VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

LETECKÁ HRA PRO ANDROID

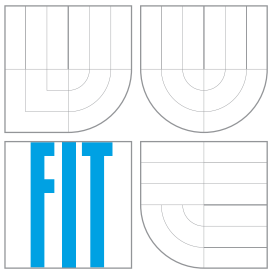
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DAVID ŠABATA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

LETECKÁ HRA PRO ANDROID

FLIGHT GAME FOR ANDROID

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DAVID ŠABATA

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2013

Abstrakt

Tato práce se zabývá vývojem letecké hry pro platformu Android. Nejdříve budou popsány možnosti nativního vývoje a vývoje s využitím knihovny Libgdx. Dále budou vysvětleny principy které se uplatňují při letu skutečného letadla a tyto budou zjednodušeny pro použití v leteckých hrách. Práce také shrne současné trendy v ovládání leteckých her na mobilních zařízeních a navrhne novou metodu založenou na dotykovém ovládání. V závěru bude popsán aktuální stav implementace.

Abstract

This work deals with flight game development on Android platform. Firstly the possibilities of native development and development using Libgdx library will be discussed. Then flight mechanics of a real aircraft will be explained and simplified for the use in flight games. The work will also summarize current trends in mobile flight game controls and will propose a new control method based on touch input. In the end the current state of implementation will be described.

Klíčová slova

vývoj her, mobilní zařízení, Android, mechanika letu, dotykové ovládání

Keywords

game development, mobile devices, Android, flight mechanics, touch input

Citace

David Šabata: Letecká hra pro Android, diplomová práce, Brno, FIT VUT v Brně, 2013

Letecká hra pro Android

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Doc. Ing. Adama Herouta, Ph.D.

.....
David Šabata
17. dubna 2013

Obsah

Kapitola 1

Úvod

TODO: revize

Hry pro mobilní zařízení, zejména telefony a tablety, v současnosti zažívají nebývalý rozmach. Dalo by se říct, že jsou nyní tam, kde byly před patnácti až dvaceti lety hry počítačové. Zatímco typický tým vyvíjející hru pro počítač nebo herní konzoli čítá desítky, ale často i stovky členů, hru pro mobilní telefon je stále možné stvořit jen v několika málo lidech.

Neustále narůstající výkon dovoluje přiblížit se kvalitě počítačových her. Těmi se nově díky mobilním zařízením může hráč zabavit například na cestách anebo při čekání na autobus. Velká většina lidí s sebou neustále nosí výkonný počítač ve velmi malém balení, který je navíc vybaven množstvím sensorů a často i soustavným připojením k internetu. To nabízí prostor pro velké množství inovací a je dobře že se takové inovativní přístupy skutečně objevují.

Tato práce se pokusí přispět svým dílem navržením nové metody ovládání leteckých her pro mobilní zařízení. Nejdříve popíše platformu na které bude implementace probíhat, dále vysvětlí principy díky kterým letí a manévruje skutečné letadlo. Tyto principy poté zjednoduší pro použití v prostředí her a shrne současné možnosti ovládání herních letadel. Nakonec navrhne novou metodu ovládání založenou na dotyku a popíše současný stav její implementace a hry na ní založené.

Kapitola 2

Vývoj na platformě Android

TODO: revize, aktualizovat procenta

Andriod je operační systém pro mobilní zařízení, založený na Linuxovém jádře a poskytovaný ve formě otevřeného zdrojového kódu. Poprvé byl představen v roce 2007, první mobilní telefon s Androidem se dostal na trh o rok později. Aktuálně dle statistik za třetí čtvrtletí roku 2012 má na poli *chytrých mobilních telefonů* (smartphone) Android jakožto operační systém podíl 75% [1]. Ačkoliv je Android použitelný pro množství různých zařízení (a implementace mimo pole mobilních telefonů se už nyní objevují), bude se tato práce soustřeďovat pouze na jeho využití v mobilních zařízeních typu mobilního telefonu nebo tabletu. Informace budou vycházet z [2].

2.1 Architektura systému

OK r1

Architekturu systému lze rozdělit do několika úrovní. Na té nejvyšší jsou aplikace psané v jazyce Java, které používají systémové funkce a rozhraní. Distribuují se v balíčcích v tzv. *mezikódu* (Java bytecode). Mezikód je výstup kompilace a jako takový je přenositelný mezi platformami. Pro spuštění však potřebuje interpret, kterým je v tomto případě virtuální stroj pojmenovaný Dalvik.

Dalvik samotný se nachází na nižší úrovni, tedy mezi programy a jejich součástmi v nativním kódu psanými v jazyce C, C++ či jazyce symbolických instrukcí. Zde můžeme najít především sadu systémových knihoven od kterých je vyžadován vysoký výkon. Jsou jimi například knihovny pro vykreslování (OpenGL), databáze (SQLite), šifrování (SSL) anebo multimédia. Jejich rozhraní je dostupné jak v prostředí Javy, tak skrze *JNI* (Java Native Interface). JNI je rozhraní umožňující psát programy v některém z výše uvedených jazyků s možností přímého propojení se součástmi psanými v Javě. Aplikace jako celek je pak rozdělená do dvou samostatně kompilovaných částí, jejichž spolupráci zajišťuje Dalvik.

Do nativního kódu je vhodné převést především provádění náročných operací v kritickém čase, například simulaci fyziky ve 3D scéně anebo výpočty pro umělou inteligenci. Není ovšem vhodné, ačkoliv by se to mohlo zdát, implementovat zde celou aplikaci. Výkon nemusí být oproti interpretovanému kódu nutně vyšší, a naopak mohou nastat problémy s přenositelností a optimalizací pro konkrétní zařízení [3]. Tu by běžně prováděl Dalvik, zde ale záleží plně na programátorovi. Konkrétní přínosy i slabiny použití JNI budou dále rozvedeny v kapitole 2.4.

Na nejnižší úrovni je již zmíněné jádro psané v jazyce C. To zajišťuje přístup k hard-

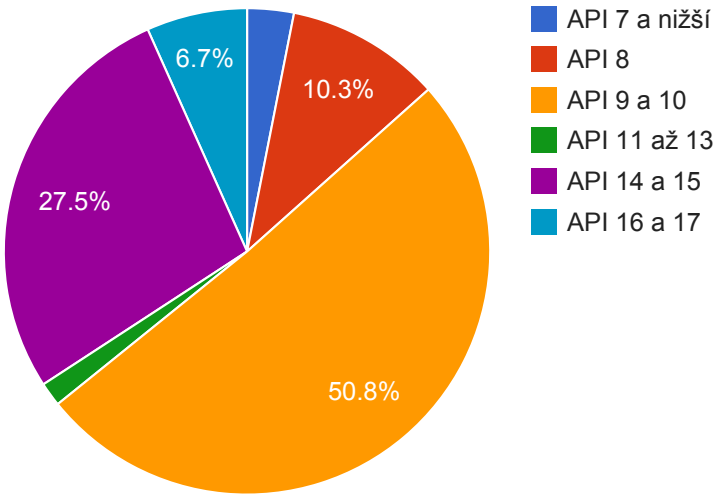
waru, multitasking, správu souborového systému a další základní úlohy. Přímo k jádru však programátor v drtivé většině případů nepotřebuje přistupovat. Funkce jádra programátor využije jen při výjimečných případech kdy vytváří systémovou aplikaci. Taková aplikace vyžaduje pro svůj běh vyšší oprávnění a běžný uživatel ji nemůže spustit. Své uplatnění ale nachází při modifikacích systému, kde jakožto předinstalované aplikace získávají vyšší oprávnění automaticky. Jelikož je Android otevřená platforma, je možné její součásti libovolně upravovat a takto vzniklý systém distribuovat.

2.2 Verze systému a cílová zařízení

TODO: revize, nové procenta a grafy

Vývoj operačního systému vede firma Google, která vydává zdrojové kódy nových verzí. Ty se ovšem přímo do zařízení dostávají jen velmi zřídka (jde o referenční modely telefonů a tabletů které si nechává vyrábět přímo Google). Typicky přebírají kód jednotliví výrobci a upravují či doplňují jej tak, aby fungoval pro konkrétní modely (nekompatibilita hrozí především u specifického hardwaru jako je 3D displej). Často také výrobce přidává dodatečné aplikace, ať už pro plné využití možností zařízení nebo čistě pro reklamní účely. Jak bude výsledný systém vypadat, jaké funkce bude obsahovat a jaká zařízení bude podporovat je plně v zodpovědnosti výrobců. Právě kvůli tomu jsou nejčastěji aktualizovány modely z vyšší cenové třídy a levnější zařízení bývají opomíjena.

Důsledkem otevřenosti Androidu je jeho roztříštěnost co se verzí týče. Aktuální rozložení verzí *API* (programátorské rozhraní, Application Programming Interface) k prosinci 2012 je na obrázku 2.1, data pocházejí z oficiálního zdroje [4]. Je jasně vidět dominance verzí 9 a 10 (v uživatelském značení jde o Android verze 2.3.x) které jsou i nyní, po téměř dvou letech od prvního vydání, stále nasazovány do nově prodávaných zařízení.



Obrázek 2.1: Rozložení verzí systému Android k prosinci 2012

Je často připomínanou pravdou, že právě zpětná kompatibilita se staršími verzemi systému znamená pro tvůrce aplikací množství práce navíc. V případě vývoje her však toto neplatí, neboť ve skutečnosti využívají jen malou část API. Zde je z programové části důležitá především podpora knihovny OpenGL ES ve verzi 2.0. Ta je nyní dostupná na více

než 90% zařízení a můžeme ji tedy považovat za standard. Mnohem důležitější je však technická vybavenost zařízení, zejména pak výkon procesoru, výkon grafického čipu a velikost jeho paměti. Zde jsou parametry často velmi rozdílné a relevantní srovnání či statistiky bohužel chybí. Nezanedbatelným parametrem jsou také rozměry displeje. Vzhledem k různým hustotám zobrazení není vhodné uvádět pouze rozlišení. Větší vypovídací hodnotu z hlediska dotykového ovládání má fyzická úhlopříčka displeje, která je v současnosti nejvíce zastoupená v rozměrech okolo 4 palců. [4]

2.3 Android při tvorbě her

OK r1

Pokud se zaměříme konkrétně na vývoj grafických her, zjistíme že hluboké znalosti systému nejsou nezbytné. Android umožňuje navrhovat velmi složitá a propracovaná uživatelská rozhraní, hernímu vývojáři však bude stačit jen nezbytné minimum. Podobně je tomu s dalšími vymoženostmi systému.

Základním stavebním kamenem každé aplikace je tzv. *Aktivita* (třída `Activity`). Zjednodušeně lze říct že Aktivita je okno aplikace, typicky zobrazované přes celý prostor displeje (s výjimkou informační lišty a případných virtuálních kláves), se kterým může uživatel interagovat. Aktivita je rodičem jednotlivých elementů uživatelského rozhraní, případných dialogů, nabídek a dalších prvků. Pro potřeby hry běžně postačuje jediná aktivita jejímž potomkem je speciální oblast do které může vykreslovat OpenGL (oblast je třídy `GLSurfaceView`). Při vytoření této oblasti systém současně spustí speciální vlákno, které má přístup k OpenGL kontextu, a může tedy provádět grafická volání. Toto vlákno je poté systémem pravidelně probouzeno tak, aby vykreslování probíhalo v rychlosti co nejbližší ideálním 60 snímkům za vteřinu. S uživatelským rozhraním systému běžící hra až na výjimky nepracuje, vše probíhá v oblasti vyhrazené pro vykreslování.

Kromě zobrazování je Aktivita důležitá také pro příjem vstupních událostí. Má možnost *přihlásit* se u systému ke sledování dotyků na obrazovce, stisků tlačítek, naklonění zařízení, případně stavů dalších sensorů. Tyto události přicházejí v odděleném vlákně které nemá možnost vykreslování. Vlákno typicky upraví vnitřní stav hry a poté se opět uspí a čeká na další událost. Platí zde samozřejmě obdobné principy jako u jiných vícevláknových prostředí. Jak vlákno pro vykreslování, tak i vlákno pro zpracování události nesmějí provádět blokující volání a neměly by provádět časově náročné výpočty. Pro takové případy nabízí Android možnost spuštění odděleného vlákna a postupy jak s takovým vláknem asynchronně komunikovat, aniž by docházelo k narušení plynulosti běhu aplikace, v tomto případě poklesu počtu vykreslených snímků za vteřinu.

2.4 Nativní vývoj

OK r1

V případě že zvolíme tvorbu hry čistě na základě Androidu bez mezičlánků typu frameworku nebo herního engine, můžeme narazit na některé nepříjemnosti. Především nemůžeme počítat s jakýmkoliv rozšířením nad možnosti OpenGL ES. Je plně v kompetenci programátora aby si připravil potřebné datové struktury a pomocné třídy. Systém jako takový sice obsahuje několik základních tříd pro popis vektorů a matic a operace s nimi, ty ale nebyly navrženy pro práci s OpenGL a dochází zde k nesrovnalostem (příkladem může být vnitřní reprezentace matice, která je jednou řádková a jindy sloupcová).

Jako jedna z výhod aplikací v jazyce Java bývá uváděna automatická správa paměti použitím tzv. *garbage collectoru*, tedy součásti virtuálního stroje, která počítá reference na každý alokovaný objekt a v případě odstranění posledního výskytu tento objekt uvolní z paměti. Ačkoliv může být tento přístup pohodlný, přináší množství rizik. Vzhledem ke složitosti operačního systému (a často i neznalostí jazyka Java) je pro programátora relativně snadné držet referenci na objekt, který již nepoužívá a o kterém předpokládá, že již byl z paměti uvolněn. Tím sice nedochází k nenávratně ztraceným blokům paměti jako při práci v nativním kódu, paměťové nároky aplikace však zbytečně rostou. Čím více paměti pak aplikace používá, tím častěji se bude garbage collector obracet právě na ni a její paměť se bude snažit uvolňovat. A právě při úklidu paměti dochází ke krátkému pozastavení všech vláken dané aplikace. Toto pozastavení běžně trvá jednotky až desítky milisekund a nemusí být pro uživatele viditelné. Zda se ale aplikace pozastaví a jak dlouho tento stav potrvá závisí na velkém množství faktorů které programátor nemá pod kontrolou. Je tedy velmi důležité zvolit vhodnou taktiku práce s pamětí a té se striktně držet. Jednou z možností je předběžné vytvoření všech objektů které mohou být potřeba s tím, že v průběhu hry již nové objekty nevznikají. Toto je však ideální stav, kterého není snadné dosáhnout. Sofistikovanější správa paměti bude popsána v kapitole **??**.

S problematikou paměti souvisí také načítání dat potřebných ve hře, zejména pak geometrie scény a textur. V případě 3D modelů opět narazíme na neexistenci jakýchkoliv pomocných tříd. Ve vývojářské komunitě navíc není úplná shoda na tom, který z formátů uložení modelů používat. Ve výsledku je opět na programátorovi, aby zvolil vhodný formát, ten nastudoval a implementoval jeho načítání. To vzhledem ke správě paměti popsané v předchozí části rozhodně není triviální úkol. (U mnohých formátů existují více či méně zdařilé implementace jejich načítání, téměř vždy ale diktují v jakých strukturách budou načtená data uložena, což nemusí být optimální.)

V případě textur je situace o něco jednodušší. Jelikož Android běžně využívá rastrová data obvyklých formátů ve svém uživatelském rozhraní, umí takové soubory přirozeně i načítat. Nedostatkem zde mohou být zvýšené paměťové nároky ve chvíli kdy se textura dekóduje ze svého formátu do interní reprezentace (velmi podobné bitmapě) a následně přesouvá do grafické paměti. V tento okamžik jsou současně alokovány dva až tři objekty reprezentující stejnou texturu, ovšem s rozdílnou vnitřní reprezentací. S velkou pravděpodobností bude opakovaně docházet ke spuštění automatického úklidu paměti. V krajních případech při použití rozměrných textur může dojít i k ukončení aplikace z důvodu překročení maximálního množství přidělené paměti.

Zmíněné slabiny nízkoúrovňového přístupu k tvorbě her jsou jen jedny z mnoha. Zcela jistě záleží na subjektivním přístupu a zkušenostech, zda se můžou rozvinout do skutečných problémů či nikoliv. Drtivá většina těchto i jiných nedostatků je samozřejmě řešitelná a vývojová prostředí nabízejí množství pomocných nástrojů. V extrémním případě může programátor zajít až tak daleko, že vytvoří vlastní framework či engine. Je však otázkou, zda by měl své síly soustředit na ladění a obcházení záludností systému nebo použít existující řešení a své síly zaměřit na konkrétní prvky, které odliší jeho hru od konkurence.

Z výše uvedeného je zřejmé, že u většiny grafických aplikací je žádoucí použít existující a ověřený kód na kterém se bude dále stavět. Není podstatné zda půjde o malou knihovnu, framework či herní engine – názvosloví zde není ustálené (v dalším textu se proto budu držet obecného pojmu *knihovna*). Důležitý je přínos pro programátora, úspora času a prostředků, možnost soustředit se na konkrétní produkt a ignorovat podružné problémy. V neposlední řadě také použití knihovny znamená přenesení odpovědnosti za část aplikace na někoho

jiného. To může znamenat zefektivnění vývojového procesu, ale také jeho zásadní zpomalení v případě že se vývoj knihovny zastaví nebo obsahuje chyby které autor nechce či nemůže opravit.

2.5 Komplexní vývojová prostředí

Jistou ochranou před hrozbou budoucích problémů je využití komerčně nabízených řešení. Dva nejrozšířenější zástupci této kategorie jsou Unity 3D a Unreal Engine, kde oba sami sebe označují za herní engine. V obou případech jde o specializovaná vývojová prostředí která se snaží upozadit programování ve smyslu psaní kódu a naopak upřednostňují přímé sestavování scény a definice chování. Vývojář může hru hrát a souběžně v reálném čase upravovat. K tomu slouží interaktivní pohled do scény, kde je možné pracovat na úrovni objektů. Samozřejmostí je množství předdefinovaných chování, fyzikálních vlastností, materiálů a často i celých objektů se všemi vlastnostmi (například celé pojízdné auto). Na skutečné programování přichází čas pouze ve chvíli, kdy je potřeba nadefinovat vnitřní logiku hry, případně vytvořit nové chování některého z objektů. V takovém případě používá engine vlastní skriptovací jazyk. Jak Unity 3D, tak i Unreal Engine jsou komplexní vývojová prostředí produkující multiplatformní aplikace a Android je pouze jedním z mnoha výstupů. Vývojář ve skutečnosti přichází s architekturou Androidu do kontaktu jen minimálně nebo vůbec. I přesto není vhodné tento přístup opomíjet, neboť s použitím právě Unity 3D vznikají v současnosti nejlépe hodnocené a po vizuální stránce nejpropracovanější hry. Záporem je zde vysoká počáteční investice, a to v řádu minimálně stovek dolarů.

Obrázek vývojového prostředí UDK / Unity ?

2.6 Knihovna Libgdx

Alternativou k placeným vývojovým nástrojům jsou menší knihovny s těsnější vazbou na systém Android. Nejpoužívanější z nich jsou AndEngine a Libgdx. První z jmenovaných se nazývá herním engine, druhý se označuje za framework. Oba jsou však k dispozici zdarma ve formě otevřených zdrojových kódů s možností komerčního i nekomerčního využití. Jelikož jsou si obě knihovny velmi podobné, soustředím se v dalším popisu pouze na Libgdx, kterou také použiji při implementaci. Zdrojem bude z velké části oficiální dokumentace [5] a také kniha ?? ???? ? [**?**]. Jejím autorem je Mario Zechner, který je současně tvůrcem knihovny Libgdx a jejím hlavním přispěvatelem.

TODO: Citovat Mariovu knihu! TODO: Říct že kapitola čerpá z dokumentace libgdx a scene2d.ui

Na první pohled vypadá knihovna Libgdx podobně jako dříve zmíněné nástroje – stejně jako ony je schopná produkovat aplikační balíčky pro více platforem. Původně však vznikala jako knihovna pro tvorbu 2D her pro Androida a až postupem času došlo rozšíření o podporu dalších platforem. Aktuálně je kromě systému Android podporovaná i tvorba webových appletů a klasických aplikací pro počítače. Jak konkrétně multiplatformní vývoj probíhá bude popsáno v kapitole 5.2.1.

Narozdíl od jiných řešení knihovna neobsahuje žádné vývojové prostředí ani předpřipravená data. Zaměřuje se více na usnadnění vývoje nabízením pomocných algoritmů a tříd. Je pak čistě na programátorovi, zda výhod knihovny využije. Díky tomu má prostor pro detailní optimalizace na nízké úrovni pro dosažení maximálního výkonu hry, stejně jako se může pohybovat na vyšší úrovni, která umožní velmi rychlé prototypování a vytvoření

Obrázek 2.2: Hra Shadowgun od brněnského studia Madfinger Games, vytvořená v Unity 3D

funkční aplikace. Mezi přednosti Libgdx patří vysoko- i nízkoúrovňové rozhraní pro 2D a 3D grafiku, podpora fyziky, propracovaný systém uživatelského rozhraní, načítání množství formátů, práce se zvuky a mnoho dalších. V neposlední řadě se pak snaží stírat rozdíly mezi jednotlivými zařízeními a verzemi systému Android.

Samozřejmostí je řešení výše zmiňovaných nedostatků při vývoji bez pomocných knihoven. Velká síla Libgdx spočívá v tom, že je implementována částečně přímo v Javě a částečně v nativním kódu. Tím dosahuje vyššího výkonu zejména u již zmiňovaných fyzikálních simulací, ale především má plnou kontrolu nad alokací paměti a jejím následným uvolňováním. Automatické uvolnění přirozeně probíhá pouze v části aplikace bežící ve virtuálním stroji, zatímco v nativním kódu je práce s pamětí plně v režii programátora. Vhodným propojením těchto dvou částí lze bezpečně alokovat paměť kterou garbage collector *neuvidí*, a dokonce dosáhnout stavu kdy framework jako takový uvnitř hlavní herní smyčky žádnou paměť nealokuje. To samozřejmě nemůže úplně zamezit spouštění úklidu paměti, příčinou však bude běh ostatních aplikací a systému, nikoliv samotné hry. V praxi se pak paměť uklízí jen velmi výjimečně, probíhá velmi rychle (garbage collector spravuje jen velmi malé množství paměti), a hra tedy může běžet plynule.

2.6.1 Struktura aplikace

Vstupním bodem aplikace je instance třídy `Application`, která obsahuje základní herní smyčku. Ta je volaná systémem právě na dříve zmiňované frekvenci blízké 60 snímkům za

sekundu. Při tvorbě grafických aplikací se často využívá oddělené volání pro aktualizaci scény a její vykreslení, Android však takové rozdělení nepodporuje (z důvodu návaznosti na systémový element s OpenGL kontextem, popsaný v kapitole 2.3. Je samozřejmě možné aktualizaci scény a její vykreslení oddělit ručně, například v případě že není nutné se scénou manipulovat příliš často, ale chceme zachovat plynulost vykreslování. Typické využití může být u tahových her, kdy se herní logika provádí jen v okamžiku tahu, zatímco zobrazování probíhá v každém snímku, například kvůli probíhajícím animacím. Je samozřejmě možné optimalizovat ještě více, animace nepoužívat a vykreslovat jen ve chvíli, kdy se scéna změní. V takovém případě dochází k maximální úspoře energie. Aplikace ale může působit značně statickým dojmem a pro akční hry tento přístup není vhodný vůbec.

Zatímco samostatná třída aplikace je vhodná jen pro nejtriviálnější aplikace, u složitějších bude žádoucí rozdělit aplikaci do několika částí. Zde knihovna nabízí členění do tzv. *obrazovek* (potomci třídy `Screen`), které mají vlastní životní cyklus. Hlavní třída aplikace pak obstarává pouze přepínání mezi jednotlivými obrazovkami. Ty lze považovat za oddělené logické celky a často je možné je i samostatně vyvíjet, což bude přínosem zejména pro vícečlenný tým vývojářů. Obrazovka jako taková však žádné vyšší funkce nemá a kromě zmíněného zpřehlednění kódu nabízí jen možnost jednoduchého vykreslování texturovaných obdélníků (tzv. *sprite*).

2.6.2 Zpracování vstupních událostí

Vstupní události jsou závislé na aktuální platformě, a proto jsou v Libgdx skryty pod jednotným rozhraním, kdy konkrétní implementaci doplňuje sama knihovna právě podle aktuálního prostředí. Programátor díky tomu nemusí řešit, zda bude aplikace ovládaná dotykem či myší, klávesnicí nebo hardwarovými tlačítky mobilního telefonu nebo třeba ovladačem typickým pro herní konzole. Je zřejmé, že všechny typy událostí nebudou podporovány na všech platformách. Knihovna však zavádí vhodné substituce a sjednocení, kde například kliknutí na dotykovém displeji odpovídá kliknutí levého tlačítka myši. Na vyšší úrovni je pak možné detekovat celá gesta jako chycení a táhnutí nebo zoom.

Přirozeně lze definovat i vlastní zpracování vstupu. Příkladem může být křížový ovladač vykreslovaný na obrazovku v případě, že aplikace běží na dotykovém zařízení. Vnitřně pak může takový ovladač *překládat* dotykové události na události stisku klávesy. Ty pak aplikace obslouží jako běžné události a programátor tak velmi snadno získává podporu pro dotyková zařízení, počítačové klávesnice i konzolové ovladače.

TODO: lépe přeložit "vstupní procesor"?

Toto chování je umožněno tzv. *vstupními procesory* (třída `InputProcessor`). Vstupní procesor je třída obsahující kód pro zpracování vstupních událostí. V okamžiku zpracování v procesoru už je známo o jaký typ události jde. Procesor jako takový tedy žádnou detekci neprovádí. Typicky obsahuje referenci na objekty aplikační logiky a s těmi nějak pracuje. Na konci svého běhu vrací návratovou hodnotu podle toho, zda událost zpracoval (a měla by tedy zaniknout) anebo ne (což nutně neznamená že na událost nijak nereagoval).

Vstupních procesorů může být v aplikaci několik, aktivní je ale vždy pouze jeden. Pro komplexní zpracování vstupů se proto zavádí tzv. multiplexer (třída `InputMultiplexer`), který představuje kontejner vstupních procesorů, přičemž rozhraní procesoru také sám implementuje. Při příchodu události ji pouze předává jednotlivým procesorům do té doby, dokud ji některý z nich nezpracuje.

Tím je možné vytvářet kaskády vstupních procesorů ať už pro zmíněné sjednocení vstupů napříč platformami anebo pro obsluhu událostí ze složitějších scén. Příkladem může

být dotykem ovládaná hra využívající doplňkové ovládací prvky na obrazovce. Vhodným řešením by mohl být multiplexer zastřešující vstupní procesor pro doplňkové prvky (přicházející na řadu jako první, tedy může událost zpracovat a nepustit dál) a procesor pro ovládání samotné hry. Takto oddělené zpracování vede nejen k přehlednějšímu kódu, ale i k jednodušší obsluze událostí ve chvílích, kdy existuje více prvků souběžně či postupně reagujících na stejný vsutp.

2.6.3 Tvorba uživatelského rozhraní

Velmi silnou součástí Libgdx z hlediska tvorby uživatelského rozhraní je knihovna Scene2D.ui (TODO: citace), která je použitelná i samostatně v obecných projektech v jazyce Java. Jejím základním prvkem je tabulka (třída `Table`), pro kterou se definují jednotlivé buňky a jejich rozdělení do řádků. Chování je podobné tabulkám vytvořeným v jazyce HTML v kombinaci s CSS. Buňkám lze nastavovat pevné či poměrné rozměry, vnitřní okraje, pozadí a některé další atributy. Je také možné buňky v rámci jednoho řádku slučovat; vertikální slučování podporované není, ale lze jej dosáhnout jinou cestou. Specifickým parametrem buněk je pak *rozpínavost* v rámci řádku a v rámci sloupce. Vhodnou kombinací je možné vytvořit dynamickou, tzv. *fluidní* tabulku, která se bude přizpůsobovat svému obsahu i rozměrům zobrazovací plochy.

Nad tabulkami jsou dále vystavěny základní prvky uživatelského rozhraní jako tlačítka, dialogy, pole pro zadání textu, obrázky, bloky textu, posouvatelné panely a další. Využití tabulky lze demonstrovat na tlačítku. To je ve skutečnosti tabulka o třech řádcích a třech sloupcích, kde prostřední buňka obsahuje textový popisek čí obrázek. Všechny jeho buňky pak mají vhodně nastavené pozadí, čímž lze vytvořit například i tlačítka se zakulacenými okraji. Samozřejmě si opět zachovává schopnost přizpůsobit rozměry svému obsahu i okolí. Složitější prvky vznikají obdobně, přičemž často využijí i možnost tabulky do sebe libovolně vnořovat.

Prvky rozhraní také nemusejí být nutně statické. Každý prvek může mít nadefinováno své vlastní chování a v rámci scény se chovat zcela autonomně. Současně knihovna nabízí množství předdefinovaných efektů, které lze prvkům přiřazovat. Typicky jde o geometrické transformace nebo práce s barvami či průhledností. Efekty dostávají jako parametr dobu trvání a krajní hodnoty, mezi kterými v čase interpolují. Podobně jako při zpracování událostí i zde existují speciální efekty chovající se jako kontejnery efektů běžných. Ty umožňují nechat efekty probíhat současně anebo v řadě za sebou. Samozřejmostí je opět libovolné vnořování, čímž lze vytvořit velmi komplexní chování.

Tento odstavec je divný, protože nevím v jakém rodě o Stage mluvit Kořenovou komponentou uživatelského rozhraní je Stage překlad? (třída `Stage`). Představuje kontejner na výše popsané prvky rozhraní. - zabírá celou plochu - řeší kameru - řeší z-order (podle pořadí přidání nebo explicitně)

obrázek - menu UI se zapnutým debugem

popsat skiny?

popsat bitmapové fonty?

scene2d.ui umí dělat i hry

tmp Libgdx také obsahuje pokročilou práci s texturami, kdy programátora odstiňuje od zmiňovaného převodu mezi formáty a přesouvání do grafické paměti. Navíc přidává funkce pro práci s *texturovými atlasy* (několik samostatných textur seskládaných v jediném souboru), včetně možnosti jejich automatického vygenerování. K úspoře poté dochází pře-

devším při vykreslování, které je v případě jediné textury mnohem rychlejší než při použití jednotek či desítek menších textur.

2.7 Distribuce aplikací

- podepisování - obfuskace? licencování? root? - free markety - vývojářský účet u Google - Google Play - nemožnost beta testerů nebo obecně dělení do skupin - featuring na hlavní straně / hlavní straně kategorie

Kapitola 3

Ovládání skutečného letadla

TODO: revize

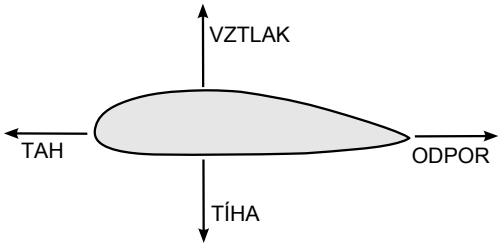
Tato kapitola popíše principy ovládání letadel a leteckých her. Shrne existující metody a představí novou metodu zaměřenou na ovládání akčních leteckých her v mobilních zařízeních.

Před návrhem herního ovládání letadla je vhodné se seznámit s tím, jak létají skutečná letadla. Více než o konkrétní rovnice zde půjde o základní principy díky kterým se letadlo udržuje ve vzduchu. Rovnice by totiž nebylo možné vyčíslit bez znalosti konkrétních koeficientů, které jsou pro každou konstrukci odlišné. I s nimi by však bylo exaktní matematické popsání letícího letadla velmi obtížné. Následující kapitoly budou čerpat především z [6] a [7].

3.1 Síly působící za letu

TODO: revize

Vyjdeme nyní z takzvaného *ustáleného vodorovného letu*. V tuto chvíli se letadlo pohybuje konstantní rychlostí s nulovým zrychlením, nestoupá, neklesá a není v náklonu. Působí na něj síly vyznačené na obrázku 3.1. Jsou jimi vzlak, gravitační tíha, tah a odpor. Při ustáleném letu jsou tyto síly v rovnováze. Ne však všechny navzájem, jak je někdy mylně uváděno, ale pouze dvojice sil působících proti sobě. Pro ilustraci můžeme použít namísto celého letadla pouze křídlový profil, na kterém jsou veličiny více zřejmé.



Obrázek 3.1: Síly působící na letadlo při ustáleném vodorovném letu

Nejdůležitější silou je vztlak. Ten působí kolmo vzhůru od křídla a při vodorovném letu vyrovnává tíhu letadla. Dochází k němu díky typickému profilu křídla, kdy na jeho spodní straně vzniká vyšší tlak vzduchu a současně na horní se tlak vzduchu snižuje. Pro zjištění

celkového vztlaku křídla je potřeba znát množství parametrů, jako je hustota prostředí či rychlost pohybu křídla. Především ale záleží na koeficientech popisujících tvar křídla, jeho plochu, rozpětí a tvar jeho profilu. Pokud by nás zajímal celkový vztlak letadla, bylo by samozřejmě nutné započítat dílčí vztlakové síly na celém jeho povrchu.

Další z hlavních sil je tah motoru. Ten je tvořen jednou či více vrtulemi. Lopatky vrtule svým tvarem ženou vzduch za sebe (tedy směrem na trup a křídla letadla), čímž vzniká síla táhnoucí vrtuli, a tím i celé letadlo, dopředu. Tažná síla opět závisí na množství parametrů které popisují stavbu vrtule, počet lopatek a jejich aerodynamické vlastnosti, a samozřejmě také na rychlosti jakou se vrtule otáčí. S vyšší rychlostí stoupá tah a současně se zvyšuje rychlost proudění vzduchu v oblasti za vrtulí. Tím se nepřímo zvyšuje vztlak křídel, ale také se mění citlivost ovládacích prvků letadla.

Gravitační síla působící na letadlo je zřejmá a vždy směřuje ke středu Země.

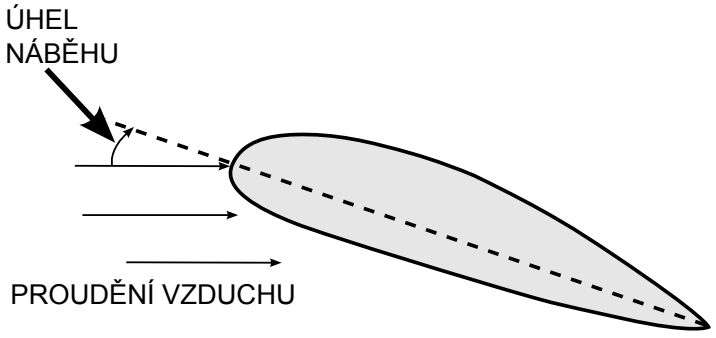
Poslední silou je odpor který klade vzduch letícímu letadlu a působí vždy v opačném směru než je směr jeho pohybu. Je složený ze dvou hlavních složek. Jednou je odpor způsobený tvarem křídla, potažmo celého letadla, a jeho pohybem v prostředí. Druhou složkou je odpor vznikající na koncích křídel. Jak již bylo zmíněno, u horní části křídla se nachází podtlak, zatímco u spodní části je naopak přetlak. V místě kde je křídlo spojeno s trupem letadla se tyto tlaky *nepotkávají*, na opačném konci křídla však ano. Dochází k jejich přirozenému vyrovnávání a vznikají víry, které pokrucují proudění vzduchu a mění směr vztlakové síly křídla. Její pomyslný vektor pak není kolmý ke křídlu, ale mírně sklopený, čímž přispívá k velikosti odporové síly. Čím vyšší vztlak křídlo tvoří, tím více odporové síly současně vzniká. Obecně je pro zjištění odporové síly důležitá rychlost pohybu a charakteristika prostředí, především pak samotný tvar křídel a celého letadla.

Výsledné chování letadla záleží vždy na poměru zmíněných sil. Pokud by křídla netvořila dostatečný vztlak (z důvodu konstrukce nebo nízké rychlosti pohybu), letadlo se nevznese, případně spadne. Je však nutné si uvědomit, že s výjimkou tahu motoru nemá pilot letadla možnost žádné z uvedených parametrů změnit, neboť jsou dány konstrukcí letadla.

3.2 Ovládací prvky a manévry

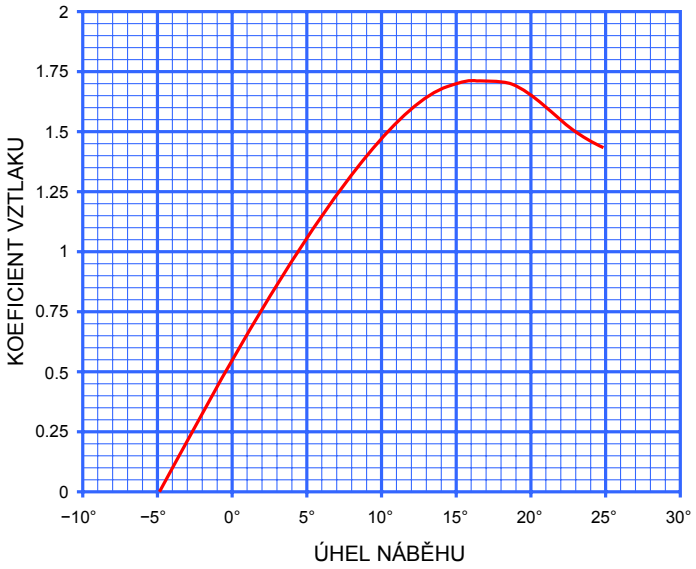
TODO: revize

Pro popis jiného než vodorovného letu je potřeba zavést nový pojem, a sice *úhel náběhu*. Jde o úhel, který svírá tětiva profilu křídla s proudem vzduchu, který na ni dopadá. Úhel náběhu je vyznačený na obrázku 3.2.



Obrázek 3.2: Úhel náběhu křídla

Navzdory časté představě ve skutečnosti neplatí, že by směr letu letadla byl vždy totožný s osou jdoucí jeho trupem. Při dostatečné rychlosti může letadlo letět vodorovným letem, přičemž trup bude skutečně souběžný s dráhou letu. Vztlak křídla bude v tomto případě dostatečný právě díky vysoké rychlosti proudění. Pokud je ale cílem nižší letová rychlost (například z důvodu úspory paliva), bude pro zachování vodrovného směru nutné aby pilot zvýšil úhel náběhu křídel (použije k tomu výškové kormidlo – viz dále). Tím zvýší i jejich koeficient vztlaku a zamezí klesání letadla. V této chvíli se letadlo pohybuje vodorovně, ale jeho příď je výš než záď a míří tedy mírně nahoru (úhel o který je letadlo nakloněno odpovídá úhlu náběhu). Přibližný vztah mezi úhlem náběhu a vztlakovým koeficientem křídla je znázorněn na obrázku 3.3. Přesný tvar průběhu je opět dán konstrukčními vlastnostmi konkrétního křídla.



Obrázek 3.3: Závislosti vztlakového koeficientu na úhlu náběhu u neidentifikovaného křídla. Zdroj: Wikimedia Commons na základě dat z proframu XFOIL

Nejvyšší hodnota vztlakového koeficientu je vidět v oblasti kolem 17° (opět je nutné připomenout že tento průběh je jen jedním z mnoha možných a například sportovní letadla mohou efektivně využívat i vyšších úhlů náběhu). Při dalším zvyšování úhlu náběhu se mění proudění kolem horní části křídla z *laminárního* (proudnice jsou rovnoběžné a na pohled vypadají jako by byly rozdělné do vrstev) na proudění *turbulentní* (proudnice se trhají a vzájemně prolínají, vznikají vzdušné víry). V důsledku vznikajících turbulencí klesá vztlak křídla a současně roste jeho povrchové tření, čímž stoupá i jeho odpor. Na hranici blízké 25° pak křídlo ztrácí vztlak úplně a letadlo se začne propadat. Této situaci se často říká *přetažení*.

Pilot k řízení letadla používá takzvané *primární ovládací prvky*. Jsou jimi klapky na křídlech a výškové kormidlo, obojí ovládané kniplem, a dále směrové kormidlo ovládané pomocí pedálů. Všechny tyto ovládací prvky ve skutečnosti plní jedinou činnost, a sice že mění úhly náběhu jednotlivých částí letadla. Přestože jsou křídla pevně spojená s trupem, jsou na nich pohyblivé klapky. Stejně tak vodorovné výškové i svislé směrové kormidlo má tvar křídla, jehož část u odtokové hrany je pohyblivá.

Pokud nyní přestaneme dosud popisované síly vztahovat k celému letadlu, ale budeme je uvažovat jako samostatné síly vznikající vždy na konkrétním křídle (za křídla zde považujeme i výškové a směrové kormidlo), můžeme snadno popsat základní letecké manévry.

Proces stoupání či klesání pilot reguluje nastavením klapek výškového kormidla. V případě stoupání klapky zvedne, čímž sníží úhel náběhu kormidla a tím i jeho vztlak. Záď letadla začne klesat, úhel náběhu hlavních křídel letadla se začne zvyšovat a s ním i jejich vztlak, potažmo vztlak letadla jako celku. Při klesání je situace analogická, pouze se veličiny pohybují opačným směrem.

U zatáčecího manévru často panuje přesvědčení, že k jeho provedení slouží směrové kormidlo, od kterého se už podle názvu očekává že bude řídit směr letu. Z předchozích odstavců již můžeme tušit že tomu tak není. Pokud by pilot při vodorovném letu použil směrové kormidlo, záď letadla by se sice vůči směru letu natočila, samotný směr letu by se však změnil jen málo. Tím jak se osa letadla natáčí vůči směru letu současně roste proudění kolem křídla které je více vystaveno přímému směru letu. Tím na křídle vzniká větší vztlak a letadlo začíná podélně rotovat, ačkoliv je knipl stále ve výchozí poloze.

Správný průlet zatáčkou probíhá v náklonu. Pilot nejdříve zatočením kniplu nakloní klapky na křídlech (které se narozdíl od výškového a směrového kormidla pohybují navzájem protichůdně) tak, že na jednom z křídel se vztlak zvýší, zatímco na druhém se sníží. Letadlo tedy letí v náklonu a vztlak vznikající na křídlech nepůsobí kolmo vzhůru, ale kolmo k horním stranám křídel (vztlak vzniká stále stejně, pouze jeho směr již není přímo opačný ke gravitační síle). Můžeme tedy vztlakovou sílu pomyslně rozložit na skutečný vztlak držící letadlo ve vzduchu a složku vodorovnou, která působí jako dostředivá síla a nutí letadlo skutečně zatáčet. Směrové kormidlo při průletu zatáčkou slouží regulaci vzájemné polohy přídě a zádi letadla. Některá letadla mají totiž tendenci v důsledku různě velkého odporu na křídlech (například pokud letadlo automaticky nenastavuje klapky do vzájemně opačných poloh) *sklouzávat* po přídi nebo zádi do středu pomyslného kužele po jehož vnitřní stěně letadlo zatáčkou prolétává. Výškové kormidlo může zatáčecí manévr doplnit na vzestupný či sestupný.

3.3 Stabilita letadla

TODO: revize

Každé letadlo má svou konstrukcí dané stabilizační schopnosti, které se uplatňují aniž by pilot zasahoval do řízení. Stabilitu dělíme na statickou a dynamickou.

Statická stabilita popisuje reakci letadla ve chvíli kdy dojde k narušení jeho dosavadního směru letu například poryvem větru. U staticky stabilního letadla vznikne opačná síla snažící se letadlo vrátit do původní polohy. Staticky neutrální letadlo bude pokračovat v pozměněném směru a u staticky nestabilního letadla se dokonce objeví síla působící spolu s rušivým vlivem, která odklon od původního směru ještě více posílí.

Dynamická stabilita popisuje stabilitu letadla v dlouhodobém horizontu. Dynamicky stabilní letadlo sice bude při vyrovnávání svého směru oscilovat kolem rovnovážné polohy (vodorovného letu), ale po čase se v něm ustálí. Dynamicky neutrální letadlo bude oscilovat s pořád stejnou sílou a dynamicky nestabilní letadlo bude od rovnovážné polohy dokonce divergovat.

Letadlo které je staticky i dynamicky stabilní je většinou snáze ovladatelné a vhodné pro začínající piloty. Pokud je však tendence letadla dosáhnout stabilní polohy příliš velká, může to být ke škodě, neboť bude k ovládání letadla nutné vynaložit vyšší sílu. Ač se to

nemusí zdát, není vždy cílem zkonstruovat dokonale stabilní letadlo a v některých případech může být vhodné pokud se letadlo chová neutrálně nebo dokonce nestabilně.

Kapitola 4

Návrh dotykového ovládání a letecké hry

4.1 Herní ovládání letadla

TODO: revize

V následujících podkapitolách bude popsáno zapojení výše vysvětlených mechanik letu do leteckých her. Zaměřím se přitom především na takzvané *causual* hry, tedy oddechové hry které od hráče nevyžadují vysokou úroveň zkušeností. U her a programů profilujících se jako simulátory je zřejmá snaha co nejvíce se přiblížit reálné předloze jak u letové dynamiky, tak i z hlediska ovládání letadla. I přesto lze u simulátorů pozorovat využití některých zjednodušení zejména z oblasti ovládání.

4.1.1 Chování letadla

TODO: revize V předchozích kapitolách byla zmíněna složitost numericky přesného výpočtu všech sil působících na letadlo. Nejen že by bylo potřeba znát až desítky reálných koeficientů pro každý jeden typ letadla, ale požadavky na výpočetní výkon by byly velmi vysoké (tím spíše pokud uvažujeme mobilní zařízení). Proto se v případě leteckých simulátorů a her používají matematické modely s různou úrovní věrnosti fyzikální předloze [8].

Je pochopitelné, že v případě simulátorů pro prvotní výcvik pilotů je maximální věrnost simulace nezbytná. V případě leteckých her se však ukazuje, že příliš reálně ovládané letadlo je spíše na obtíž. Hráč causual her neumí a často ani nechce umět plnohodnotně ovládat letadlo. Jeho prvotním cílem je splnění herních úkolů, typicky sestřelení protivníka nebo let na dané místo. Letadlo je v této kategorii her pouze prostředkem k dosažení cíle, nikoliv cílem samotným.

Do matematických modelů se proto vkládají zjednodušující předpoklady, jako například stabilita letadla při zatáčení. To potom nemá tendence v zatáčce sklouzávat do jejího středu (tento jev byl detailněji popsán v kapitole 3.2). Častým zjednodušením je také nemožnost dosáhnout bodu přetažení (opět viz kap. 3.2). Pomyslný omezovač nedovolí další zvyšování úhlu náběhu a letadlo přetažení nikdy nedosáhne. Tím odpadne část simulace, kdy se letadlo ocitá ve vzduchu s minimálním nebo žádným vztlakem a mělo by začít padat. Jelikož však běžný hráč nemá nejmenší ponětí o úhlu náběhu a jeho důsledcích na let, je tohoto zjednodušení často využíváno. Nevýhodou tohoto přístupu může být pro hráče viditelné omezení manévrovacích možností. Pokud se ale stále držíme v kategorii her které se nesnaží býti skutečnými simulátory, není tento problém nijak závažný. Navíc lze vhodným

návrhem hry takové omezení ospravedlnit, například použitím grafického modelu letadla který v hráči vyvolá dojem těžkopádnosti a nízkého výkonu. Od takového letadla potom podvědomě nebude očekávat velké manévrovací schopnosti a snáze pochopí, když narazí na jejich hranice, které ve skutečnosti slouží primárně ke zjednodušení celého výpočtu.

Podobně lze mlčky předpokládat existenci nějakého inteligentního řízení, které hráči asistuje. To může kontrolovat další ovládací prvky letadla sloužící zejména ke zjednodušení práce pilota v některých specifických situacích. (Jde mimo jiné o tzv. *sekundární ovládací prvky*, jejichž popis je však nad rámec této práce.) Takový pomocný systém je zcela jistě reálný a můžeme na něj nahlížet jako na jistou formu autopilota.

Z pohledu ovládání letadla se hry v drtivé většině omezují výhradně na klapky na křídlech a výškové kormidlo. Pokud uvažujeme zmíněné *asistované řízení*, lze těmito dvěma prvky letadlo plnohodnotně (stále pouze z hlediska hry, nikoliv simulátoru) ovládat. V některých případech bývá doplněna i možnost ovládání směrového kormidla. U něj však nastává časté nepochopení jeho funkce a bývá implementováno jako možnost zatáčení letadla bez náklonu při zachování vodorovného letu. Takové letadlo pak zatáčí podobně jako například ponorka. Je otázkou, zda toto pramení z neznalosti vývojáře hry anebo zda jde o objednávku samotných hráčů. Průlet zatáčkou je totiž při použití *nepochopeného* směrového kormidla mnohem snazší, dává hráči větší kontrolu nad letadlem a přirozeně snižuje požadavky na hráčovu zručnost. Ve výsledku záleží na herním vývojáři a na návrhu a zaměření hry, zda se bude držet reálného (byť zjednodušeného) ovládání letadla anebo se podvolí poptávce po snadno ovládaném letadle, přestože by tím popíral fyzikální zákonitosti.

V neposlední řadě u většiny leteckých her chybí možnost regulovat tah motoru. To je veličina se kterou pilot skutečného letadla v průběhu letu a především pak manévrů aktivně pracuje. Je ale také jednou z veličin ovlivňujících celkový vztlak letadla a při neznalosti všech důsledků by pro hráče bylo snadné ztratit nad letadlem kontrolu. Stejně tak se často ignoruje dříve zmiňovaná změna citlivosti ovládacích prvků letadla v závislosti na tahu motoru. Obvyklým řešením bývá opět předpoklad existence nějakého palubního systému který tah motoru automaticky reguluje. Ve výsledku bývá v průběhu letu tah konstantní a k jeho změnám dochází pouze při vzletu nebo přistávání. Často také informace o otáčkách motoru úplně chybí a jedinou zpětnou vazbou hráči je zvuk který motor vydává.

4.1.2 Vstupní metody

TODO: revize

Pokud se podíváme na letecké hry pro počítače, případně pro herní konzole, jsou ovládací možnosti vcelku omezené. Prakticky vždy se omezují na ovládání ve dvou osách, kdy pohyb v jedné ose řídí klapky na křídlech letadla a pohyb v druhé ose ovládá výškové kormidlo. Subjektivně je asi nejméně pohodlnou a současně nejméně přesnou metodou ovládání klávesnice počítače. Reakce na řízení jsou z povahy kláves skokové a nelze ovlivňovat jejich intenzitu. Pozitivem použití klávesnice může být velké množství kláves, které můžou ovládat další prvky letadla. Tyto možnosti však využijí spíše simulátory.

O mnoho lepší kontrolu nabízí joystick, případně jeho zmenšená varianta na ovladačích současných herních konzolí. Ovládání je opět dvouosé, ovšem s možností plynule regulovat intenzitu. Nejdůležitější vlastností je ale podobnost se skutečným řízením letadla pomocí kniplu. To umožňuje především v případě joysticku mnohem vyšší vtáhnutí hráče do hry. Nevýhodou může být omezený počet doplňkových ovládacích prvků, typicky je na ovladači pouze několik málo dalších tlačítek. Pokud by hra nabízela možnost detailnějšího ovládání

letadla (například vysouvání podvozku nebo vypouštění světlic pro odlákání nepřátelských střel), mohl by to být problém.

Rozšíření výkonných mobilních zařízení umožnila příchod nových metod ovládání. Můžeme ignorovat generaci mobilních telefonů s klávesnicí, jelikož ty ještě neměly dostatek výkonu a k rozšíření leteckých her u nich nedošlo, a zaměřme se pouze na zařízení s velkým dotykovým displejem. Taková zařízení často nemají hardwarová tlačítka nebo taková tlačítka slouží systému a hra je pro své potřeby nemůže využít. Novým prvkem je ale zapojení sensorů. Konkrétně gyroskopu a akcelerometru, které sledují orientaci, respektive zrychlení zařízení v trojrozměrném prostoru. Hráč ovládá letadlo nakláněním zařízení přičemž se typicky neuvažuje vodorovná osa, natočení do stran (pohyb podobný zatáčení volantem) ovládá klapky letadla a naklonění od sebe či k sobě ovládá výškové kormidlo.

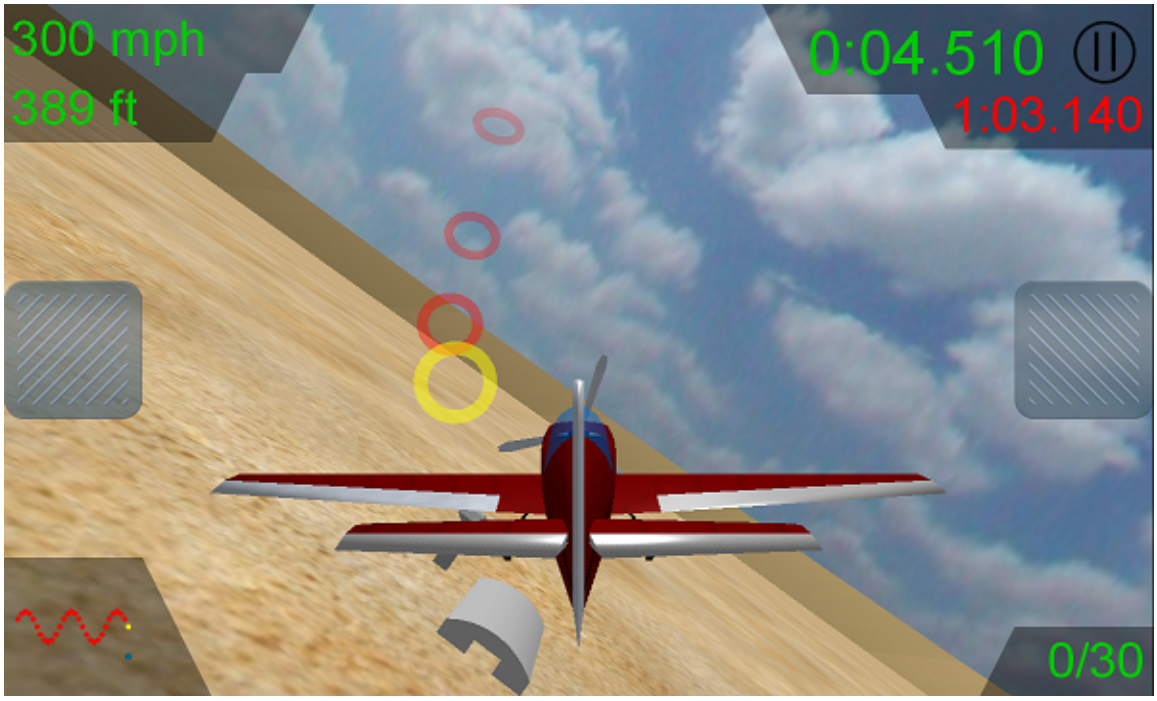
Ovládání her pomocí sensorů se rychle rozšířilo pravděpodobně díky své inovativnosti a v současnosti si udržuje přinejmenším u leteckých her většinový podíl. Má však také několik nedostatků. Asi nejzásadnějším je výchozí pozice zařízení. Při používání sensorů je potřeba stanovit referenční bod ke kterému naměřené hodnoty vztahujeme. Častým řešením je nastavit bod na začátku každého letu, poté se již měnit přirozeně nemůže. Problém nastává ve chvíli, kdy se hráč hrající hru sám pohybuje ve skutečném světě. Může jít například o houpání se v křesle anebo přesun v nějakém dopravním prostředku. Hry se v takovém případě stávají nehratelnými a zatím se neobjevila metoda, která by uměla tento jev odstranit.

Druhý nedostatek stojící za zmínku je změna úhlu pohledu při pohybu se zařízením. Displeje mají většinou relativně velký pozorovací úhel, i tak ale může docházet ke snižování viditelnosti pokud je zařízení až moc přikloněno, respektive odkloněno od pohledu hráče. Řešením může být vhodná volba citlivosti tak, aby stačilo zařízení naklánět jen v *pohledově bezpečném* rozsahu. Stejně tak může být problémové i otáčení displeje. Přestože zde se úhel pohledu nemění, dochází k otáčení celého obrazu včetně uživatelského rozhraní. Na tuto skutečnost je nutné myslet při návrhu hry tak, aby uživatel špatnou čitelností prvků rozhraní nepřicházel o důležité informace.

Dalším problémem je relativně pomalá odezva sensorického vstupu a tvar jeho průběhu. Zpoždění není zásadní, ale je pozorovatelné. Hru je navíc možné navrhnout tak, aby pro hráče bylo pochopitelné. Při letu s velkým těžkopádným letadlem asi nikdo nebude očekávat reakce v řádu zlomků vteřiny. U rychlých a akčních her by však toto mohlo být omezující. Průběh signálu vznikajícího v sensoru navíc není ideální a běžně obsahuje výkyvy a šum. Je proto nutné jej filtrovat, čímž může docházet ke ztrátě přesnosti.

Poslední slabinou sensorového ovládání je společenská vhodnost. S tím jak se herní zařízení stala mobilními se rozšířila i místa kde může člověk hry hrát. Ne vždy a všude bude okolí tolerovat hráče který před sebou mává mobilním telefonem či tabletem. Vývojář sice nemůže ovlivnit na jakých místech budou hráči jeho hru hrát, může se však přinejmenším zamyslet kdo bude cílovou skupinou a nakolik jsou členové této skupiny ochotní se ve společnosti tímto způsobem zviditelňovat.

Z pohledu mobilních zařízení je asi nejzásadnější možnost ovládání dotykem. Ta se neomezuje pouze na tlačítka a polohovací páčky a je zde možné navrhnout téměř libovolné ovládací prvky. K tomu lze využít položení prstu na displej, který odpovídá stisku klávesy, a může tedy řídit dvoustavové veličiny. Současně je možné sledovat pohyb jednoho či více prstů v nějaké předem dané oblasti a získávat plynulé změny hodnoty (například vzdálenost bodu dotyku od referenčního bodu). Dalším velmi silným prostředkem je vzájemná poloha



Obrázek 4.1: Screenshot hry Race Pilot ovládané pomocí sensorů. Šedé čtverce po stranách ovládají směrové kormidlo, které je však implementováno nesprávně

dvou a více bodů dotyku.

Ve srovnání se sensorickým ovládáním netrpí zmiňovanými nedostatky jako je nutné natáčení a naklánění zařízení a nepoužitelnost v případě že je hráč v pohybu. Při návrhu dotykového ovládání je však nutné počítat s tím, že hráč může svými prsty překrývat značnou část displeje. To se může projevit zejména u displejů s menší úhlopříčkou. Stejně tak je vhodné se při návrhu ovládání zamýšlet nad tím, kdo bude typickým hráčem hry. Zcela jistě bude rozdíl mezi prstem dítěte a dospělého jedince, navíc s přihlédnutím k vykonávané profesi, a je nutné tomuto ovládací metodu přizpůsobit.

Přestože se zdá že právě dotykové ovládání nabízí široké možnosti pro inovace, herní vývojáři je zatím spíše nevyužívají. V případě leteckých her drtivě převládá využití sensorů, případně dotykového ovladače na displeji – bodu který hráč dotykem posouvá a simuluje tak pohyb joysticku.

4.2 Nově navrhovaná metoda ovládání

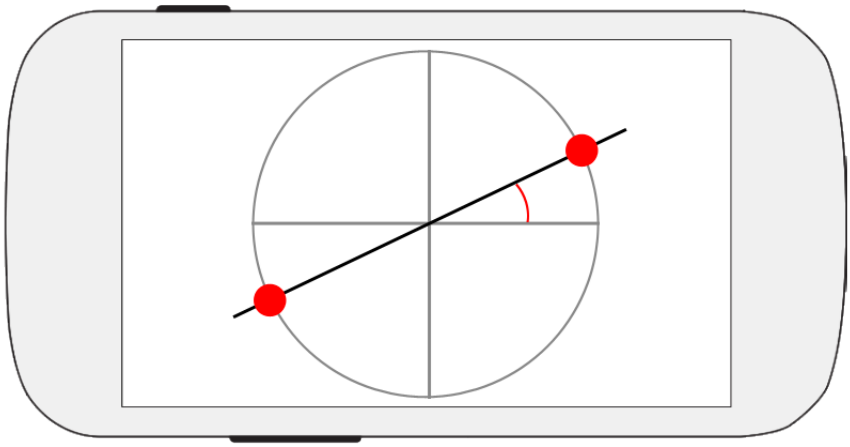
TODO: revize

Při uvažování o inovativním přístupu k ovládání letadla jsem téměř okamžitě vyřadil vstup pomocí sensorů. Důvody zmíněné v předchozí kapitole mi přijdou příliš omezující. Současně v této oblasti nejsou velké možnosti k dalšímu rozšiřování tak, aby pro hráče zůstala zachovaná pohodlnost a intuitivnost ovládání.

Navrhované ovládání tedy bude dotykové. Tím bude možné jej používat kdekoliv a jeho hlavními přednostmi bude vysoká přesnost a rychlá odezva. Díky tomu na něm půjde vystavět i velmi rychlou a akční hru. Oproti existujícímu dotykovému vstupu by mělo hráči

nabídnou větší vtažení do hry a snad i navodit pocit jisté jedinečnosti, tedy že hráč používá něco nového a neobvyklého. Proti této myšlence jde částečně požadavek na nízké znalosti a zkušenosti. Hráč by měl princip ovládání pochopit prakticky ihned, v nejlepším případě pak bez jakékoliv nápovědy nebo předchozího proškolení. Současně musí být ovládání maximálně pohodlné. To totiž opět není cílem hry, ale pouze prostředkem k jeho dosažení. V ideálním případě by mělo být pro hráče natolik přirozené, že nad ním nebude muset přemýšlet či ještě lépe vůbec si uvědomovat jeho existenci. Pochopitelně jej také nesmí omezovat a mělo by postihnout všechny podstatné funkce ovládaného objektu, v tomto případě letadla.

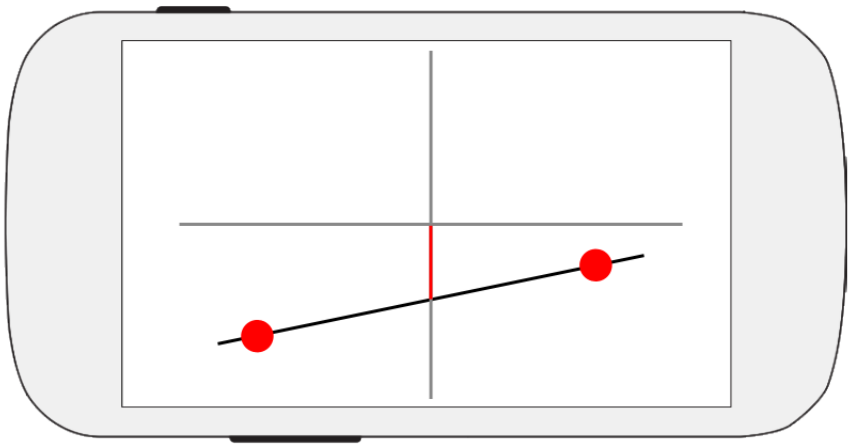
Pro další popis rozdělme ovládání do jednotlivých os. První z nich bude osa ovládající klapky na křídlech a tím i náklon celého letadla. V případě joysticku a z něj vycházejících metod jde o pohyb ukazatele či ovladače do stran. Navrhovaná metoda ovládání vyžaduje aby existovaly právě dva body ve kterých se hráč dotýká displeje. Tyto se použijí k sestrojení pomyslné kružnice, jejíž střed bude ležet uprostřed mezi oběma body. Úhel náklonu letadla pak odpovídá úhlu o který se spojnice bodů odchyluje od vodorovné osy kružnice. Situace je zachycená na obrázku 4.2. Ačkoliv to z popisu nemusí být zřejmé, na poloměru kružnice ve skutečnosti vůbec nezáleží. Hráč může průběžně vzdálenost bodů měnit a pokud oba body zůstanou na původní přímce, náklon letadla se nezmění. Vedlejším efektem je pak možnost takto měnit citlivost ovládání. Zatímco u kružnice o velkém poloměru je možné dosáhnout velmi jemné regulace náklonu, při přibližování bodů a zmenšování poloměru se citlivost přirozeně zvyšuje. Důležité je, že nezáleží na pozici kružnice v rámci dotykové plochy a v každém okamžiku se vytváří tak, aby její střed ležel ve středu spojnice mezi oběma body dotyku.



Obrázek 4.2: Ovládání náklonu letadla, červená místa značí body dotyku

Druhou z ovládaných veličin je výškové kormidlo. To bývá u joysticku a podobných metod reprezentování přitáhnutím nebo odtlačením ovládací páčky. Princip nastiňuje obrázek 4.3. Opět zde pracujeme se středem spojnice mezi dvěma body dotyku a odečítáme jeho vzdálenost od referenčního bodu, který je na obrázku znázorněn uprostřed displeje. Střed displeje je zvolen pouze pro ilustraci a ve skutečnosti se referenční bod přemístí pokaždé když se displeje dotýkají právě dva prsty. V okamžik prvotního dotyku se referenční bod nastaví a nemění se dokud se oba prsty nepřestanou displeje dotýkat. Jakmile se znovu dotknou, referenční bod se přesune na novou pozici na které opět po dobu kontaktu prstů

s displejem zůstává. Hráč tak při pouhém položení prstů na displej bez dalšího pohybu nijak nezasahuje do ovládání letadla. Současně tento přístup umožňuje hráči začít letadlo ovládat dotykem v libovolné části displeje, což může být výhodné zejména u tabletů a obecně zařízení s větší úhlopříčkou.



Obrázek 4.3: Ovládání vzestupu nebo poklesu letadla, červená místa značí body dotyku

Kombinací těchto dvou principů vzniká ovládání schopné pokrýt stejný rozsah veličin jako zmiňovaný *klasický dotykový* anebo sensorový vstup. Skládá se přitom ze známých prvků – ovládání náklonu může hráči evokovat zatáčení volantem, změna výšky pak gesto běžně používané pro svislé posouvání obsahu. Je proto možné předpokládat, že navrhované ovládání bude pro hráče skutečně intuitivní a k jeho vysvětlení nebude potřeba více než dva nákresy uvedené v této kapitole. Tento předpoklad se potvrdil při testování na jednotlivcích, které v průběhu vývoje probíhalo. Zatím jej však nelze považovat za dostatečně průkazné neboť probíhalo ve velmi omezeném rozsahu i počtu účastníků a není ani nijak zdokumentované. Detailnější průzkum bude probíhat v pozdějších stádiích vývoje.

4.3 Návrh hry

TODO: revize

V této části práce popíši jak by měla výsledná hra vypadat, jaké budou její herní mechanismy a v jakém stavu se nachází implementace v době psaní této zprávy. Stále je potřeba mít na paměti, že se pohybujeme v oblasti her pro mobilní zařízení. Je proto vhodné hru dělit do krátkých úseků a počítat s častým přerušováním. Dalo by se říct že jedna *epizoda* mobilní hry by měla trvat v řádu desítek vteřin až jednotek minut.

Při navrhování hry jsem vycházel zejména z [9] a [10].

4.4 Žánrové zařazení a herní mechanismy

TODO: revize

Hra se bude odehrávat v prostředí leteckých závodů. Hráč je v roli pilota akrobatického letadla který navštěvuje jednotlivé závody. Samotný závod pak spočívá v průletu branek ve stanoveném pořadí (přesná dráha letu se nevyžaduje), vzlet ani přistávání se neuvažuje.

Branky můžou být různého typu: obyčejná, jednosměrná nebo branka vyžadující průlet v předem dané poloze letadla. Právě poslední typ branky je typickým prvkem leteckých závodů, kdy nezáleží pouze na času letu, ale hodnotí se také přesnost se kterou pilot brankou proletí. Rozlišuje se průlet vodorovný, svislý (takzvaný *nožový let*) a průlet vzhůru nohama.

Cílem hráče je proletět všemi brankami v co nejkratším čase. Při průletu brankou vyžadující konkrétní polohu letadla se kontroluje přesnost průletu. Pokud je nízká, přičte se k času průletu postih závisející na velikosti odchylky. Průlet brankou se však započítá i v případě, že bude odchylka maximální. To je rozdíl oproti jednosměrné brance, která se považuje za splněnou pouze při průletu ze správného směru. U obyčejné branky žádná omezení ani postihy nejsou. U každého závodu jsou definovány tři časy průletů, které se hráč snaží překonat. Tyto časy vzniknou pravděpodobně v prvotní fázi odhadem, později se upraví na základě testování a zpětné vazby od hráčů.

Celá hra se pak sestává s množství závodů, kde každý se může odehrávat v jiném grafickém prostředí (ne však nutně vždy unikátním). Jednotlivé závody spolu ale nijak nesouvisejí. Na začátku má hráč zpřístupněné omezené množství závodů (například tři) do kterých může vstoupit, ostatní závody jsou uzamčené. Ve chvíli kdy hráč v některém ze závodů překoná alespoň jeden z předdefinovaných časů, považuje se závod za splněný a zpřístupní se jeden další závod. Tím se u hráče udržuje pocit soustavného pokroku, radost z rozšiřování možností a současně se omezuje riziko stavu kdy hráč není schopný žádný z časů překonat a ve hře prakticky nemůže pokračovat. Motivací pro nejlepší možné splnění závodu, tedy překonání nejlepšího z časů, pak budou speciální odměny, jako například nové a výkonnější letadlo za každých deset závodů. To navíc dovolí postupně zvyšovat obtížnost závodů a tlačit hráče do opakovaného hraní s cílem dosáhnout nejlepšího času. V jisté chvíli totiž zjistí že se současným letadlem nemůže závod splnit a potřebuje svůj stroj vylepšit. Jde o široce používaný princip, kdy se zvyšují nároky na hráče až do stavu velmi vysoké obtížnosti a následně se uměle zvýší hráčovy možnosti dodáním nového vybavení jako jsou zbraně nebo v tomto případě rychlejší letadlo. U hráče se při hraní střídá pocit výzvy a následného uspokojení z odměny a hra pro něj nepůsobí jednotvárně a nudně.

Hra by měla působit akčně a rychle, čemuž musí odpovídat i metoda ovládání. Právě u akrobatických letů je potřeba aby odezva na vstup byla minimální a hráč neustále cítil že má letadlo plně pod kontrolou. Takovouto hru by zcela jistě bylo velmi obtížné ovládat pomocí sensorického vstupu, a je proto dobrým prostředkem pro prezentaci předností nově navrhovaného ovládání.

4.5 Rozšiřitelnost

TODO: revize

Podstatným faktorem je také následná rozšiřitelnost hry. Aktualizace aplikace se nemusí zaměřovat pouze na opravování chyb, ale mohou současně přidávat další herní obsah. To je z pohledu hráče velmi žádoucí a právě informaci o aktualizaci mu může připomenout dávno dohranou hru. Aktualizace se pak stávají menšinovým propagačním nástrojem a je nutné s nimi takto i pracovat. Příliš časté vydávání nových verzí může být obtěžující, zejména pokud hráč většinu herního obsahu zkonzumoval a nového se mu nedostává. I v případě že má vývojář větší množství nevydaného obsahu, je vhodné jeho zveřejňování dávkovat a snažit se tak udržovat dlouhodobý zájem o hru.

Konkrétně u navrhované hry je relativně jednoduché doplnit další herní režimy. Ve chvíli kdy je implementováno ovládání letadla a jeho interakce s herním světem, je jen otázkou

téměř triviální logiky, zda se budou počítat body za prolétnuté branky nebo zda bude cílem proletět trasu v co nejkratším čase, případně cokoliv jiného.

Podobně je tomu v případě vytváření nových herních úrovní, tedy závodů. Hra nepočítá s unikátním prostředím pro každý jeden závod a scenérie se nutně budou opakovat. Nejjednodušším doplněním obsahu tedy může být využití existujícího prostředí a vytvoření pouze nové sady branek. O něco náročnějším rozšířením by pak bylo doplnění nového prostředí. Tam je náročnost zejména na straně grafických prací. Tímto způsobem je možné připravovat tématické balíčky s novými úrovněmi k různým významným příležitostem, což může být opět zajímavý propagační prvek.

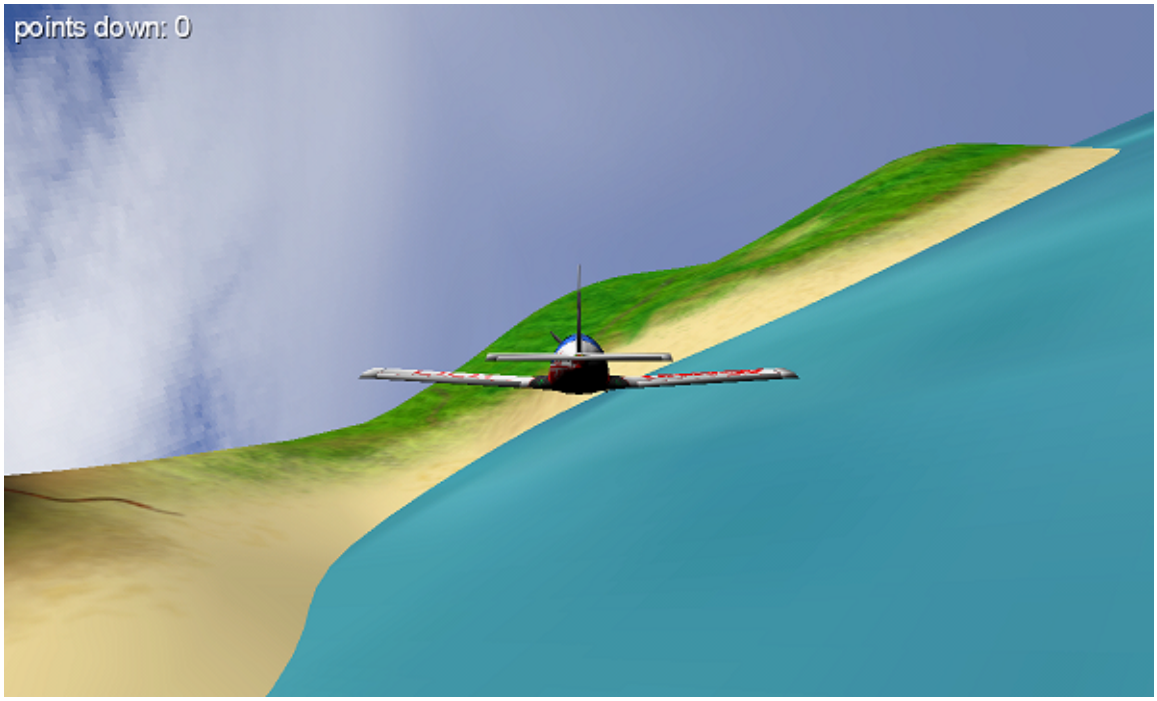
4.6 Stav implementace

TODO: revize - ponechat?

V průběhu vývoje vznikl prototyp pro ověření základní funkcionality a principů ovládání. Ten nepoužíval žádnou podpůrnou knihovnu a byl tvořený pouze plochou pro vykreslování. Jako uživatelské rozhraní využíval standardní systémové prvky. To bylo sice velmi neefektivní, ale současně velmi snadno implementovatelné. Prototyp obsahoval jediné herní prostředí a jediné letadlo s možností neomezeného letu s pouze základní kontrolou kolizí. Jeho cílem bylo zjistit možnosti a omezení z hlediska výkonu a vykreslování a především experimentovat s různými možnostmi ovládání. Podle předpokladů se potvrdilo že ovládání pomocí sensorů je pro tento typ hry velmi pomalé a těžkopádné. Prototyp byl nahodile testován na několika hráčích, plnohodnotné uživatelské testování však zatím z důvodu chybějících herních prvků není možné.

Let letadla je implementován velmi jednoduše. Orientace letadla ve scéně je reprezentovaná pomocí kvaternionů. Ty nabízejí možnost pohodlného skládání dílčích rotací podél jednotlivých os do výsledného směru letu. V každém průchodu herní smyčkou dochází k výpočtu těchto dílčích rotací na základě vstupu z ovládání, aktualizace natočení letadla a jeho posun ve směru letu. Namísto výpočtu skutečných sil působících na letadlo se hodnoty ze vstupu normalizují do rozsahu 0–1 a následně se přímo převádějí na hodnoty natočení letadla. Všechny veličiny jsou samozřejmě vážené svými koeficienty, což umožňuje definovat více letadel s odlišnými parametry. Je také zapojena funkce stabilizace letadla, která v každém okamžiku kdy hráč nepoužívá ovládání pomalu vrací letadlo do rovnovážné polohy, tedy do vodorovného letu.

V současnosti pracuji na verzi hry vystavěné nad zmiňovanou knihovnou Libgdx. Ta zcela přejímá principy ovládání a pohybu letadla které se osvědčily v prototypu. Současně využívá pokročilé funkce knihovny jako například detekci kolizí nebo předdefinované prvky uživatelského rozhraní. Tato implementace je však v ranném stádiu s funkcionalitou jen o málo větší než kterou nabízel prototyp. Aktuálním cílem je vytvoření jednoduchého závodu obsahujícího alespoň obyčejné branky a provedení prvního rozsáhlejšího uživatelského testování pro stanovení optimálních parametrů letadla a citlivosti ovládání.



Obrázek 4.4: Současná podoba hry

Kapitola 5

Implementace a publikování hry

5.1 Prototyp bez použití knihoven

Lorem Ipsum Obrázek z prototypu

5.2 Struktura aplikace

5.2.1 Oddělení platforem

Jak gdx odlišuje/sjednocuje vývoj pro jednotlivé platformy. Jaké jsou výhody a jaké jsou nedostatky.

5.2.2 Objektová struktura aplikace

Diagram tříd, popis jednotlivých komponent

5.3 Herní logika

Herní mechanismy a jejich implementace

5.4 Zveřejnění hry a zpětná vazba

5.4.1 Veřejné testování a průzkum

Co je potřeba zařídit před vydáním. Jak probíhala public beta. Ohlasy, dotazník, shrnutí výsledků.

5.4.2 Zveřejnění na Google Play

Co se změnilo od public bety. Na co pozor u publikování na Play. Jak probíhala propagace. Jaké jsou statistiky, zpětná vazba a hodnocení. Jak se měnila pozice v Play.

5.5 Možnosti dalšího vývoje

Shrnout slabiny, rozepsat co by se dalo dál rozšiřovat, co má smysl a co spíš ne.

Kapitola 6

Závěr

TODO: revize

Tento text shrnul možnosti vývoje 3D her na platformě Android, jeho přínosy a nedostatky. Dále se věnoval popisu letu skutečného letadla a z něho vycházející zjednodušené simulaci letu používané v hrách. Srovnává také existující metody ovládání leteckých her a navrhuje novou metodu, která závažnými nedostatky oproti ostatním netrpí. Navrhovaná metoda je založená na dotykovém vstupu za použití dvou prstů a nabízí velmi rychlé a přesné ovládání, čímž posiluje dojem že má hráč letadlo plně pod kontrolou.

Dále tato práce popsala návrh a prvotní implementaci hry, která toto ovládání využívá. Jde o hru kde hráč s akrobatickým letadlem prolétává brankami a snaží se doletět do cíle v co nejkratším čase. Hra je navržená tak, aby co nejvíce těžila z navržené ovládací metody, podpořila tak její platnost a v ideálním případě i její další rozšíření na poli leteckých her. V průběhu vývoje vznikl prototyp hry, na kterém se ověřily základní myšlenky a principy ovládání. Na několika testovacích subjektech se potvrdila jeho použitelnost a v současnosti probíhá vývoj další verze hry, tentokrát vystavené na knihovně Libgdx. Ta by měla implementovat popsané herní mechanismy a následně se rozvinout v plnohodnotnou hru. Detailní popis implementace v této práci chybí, neboť ta ještě není zcela ustálená a s velkou pravděpodobností se bude dále vyvíjet. Současně je v plánu před vydáním hry provést nejméně jedno rozsáhlejší a řádně zdokumentované uživatelské testování.

Literatura

[1] IDC. *Android Marks Fourth Anniversary Since Launch with 75.0% Market Share in Third Quarter* [online]. listopad 2012 [cit. 28.12.2012]. Dostupné na: <http://www.idc.com/getdoc.jsp?containerId=prUS23771812>.

[2] *Android Developers: Guides and reference* [online]. [cit. 28.12.2012]. Dostupné na: <http://developer.android.com/develop/>.

[3] *Performance Tips: Use Native Methods Carefully* [online]. [cit. 28.12.2012]. Dostupné na: <http://developer.android.com/training/articles/perf-tips.html>.

[4] *Platform Versions* [online]. [cit. 28.12.2012]. Dostupné na: <http://developer.android.com/about/dashboards/index.html>.

[5] *Libgdx: Desktop/Android/HTML5 Java game development framework* [online]. [cit. 28.12.2012]. Dostupné na: <http://libgdx.badlogicgames.com>.

[6] Langewiesche, W. *O umění létat.* [b.m.]: Baronet, 2010. ISBN 978-80-7384-307-6.

[7] NASA. *The Beginner's Guide to Aeronautics* [online]. [cit. 28.12.2012]. Dostupné na: <http://www.grc.nasa.gov/WWW/K-12/airplane/>.

[8] Banks, C. *A discussion of methods of real-time airplane flight simulation* [online]. [b.m.]: The Pennsylvania State University, 2000 [cit. 28.12.2012]. Dostupné na: <http://www.aerojockey.com/files/meng.pdf>.

[9] Jirkovský, J. *Game industry.* [b.m.]: D.A.M.O., 2011. ISBN 978-80-904387-1-2.

[10] Jirkovský, J. *Game industry 2.* [b.m.]: D.A.M.O., 2012. ISBN 978-80-904387-3-6.