/UDP). Zná tedy seznam dostupných her a nic mu nebrání přihlášení do zvolené hry zprávou *ConnectionRacerClientMessage*. V okamžiku připojení a potvrzení plného počtu klientů ke hře se rozbíhá na serveru herní smyčka a začíná hra.

5.8.4 Připojení ke hře

Po odeslání zprávy *ConnectionRacerClientMessage* serveru získá klient ještě potvrzení, že byl do hry skutečně připojen. To mu server pošle ve zprávě *ConnectionEstablishedServerMessage*. Může se stát, že ho jiný klient předběhne a už ve hře nezbývá volné místo nebo že nesouhlasí verze komunikačního protokolu obou stran. K tomu server používá zprávy *AllSlotsTakenServerMessage* respektive *ConnectionRejectedProtocolMissmatchServerMessage*.

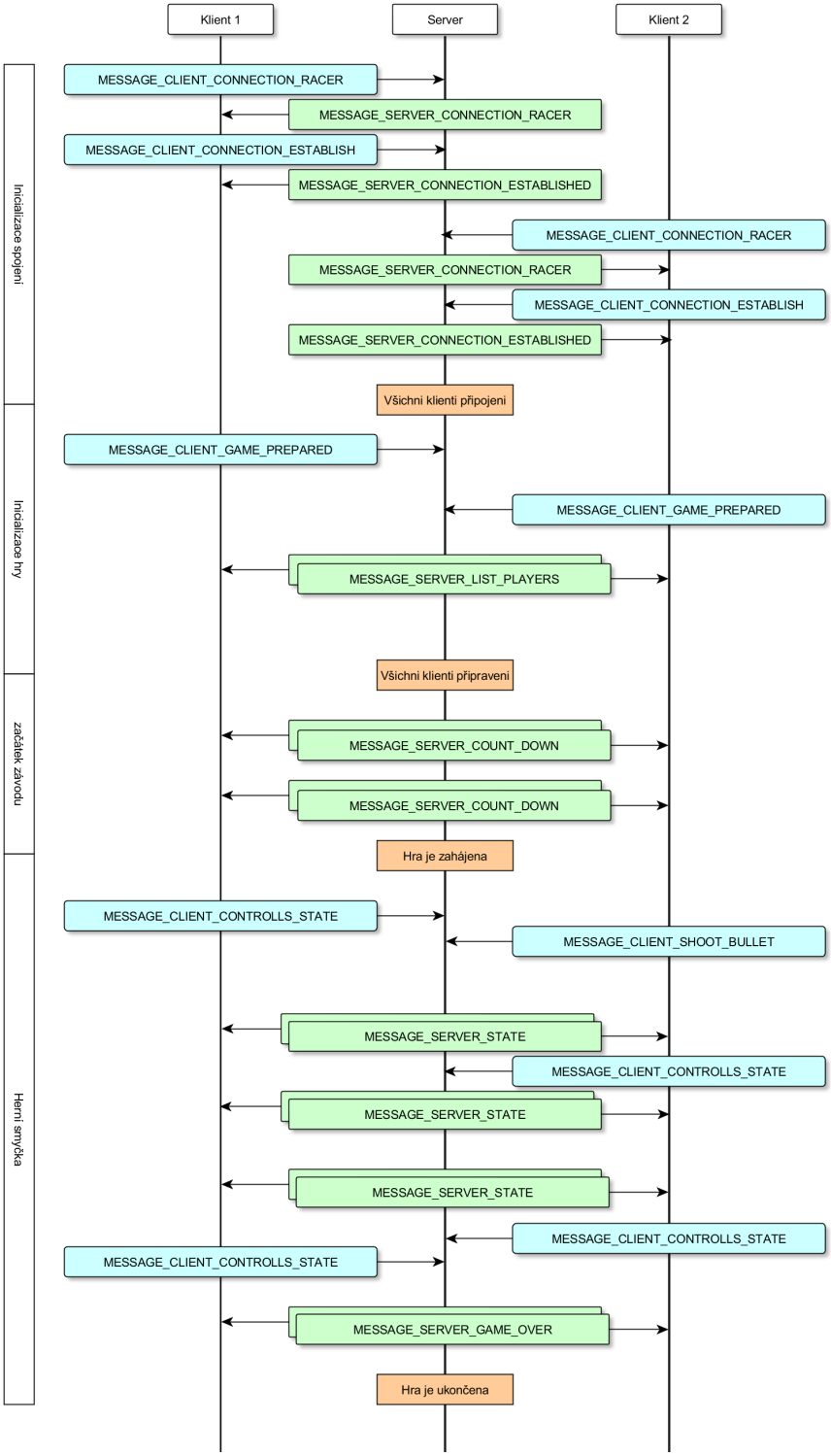
Pokud byl úspěšně připojen, může si připravit herní scénu podle ID herní mapy a počkat na ostatní hráče.

5.8.5 Zahájení hry

V okamžiku dokončení inicializace herní scény na straně klienta dochází k odeslání zprávy *GamePreparedClientMessage* serveru. Když server obdrží tato potvrzení od všech klientů, rozešle jim ještě zprávu se seznamem jmen soupeřů *ListPlayersServerMessage* a začne odpočítávat start. Každých 100ms je ze serveru všem klientům odeslána zpráva *CountDownServerMessage* s časem zbývajícím do začátku závodu. S dosažením nuly se spouští herní smyčka na obou stranách. Zatímco na straně klienta dochází v této smyčce k aktualizaci herní scény a odesílání konfigurace ovladače, na straně serveru se počítají stavy fyzikálního světa, který je následně rozesílán všem klientům.

5.8.6 Herní smyčka

Na následujícím obrázku 5.13 je zobrazena ukázka možné posloupnosti zpráv při komunikaci serveru se dvěma klienty od požadavku k připojení až po konec hry.



Obrázek 5.13: Ukázka posloupnosti zpráv během hry

Jak je z obrázku 5.13 patrné, nejprve dojde k potvrzenému připojení prvního a následně i druhého klienta. V další fázi dochází k potvrzení inicializace herní scény u obou klientů a odeslání informace o seznamu připojených hráčů ze serveru. Následuje odpočet do začátku hry, po kterém už server v pravidelných intervalech informuje hráče o aktuálním stavu světa, ti si jej podle obdržených zpráv upravují. K tomu je používána zpráva *CurrentWorldStateServerMessage*. V opačném směru (od klientů k serveru) proudí zprávy o aktuální konfiguraci ovladače pro řízení pohybu vozidla hráče. Hra končí oznámením ze serveru.

5.8.7 Detekce přerušení spojení

TCP protokol díky udržovanému virtuálnímu spojení umožňuje přímo reagovat na odpojení druhé strany. Konkrétně dojde k zavolání metody *onTerminate()* z rozhraní *ISocketConnectionClientConnectorListener*. Toto rozhraní využívá *SocketServer<SocketConnectionClientConnector>*, který je použit pro spojení z pozice klienta. Ze strany serveru se využívá konektor *ServerConnector<SocketConnection>*, který obsahuje instanci *ISocketConnectionServerConnectorListener* s metodou *onTerminate()*. Ta je opět volána v okamžiku detekce přerušení spojení.

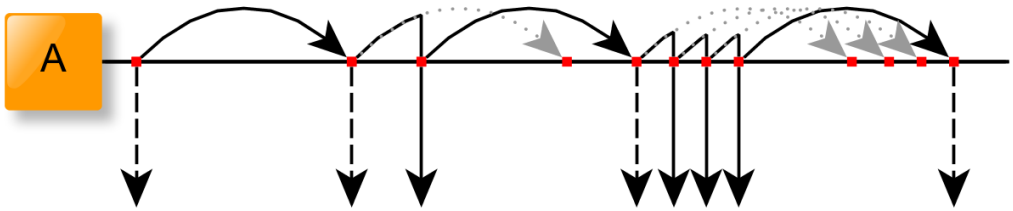
U protokolu UDP už je nutné stanovit časový interval nečinnosti druhé strany, který signalizuje ztrátu spojení. Tento interval by měl být nastaven s ohledem na frekvenci komunikace tak, aby k oznámení výpadku nemohlo dojít předčasně. Konkrétní implementace vypadá tak, že s každou obdrženou zprávou dojde k zaznamenání času přijetí zprávy (*UDPConnectionMonitor*) a před každým odesláním zprávy dojde k ověření, že doba uplynulá od poslední přijaté zprávy nepřekročila stanovené maximum.

Na obou stranách UDP spojení je také nutné ošetřit situaci, kdy se jedna strana z nějakého důvodu odmlčí, ale nechce přerušit spojení. Taková situace nastává například před začátkem hry, kdy je jeden klient úspěšně připojen, ale stále se čeká na přihlášení druhého klienta. V takové chvíli neposílá klient serveru žádné příkazy na pohyb ani server neposílá klientovi informace o stavu světa. Protože může být tato doba delší než interval pro detekci výpadku, je nutné zajistit alespoň nějakou výměnu informací.

K tomu slouží v aplikaci třída *UDPConnectionHolder*, která obsahuje časovač s nastaveným intervalem pro generování odpovědi. S každým odesláním zprávy je tento časovač resetován bez jakékoliv další akce. Ve chvíli, kdy

vytiká, znamená to, že po stanovenou dobu nebyla odeslána žádná zpráva. Reakcí na vytikání časovače je vykonání události určené v konstruktoru, tedy odeslání zprávy typu KeepAliveServerMessage nebo KeepAliveClientMessage (podle role odesílatele).

Na obrázku 5.14 je naznačeno řízení časovače (šipky nad časovou osou) vzhledem k odesílaným zprávám (šipky pod časovou osou). Přerušovaou čarou jsou zvýrazněny zprávy typu KeepAlive odesílané na základě vytikání časovače, plnou čarou potom zprávy odesílané z jiného důvodu. Šedé tečkované šipky znázorňují časovač, který byl přerušen při odesílání zprávy.

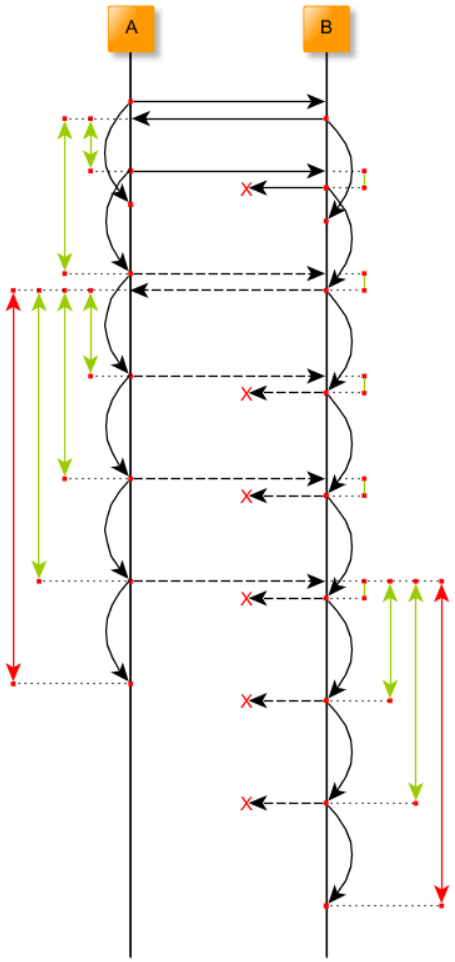


Obrázek 5.14: Ilustrace funkce časovače ze třídy *UDPConnectionHolder*

Ani zprávy pro udržení spojení nemají zajištěné bezpečné doručení, proto musí být interval pro jejich generování vždy menší než časový interval pro detekci výpadku, aby bylo případně možné jejich přenos opakovat.

UDPConnectionHolder může být zastaven nebo spuštěn podle potřeby a společně s třídou *UDPConnectionMonitor* tak umožňuje detekovat přerušené spojení i při použití protokolu UDP.

Na obrázku 5.16 je zachycena detekce ztráty spojení. Scénář komunikace je takový, že si strana A vymění se stranou B postupně dvojici zpráv v obou směrech a dál už pouze udržují spojení. Zprávy od B se ale při přenosu sítí ztrácejí a strana A po uplynutí intervalu spojení ukončí. To vede následně k detekci přerušení i na straně B. Přerušované šipky opět značí zprávy pro udržení spojení, zelenou barvou je zobrazen interval pro aktivní spojení. Červenou barvou je označen interval, který signalizuje ztrátu spojení, protože doba, která uplynula od posledního příjmu zprávy je větší než stanovené maximum.



Obrázek 5.15: Ilustrace detekce přerušení

5.8.8 Ošetření výpadků spojení v UDP

V multiplayerových hrách může docházet ke ztrátám spojení způsobeným různými faktory. Následující scénáře popisují reakce aplikace FavRacer na vzniklé situace při detekci přerušení spojení nebo vedoucí k její identifikaci. Stanovený interval pro detekci ztráty spojení po neaktivitě druhé strany byl nastaven na $2s$. Tento čas vychází z frekvence pravidelně odesílaných zpráv a velikosti bitového pole pro sledování ztracených zpráv (viz 3.1.3). V následujících bodech bude označován t_x.

- Předpokládáme, že je klient odpojen, pokud neodpovídá (nedostávámě od něj žádné zprávy po stanovenou dobu) a ve stanovené době se ani

nepřihlásil z jiné IP adresy (nejedná se tedy pouze o změnu IP po změně sítě klienta).

- Pokud klientovi od serveru za stanovený interval t_x nepřišla od serveru žádná zpráva, můžeme server považovat za odpojený a hru ukončit.
- Pokud se klient přihlásí během hry z jiné adresy (poznáme ho podle unikátního ID, kterým podepisuje zprávy), ukončí server původní spojení a pomocí zjednodušeného procesu přihlášení s ním spojení naváže na nové adrese.
- Při změně adresy serveru dojde k ukončení hry (server nežádá všechny klienty o navázání nového spojení na nové adrese). Na straně klientů se výpadek projeví po uplynutí intervalu t_x.
- Ukončení serveru nebo jakákoliv ztráta spojení vede na straně klientů po uplynutí intervalu t_x k detekci výpadku a k signalizaci ztráty spojení.

5.9 Herní scéna

Aplikace obsahuje definici základní herní scény a další mohou být snadno definovány pomocí XML konfiguračního souboru *levelXML.xml*. Pochopitelně je nutné stejný soubor používat jak u klientské aplikace, tak v serverové aplikaci. Toto XML má jednoduchou strukturu, která může vypadat následovně:

```
<?xml version="1.0" encoding="UTF-8" ?>
<levels>
<level name="Second Level">
   <track x="0" y="0" w="4048" h="2864" r="0" texture="LEVEL_2" />
   <checkpoint x="200" y="10" w="100" h="150" r="200"
      texture="TRACK_STRAIGHT" />
   <checkpoint x="400" y="10" w="100" h="150" r="200"
      texture="TRACK_STRAIGHT" />
   <wall x="10" y="220" w="100" h="150" r="0"/>
   <building x="400" y="400" w="100" h="150" d="20" r="0"
      roofTexture="BOX" sideTextureHor="TRACK_STRAIGHT"
      sideTextureVert="WALL_BUILDING_VERT_1" />
   <vehicle x="0" y="20" w="30" h="30" r="50" texture="VEHICLE" />
</level>
</levels>
```

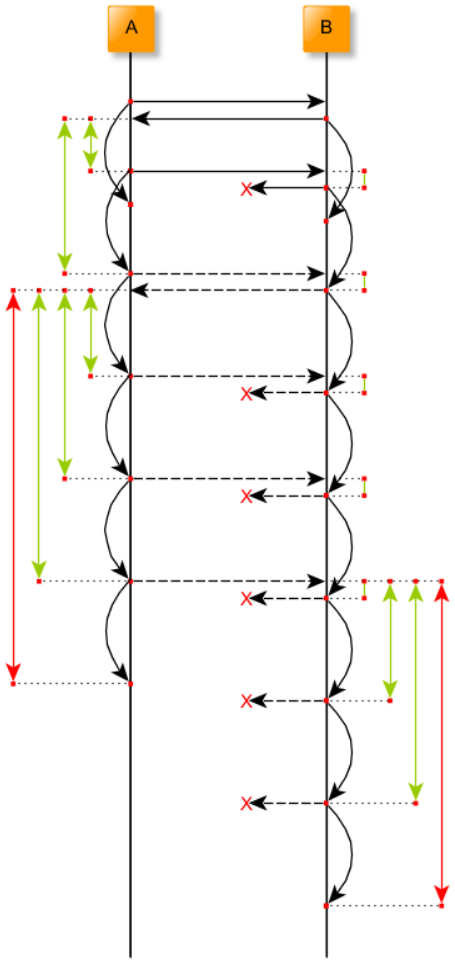
Jak je z uvedené ukázky patrné, každý objekt herní scény má definovanou pozici, velikost, natočení a klíč použité textury. U budov (objekt building) je navíc zadávána výška objektu a textury použité na stěnách. Všechny číselné hodnoty jsou v obrazových bodech, přičemž jejich počet na obrazovce je nastaven na 800 bodů šířky a 480 bodů výšky. Textury jsou načítány ve třídě *AndroidData* z adresáře *Assets/gfx*.

Entity implementují různá rozhraní, která jim přidávají následující vlastnosti:

- IHasBody - mají definované charakteristické fyzikální těleso
- IMovingBody - mohou se pohybovat
- ISceneEntity - jsou vykreslovány v herní scéně
- IDynamic - mohou být do scény přidány nebo odebrány v průběhu hry
- ILevelEntity - může být načten v definici herní mapy
- ISceneMovingBodyEntity - obsahuje vlastnosti prvích tří rozhraní

Výčet používaných entit

- Auto (Fyzikální model, tvar, vzhled) - Vehicle
- Budova - *Building*
- Kontrolní bod (alespoň dva, první je start i cíl) - Checkpoint
- Prvek střelby (typ střely, efekty, důsledky) - AbstractBullet
- Vizuální prvek (povrch tratě v nejnižší vrstvě) - RaceTrack
- Stěna (neviditelná překážka ohraničující herní scénu) - Wall



Obrázek 5.16: Ilustrace detekce přerušení

5.10 Fyzikální svět

Pro simulaci fyzikálního světa ve 2D je možné využít několik různých postupů. Pro jednoduché interakce a pohyb těles se hra často obejde i bez složitého fyzikálního modelu. Případné kolize se dají řešit už na základě kolize objektů scény a jejich pohyb je určen základními fyzikálními rovnicemi (viz [43]). V takovém případě nemá smysl využívat komplexní fyzikální knihovny. Nevýhodou tohoto přístupu je celková náročnost implementace i v případě výrazně omezených možností. Další překážkou pak bývá náročnost implementace složitějších mechanizmů interakce (pružné objekty, těžiště hmoty, tření, deformace, složené objekty, atd.). Z toho vyplývá, že tento postup je vhodný pro hry s malým počtem jednoduchých objektů a nízkými nároky

na celkový dojem z chování objektů vzhledem k reálnému světu. Příkladem mohou být arkády typu Pong nebo Arkanoid. Vzhledem k náročnosti a navrženému scénáři tato varianta není vhodná.

Druhou možností je využít některou z dostupných fyzikálních knihoven, které jsou schopné řešit složitější modely a situace. Existuje řada komerčních i volně dostupných řešení pro různé platformy. Pro použití v mobilních zařízeních na platformě Android lze využít knihovny napsané v jazyce Java nebo v nativním C/C++, které jsou používány přes wrapper JNI (Java Native Interface [36]). Některé fyzikální knihovny jsou přímo součástí herního enginu (např. Unity3D [37]).

Zvolený framework AndEngine se dá velmi snadno rozšířit o rozšíření AndEnginePhysicsBox2DExtension, který obsahuje JNI wrapper knihovny Box2D napsaný v jazyce C++. Toto řešení ale nelze snadno použít v desktopové variantě aplikace, protože je u ní nutné zkompilovat knihovnu pro architekturu procesoru počítače a následně zajistit správné napojení JNI. Použití tohoto doplňku bylo navíc připraveno pro použití přímo v hlavní herní smyčce, což není vhodné řešení vzhledem k návrhu aplikace.

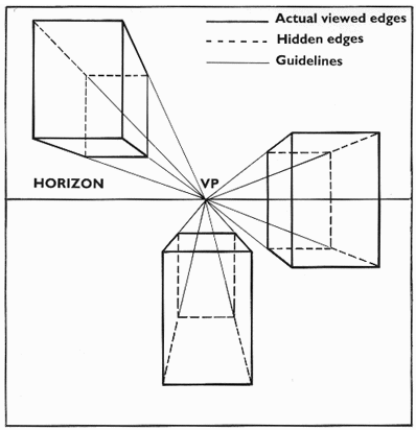
Pro naše potřeby (shodné jádro mobilní a desktopové aplikace) je vhodnější využít knihovnu vytvořenou kompletně v jazyce Java. Dostupná jsou řešení JBox2D [45], JBullet [46] a dyn4j [47]. Zatímco JBullet a JBox2D jsou přeložené verze knihoven pro jazyk C++ (Bullet respektive Box2D), dyn4j je vyvíjen od počátku v jazyce Java.

Z výše uvedených možností má jednoznačně nejširší uživatelskou základnu a komunitní fórum knihovna Box2D. Vzhledem k tomu, že JBox2D je její portovanou verzí a obsahuje stejné funkce i postupy, je možné rovnou použít většinu postupů a návodů věnovaných Box2D. Z toho důvodu byla nakonec pro výslednou implementaci zvolena právě knihovna JBox2D ve verzi v2.2.1.1.

Během jejího použití se sice ukázalo, že ne všechny části odpovídají poslední verzi knihovny Box2D, ale vzhledem ke kvalitě starších verzí Box2D to není důvod k obavám. Za zmínku tak stojí snad jen nutnost volat po každém kroku metodu *clearForces* nad fyzikálním světem, což je v poslední verzi Box2D v2.3.1 prováděno automaticky.

5.11 Vykreslování 3D objektů

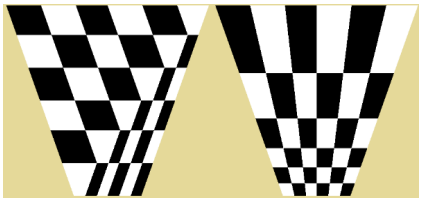
Návrh hry je inspirován hrou Death Rally, která se vyznačuje pohledem shora (ptačí perspektiva) a simulací 3D pomocí perspektivní projekce. Na následujícím obrázku 5.17 je zachycen její princip.



Obrázek 5.17: Princip projekce podle jednoho ohniska, zdroj [48]

Z obrázku je patrné, že každý objekt, který modelujeme do 3D, je možné složit ze sady čtyřúhelníků, které jsou určeny projekcí podle bodu ve středu obrazovky. Grafický framework AndEngine nepravidelné čtyřúhelníky jako základní primitivum nemá, ale překrytím vykreslovací funkce OpenGL ES bylo možné nepravidelný čtyřúhelník vytvořit.

Dále bylo ještě nutné definovat grafický shader [51], který pokrývá čtyřúhelník texturou ve směru projekce (Perspective correction - [49]), což znázorňuje obrázek 5.18.



Obrázek 5.18: Oprava perspektivy vlastním shaderem, zdroj [50]

Popsaná primitiva jsou ve třídách *StretchSprite* a *StretchShader*.

Oddělením výpočtu logiky na samostatné vlákno je zamezeno situacím,

kdy by herní smyčka musela čekat na výsledek 3D projekce. S tím souvisí potřeba synchronizace souřadnic popsaná podrobněji v kapitole 5.7.

6 Testování

Vytvořená hra byla otestována při různých typech spojení. Tato kapitola popisuje testovací prostředí a základní scénář. V následujícím částech jsou představeny výsledky jednotlivých měření.

6.1 Testovací prostředí

Následující podkapitoly obsahují popis testovacího zařízení s výčtem použitých hardwarových prvků.

6.1.1 Testovací zařízení

Pro testování mobilní aplikace byly používány následující tři zařízení:

- tablet ASUS MeMO Pad HD7, Android 4.2.2
- mobilní telefon Motorola Moto G XT1039, Android 4.4.4
- mobilní telefon Huawei Ascend G300, Android 2.3.6

Ve hře jednoho hráče bylo použito první jmenované zařízení dále v textu označované jako ASUS. Druhé zařízení bylo použito pro měření s mobilním připojením a jako druhé zařízení ve hře více hráčů. V dalším textu bude označováno Motorola. Poslední testovací zařízení (Huawei) bylo použito při hře tří hráčů a pro porovnání slabého zařízení s mobilním připojením ve hře dvou hráčů s UDP protokolem.

6.1.2 Konfigurace pro testování WLAN

V lokální síti byl použit klasický stolní počítač, ke kterému byl připojen router v základním nastavení bez připojení k internetu. Router umožňoval propojení počítače s mobilními zařízeními v dosahu jeho Wi-Fi sítě. Stejná

konfigurace byla použita pro měření Peer-to-Peer, kde se mobilní zařízení propojovaly přímo mezi sebou. Kromě mobilních zařízení tedy byly použity následující prvky:

- Osobní počítač, OS Windows 8.1
- Wi-Fi router ASUS WL-500G Premium

6.1.3 Konfigurace pro testování připojení přes internet

Pro testování přenosu dat přes internet na vzdálený server byl použit opět klasický stolní počítač. Ten musel mít veřejnou statickou IP adresu, aby bylo možné se k němu připojovat z vnější sítě. Dalšími prvky spojení byly dva Wi-Fi routery, z nichž jeden připojoval do internetu server a druhý klienty.

- Osobní počítač, OS Windows 8.1, statická veřejná IP adresa
- Wi-Fi router Techicolor TC7200.U na straně klienta
- Wi-Fi router Zyxel VMG1312-B30B na straně serveru

Latence sítě naměřená programem *ping* mezi koncovými body spojení se pohybuje okolo 30ms, což je poměrně vysoká hodnota daná především typem připojení koncového routeru (VDSL). Na druhou stranu taková hodnota v internetu není nijak extrémní a naměřená data tak lépe reflektují podmínky použití v reálném prostředí.

6.2 Scénář hry

Měření je založeno na obvyklém průběhu hry, což znamená pohyb vozidla s občasným výstřelem a průjezdem alespoň jednoho kola základní herní mapy. Během pohybu dochází často ke kolizím s budovami, případně dalšími hráči. Každé měření probíhá pro stejný scénář opakovaně, aby byly odhaleny případné výjimečné stavy. Hra končí buď standardním způsobem (průjezd cílem, zničení soupeře nebo kombinace obou případů vedoucí k rozhodnutí

a ukončení hry) nebo ukončením aplikace z důvodu nedostatečné kvality spojení.

Aplikace (pro Android a pro PC), na kterých byly jednotlivé testovací scénáře měřeny, se od výsledných implementací liší v rozšíření o zápis statistik do logovacích souborů. Použití souborů eliminuje následky celkového zpomalení aplikací při výpisu hodnot přímo do konzole. Zřejmě i při tomto způsobu sběru dat dochází k jistému zpomalení, ale jeho efekt je minimální a neovlivňuje znatelně chod hry.

6.3 Měřená data

Základním měřeným parametrem je u multiplayerových her RTT (Round Trip Time). To je doba mezi odesláním zprávy a přijetím jejího potvrzení druhou stranou. RTT se v rpogramu zjišťovalo na základě zpráv *PingClientMessage*, které klienti odesílali na server po každých 10 iteracích herní smyčky a *PongServerMessage*, kterými server na tyto zprávy reagoval. Samozřejmě platí, že čím menší je RTT, tím kratší dobu trvá přenos zprávy sítí a tím lepší je celková hratelnost (tím aktuálnější a plynulejší je herní svět hráčů).

Ideálním RTT by bylo $0ms$, což je stav, kterému se blíží scénář s přímým předáváním odesíláné zprávy do příjmového kanálu (i ten ale provádí serializaci a deserializaci každé zprávy na obou stranách spojení). Za maximální hratelné lze považovat hodnoty okolo $50ms$, kdy už je ale třeba počítat s menší plynulostí hry. Větší hodnoty se pak zřetelně projevují zaseknutím aplikace a to i v případě ojedinělých výkyvů.

Dalším ukládaným údajem byly celkové počty odeslaných a přijatých zpráv podle jejich typů společně s jejich celkovou velikostí.

Vytvářeny byly také soubory s časy aktualizace grafického vlákna a časy aktualizace vlákna fyziky. Poslední logovací soubor obsahoval časy odeslání zpráv a čas přijetí jejich potvrzení druhou stranou. Kombinací této trojice souborů bylo možné vizualizovat chování celé aplikace v různých podmínkách.

Tyto soubory také umožňovaly počítat snímkování obrazovky (FPS - Frames per Second) grafického vlákna a fyzikálního vlákna. Vzhledem ke krokování v pevných intervalech obou vláken se ale naměřené údaje nelišily

od stanovených kroků. Ani fyzikální krok, ani překreslování scény totiž nepřekračuje poskytnutý interval a případné zasekávání je způsobováno přímo komunikační vrstvou nebo následky jejího chování.

Data získaná z logovacích souborů byla načtena v programu Matlab, pomocí nějž pak byly vytvořeny grafy v následujícím textu.

6.4 Množství a velikost dat podle zpráv

V kapitole 5.8 byl popsán typ zpráv posílaných během hry a jejich funkce. Pro představu o využití sítě je ale potřeba ještě specifikovat jejich velikost a počet v závislosti na délce hry.

Jako referenční situace v tomto případě dobře poslouží data ze hry dvou hráčů s využitím protokolu UDP a připojených ke vzdálenému serveru. RTT této konkrétní hry je znázorněno na obrázku 6.8 a 6.9.

Ačkoliv oba hráči hráli stejnou hru ve stejných rolích a za srovnatelných podmínek, v počtu odeslaných a přijatých zpráv je možné najít odchylky. Konkrétní počty zpráv a součet jejich velikostí zobrazuje tabulka 6.1 a 6.2. V tabulkách nejsou uvedeny typy zpráv, které nebyly během hry použity.

Typ zprávy	**Odesláno**	**Přijato**
ConnectionEstablished_SM	0 (0B)	1 (58B)
ShootBullet_CM	6 (342B)	0 (0B)
CountDown_SM	0 (0B)	60 (3 300B)
ConnectionEstablish_CM	1 (60B)	0 (0B)
GamePrepared_CM	1 (53B)	0 (0B)
ListPlayers_SM	0 (0B)	1 (71B)
ServerState_SM	0 (0B)	3634 (473 396B)
ControlsState_CM	3512 (200 184B)	0 (0B)
ConnectionAlive_CM	11 (583B)	0 (0B)
Ping_CM	667 (40 687B)	0 (0B)
Pong_SM	0 (0B)	667 (393 53B)
Celkem	4198 (214 909B)	4363 (516 178B)

Tabulka 6.1: Celkový počet a velikost zpracovaných zpráv - hráč 1

Typ zprávy	**Odesláno**	**Přijato**
ConnectionEstablished_SM	0 (0B)	1 (65B)
ShootBullet_CM	8 (456B)	0 (0B)
CountDown_SM	0 (0B)	60 (3 300B)
ConnectionEstablish_CM	1 (67B)	0 (0B)
GamePrepared_CM	1 (53B)	0 (0B)
ListPlayers_SM	0 (0B)	1 (71B)
ServerState_SM	0 (0B)	3635 (473 485B)
ControlsState_CM	3520 (200 640B)	0 (0B)
ConnectionAlive_CM	11 (583B)	0 (0B)
Ping_CM	720 (43 920B)	0 (0B)
Pong_SM	0 (0B)	720 (42 480B)
Celkem	4261 (245 719B)	4417 (519 401B)

Tabulka 6.2: Počet a velikost zpracovaných zpráv - hráč 2

Z obou tabulek je vidět, že objem přijímaných dat je výrazně vyšší než objem odeslaných dat (více než dvojnásobně). Celkový počet odeslaných a přijatých zpráv je ale pro oba směry a oba hráče přibližně stejný.

Počty i velikost dat odpovídají ostatním měřením. Na základě toho je tedy možné předpokládat, že mimo chybové stavy (kdy bude přenesený počet menší) bude sítí za stejnou jednotku času přeneseno přibližně stejné množství zpráv a poměr jejich velikostí bude zhruba odpovídat uvedenému příkladu.

Zprávy mohou být delší v případě hry s vysokým počtem pohybujících se předmětů (více objektů, o kterých bude server pravidelně informovat hráče).

Na základě měření můžeme očekávat přenos přibližně 2KB/s v těle zprávy ve směru od klienta na server a 4KB/s od serveru ke každému klientovi. Tato data jsou balena do paketů, jejichž celková velikost záleží na typu sítě a cílové adrese (pro různé typy sítí mohou být adresovací hlavičky paketů různě dlouhé).

6.5 Výsledky měření RTT

Nejprve byl otestován vliv Nagleho algoritmu na RTT u protokolu TCP, který určil vhodné nastavení parametru TCP_NODELAY.

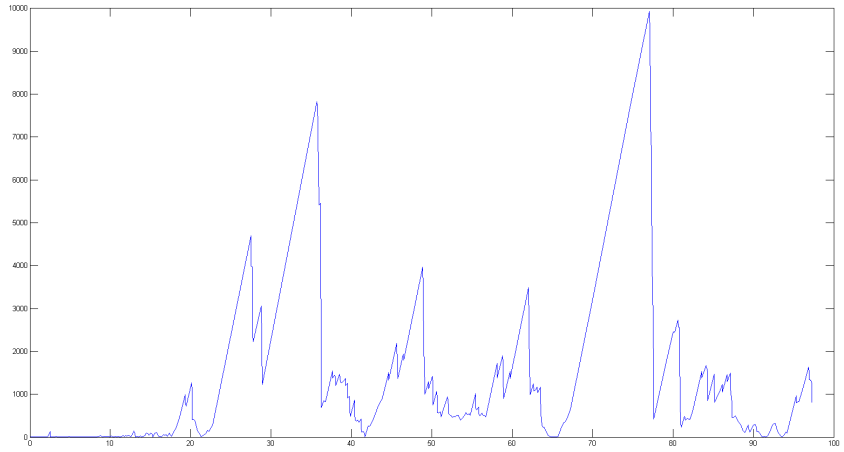
Na základě výsledků byl následně tento algoritmus deaktivován vlajkou TCP_NODELAY a byly porovnány hodnoty RTT protokolu TCP s různými druhy síťového spojení.

V dalších kapitolách následuje měření RTT pro protokol TCP, kde jsou nejprve naměřené hodnoty pro hru jednoho hráče ve všech typech spojení. Výsledky základního typu spojení, tedy spojení se vzdáleným serverem, byly porovnány s RTT při hře více hráčů.

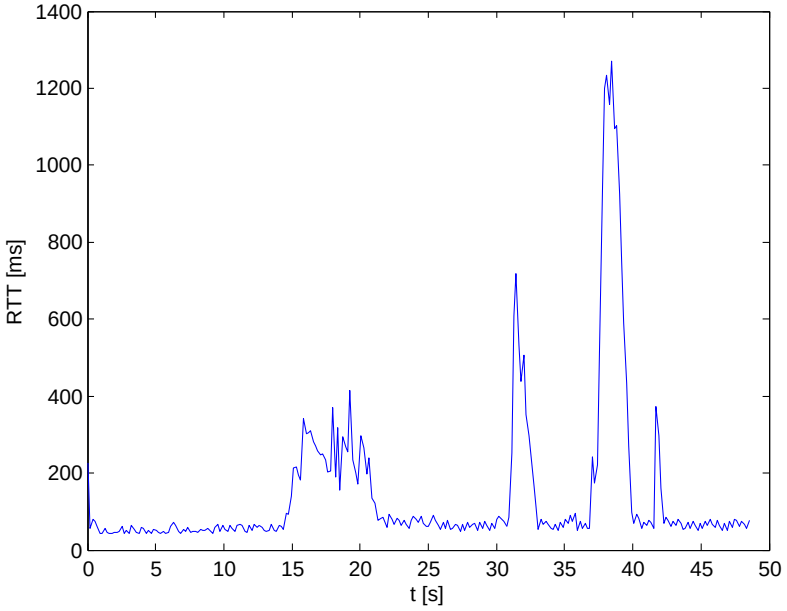
Poslední část měření se věnuje připojení s využitím mobilní sítě a zachycuje rozdíl RTT v porovnání s UDP.

6.6 Vliv Nagleho algoritmu

Jak je uvedeno v 3.1, TCP protokol používá algoritmus pro shlukování malých zpráv do větších celků, čímž redukuje zahlcování sítě a snižuje režii spojenou s bezpečným doručením každé zprávy. Následující obrázky 6.1 a 6.2 ale ukazují důvod, proč není vhodný pro hry v reálném čase. Oba jsou vytvořeny z dat nasbíraných při hrách jednoho hráče na zařízení ASUS s připojením ke vzdálenému serveru. Nagleho algoritmus byl samozřejmě zapnut respektive vypnut na obou stranách spojení.



Obrázek 6.1: Graf RTT s aktivním Nagleho algoritmem



Obrázek 6.2: Graf RTT bez aktivního Nagleho algoritmu

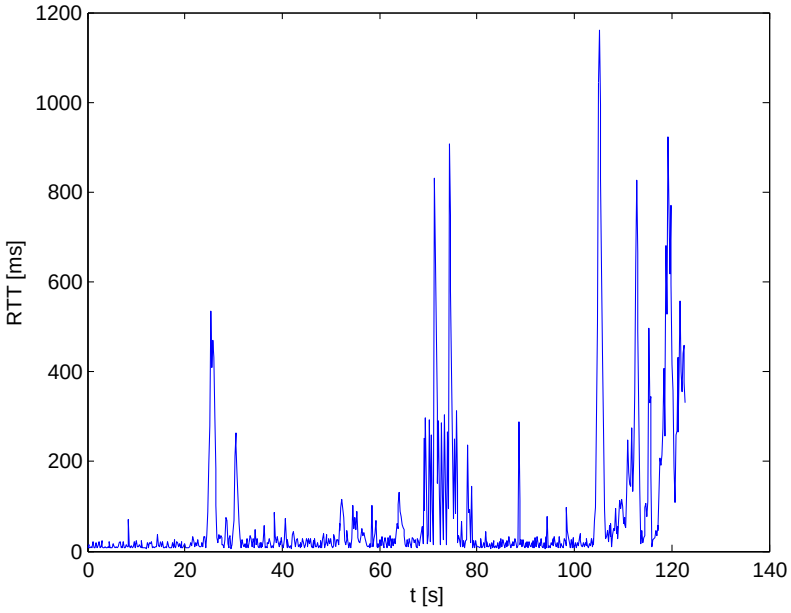
Mezi oběma grafy je velký rozdíl. Zatímco s Nagleho algoritmem šplhá RTT až k $10s$, po jeho deaktivaci jsou hodnoty většinu herního času celkem dostačující, řádově v rozmezí od 20 do $60ms$. Důvodem tak vysokých hodnot prvního případu může být skutečně malá velikost odesílaných zpráv ze strany klienta (řádově 60B velká datová část) a jejich frekvence. Pro všechny hry s protokolem TCP jsou ale typická občasná zaseknutí i na delší časová interval (viz obrázek 6.2), což už je velmi nepříjemná vlastnost.

Při vypnutí Nagleho algoritmu se projevil pro změnu problém na straně serveru. Velké množství krátkých zpráv, kterými je zaplavována síť vede především při hře více hráčů k jejímu zahlcení. To se projevovalo tak, že herní server sice tyto pakety přijímá a hra normálně pokračuje, ale ostatní služby na fyzickém serveru mohou být po krátké době omezeny nebo zcela odstřiženy. To je dáno pravděpodobně charakteristikou směrování TCP/IP paketů od klientského routeru k serverovému.

Měření tedy potvrdila původní očekávání, že protokol TCP pro realtimový multiplayer nebude vhodný. Pro porovnání jeho RTT s protokolem UDP byla ale provedena ještě další měření s použitím dalších typů připojení.

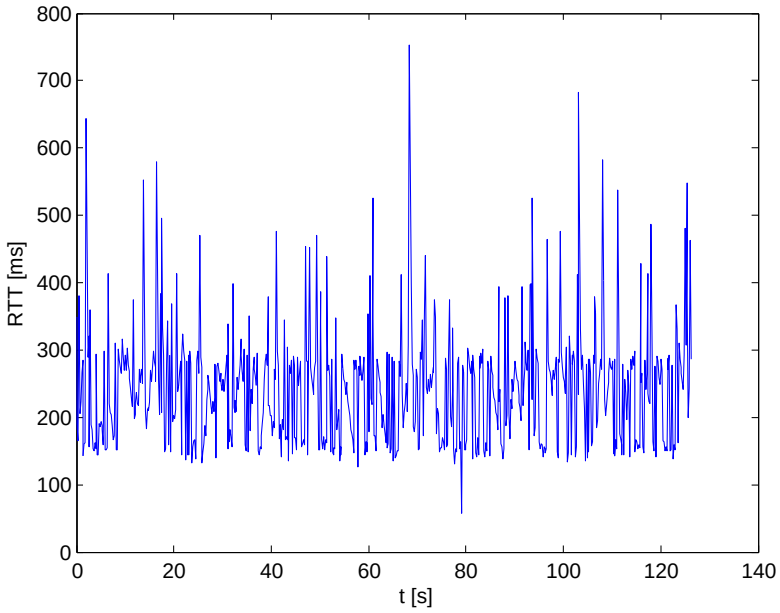
6.7 Hodnoty RTT pro TCP

V předchozí kapitole byly uvedeny hlavní problémy protokolu TCP a důvod, proč je v programu u TCP Nagleho algoritmus deaktivován. Následující měření jsou zaměřena na porovnání konkrétních typů spojení klienta se serverem. V prvním obrázku 6.3 je zachycen typický průběh RTT při hře jednoho hráče na lokální síti.



Obrázek 6.3: Graf RTT při hře jednoho hráče v místní síti

Následující obrázek zachycuje hru dvojice hráčů připojených ke vzdálenému serveru. Oproti grafu na obrázku 6.2 je patrný výrazný nárůst RTT, který je pravděpodobně způsoben zmíněným zahlcením sítě na straně serveru. Tento efekt se projevoval ve větší či menší míře v každé hře.

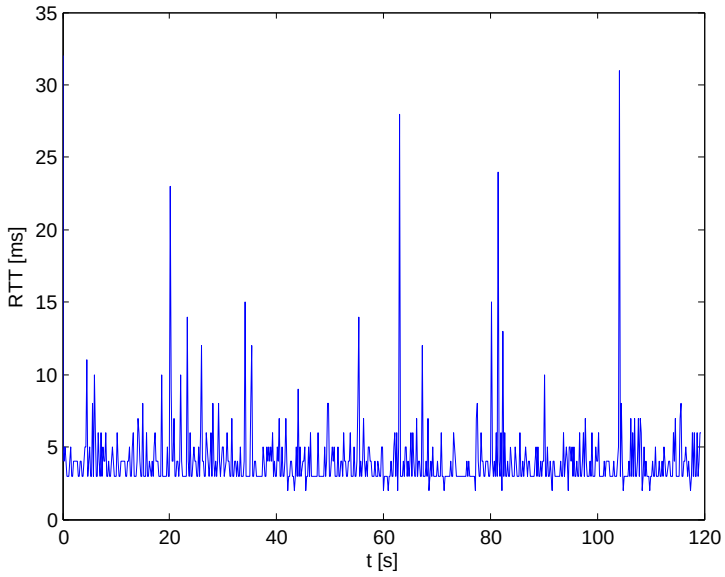


Obrázek 6.4: Graf RTT při hře dvou hráčů na vzdáleném serveru

6.8 Hodnoty RTT pro UDP

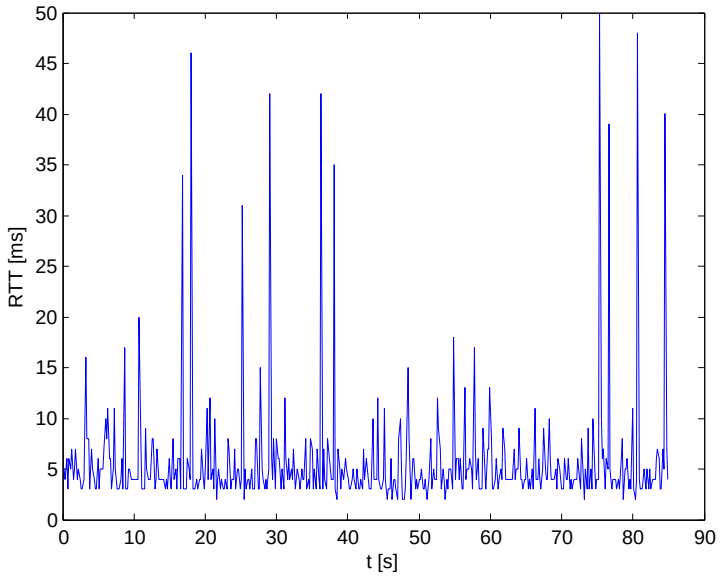
Tato kapitola popisuje hodnoty RTT při použití protokolu UDP, které byly naměřeny pro jednoho hráče s různým typem připojení a pro dvojici hráčů s připojením ke vzdáleném serveru.

První graf 6.5 zobrazuje situaci, kdy hraje jeden hráč v roli serveru i klienta bez připojení do sítě. Obvyklá komunikace tedy probíhá lokálně a RTT není ovlivněno přenosem sítí (narůstá podle režie se zpracováním a přeposíláním zpráv, synchronizace vláken aplikace nebo aktuálními zdroji aplikace).



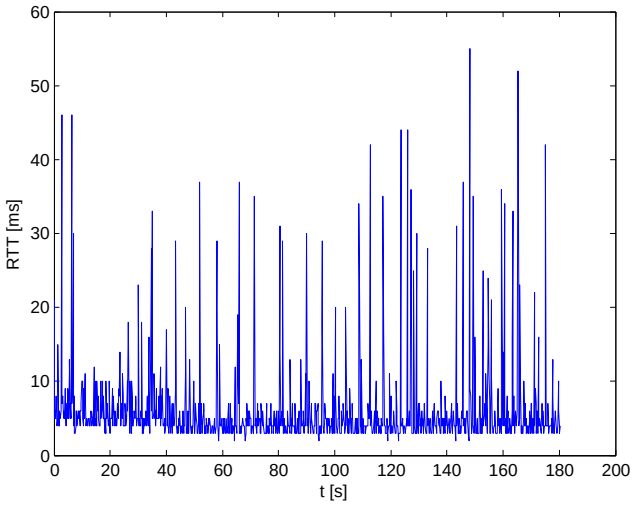
Obrázek 6.5: Graf RTT pro hru jednoho hráče lokálně

Následující graf 6.6 zachycuje hru jednoho hráče na vzdáleném serveru.



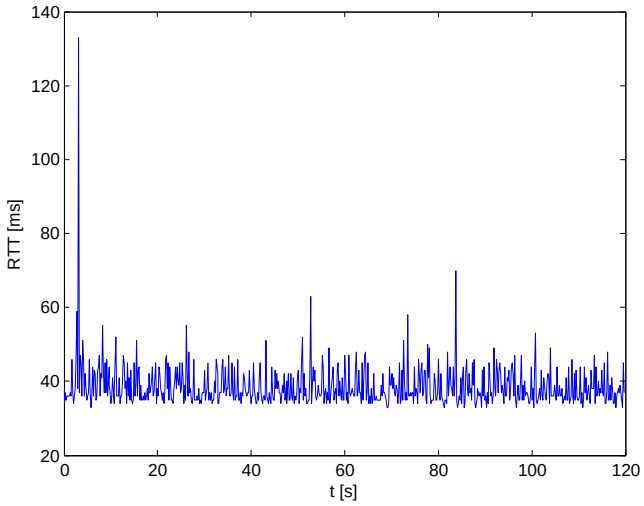
Obrázek 6.6: Graf RTT pro hru jednoho hráče na vzdáleném serveru

Rozdíl oproti hře jednoho hráče v lokální síti (obrázek 6.7) je minimální a předpokládaná výhoda lokální sítě se od připojení ke vzdálenému serveru ani pro opakovaná měření výrazně neprojevila.

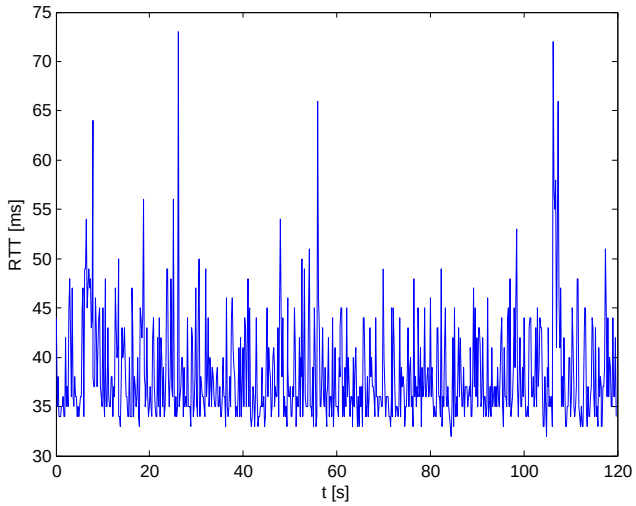


Obrázek 6.7: Graf RTT pro hru jednoho hráče v místní síti

Špička na začátku dalšího grafu 6.8 značí okamžik, kdy byl připojen druhý klient, jehož RTT je zachyceno na obrázku 6.9. Oba grafy jsou naměřeny na stejné hře a jak je patrné, mezi jejich průběhem není nijak výrazný rozdíl (oběma se RTT pohybuje v rozmezí $32ms$ až $60ms$).

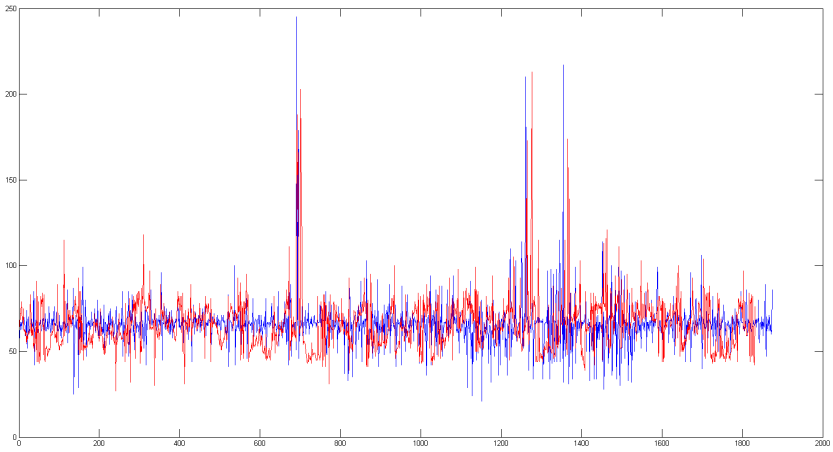


Obrázek 6.8: Graf RTT hry dvojice hráčů na vzdáleném serveru - ASUS



Obrázek 6.9: Graf RTT hry dvojice hráčů na vzdáleném serveru - Motorola

Poslední obrázek kapitoly zachycuje graf RTT pro dvojici hráčů připojenou k zatíženému serveru. Ve srovnání s předchozí dvojicí grafů je patrné zhoršení o přibližně $25ms$. To už má samozřejmě nepříjemný dopad na hratelnost, ale stále se nejedná o nijak extrémní hodnoty. Zcela jistě to ale poukazuje na potřebu zajištění dostatečně výkonných prvků na straně serveru.

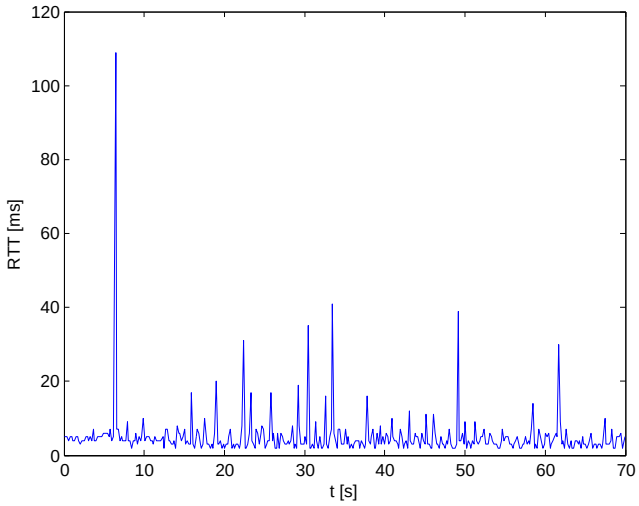


Obrázek 6.10: RTT hry dvojice hráčů na zatíženém vzdáleném serveru

6.9 Hodnoty RTT při Peer-to-Peer

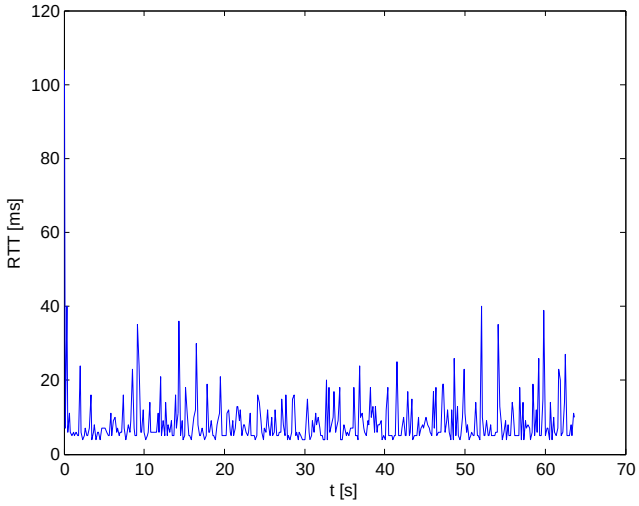
Doba mezi odesláním zprávy a přijetím jejího potvrzení byla pro Peer-to-Peer spojení v lokální síti testována ve hře dvojice hráčů pro protokol TCP i UDP. Pro porovnání pak bylo změřeno RTT i pro hru tří hráčů.

Na prvním grafu 6.11 je zachyceno RTT komunikace prvního zařízení v režimu server i klient. Jsou to doby potvrzení zpráv, které je možné porovnat se hrou jednoho hráče lokálně (viz 6.5). Rozdíl od uvedené hry je v tom, že na zařízení navíc probíhá komunikace s dalším klientem. Naměřené výsledky ale výrazný nárůst RTT nevykazují.



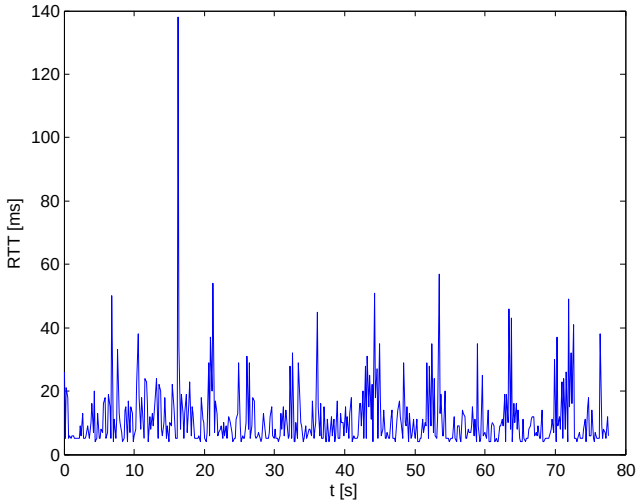
Obrázek 6.11: Hodnoty RTT hry dvou hráčů u prvního hráče (server i klient)

Další graf 6.12 zobrazuje stejnou hru z pozice druhého klienta. Ani ten při srovnání s grafem 6.7 neukazuje nárůst RTT. Z toho můžeme vyvodit, že není výrazný rozdíl v RTT při komunikaci klienta se serverovou aplikací na počítači v lokální síti a při spojení Peer-to-Peer k jinému zařízení.



Obrázek 6.12: Hodnoty RTT hry dvou hráčů u druhého hráče (kleint)

Následující obrázek 6.13 zachycuje hru tří hráčů z pozice druhého klienta. U prvního klienta (v roli server i klient) se situace nijak nelišila od Peer-to-Peer dvojice hráčů. Stejně tak nebyl patrný rozdíl mezi RTT druhého a třetího připojeného klienta. Celkově byly ale RTT klientů lehce vyšší a na zobrazeném grafu je patrný výkyv až ke $140ms$. To odpovídá předpokladu, že první zařízení řídící hru i komunikaci má při vyšším počtu hráčů horší časy reakcí na příchozí zprávy.

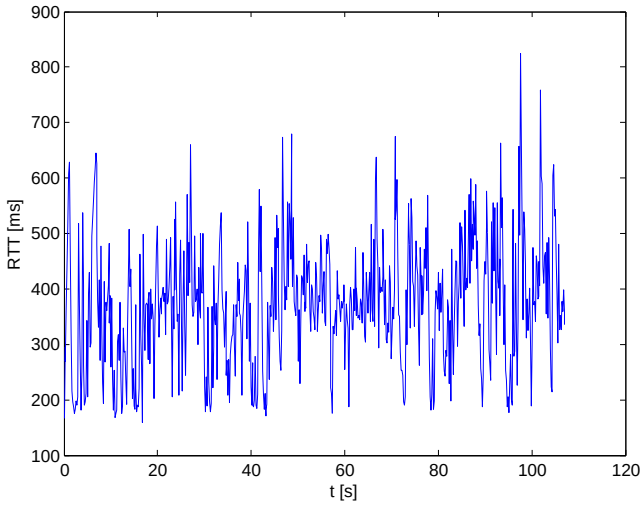


Obrázek 6.13: Hodnoty RTT druhého hráče ve hře tří hráčů

6.10 Hodnoty RTT s mobilním připojením

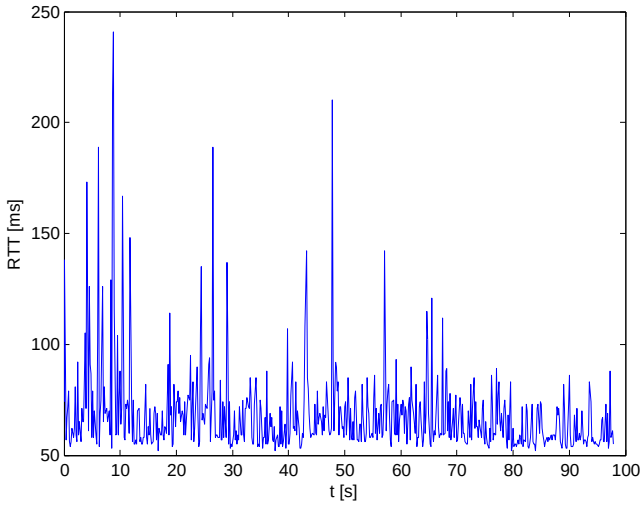
U mobilního připojení byly měřeny hry jednoho a dvou hráčů s připojením ke vzdálenému serveru. U hry dvou hráčů byly navíc porovnány hodnoty RTT mobilního spojení pomocí 2.5G technologie EDGE s připojením přes Wi-Fi na nejpomalejším testovacím zařízení (Huawei).

První obrázek 6.15 zobrazuje hodnoty RTT pro hru jednoho hráče, který je pomocí mobilní sítě EDGE připojen ke vzdálenému serveru.



Obrázek 6.14: Hodnoty RTT pro hru jednoho hráče přes EDGE

Následující obrázek 6.15 ukazuje graf RTT s připojením do mobilní sítě pomocí HSDPA. Oproti EDGE se jedná o výrazné zlepšení a hra měla po většinu času sice lehce zpožděné reakce, ale byla plynulá a hratelná, což se o EDGE říct bohužel nedá.

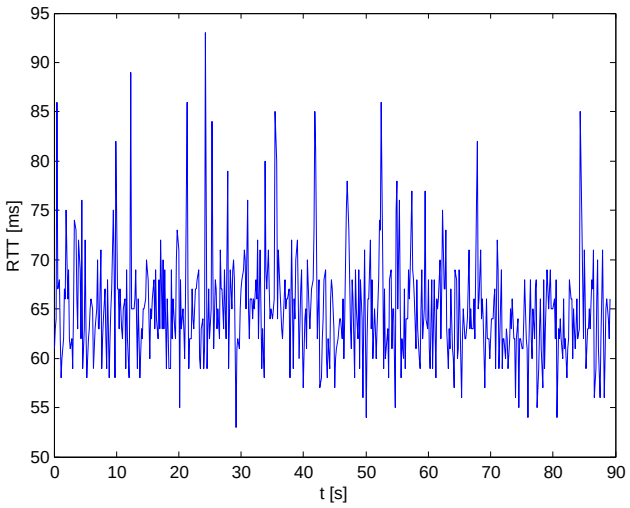


Obrázek 6.15: Hodnoty RTT pro hru jednoho hráče přes HSDPA

Pro hru můžeme tento typ sítě považovat za dostatečný, protože hodnoty

RTT se i přes občasné výkyvy pohybují stále okolo $60ms$ a hra je poměrně plynulá.

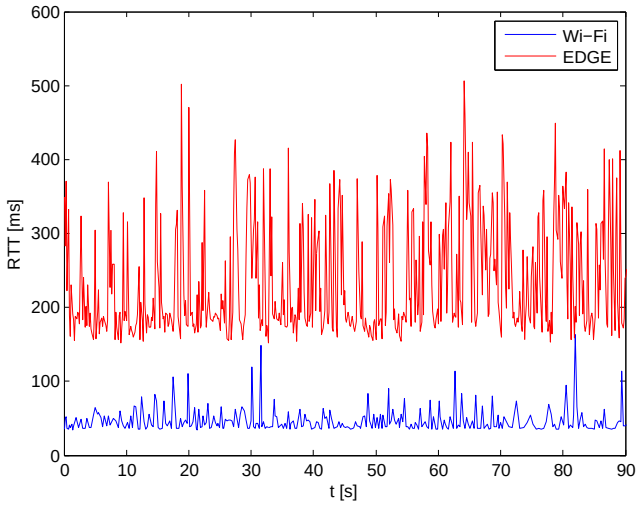
Následující obrázek 6.16 zobrazuje hodnoty RTT s mobilním připojením standardu LTE na frekvenci 1800MHz, více v [58].



Obrázek 6.16: Hodnoty RTT pro hru jednoho hráče přes HSDPA

Průběh grafu je zřetelně lepší než u dříve měřeného HSDPA. Výkyvy RTT nejsou tak velké ani časté a průměrné RTT je podle grafů také nižší.

V následujícím obrázku 6.17 je zřetelně zachycen rozdíl mezi velikostí RTT u mobilního připojení EDGE a u připojení pomocí Wi-Fi. Zatímco klient připojený ke hře přes Wi-Fi měl hru ještě celkem hratelnou, u hráče připojeného přes EDGE to ji hrát v podstatě nebylo možné. Dobře to ukazuje rozdíl mezi oběma technologiemi a ukazuje to, že pro síťové hry takové mobilní připojení není vhodné.



Obrázek 6.17: Hodnoty RTT pro dvojici hráčů s různým typem připojení

7 Závěr

Úvodem práce bylo poskytnuto teoretické zázemí k problematice síťových multiplayerových her na platformě Android. Prvotně byla zkoumána specifika této platformy a rozdíly oproti tradičním herním platformám jako jsou herní konzole a PC. Následně byly podrobně popsány běžně používané technologie pro přenos dat a porovnány dva základní modely komunikace: Klient-Server a Peer-to-Peer.

Základem multiplayerových her je přenos herních dat. V další část textu tedy byly prozkoumány protokoly, které se běžně pro přenos dat ve hře používají. Byly detailně rozebrány jejich výhody a nevýhody. Po jejich analýze byla prozkoumána i samotná data, která ve hrách obvykle proudí sítí.

Další, stále ještě teoretická část, se věnuje možnostem řízení fyzikálního světa. Popisovány jsou dva nejčastější způsoby řešení: fyzikální svět pouze na straně serveru a model s predikcí na straně klienta. Byl zde vysvětlen jejich princip a na závěr byly shrnuty jejich výhody a nevýhody.

Na základě provedené analýzy byla navržena komplexní multiplayerová realtimová hra inspirovaná počítačovou hrou Death Rally. Pro komunikaci byl sestaven protokol, který definuje zprávy posílané sítí a jejich efekty. Zprávy byly posílány protokolem TCP nebo UDP, který byl navíc rozšířen o mechanismus bezpečného doručení. Hra používala fyzikální knihovnu JBox2D, což znatelně zvyšovalo složitost architektury aplikace. Dalším atributem hry bylo používání projekce do 3D zobrazení. Z důvodu kombinace s fyzikou a komplexní strukturou komunikační vrstvy bylo nezbytné navrhnout kompletní aparát správy souřadnic a synchronizace používaných vláken. Pro rozšíření možných testovacích scénářů pro spojení byl z mobilní aplikace odvozen ještě konzolový desktopový program, který umožňoval i simultánní hry.

Takto vytvořená testovací platforma umožnila porovnat širokou škálu scénářů s různými typy připojení. Měření ověřila předpoklad, že protokol TCP se pro hry v reálném čase příliš nehodí. Protokol UDP ve všech případech vykazoval lepší výsledky. Připojení ke hře pomocí mobilní sítě také splnilo přepdoklady v tom smyslu, že u sítí typu 2G nebylo prakticky možné hru hrát, zatímco 4G síť už byla pro plynulou hru dostačující. Připojení ke vzdálenému serveru vykazovalo lepší výsledky než nejrychlejší testovaná mobilní síť, ale pochopitelně ne tak dobré jako spojení v lokální síti. To bylo dobré jak pro spojení server-klient, tak pro Peer-to-Peer.

Literatura

[1] **Angry Birds**. (Červen 2015).
https://www.angrybirds.com/

[2] **Candy Crush Saga**. (Červen 2015).
http://candycrushsaga.com/

[3] **Fruit Ninja**. (Červen 2015).
http://fruitninja.com/

[4] **Mobilenet.cz: Váš telefon je prošpikovaný senzory**. (Červen 2015).
http://mobilenet.cz/clanky/techbox-vas-telefon-je-prospikovany-senzory-12496

[5] **Google Play: Asphalt 7**. (Červen 2015).
https://play.google.com/store/apps/details?id=com.gameloft.android.ANMP.GloftA7HM&hl=cs

[6] **Ingress**. (Červen 2015).
https://www.ingress.com/

[7] **Dumb Ways To Die**. (Červen 2015).
http://dumbwaystodie.com/

[8] **Andoid Developers: Sensors Overview** (Červen 2015).
http://developer.android.com/guide/topics/sensors/sensors_overview.html

[9] **Mobilmania.cz: Konec zmatků v rozlišení displejů**. (Červen 2015).
http://www.mobilmania.cz/clanky/konec-zmatku-v-rozliseni-displeju-prehled/sc-3-a-1319962/default.aspx

[10] **World of Tanks Blitz**. (Červen 2015).
https://wotblitz.eu/

[11] **Android Developers: Activities**. (Červen 2015).
http://developer.android.com/guide/components/activities.html

[12] **Bluetooth® Core Specification 4.2**. (Červen 2015).
https://www.bluetooth.org/en-us/Documents/Bluetooth4-2QuickRefGuide.pdf

[13] **Bluetooth®**. (Červen 2015).
http://www.bluetooth.com/Pages/Bluetooth-Home.aspx

[14] **Wi-Fi Alliance: Wi-Fi Direct**. (Červen 2015).
http://www.wi-fi.org/discover-wi-fi/wi-fi-direct

[15] **Android Central: Tethering**. (Červen 2015).
http://www.androidcentral.com/android-internet-tether

[16] **Android Developers: Wi-Fi P2P**. (Červen 2015).
http://developer.android.com/guide/topics/connectivity/wifip2p.html

[17] **Wi-Fi Alliance: How fast is Wi-Fi Direct**. (Červen 2015).
http://www.wi-fi.org/knowledge-center/faq/how-fast-is-wi-fi-direct

[18] **ITBIZ: Průvodce po mobilních sítích nejen čtvrté generace (1.)**. (Červen 2015).
http://www.itbiz.cz/pruvodce-po-mobilnich-sitich-nejen-ctvrte-generace-aneb-co-znamena-3g-4g-hspa-lte-ci-wimax-1-cast

[19] **ITBIZ: Průvodce po mobilních sítích nejen čtvrté generace (2.)**. (Červen 2015).
http://www.itbiz.cz/pruvodce-po-mobilnich-sitich-nejen-ctvrte-generace-aneb-co-znamena-3g-4g-hspa-lte-ci-wimax-2-cast

[20] **ITBIZ: Průvodce po mobilních sítích nejen čtvrté generace (3.)**. (Červen 2015).
http://www.itbiz.cz/pruvodce-po-mobilnich-sitich-nejen-ctvrte-generace-aneb-co-znamena-3g-4g-hspa-lte-ci-wimax-dokonceni

[21] *L. Pešička:* **Přednášky k předmětu KIV/MKZ**.
ZČU, 2015.

[22] **ExtremeTech: ITU designates LTE-Advanced as "True 4G"**. (Červen 2015).
http://www.extremetech.com/mobile/114953-itu-designates-lte-advanced-as-true-4g

[23] **Gaffer on Games: What every programmer needs to know about game networking**. (Červen 2015).
http://gafferongames.com/networking-for-game-programmers/what-every-programmer-needs-to-know-about-game-networking

[24] **Game Development Stack Exchange: Limitations of p2p multiplayer games vs client-server**. (Červen 2015).
http://gamedev.stackexchange.com/questions/67738/limitations-of-p2p-multiplayer-games-vs-client-server

[25] **RFC 1151 - Version 2 of the Reliable Data Protocol (RDP)** . (Červen 2015).
http://tools.ietf.org/html/rfc1151

[26] **RFC 3286 - An Introduction to the Stream Control Transmission Protocol (SCTP)**. (Červen 2015).
https://tools.ietf.org/html/rfc3286

[27] **ENet: Features and Architecture** . (Červen 2015).
http://enet.bespin.org/Features.html

[28] **Gaffer on Games: UDP VS. TCP**. (Červen 2015).
http://gafferongames.com/networking-for-game-programmers/udp-vs-tcp/

[29] **What is Nagle's algorithm - TechTarget**. (Červen 2015).
http://searchnetworking.techtarget.com/definition/Nagles-algorithm

[30] **RFC 1122 - Requirements for Internet Hosts - Communication Layers**. (Červen 2015).
https://tools.ietf.org/html/rfc1122

[31] **RFC 896 - Congestion Control in IP/TCP Internetworks**. (Červen 2015).
https://tools.ietf.org/html/rfc896

[32] **Know a Delay: Nagle's Algorithm and You. - Boundary**. (Červen 2015).
http://www.boundary.com/blog/2012/05/know-a-delay-nagles-algorithm-and-you/

[33] **TCP Performance problems caused by interaction between Nagle's Algorithm and Delayed ACK**. (Červen 2015).
http://www.stuartcheshire.org/papers/nagledelayedack/

[34] **Gaffer on Games: Reliability, Ordering and Congestion Avoidance over UDP**. (Červen 2015).
`http://gafferongames.com/networking-for-game-programmers/reliability-and-flow-control/`

[35] **RFC 1122 - Requirements for Internet Hosts - Communication Layers**. (Červen 2015).
`https://tools.ietf.org/html/rfc1122`

[36] **Oracle: Java Native Interface**. (Červen 2015).
`https://docs.oracle.com/javase/6/docs/technotes/guides/jni/`

[37] **Unity - Game Engine**. (Červen 2015).
`http://unity3d.com/`

[38] **Tuts+ Game Development Article: What Is "Client-Side Prediction"?**. (Červen 2015).
`http://gamedevelopment.tutsplus.com/articles/gamedev-glossary-what-is-client-side-prediction--gamedev-3849`

[39] **Best Old Games: Death Rally**. (Červen 2015).
`http://www.bestoldgames.net/death-rally`

[40] **AndEngine**. (Červen 2015).
`http://www.andengine.org/blog/`

[41] **National Instruments: Application Design Patterns - Producer/-Consumer** `http://www.ni.com/white-paper/3023/en/`

[42] *R. Pecinovský:* **Návrhové vzory**.
ISBN 9788025115824, Computer Press, 2007.

[43] **Teach Yourself Game Programming in 21 Days**
ISBN 9780672305627, Sams Pub., 1994.

[44] *J. Schroeder, B. Broyles:* **Andengine for Android Game Development Cookbook**. Packt Publishing, Limited 2012. ISBN 9781849518987.

[45] **JBox2D: A Java Physics Engine**. (Červen 2015).
`http://www.jbox2d.org/`

[46] **JBullet: Java port of Bullet Physics Library**. (Červen 2015).
`http://jbullet.advel.cz/`

[47] **dyn4j: Java Collision Detection and Physics Engine**. (Červen 2015).
`http://www.dyn4j.org/`

[48] **ACM SIGGRAPH: Perspective Viewing Projection**. (Červen 2015).
https://www.siggraph.org/education/materials/HyperGraph/viewing/view3d/perspect.htm

[49] **OSU.EDU: Texture Mapping**. (Červen 2015).
http://web.cse.ohio-state.edu/~whmin/courses/cse5542-2013-spring/15-texture.pdf

[50] **Stack Overflow: Perspective correct texturing of trapezoid in OpenGL ES 2.0**. (Červen 2015).
http://stackoverflow.com/questions/15242507/perspective-correct-texturing-of-trapezoid-in-opengl-es-2-0

[51] **OpenGL.org: Shader. (Červen 2015).**
https://www.opengl.org/wiki/Shader

[52] **fabiensanglard.net: Quake Source Code Review. (Červen 2015).**
http://fabiensanglard.net/quakeSource/quakeSourcePrediction.php

[53] **Unlagged: Quake 3 Networking Primer. (Červen 2015).**
http://www.ra.is/unlagged

[54] **OpenGL ES 2_X - The Standard for Embedded Accelerated 3D Graphics. (Červen 2015).**
https://www.khronos.org/opengles/2_X/

[55] **Andoid Developers: OpenGL ES. (Červen 2015).**
http://developer.android.com/guide/topics/graphics/opengl.html

[56] **Box2D | A 2D Physics Engine for Games. (Červen 2015).**
http://box2d.org/

[57] **Andoid Developers: Storage Options. (Červen 2015).**
http://developer.android.com/guide/topics/data/data-storage.html#pref

[58] **LTE1800: A Versatile Platform for Connected Devices and Applications. (Červen 2015).**
http://www.gsma.com/spectrum/wp-content/uploads/2012/03/whitepapersierrawirelesslte1800.pdf

Přílohy na CD

Příloha č.1 Zdrojové soubory mobilní aplikace

Příloha č.2 Zdrojové soubory desktopové aplikace

Příloha č.3 Spustitelná mobilní aplikace

Příloha č.4 Spustitelná desktopová aplikace

Uživatelská příručka

Instalace mobilní aplikace FavRacer

Zkompilovaná aplikace je přiložena v souboru *FavRacer.apk*. Pro její instalaci je nutné mít na zařízení povolenou instalaci z neznámých zdrojů. To se provádí v *nastavení* mobilního zařízení v položce *zabezpečení*, kde se nachází nabídka *neznámé zdroje*.

V dalším kroku je nutné aplikaci přenést do mobilního zařízení. Toho je možné dosáhnout buď aplikací pro správu souborů, připojením zařízení v režimu Media Device (MTP) pomocí kabelu nebo například sdílením souboru na cloudové úložiště a eho následní stažení ze zařízení.

Instalace se spustí po otevření přeneseného souboru.

Spuštění aplikace už pak probíhá standardní cestou.

Spuštění desktopové aplikace

Pro spuštění serverové aplikace je nutné mít nainstalované prostředí Java JRE (aktuální je ve verzi v8u45).

Program se spouští z příkazové řádky následujícím příkazem:

java -jar FavRacer.jar

Program se ukončuje stiskem Ctrl+C.

Pokud všechno probíhá podle očekávání, dojde ke spuštění sady her definovaných v konfiguračním souboru *RunConfig.xml* uvnitř archivu *FavRacer.jar*. Tento soubor je možné měnit podle potřeby.

Spuštěné hry by měly být dostupné v lokální síti pro připojení. Kdykoliv dojde k ukončení hry, vyčká aplikace 10s a spustí ji znovu.

Parametry desktopové aplikace

Pokud nastavení sítě z nějakého důvodu nepředá aplikaci korektně dostupnou IP adresu, nepodaří se přihlásit klienty ke hře. V takovém případě je nutné IP adresu serveru zadat ručně. K tomu slouží parametr **-ip=** následovaný korektní IPv4 adresou.

Pokud je z nějkaého důvodu nedostupný port, na kterém server přijímá spojení, je možné nastavit mu jiný. K tomu slouží parametr **-p=** následovaný číslem cílového portu. V případě změny portu je potřeba informovat klienty, na kterém portu mohou dostupné hry vyhledávat.

Poslední parametr je experimentální a slouží k zapínání Nagleho algoritmu na straně serveru. Tím je parametr **nagle_on**.

Hlavní Menu

Hlavní menu hry obsahuje možnost změnit jméno hráče (zůstane uloženo) a trojici dalších možností.

Možnost Server

V tomto menu je možné nastavit parametry Peer-To-Peer hry. První řádek obsahuje lokální IP adresu, ke které se budou moci ostatní klienti připojovat. Nabídka s možnostmi nastavení zakládané hry obsahuje následující položky:

- Server name: Název hry zobrazovaný klientům při hledání her
- Players: Nastavení počtu hráčů, který je potřeba pro zahájení hry (2)
- Laps: Počet kol závodu (1)
- Level ID: Výběr mapy závodu (1)
- Game Port: Port, na kterém bude hra probíhat (3000)
- Discovery port: Port, na kterém je možné hru najít (6666)
- Protocol: TCP nebo UDP protokol (UDP)

Pokud uživatel nastaví počet hráčů na 1 a zvolí TCP protokol, bude spuštěna hra s přímým předáváním zpráv.

Možnost Klient

V tomto menu je možné zvolit konkrétní IPv4 adresu, na které budou hledány otevřené hry. Při potvrzení prázdného pole jsou hry hledány v lokální síti.

Možnost Test local

Tato volba umožňuje spuštění lokální hry s přímým předáváním zpráv. Je spouštěna bez možnosti další konfigurace s mapou jedna a závodem bez soupeřů na jedno kolo.

Ovládání

Ve hře každý hráč řídí přidělené vozidlo (podle pořadí přihlášení). Kromě toho má možnost střílet a měnit přiblížené kamey (výšku nad budovami).

- Pohyb

  Pro pohyb slouží ovládací prvek v pravém dolním rohu obrazovky. Ten má dotykem v jeho horní části se vozidlo hráče pohybuje dopředu (plyn). Dotek v dolní části působí na vozidlo jako zpátečka. Dotekem v pravé části Důležité je si uvědomit, že směr výsledného pohybu je určen na základě modelu vozidla, ne vzhledem k obrazovce.

- Střelba

  Ke střelbě slouží čvteřice tlačítak v pravém horním rohu obrazovky. Každé z nich aktivuje jiný typ zbraně. Tlačítko označené číslem 1 vystřelí raketu, která při zásahu soupeřům odebírá 40 bodů zdraví z celkového 100 a dobíjí se po 10s. Pod tlačítkem 2 se nachází mina, kterou je možné umístit na závodní trat'. Ta při kontaku odebírá 33 bodů a je možné ji opětovně použít po 10s. Tlačítko 3 spouští výstřel z pistole, který se obnovuje po 1.5s a ubírá 10 bodů. Poslední tlačítko slouží ke střelbě z kulometu. Ten se sice spouští každých 300ms, ale má po zásahu ubírá pouze 3 body zdraví.

Zásahy jsou doprovázeny grafickým i zvukovým efektem.

Obrázek 7.1: Dva hráči během přestřelky

- Přiblížení kamery

  K přiblížení nebo oddálení kamery je možné použít tlačítka v levém horním rohu. Tlačítko se symbolem plus obraz přiblíží, tlačítko se symbolem mínus naopak oddálí. Kamera zůstává zafixovaná na vozidlo hráče tak, aby bylo přesně ve středu obrazovky.

Kontrolní body

Na mapě se nachází kontrolní body, které musí být projety ve správném pořadí. Příští kontrolní bod je vždy zvýrazněn zelenou barvou.

Další menu a obrazovky

Hra obsahuje řadu obrazovek, které informují uživatele o vzniklé chybě nebo problému. Při ztrátě spojení se například objeví sytě červená obrazovka, při vyprchání časovače při čekání po odeslání žádosti o spojení fialová. Obrazovka výsledků závodu je sytě žlutá. Obrazovka s nalezenými hrami je bílá. Často je ji nutné překreslit tlačítkem "Search Again", aby se načetly všechny dostupné hry.