FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Vytvoření počítačové verze deskové hry v jazyce Java
Student:	Ing. Petr Lohonka
Vedoucí:	Ing. Josef Pavlíček, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Vytvořte softwarovou verzi deskové hry vzdušný souboj stihačů.
- Navrhněte základní kostru hry formou diagramu tříd (UML)
- Navrhněte způsoby hry (sám proti počítači, dva u počítače)
- Hru naprogramujte dle pravidel JEE pro běh na aplikačním serveru a s využitím prezentační vrstvy formou tenkého klienta (Vaadin framework , či jiný dle výběru a konzultace se školitelem)
- Hru jako funkční blok umístěte na aplikační server běžící v prostředí firmy USPIN.cz

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 4. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Vytvoření počítačové verze deskové hry v jazyce Java

Ing. Petr Lohonka

Katedra softwarového inženýrství

Vedoucí práce: Ing. Josef Pavlíček, Ph.D.

3. ledna 2020

Poděkování

Mé poděkování patří Ing. Josefu Pavlíčkovi, Ph.D. za odborné vedení práce a cenné rady, které mi v průběhu zpracování bakalářské práce pomohly celé dílo zkompletovat a Ing. Petru Netíkovi za poskytnuté grafické podklady. Dále bych chtěl poděkovat své ženě a svým blízkým za jejich bezbřehou trpělivost.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 3. ledna 2020 .....

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

Lohonka, Petr. *Vytvoření počítačové verze deskové hry v jazyce Java.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020. Dostupný také z WWW: ⟨`https://portal.uspin.cz/planes/`⟩.

Abstrakt

Hlavním smyslem celé práce je převést deskovou hru „Vzdušný souboj stíhačů“ do virtuální podoby webové aplikace. Nejdříve analyzuji danou problematiku z hlediska historie a existujících řešení, poté vytvořím základní návrh vlastní aplikace s pomocí architektonického vzoru MVP. Při implementaci je nejzajímavější komponentou hry umělá inteligence, která simuluje lidského protivníka. Frontend je proveden v moderním frameworku Vaadin, backend v klasickém jazyce Java. Celá aplikace je připravena na nástup technologie Progressive Web Apps a nasazena na aplikační server. V závěru nalezneme i návrhy pro budoucí rozvoj celé hry.

Klíčová slova webová aplikace, návrh hry, webová desková hra „Vzdušný souboj stíhačů“, hratelnost, uživatelská přívětivost, umělá inteligence, JAVA EE, Vaadin

Abstract

The main goal of my Bachelor's thesis is to transform the uniqu board game „Dogfight“ to a web application. Firstly I analyze the history of board games and the technology, then I study some examples of existing solutions. After that I continue with the design of my own application, where I use the architectural pattern MVP. The most interesting part of the implementation is an artificial intelligence, which simulates a human opponent. Frontend is created with the modern framework Vaadin, backend is written in classic Java. The whole app is prepared for Progressive Web Apps and deployed on an application server. In the end of the whole work I suggest some future improvements for this app.

Keywords web application, game design, web board game „Dogfight“, gameplay, user-friendly experience, JAVA EE, Vaadin.

Obsah

Seznam obrázků

Úvod

Hraní deskových her patří k oblíbeným kratochvílím odpradávna. Zatímco v hluboké minulosti bylo výsadou králů a vzdělanců, v tomto století se rozšířilo i mezi běžný lid a v současnosti se začíná úspěšně adaptovat i na vyvíjející se technologie.

Tento trend považuji za velmi zajímavý, má bakalářská práce má tedy za cíl prohloubit toto spojení mezi klasickým deskovým hraním a výhodami technologického pokroku, které celý herní zážitek dokáží zpřístupnit a ozvláštnit.

Výsledek práce bude prospěšný všem, kteří si chtějí procvičit mozek a ukrátit chvíli s deskovou hrou, ale nemají spoluhráče, či si nechtějí pořizovat fyzickou verzi hry. Také bych rád podpořil tvůrce deskových her, kteří mohou svůj nápad distribuovat i bez nutnosti hru fyzicky vydat.

Téma práce jsem si zvolil ze tří důvodů. Prvním z nich byla touha po prozkoumání složitosti tvorby umělé inteligence, jejíž vznik byl jedním z důležitých bodů zadání. V současnosti celým technologickým světem rezonují otázky i řešení opravdové umělé inteligence, přišlo mi tedy zajímavé pokusit se v daleko menším měřítku o vytvoření své vlastní, specializované.

Druhým důvodem je samotná hra „Vzdušný souboj stíhačů“, její originální design a možnost vytvořit si vlastní pravidla z jejich hrubého obrysu, který mi předali její tvůrci. Po odborné stránce je to ochutnávka reálného vývoje vlastní aplikace, kdy ještě před počátkem tvorby musí být jasně rozvržen celý projekt tak, aby přinesl co největší míru užitku (v tomto případě zábavy) za co nejmenší náklady (v tomto případě zejména časové).

Posledním důvodem, proč je mi zvolené téma velmi blízké, je to, že jsem již odmala deskové hry přímo miloval a hrál zejména s rodiči všechny, které v té době byly dostupné (Dostihy a sázky, Maršál a špion, Fantom staré Prahy).

Proto jsem při domýšlení pravidel vsadil na zvýšení akčnosti, zábavy i hratelnosti tak, aby připomínala právě výše zmíněné hry mých raných začátků.

Za hlavní cíl celé práce považuji převedení originální deskové hry „Vzdušný souboj stíhačů“ do virtuální podoby tak věrně, aby byl zážitek identický s tím, který lze zažít při klasickém hraní.

Ke splnění tohoto cíle je třeba prozkoumat hraní deskových her obecně, poté zvolit vhodné technologie, důkladně přemýšlet nad přívětivostí uživatelského rozhraní a vybrat herní možnosti, které hra bude nabízet (hra proti počítači, hra více hráčů na jednom počítači, hra přes internet). Také se bude třeba soustředit na vhodný návrh z hlediska softwaru – provázanost tříd, akce uživatelů, návrhové vzory. Tyto oblasti budou podrobně prozkoumány v první části celé práce.

Praktická část bude obsahovat samotnou implementaci a nasazení na server.

Kapitola 1

Cíl práce

Cílem práce je převést originální deskovou hru „Vzdušný souboj stíhačů“ do virtuálního světa jako webovou aplikaci a zpřístupnit ji na webu pro širokou veřejnost.

V rešeršní části práce budu v prvním dílčím cíli zkoumat pravidla hry, požadavky a způsoby hraní (sám proti počítači, dva u počítače). V dalším kroku navrhnu základní kostru hry formou diagramu tříd (UML) a poté i prozkoumám technologie (JEE, Vaadin), které použiji pro implementaci.

V praktické části nejdříve naimplementuji samotnou hru dle výsledků rešeršní části, představím použité testovací metody, stručnou dokumentaci a prozkoumám možnosti budoucího rozvoje. Poslední cíl představuje nasazení hry jako funkčního bloku na aplikační server.

Hru zpracuji dle pravidel platformy Java Enterprise Edition s pomocí UI frameworku Vaadin a nasadím na aplikační server firmy USPIN.CZ.

Kapitola 2

Analýza externích faktorů

Analýza projektu je základem, na kterém celá realizace a implementace stojí, protože předem zmapována by měla být všechna témata týkající se projektu. Mezi takové oblasti patří nejdříve zkoumání vnějšího okolí, tedy historie a společenský kontext celé aplikace a srovnání a inspirace podobnými, již uskutečněnými projekty. To mi poskytne prvotní přehled o možnostech realizace, ať už z hlediska návrhu či technologie, které poté využiji při samotné realizaci.

2.1 Deskové hry a technologie v čase

Deskové hry jsou společníkem člověka již několik tisíciletí (čínská hra Go, šachy a mnohé další). Jejich používání, variabilita i dostupnost se v každé éře lišily, stejně jako zapojení soudobé technologie. Velkým krokem k nové éře hraní byl vývoj umělé inteligence, která umožnila hru i bez spoluhráče.

2.1.1 Minulost deskových her

Mnoho století si hráči deskových her museli vystačit bez technologií. Deskové hry byly vzácností, snad jen karty neboli „ďáblovy obrázky“ byly rozšířenější mezi širšími vrstvami obyvatelstva. V nich šlo ovšem spíše o hazard, než o pouhé potěšení ze hry jako u klasických deskových her.

S postupujícím pokrokem se taktéž hraní deskových her modernizovalo. Vznikly nejen mechanické pomůcky pro usnadnění hry (např. vynález míchačky karet, šachových hodin), ale i hry na mince pro jednoho hráče, většinou umístěné v restauracích. Začala tak éra nechvalně známých výherních automatů.

Samostatnou kapitolou jsou ovšem pokusy o první umělé hráče. Velice zajímavým příběhem snahy o vytvoření mechanického hráče je šachový automat zvaný „Turek“, který udivoval svými schopnostmi celý svět v 18. a 19. století. Mělo

jít o automat se vzhledem Turka, který seděl u široké bedny, na které byla šachovnice a rukou ovládal figurky. Ačkoliv byl v polovině 19. století celý stroj odhalen jako podvod, neboť v bedně byl ukryt špičkový živý šachista, který sledoval hru pomocí magnetů z druhé strany šachovnice a ovládal Turka táhly, celý koncept jistě rozjitřil myšlenky na umělou inteligenci. [1]

První opravdovou šachovou umělou inteligencí byl „El Ajedrecista“ postavený v roce 1912. Ačkoliv řešil pouze finální situace šachové partie s omezeným počtem figurek (král a věž) ne nutně nejefektivnější cestou, jednalo se o velký průlom v použití technologie pro deskové hry. [2]

Ve 20. století šel vývoj mílovými kroky a postupně začal nabízet hraní deskových her s pomocí technologií i široké veřejnosti na PC či různých herních televizních konzolích.

2.1.2 Deskové hry v současnosti

Svého největšího rozkvětu dosáhly deskové hry právě v posledních desetiletích. Zjednodušení tvorby grafiky a tisku díky rozvinutým technologiím vedlo k tomu, že svoji deskovou hru může vytvořit prakticky každý, ale o to těžší je v této konkurenci dostat právě svoji hru ke koncovým spotřebitelům – tedy hráčům.

Jednou z možností, jak deskovou hru snadno rozšířit mezi početnou komunitu, je její převedení do virtuálního světa a zpřístupnění širokým masám lidí používajících moderní technologie jako internet či mobilní zařízení. Odpadá problém s fyzickou stránkou deskového hraní, neboť nemusíme hru nákladně distribuovat a hráči se nemusí sejít na jednom místě, stačí jim připojení na internet. Nevýhodou je menší míra sociální interakce, která je velkým bonusem klasického hraní. To se však dá alespoň částečně nahradit chatem, či ještě lépe videohovorem přes internet.

Optimálním řešením se tedy zdá hra více hráčů na jednom počítači, která odstraňuje problém s dostupností hry a zároveň zachovává plný společenský charakter deskového hraní.

Další možností je kombinace fyzické hry a virtuálních pomocníků – k některým zakoupeným deskovým hrám lze stáhnout aplikace, která snadno nahrazuje podpůrné prvky – kostky, karty, počítání, někdy dokonce i hrací desku tak, že fyzická distribuce je omezena jen na to nejnutnější, a tedy i daleko levnější. Začíná také využívání prvků rozšířené reality.

Vnímání deskových her společností se v posledních době také proměňuje. Deskové hraní již není bráno jako pouhá kratochvíle pro ukrácení volného času, ale začíná mít svou váhu i jako učební, výchovný a socializační nástroj. Je do-

poručováno osobám všeho věku (s důrazem na děti a seniory) pro rozvíjení intelektu, logického myšlení, reakce na neočekávané situace, postřehu a komunikace s ostatními. Také většinou posiluje emoční pouto mezi hráči, ačkoliv jsou známy i případy, kdy naopak soupeření při deskové hře přerostlo v případě nezdravé soutěživosti až k vážným konfliktům v reálném životě. V drtivé většině případů ale lze konstatovat, že je deskové hraní prospěšné jedinci z hlediska individuálních i vztahových hledisek.

Vysoký stupeň rozvoje umělé inteligence umožnil i překonání člověka počítačem ve většině běžných her:

- hry s úplnou informací
  - dáma: program Chinook, světový šampion od roku 1994, kompletně vyřešil hru v roce 2007 [3]
  - šachy: Deep Blue, v roce 1997 [4]
  - Go: AlphaGo, v roce 2016 [4]
- hry s neúplnou informací
  - No-Limit Texas Hold'em poker: Libratus, v roce 2017 [4]
  - Starcraft II: real-time PC strategie, program DeepMind, 2018 [4]

Pro běžné uživatele je přítomnost dostatečně konkurenceschopné umělé inteligence prakticky standardem pro většinu deskových her, které jsou natolik rozšířené, aby se její vývoj vyplatil. Překážkou v její tvorbě jsou tedy spíše náklady než samotná složitost.

2.1.3 Budoucnost deskového hraní

Deskové hraní se bude pravděpodobně stále vyvíjet a reagovat na nástup nových technologií. Až se stane virtuální realita v domácnostech běžnou stejně jako je nyní PC, jistě se i deskové hry (i například hry na hrdiny) změní a využijí ji pro zvýšení zábavnosti.

Půjde jistě zapojit i zařízení chytré domácnosti, možná se k vizuálním a sluchovým vjemům přidají další smysly jako čich či hmat nebo fyzické distribuce deskových her postupně zaniknou a i při osobním setkání se deskové hry omezí na holografickou projekci, která nasimuluje hrací desku, figurky, karty, žetony a vše ostatní.

Umělá inteligence se bude pravděpodobně dál vyvíjet raketovým tempem a postupně překonávat člověka v čím dál složitějších hrách díky strojovému učení, neuronovým sítím a funkcím více a více podobnějším lidskému mozku, akorát několikanásobně zrychlenému a odolnému proti stresu či únavě.

2.2 Analýza existujících řešení

Vzhledem k naprosté originalitě deskové hry „Vzdušný souboj stíhačů“ je nemožné najít stoprocentně shodné existující řešení. Existují ovšem 2 zásadní otázky, na které se zaměřit:

- co ovlivní vybraný architektonický vzor pro aplikaci
- co ovlivní vnitřní strukturu tříd

Architektonický vzor silně ovlivňuje podmínka daná zadáním, tedy že hra musí být zpracována formou tenkého klienta a zároveň webové aplikace.

Tenký klient je označení takové aplikace, kde všechny informace a výpočty poskytuje server. Klientská část slouží pouze k zobrazení dat uživateli. Klasickým příkladem je webový prohlížeč. [5]

Tlustý klient nepotřebuje server, nebo ho využívá pouze k dodání nezbytných dat, vlastní výpočet je prováděn na klientském počítači. Jako příklad může posloužit například klasický balíček MS Office nainstalovaný na PC. [5]

Detailnější struktura tříd je ovlivněna vlastní logikou hry. Ačkoliv implementace hry „Vzdušný souboj stíhačů“ je unikátní, každá desková hra ovšem v dnešní době většinou kombinuje některé z konkurenčních, a tak i u mnou zpracovávané hry lze najít podobnosti s klasickými hrami, které svá řešení mají.

Základní vlastnosti realizované hry jsou:

- střídání tahů – prakticky každá desková hra
- pohyb figurek neomezeně po celé ploše – dáma, šachy
- pohyb po ploše v závislosti na hodu kostkou – Člověče, nezlob se!
- pohyb více figurek v jednom tahu
- vyhazování figurek soupeře – dáma, šachy, Člověče, nezlob se!
- nasazení z „domečku“ za určitých podmínek – Člověče, nezlob se!

Jak vidno z předchozího výčtu, inspiraci mohu nalézt ve všech klasických deskových hrách a jejich návrhu. Ten by měl ideálně obsahovat i zapojení umělé inteligence.

2.2.1 Existující řešení - architektonický vzor

I webová desková hra je především webovou aplikací. Pravděpodobně nejčastějším používaným vzorem pro webové aplikace je MVC neboli Model-View-Controller případně jeho varianta MVP tedy Model-View-Presenter.[6] Podrobněji MVC a MVP viz sekce 3.4.

Není tedy divu, že nejvíce deskových i jiných her najdeme realizovaných právě pomocí MVC. Již jen zkoumání her vytvořených v rámci akademických prací v ČR nám dává jasný signál o tom, jak je tento vzor oblíbený:

- *Vytvoření webové hry za pomocí MVC architektury* [7]
- *Tahová strategická hra v internetovém prohlížeči* [8]
- *Webová aplikace – internetová online hra Bitevní pole* [9]
- *Čínská dáma* [10]

2.2.2 Existující řešení - návrh tříd

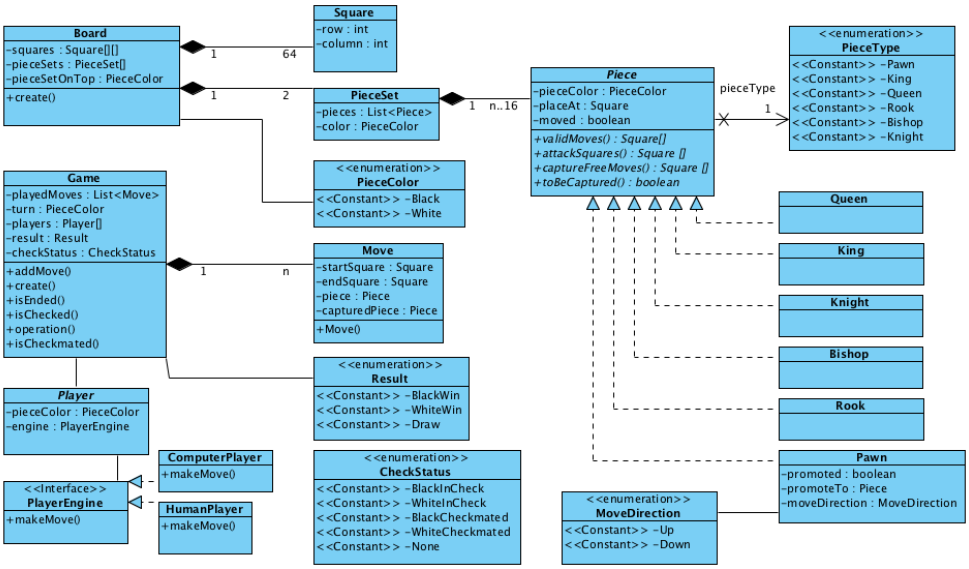
Dle kritérií shrnutých v části 2.2 je jasné, jaké projekty hledat. Díky tomu, že programování her je častou úlohou pro začínající i pokročilé programátory, řešení je velice mnoho. Pro rychlou orientaci jsem tedy zvolil analýzu dle dostupných UML diagramů logické a výpočetní části aplikace, přičemž jsem preferoval takové, které zobrazují umělé inteligence (AI), neboť moje implementace bude zahrnovat i tuto komponentu.

Prvním UML diagramem tříd je návrh šachové hry na obrázku 2.1, kde je zobrazeno i možné zapojení AI, tedy třídu ComputerPlayer jako implementace rozhraní PlayerEngine, přičemž druhou implementací je lidský protivník. Lze tedy pohodlně zaměňovat lidského a počítačového protivníka, přičemž třída Game, zastřešující celou hru, stále uchovává jen pole třídy Player, který se až vnitřně liší použitou implementací rozhraní.

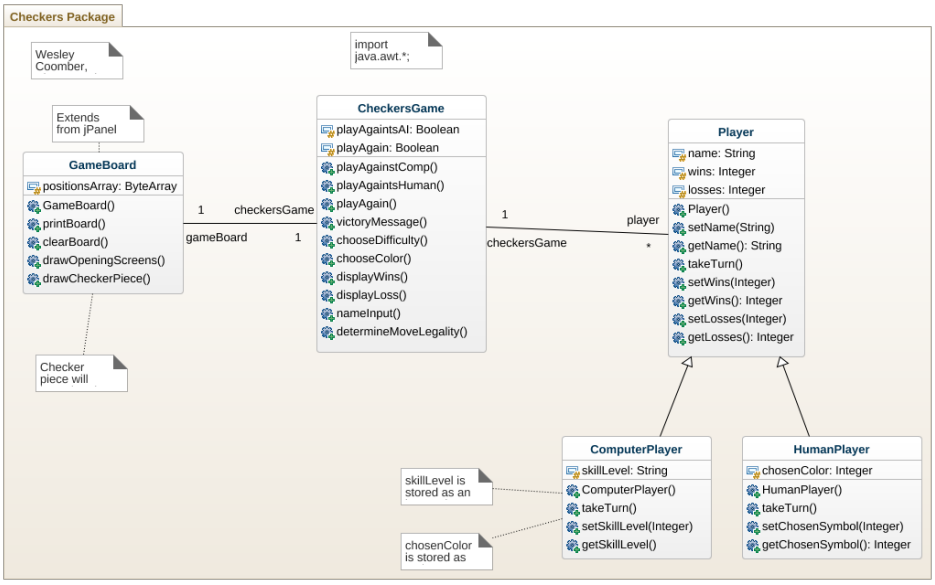
Druhý UML diagram tříd znázorňuje na obrázku 2.2 hru dáma. Zde je počítačový hráč jako potomek třídy Player, musí být tedy uložen do třídy CheckersGame přímo.

Oba způsoby jsou si podobné, liší se jen v použití interface, které by mohlo být užitečné, pokud bychom chtěli tvořit různé druhy hráčů.

Také z diagramů můžeme usoudit, že hra je obvykle rozdělena na jednotlivé třídy a celkově ovládána centrální třídou Game, která uchovává jednotlivé komponenty – hrací plochu, hráče a provádí vnitřní logiku a výpočty.



Obrázek 2.1: UML diagram tříd pro hru šachy s umělou inteligencí [11]



Obrázek 2.2: UML diagram tříd pro hru dáma s umělou inteligencí [12]

2.2.3 Existující řešení - technologie

Používaných jazyků pro webové deskové hry je celá řada a liší se případ od případu, neexistuje jedno silně preferované řešení.

Nejčastěji používané technologie pro deskové hry jsou v libovolných kombinacích tyto:

- frontend
  - CSS3 + HTML5: někdy čisté, někdy s podporou javascriptu
  - javascript: jQuery, React, Angular, pure
- backend
  - PHP: zejména frameworky Nette, Yii
  - Java
  - ASP.NET

Kapitola 3

Analýza realizované hry

Analýza realizované hry „Vzdušný souboj stíhačů“ je nezbytnou součástí celé tvorby. Nejdříve představím pravidla hry, která poskytnou představu o potřebné funkcionalitě a dle té rozhodnu o použití vhodného architektonického vzoru a rozvržení tříd. Poté vyberu vhodnou technologii ke splnění předchozích bodů analýzy, a to pro frontend i backend aplikace. Nezapomenu ani na naplánování testovací strategie v průběhu vývoje.

Všechny kroky povedu s podporou výsledků z předchozí analýzy již existujících řešení, samozřejmě s přihlédnutím ke konkrétním potřebám své aplikace.

3.1 Fyzická verze hry

Na obrázku 3.1 si můžeme prohlédnout plánek pro fyzickou verzi hry „Vzdušný souboj stíhačů“, který posloužil jako základ pro virtuální hru.

Vidíme, že pole hlavní hrací plochy, tzv. **bojová** políčka, zabírají více než polovinu hrací plochy, konkrétně 17 políček na šířku a 8 na výšku. Zbylá políčka se šipkou a čísly označujeme jako **vzletová** a plní ve hře funkci podobnou „domečku“ ze hry „Člověče, nezlob se!“.

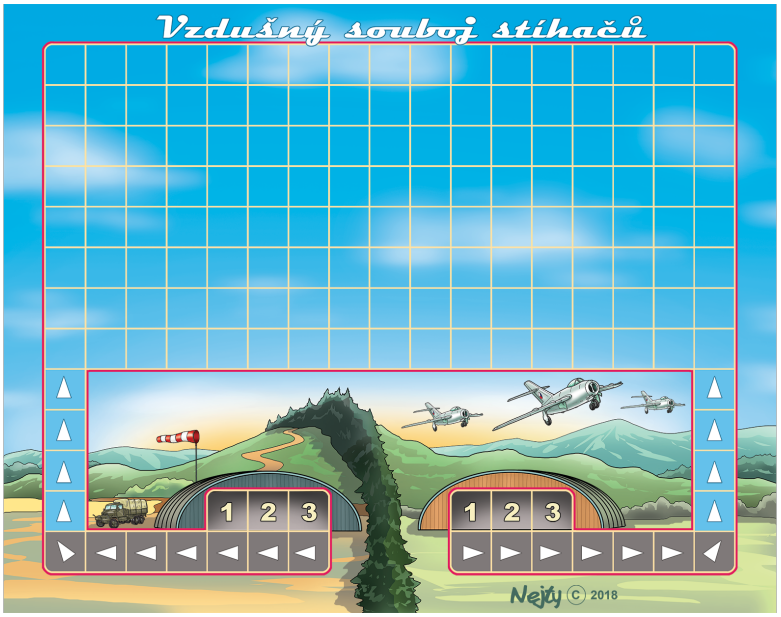
Ke hře jsou zapotřebí 2 hráči, 3 hrací kostky, 6 figurek stejné barvy znázorňující stíhače pro každého hráče (celkem tedy 12 figurek) a 1 mince.

3.2 Pravidla hry „Vzdušný souboj stíhačů“

3.2.1 Základní pravidla

Základní pravidla pro hru jsou tyto:

- Hrají 2 hráči proti sobě.



Obrázek 3.1: Herní plán fyzické verze hry „Vzdušný souboj stíhačů“

- Každý hráč má na začátku hry celkem 6 stíhaček rozlišených barvou (3 ve vzduchu a 3 v hangáru).
- Existují 2 druhy políček:

  Vzletová jsou políčka označená čísly 1, 2, 3 a políčka s bílým trojúhelníkem.

  Bojová představuje síť prázdných políček na „obloze“.
- Hráči se střídají v tazích.
- Hráč hodí v každém tahu 3 kostkami a každý jeden hod přiřadí k jedné své stíhačce, která se o hozený počet políček pohybuje.
- Po vzletových políčkách se pohybuje jen ve směru z hangáru do nebe.
- Pohyb stíhačky po bojových políčkách je možný vertikálně nebo horizontálně.
- Pokud stíhačka skončí svůj tah na políčku, kde je stíhačka protivníka, tak ji sestřelí.
- Pokud stíhačka skončí svůj tah na políčku sousedícím hranou s políčkem, kde je stíhačka protivníka, tak ji dostane do zaměřovače, zaútočí a má 50% šanci na její sestřelení.
- Vyhrává ten hráč, jenž sestřelí protivníkovi všechna letadla na bojových políčkách.

- Po hození šestky se znovu nehází.
- Po vzlétnutí se nesmí manévrovat zpátky na vzletová políčka.

3.2.2 Detailnější pravidla

Začátek hry

1. Každý hráč náhodně rozestaví 3 stíhačky ve sloupcích 1–5 (resp. 12–17) bojových políček. Tyto stíhačky hlídkují na začátku hry ve vzduchu.
2. Zbylé 3 stíhačky začínají v hangáru na políčkách 1, 2, 3 a jsou připraveny na start.
3. Začíná hráč s letkou startující vlevo.

Pohyb stíhaček obecně

Hráči střídavě házejí vždy 3 kostkami, které představují pohyb stíhaček po obloze či při vzletu. Vždy platí pravidlo přesně jeden hod pro jedno letadlo.

Pohyb po bojových políčkách

Pohyb je možný vertikálně či horizontálně. Je zakázán pohyb po diagonále. Ostatní stíhačky pohyb neblokují, tedy je možné se pohybovat i směrem, v němž stojí vlastní či nepřátelské letadlo a přeskočit ho (v realitě by se prostě vyhnuly v různé výšce).

Není dovoleno mít 2 a více stíhaček na jednom políčku. Pokud je na cílovém políčku nepřátelská stíhačka, je zničena, pokud přátelská stíhačka, pohyb není možný.

Pokud není možné se pohnout nijak (např. jsem v rohu mapy a jediná 2 možná políčka pro pohyb jsou obsazena vlastními stíhačkami), hod propadá a pokračuje soupeř.

Pokud má hráč na bojových políčkách méně než 3 letadla, stále háže 3 kostkami a po provedení pohybu stíhaček na bojových políčkách může zbývající hod (hody) použít pro letadla na vzletových políčkách.

Příklad 1: *Na bojových políčkách mám 2 letadla, na vzletových 3 letadla. Hodím 5, 3, 4 na kostkách. 2 hody musím použít na obě letadla na bojových políčkách (např. hody 3 a 4), poté si vyberu jedno z letadel na vzletu, a to posunu o zbývající hod, tedy 5 políček.*

Pokud má hráč na bojových políčkách méně než 3 letadla a na vzletových políčkách už žádné letadlo nezbývá, může si z hodů vybrat. Nepoužité hody kostkou propadají.

Příklad 2: *Na bojových políčkách mám 2 letadla, na vzletových žádné. Hodím 5, 3, 4 na kostkách. 2 hody musím použít na obě letadla na bojových políčkách (např. hody 3 a 4), hod s hodnotou 5 nelze použít nikde, proto propadá a hraje soupeř.*

Pohyb po vzletových políčkách

Po vzletových políčkách se pohybuje jen ve směru z hangáru do nebe, pohyb začíná vždy na začátku ranveje. Pohyb po vzletových políčkách nastává pouze ve 2 případech:

1. Pokud padne alespoň na jedné kostce 6, pak všechny 3 hody musí být použity pro letadla na vzletových políčkách, opět jeden hod pro jedno letadlo. Pokud jsou na vzletových políčkách méně než 3 letadla, lze zbývající hody použít pro letadlo či letadla na bojových políčkách.

   Příklad 3: *Na bojových políčkách mám 2 letadla, na vzletových 3 stíhačky. Hodím 6, 3, 1 na kostkách. Všechny hody musím použít na letadla na vzletových políčkách. Letadla na bojových políčkách se v tomto tahu nehýbají.*

   Příklad 4: *Na bojových políčkách mám 3 letadla, na vzletových také 3 stíhačky. Hodím 5, 3, 1 na kostkách. Všechny hody musím použít na letadla na bojových políčkách. Letadla na vzletových políčkách se v tomto tahu nehýbají.*

   Příklad 5: *Na bojových políčkách mám 3 letadla, na vzletových 2 stíhačky. Hodím 5, 3, 6 na kostkách. 2 hody musím použít na letadla na vzletových políčkách (např. 5 a 3). Poté si vyberu jedno z letadel na bojových políčkách, a to posunu o zbývající hod, tedy 6 políček.*

2. Pokud má hráč na bojových políčkách méně než 3 letadla, stále háže 3 kostkami a po provedení pohybu stíhaček na bojových políčkách může zbývající hod (hody) použít pro letadla na vzletových políčkách (viz Příklad 1).

Cíl hry

Vyhrává ten hráč, který protivníkovi sestřelí všechna letadla na bojových políčkách (případná startující letadla jsou snadnou kořistí, protože nemají výšku a rychlost).

Sestřelit nepřítele lze 2 způsoby:

Jasný sestřel je takový, kdy hráč dokončí svůj pohyb přímo na políčku se soupeřem a tím ho sestřelí.

Nepřítel v zaměřovači nastává v okamžiku, kdy hráč dokončí svůj pohyb přímo na políčku sousedícím hranou se soupeřem. To si můžeme představit tak, že vymanévroval soupeře a má ho v zaměřovači. Poté si ještě v tom samém tahu hodí mincí. Pokud padne panna, soupeř je zničen, pokud orel, soupeř manévrem na poslední chvíli unikl zásahu. Pokud je soupeř zničen, hráč postupuje na jeho políčko.

3.3 Výběr způsobů hry

Deskové hry lze virtuálně hrát většinou 3 základními způsoby:

Člověk proti počítači je klasickým a pravděpodobně nejčastějším způsobem hry, kdy funkci soupeře zastupuje umělá inteligence.

Člověk proti člověku přes internet se stávají čím dál populárnějším způsobem hraní, zejména díky čím dál snazší dostupnosti internetu na přenosných zařízeních (mobily, tablety).

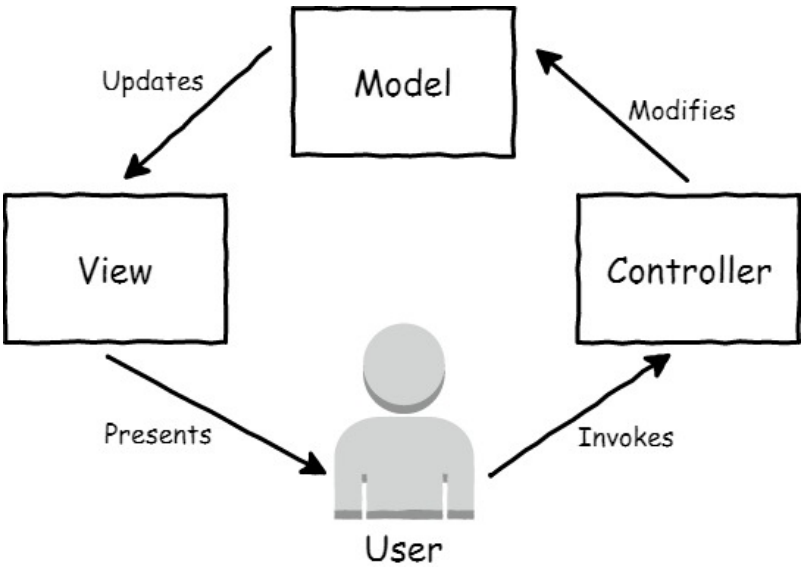
Člověk proti člověku na jednom zařízení už dnes pomalu mizí, v historii her má ale nepřehlédnutelné místo, před nástupem internetu se ani jinak hrát ve více lidech prakticky nedalo.

Tyto přístupy lze někdy i kombinovat, např. pokud desková hra umožňuje hru 3 a více hráčů, lze jich pár ovládat různými lidmi a zbylé nechat na umělé inteligenci.

Pro svou hry jsem vybral 2 způsoby hry – **člověk proti člověku na jednom zařízení**, protože to vyplyne vlastně samo z implementace hry a **člověk proti počítači**, který mi přijde jako velice zajímavý z hlediska tvorby umělé inteligence.

3.4 Architektonický vzor

Dle vzhledu i složitosti celé hry je jasné, že bude třeba velice detailně ovládat hrací plochu i logiku celé hry. Z analýzy existujících řešení vyplývá, že velice častým používaným architektonickým vzorem bývá vzor MVC (Model-View-Controller), který zároveň slouží jako základ dalším vzorům jako například MVP (Model-View-Presenter) či MVVM (Model-View-ViewModel). Nyní bych stručně představil modely MVC a z něj vycházející MVP, mezi kterými na konci zvolím.



Obrázek 3.2: Diagram jednotlivých vrstev u vzoru Model-View-Controller [6]

3.4.1 Model-View-Controller

O přesné specifikaci tohoto vzoru bylo popsáno mnoho, ovšem zdá se, že co odborník, to názor na to, jaká práva a povinnosti mají jednotlivé komponenty. Základ je pro MVC ovšem stále stejný. Skládá se ze 3 vrstev, jejichž vztahy můžeme vidět na obrázku 3.2

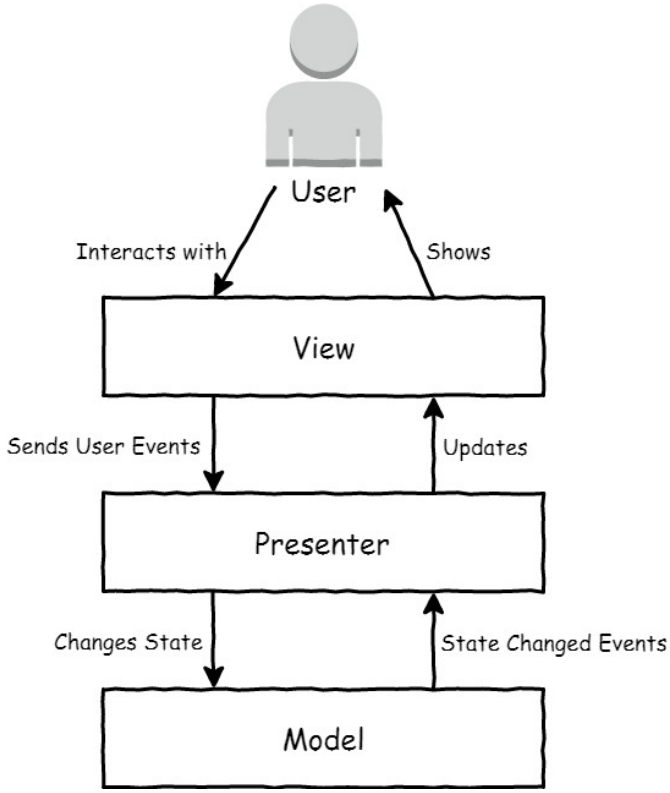
Model obsahuje vnitřní logiku aplikace, výpočetní a datovou základnu. Zde se ukládají a vyvolávají data (např. v databázi). Model nikdy nic nevykresluje a funguje na principu přijetí dat, jejich zpracování a změně uložených dat. Také často pomocí návrhového vzoru Observer upozorňuje vrstvu View na tuto změnu tak, aby se View mohlo překreslit dle aktuálního stavu modelu.

View slouží pouze k zobrazení dat a příslušného GUI uživateli, překresluje se dle notifikace z vrstvy model.

Controller slouží k zachytávání událostí způsobených uživatelem, jejich správné zpracování a předání modelu.

3.4.2 Model-View-Presenter

Z diagramu 3.3 jasně vyplývá odlišnost vzoru Model-View-Presenter od předchozího. Vztahy jednotlivých vrstev, kde vrstva Controller změnila jméno na Presenter, jsou následující:



Obrázek 3.3: Diagram jednotlivých vrstev u vzoru Model-View-Presenter [6]

Model zůstává funkcí nezměněn, pouze již nemá absolutně žádnou vazbu na View a komunikuje pouze s Presenterem.

View taktéž funkčně zůstává nezměněn, jen je zcela oddělen od vrstvy Model a také komunikuje přímo s Presenterem.

Presenter zprostředkovává veškerou komunikace mezi uživatelem a logikou aplikace. Zpracovává tedy uživatelské vstupy, které předá Modelu, ten je zpracuje, provede potřebné změny v datech a vrátí Presenteru potřebné informace. Ten je předá na správné místo ve View, které se dle nich překreslí. Presenter tedy plně ovládá tok dat a akcí v celé aplikaci.

3.4.3 Architektonický vzor - rozhodnutí

Oba výše analyzované vzory splňují dané podmínky. Pro účely realizované hry jsem ovšem vybral vzor Model-View-Presenter. Preferuji ho ze 2 důvodů:

- Dovoluje daleko podrobnější a konkrétnější ovládání View.

- Vnímám ho jako přehlednější a čistší, tok dat je jednoznačně daný.

3.5 UML diagram tříd

Před začátkem implementace je důležité si promyslet základní strukturu tříd s přihlédnutím k vybranému architektonickému vzoru. K tomu slouží UML diagram tříd, jehož detailnost závisí na účelu jeho použití.

Konceptuální model sloužící pro analýzu požadavků na aplikaci a mapuje zejména vztahy a základní atributy a metody jednotlivých tříd týkajících se business logiky (vrstvy Model v našem případě).

Designový model je detailnějším diagramem. Jako podklad ovšem slouží konceptuální model, který je dále rozšířen o viditelnost atributů a metod, datové typy atributů a zejména o ostatní vrstvy aplikace (tedy i View a Controller).

Implementační model je nejpodrobnější možnou variantou diagramu tříd, přidává všechny metody a atributy. Dá se říci, že tento diagram se dá okamžitě implementovat do funkční podoby, je tedy vlastně otisk zdrojového kódu. [13]

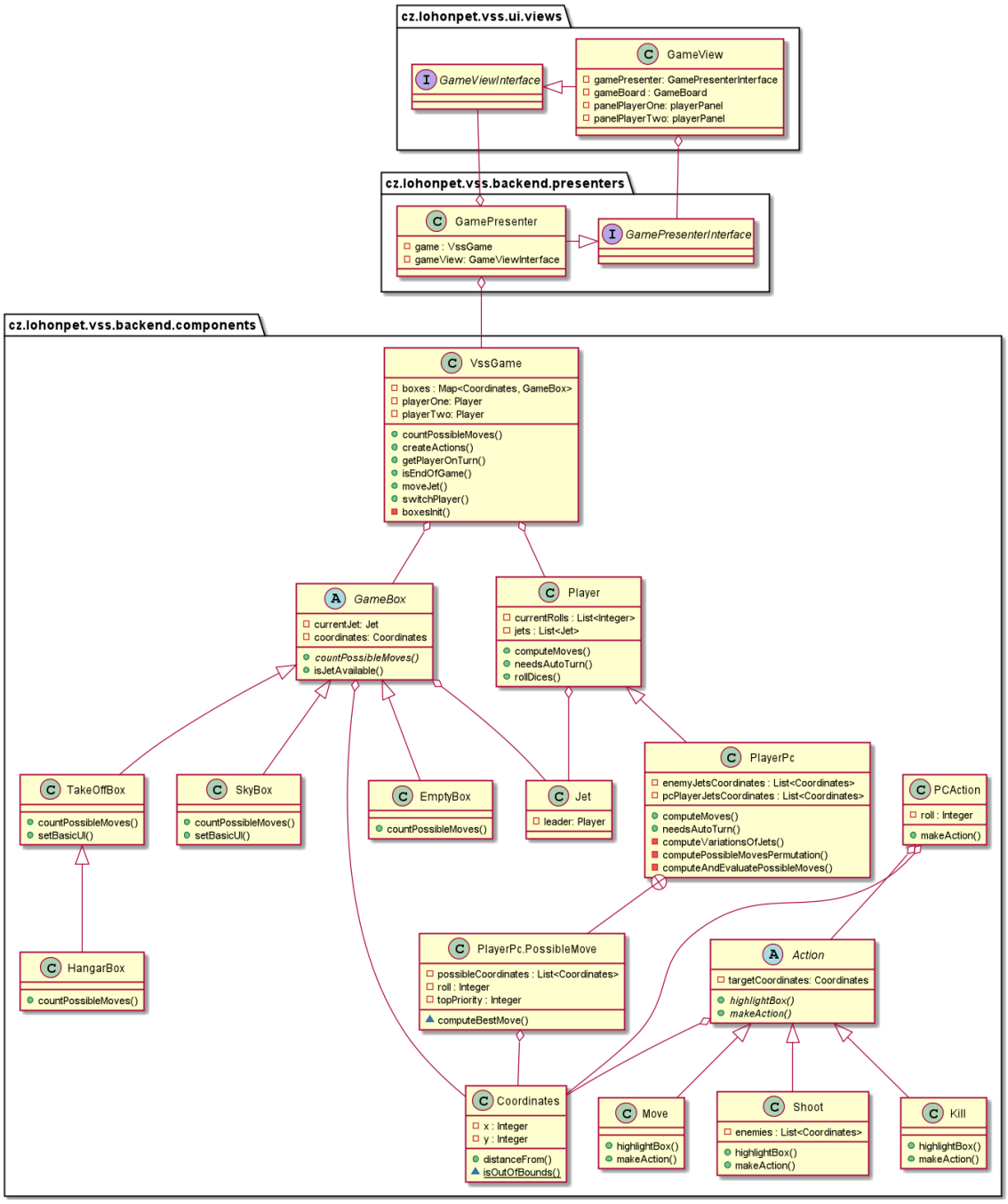
Pro mé účely je v této fázi ideální **designový model UML diagramu tříd**, neboť mi pomůže s detailnější představou o celém systému, aniž bych musel předem vymýšlet každou potřebnou funkci poměrně unikátní aplikace.

V diagramu jsou tedy vynechány konstruktory, gettery a settery a pomocné funkce tak, aby obsahoval pouze představu hlavních atributů a funkcí, které by měly jednotlivé vrstvy splňovat.

3.5.1 UML diagram tříd hry „Vzdušný souboj stíhačů“

Základem UML diagramu tříd 3.4 je vrstva Model, kterou reprezentuje balíček *cz.lohonpet.vss.backend.components*.

Zde nalezneme třídu *VssGame*, která je hlavní řídící a výpočetní třídou pro logiku celé hry a jediným bodem, ve kterém se vrstva Model stýká s Presenterem. Udržuje základní informace o samotné hře - mapu políček (třída *GameBox*) představující herní plochu a oba hráče (třída *Player*). Vždy tedy drží okamžitý stav hry, jehož momentální detail vrací přes Presenter do View uživateli, případně s ním a přijatými daty z Presenteru provede konkrétní výpočty a změny.



Obrázek 3.4: UML diagramu tříd – designový

Třída *GameBox* představuje jedno políčko na hracím plánu. Protože se funkčnost políček liší dle typu (pohyb, vzhled), třída *GameBox* je abstraktní a je implementována pouze svými potomky. Její instance je definována svými souřadnicemi (třída *Coordinates*) a přítomností letadla (třída *Jet*). Hlavní funkcí každého políčka je spočítat k danému hodu kostkou možné souřadnice cílových políček pohybu z něj.

Třída *Player* představuje lidského hráče a udržuje seznam jeho stíhačů a současný hod kostkami.

Potomkem třídy *Player* je třída *PlayerPc*, která představuje počítačového hráče, tedy umělou inteligenci, proti které si může zahrát lidský hráč. Ta si udržuje přehled o souřadnicích svých i nepřátelských letadel, ovšem hlavní a rozsáhlou zodpovědností celé třídy je výpočet nejvýhodnějšího tahu počítačového hráče. Tento výpočet se děje s pomocí vnitřní třídy *PossibleMove*, která představuje kombinaci jednoho hodu kostkou a jedné souřadnice stíhačky patřící počítačovému hráči. Vypočítá možnosti pohybu, oboduje tyto možnosti, a nakonec vybere tu nejvhodnější. Podrobněji bude rozebrána v sekci 4.5.

Poslední skupinou je abstraktní třída *Action*, která představuje všechny myslitelné akce, které lze provést při jednom tahu stíhačem, tedy pohyb (třída *Move*), okamžitý sestřel (třída *Kill*) a zaměření nepřítele (třída *Shoot*). Není nikde agregována, vytváří se dle stavu hry ve třídě *VssGame* a předává se do Presenteru, kde se přidává do akcí po kliku uživatele.

Jejím wrapperem je třída *PCAction*, která se chová podobně, ovšem pro počítačového hráče.

V diagramu vidíme i zbylé 2 vrstvy naší architektury.

Vrstvu Presenter představuje třída *GamePresenter* implementující rozhraní *GamePresenterInterface* v balíčku *cz.lohonpet.vss.backend.presenters*. Třída *GamePresenter* obsahuje jak odkaz na třídu *VssGame*, tak i na rozhraní *GameViewInterface*. Obsahuje funkce, které řídí celou aplikaci a její datový tok.

Vrsta View je vytvářena také objektově, přičemž hlavní řídící třídou je *GameView*, která implementuje rozhraní *GameViewInterface* a obsahuje odkaz na *GameViewPresenter*. Balíček *cz.lohonpet.vss.ui* bude jistě obsahovat i další pomocné třídy představující herní plochu (třída *GameBoard*) či ovládací prvky každého hráče (třída *PlayerPanel*), které jsem pro zjednodušení diagramu vynechal.

Vrstva View a Presenter je provázána ne pomocí tříd, ale pomocí dvou rozhraní, které tvoří tzv. kontrakt a velice usnadňují případnou výměnu vrstvy View, stačí pouze implementovat nové provedení dle těchto rozhraní.

3.6 Výběr technologií

Výběr technologie pro backend část aplikace je jasný ze zadání, které určuje JEE, tedy populární Java Enterprise Edition.

Pro frontend jsem zvažoval navrhovaný Vaadin, který mě zaujal natolik, že jsem ho nakonec i použil. Jedním z důvodů byla i velká podpora nově nastupujících Progressive Web Apps, které také v krátkosti představím.

3.6.1 Java Enterprise Edition

JEE je nadstavbou klasického programovacího jazyka Java Standard Edition. Přidává do Java SE knihovny, které jsou určeny primárně pro vývoj webových aplikací, přičemž umožňuje vývoj i velice rozsáhlých podnikových systémů.

Zajišťuje zejména funkčnost týkající se:

1. vývoje webových aplikací – Java API pro websocket, JSP, JSF, Servlet,
2. práce s databázemi – Java Persistence API, JTA,
3. podpory REST a webových služeb,
4. správy zabezpečení a dalších Java kontejnerů – např. EJB. [14]

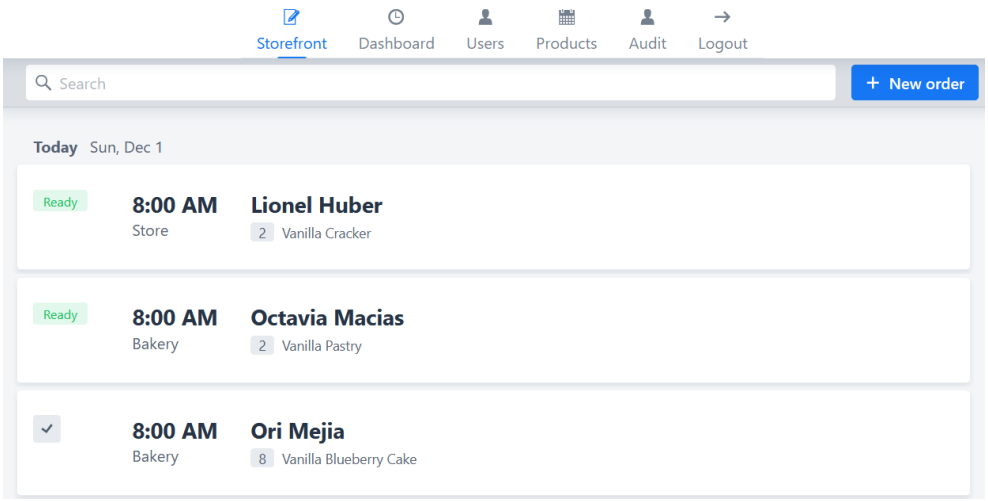
Pro vývoj deskové hry jako webové aplikace je to tedy technologie více než vhodná.

3.6.2 Vaadin

Vaadin je dlouhodobě vyvíjená platforma určená pro vývoj webových aplikací použitelná zejména pro aplikace podnikové. Hlavním znakem je používání Javy i pro UI a s ním související i server-side povaha aplikace.

Jeho hlavní výhodami jsou:

- Nejen backend, ale i UI se tvoří pomocí jazyka Java, není tedy nutné učit se další jazyk pro jeho tvorbu. Aplikace je tak napsána daleko rychleji.
- Použití Javy umožňuje snadno dodržovat objektový přístup i pro frontend.
- Je možno používat mnoho předdefinovaných komponent, které není třeba vyvíjet.



Obrázek 3.5: Vaadin – ukázka klasické aplikace

- Díky předdefinovaným „tématům“ má i defaultní podoba komponent a celého frameworku příjemný vzhled, prakticky tedy není nutné žádné stylování pomocí CSS nebo JS.
- Pokud je třeba použít CSS nebo JS, jde to s Vaadinem velice snadno.
- Podporuje a velice usnadňuje vytvoření aplikace jako Progressive Web Apps.

Framework má ovšem i zjevné nevýhody:

- Každá aplikace vytvořená Vaadinem je server-side, tedy je nemožné, aby fungovala offline.
- Protože je primárně určena pro podnikové systémy, hodí se zejména pro aplikace, které nepředpokládají větší požadavky na UI. Tvorba zcela netradičního designu (např. hry) v Javě by mohla vést k většímu množství hůře udržovatelného a přehledného kódu pro frontend, než při použití klasičtějších jazyků přímo pro UI určených.
- Klade větší nároky na vývojáře z hlediska rozdělení vrstev, protože společný jazyk pro frontend i backend umožňuje snadné vzájemné proplétání vrstev. To může bez striktního oddělení vést k velké nepřehlednosti v kódu a různým problémům v závislostech.
- Vyšší nároky na server.

Obrázek 3.6: PWA aplikace v prohlížeči Google Chrome

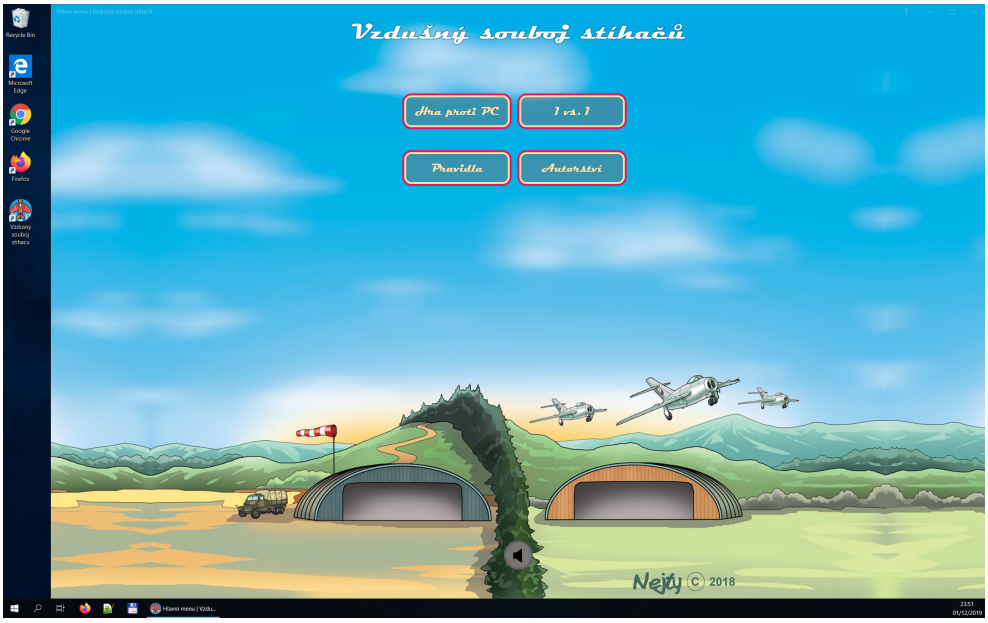
Po zvážení všech výhod a nevýhod jsem se rozhodl pro realizaci pomocí frameworku Vaadin (verze 14), neboť realizovaná hra je vlastně jen množství klikatelných boxů a velice specifické UI by mělo být realizovatelné s větším množstvím CSS. Vše převážily výhody použití jazyka Java, objektový přístup a vynikající podpora PWA.

3.6.3 Progressive Web Apps

Progressive Web Apps jsou poměrně nový termín, který představuje novou generaci webových aplikací. Tyto aplikace lze instalovat z webového prohlížeče tak, že pro uživatele jsou poté téměř k nerozeznání od nativních aplikací (např. mobilních).

To je vidět při srovnání obrázku 3.6, který zobrazuje PWA aplikaci před instalací, tedy jako obyčejnou webovou stránku v prohlížeči Chrome. Dole uprostřed okna prohlížeče je jasně vidět instalační tlačítko, které automaticky nabízí prohlížeč.

Oproti tomu na obrázku 3.7 je screenshot s nainstalovaným PWA na operačním systému Windows 10, pro názornost není hra zobrazená po celé obrazovce, aby byla vidět i automaticky vytvořená ikona na ploše. Vidíme, že se ničím neliší od ikony klasických aplikací. Samotná aplikace již velice připomíná nativní aplikaci téměř beze stopy po prohlížeči, dokonce i ve spodním panelu se hra již chová a vypadá jako samostatná aplikace s vlastní ikonou.



Obrázek 3.7: PWA aplikace nainstalovaná na OS Windows 10

Progressive Web Apps by měly mít tyto základní vlastnosti [15]:

- Progresivní – Funguje na všech prohlížečích.
- Responzivní – Vzhled je optimalizován pro všechna rozlišení obrazovky.
- Offline funkčnost – Funguje i bez přístupu k internetu.
- App-like – Uživatel má při interakci a navigaci pocit, jako by používal nativní mobilní aplikaci.
- Zabezpečená – Aplikace by měla být chráněna HTTPS.
- Fresh – Vždy aktuální data.
- Nalezitelná – Vyhledávače jsou schopny ji díky jejím náležitostem snadno nalézt.
- Znovuzapojení uživatele – Podpora funkcí jako jsou push notifikace, které motivují uživatele k užívání aplikace.
- Instalovatelná – Aplikaci lze nainstalovat z webu tak, že přístup k ní je stejný jako k nativní (tedy většinou ikonou na ploše zařízení).
- Odkazovatelná – Lze ji sdílet pomocí URL bez nutnosti instalace.

Vaadin kromě podmínky offline funkčnosti splňuje všechny tyto podmínky a stačí k tomu přidání jediné řádky v kódu.

3.7 Testovací strategie

Testování aplikace by mělo být nedílnou součástí každého vývoje, neboť řádné průběžné testování nejen zajišťuje bezchybnost kódu, ale odchytne případné chyby dříve, než se na nich postaví další funkcionalita a obtížnost správného vyřešení problému se stupňuje.

K testování lze přistoupit 2 základními způsoby, a to buď automaticky, nebo manuálně, případně lze oba přístupy mixovat.

3.7.1 Automatické testování

Zautomatizovat lze mnoho typů testů – nejčastěji jsou to testy jednotkové nebo testy komponent.

Výhody automatického testování jsou zřejmé. Při správně a bohatě napsaném testovacím frameworku je velice snadné a rychlé otestovat veškerou funkcionalitu aplikace. Pokud již máme funkční verzi a pouze přidáváme další funkcionalitu, velmi jednoduše zjistíme, zda nové úpravy nerozbily i to, co do této doby fungovalo. Řešíme tím i chyby člověka, který při stále stejných testech ztrácí pozornost.

Základní nevýhodou automatického testování je náročnost testy napsat a udržovat ve formě odpovídající aplikaci. Zejména u vývoje „na zelené louce“ se často implementace jednotlivých funkcí i komponent může drobně měnit, dokud celá aplikace není hotová. Neustálé přepisování automatických testů by tedy bylo velice časově náročnou položkou. Také je pro vývoj automatických testů nutné mít určité dovednosti a znalosti.

3.7.2 Manuální testy

Manuální testy se často používají k integračním a systémovým testům.

Lze říci, že jsou opakem automatického testování – trvají dlouho, vyžadují pozornost člověka, který může chybovat. Na druhou stranu je testování změny v aplikaci či nové funkcionality ihned možné bez psaní dalšího kódu. Testování také může provádět i nepříliš zkušený tester.

3.7.3 Testovací strategie pro hru „Vzdušný souboj stíhačů“

Náročnost implementace umělé inteligence mě přesvědčila k tomu, že svůj čas věnuji spíše jí. Tedy do automatického testování zařadím pouze nezbytné

metody a komponenty, ovšem většinu testování chci pokrýt manuálně v rámci úspory času.

Po realizaci celé aplikace by ovšem jako další krok bylo vhodné automatickým testováním celou aplikaci pokrýt, což zmíním v sekci 4.7.

Kapitola 4

Realizace hry

Při implementaci hry jsem se snažil co nejvěrněji dodržet výsledky předchozí analýzy, ještě jednou je tedy shrnu. Aplikace je webová, provedená jako tenký klient, architektonický vzor Model-View-Presenter, použiji technologie Vaadin a Java EE, přičemž celá aplikace bude vytvořena jako Progressive Web Apps.

Uživatel si může zvolit mezi hrou proti počítači, nebo proti jinému člověku na stejném PC.

4.1 Zahájení implementace

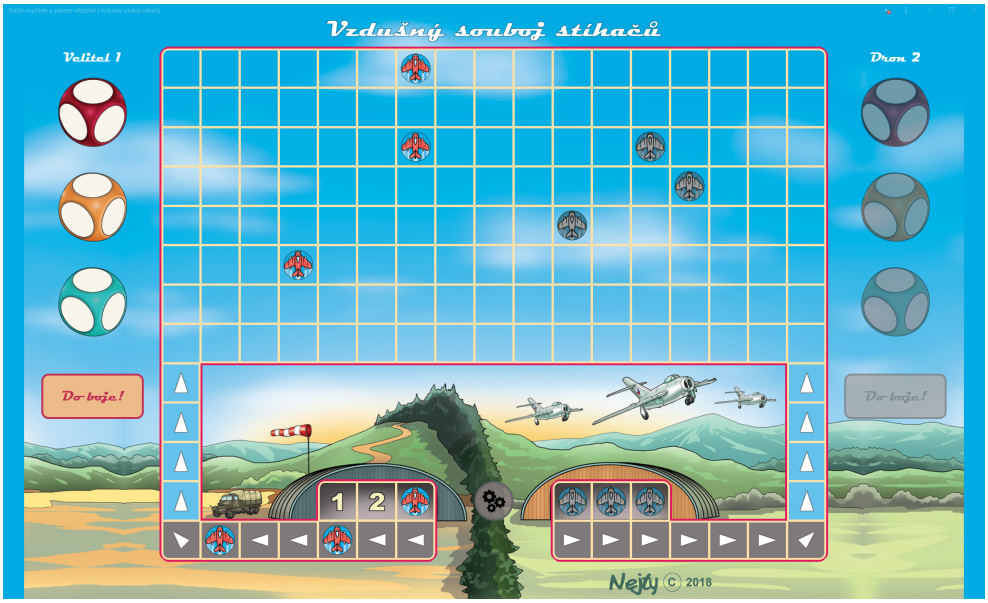
Start projektu ve Vaadinu je velice snadný, záleží ovšem na typu projektu.

Pro jednodušší klasickou podnikovou aplikaci je možné využít a pouze předělat Vaadinem přímo implementovanou vzorovou aplikaci Bakery App (`https://bakery-flow.demo.vaadin.com`), která je součástí závěrečného tutoriálu na oficiálních stránkách frameworku Vaadin. Ta již obsahuje základní funkcionalitu pro podnikové aplikace – CRUD, přihlašování, různé stupně uživatelů a rozličné způsoby manipulace s daty a jejich prezentaci v grafické podobě.

Pro složitější podnikové aplikace, či nestandardní použití celého frameworku jako v mém případě, lze stáhnout ze stránek Vaadin jednu z předpřipravených prázdných šablon (`https://vaadin.com/start/latest`). Tu lze následně vložit do IDE (doporučované je Eclipse nebo Intellij Idea) a rovnou začít vyvíjet.

Druhý způsob jsem zvolil i já. Vybral jsem šablonu využívající framework Spring boot a nahrál ji do IDE Intellij Idea. Úvodní nastavení tak trvalo jen několik málo minut.

Ještě před samotnou implementací je nutné si ujasnit, kam ukládat zdroje,



Obrázek 4.1: Hrací plocha hry „Vzdušný souboj stíhačů“ ve virtuálním prostředí

neboť pro správnou funkčnost je třeba mít CSS soubory, obrázky, zvuky a JS scripty ve správné lokaci.

Pro Vaadin 14 používající technologii Spring Boot jsou lokace následující [16]:

CSS patří do */frontend/my-styles/*.

JavaScript náleží do */frontend/src/*.

Obrázky, zvuky, soubory ke stažení a ostatní statické zdroje uchováváme v */src/main/resources/META-INF/resources/*.

4.2 Grafické rozhraní

Grafická podoba celé hry je návrhem Ing. Petra Netíka, který mi poskytl podklady ve formě souborů určených pro aplikaci Adobe Photoshop.

Drobné grafické přídavky, se kterými nebylo ve fyzické variantě hry počítáno, jsem přidal osobně. Jedná se zejména o panel ovládání každého hráče, mince, tlačítka, popup okna a zobrazení sestřelu či zaměřovače.

Pro obě obrazovky aplikace jsem použil podkladový obraz z fyzické části hry. Finální podobu hlavního menu zobrazuje obrázek 3.7. Konečná podoba hrací

plochy na obrázku 4.1 je trochu odlišná od fyzické verze hry, kterou pro srovnání najdeme na obrázku 3.1. Povšimněme si přidaných bočních ovládacích panelů, kvůli kterým bylo nutné rozšířit hrací plochu a tlačítka „Nastavení“.

Zajímavostí ve vizuálním provedení hry jsou 2 animace, jedna simulující hod kostkami, druhá představující hod mincí.

4.2.1 Technické detaily GUI

Celá aplikace včetně animací a zvuků je vytvořena pouze pomocí jazyka Java ve frameworku Vaadin a CSS. Právě široké možnosti CSS3 umožňují pokročilé funkce jako animace, zvětšování prvků při najetí apod. Pro přehlednost kódu jsem rozdělil funkcionalitu grafického rozhraní tak, že se v Javě pouze přidávají a ubírají třídy, které jsou definovány v CSS souborech. To dává jistotu, že samotný vzhled vždy řeší CSS a jeho manipulace a přiřazování jednotlivým prvkům vždy Vaadin.

Vzhledem k velikosti celého hracího plánu je nemožné, aby byla hra ve stávající podobě pohodlně hratelná na mobilních telefonech, či obecně zařízeních s nízkým rozlišením. Proto jsem responzivitu aplikace omezil zdola na rozlišení 1024 x 635, což by mělo umožňovat hru na drtivé většině tabletů a prakticky všech noteboocích a monitorech stolních počítačů.

Do této spodní hranice je aplikace plně responzivní, tedy upravuje velikost hrací plochy tak, aby pokrývala co největší část obrazovky, a to při dodržení poměrů stran původního hracího plánu.

4.3 Zvuky

Zvuky přinášejí do hry další rozměr, vtahují do děje a zvyšují celkový zážitek. Proto jsem se rozhodl, že je do hry také zařadím, bohužel Vaadin 14 žádnou komponentu ovládající zvuky nenabízí.

Vytvořil jsem tedy vlastní Vaadin komponentu SoundLibrary, která ovládá zvuky čistě na bázi HTML tagu *audio*. Samotné zvuky jsou stažené z internetové databáze `https://freesound.org/` a jejich autoři jsou uvedeni v aplikaci.

V hlavním menu hraje automaticky a ve smyčce „bojová hudba“, při hře jsou zvuky navázané na akce uživatele – pohyb, sestřel, zaměřovač, hod kostkami a hod mincí.

Zvuky lze samozřejmě v aplikaci vypnout i zapnout.

4.4 Implementace

Při implementaci samotné hry jsem se držel UML diagramu tříd (3.4) vytvořenému v analytické části realizace. Proto hrubý popis systému v části 3.5.1 odpovídá skutečnosti, některé implementační detaily bych ovšem rád rozvinul.

4.4.1 Dědičnost a polymorfismus

Dědičnost a polymorfismus jsou klasickými technikami pro přehlednost kódu a snadný vývoj aplikace. V mnou realizované hře jsou tyto postupy použity hned několikrát.

Abstraktní třídě ***GameBox*** představující jedno herní políčko přísluší potomci *SkyBox, TakeOffBox, EmptyBox*. Ze třídy *TakeOffBox* potom dále dědí třída *HangarBox*. Celá tato struktura má 2 hlavní účely:

1. Nastavení vzhledu hrací plochy – po inicializaci hrací plochy v modelu ji převezme Presenter a dle příslušného zavolání metody *setBasicUI()* na každé políčko určí základní vzhled jeho protějšku v UI.

2. Výpočet možností pohybu – dle pravidel je jasné, že jiné možnosti pohybu jsou na bojových políčkách (reprezentovaných třídou *Skybox*) a jiné na vzletových políčkách (*TakeOffBox*) či v hangáru (*HangarBox*).

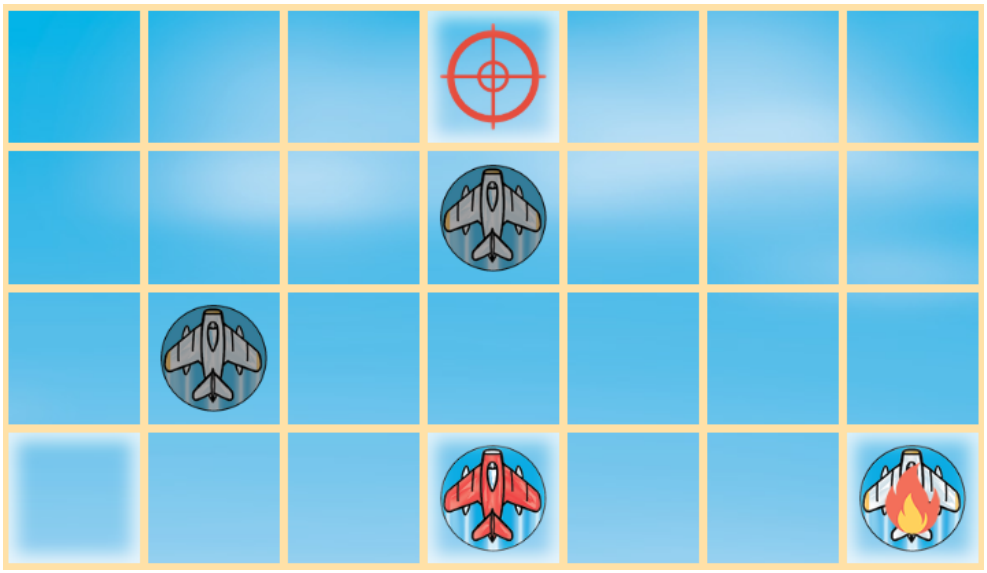
3. Pro neaktivní políčka bez vzhledu jsem zavedl třídu *EmptyBox* tak, abych si usnadnil tvorbu celé sítě hracího plánu.

Abstraktní třída ***Action*** představuje možnou akci uživatele. Jejími potomky jsou třídy *Shoot, Kill a Move*.

Tyto potenciální akce se vytvoří dle hodu a pozice všech prvků ve hře a nejdříve ovlivňují vzhled možností v UI, viz obrázek 4.2. Na tomto obrázku vidíme situaci, kdy červený hodil 3, přičemž tímto hodem se může buď pouze posunout na políčko vlevo dole, zaútočit nepřímo se zaměřovačem na letadlo nahoře, nebo rovnou sestřelit letadlo vpravo dole.

Po kliknutí na příslušné políčko se provede přetížená metoda *makeAction()* ze třídy *Action*, která vykoná příslušnou akci přes Presenter v Modelu ve třídě *VssGame*.

Další dědičnost je využita pro třídu ***Player***, kdy samotný *Player* představuje živého hráče a potomek *PlayerPC* hráče umělého. Jeho funkcionalita bude podrobněji rozebrána v sekci 4.5.



Obrázek 4.2: Všechny možné akce zobrazené v UI

Díky použití Vaadinu je dědění snadné i při stavbě UI, např. pro ovládací prvky hráče jsou v UI používány třídy *PlayerPanel* a z něj vycházející potomek *PlayerPanelPc*.

4.4.2 UI vlákna

Speciální vlákna pro ovládání UI jsou nástrojem pro specifické chování aplikace, které upgradují View, zatímco hlavní vlákno má volné ruce k práci na výpočtech.

Tento vícevláknový přístup se používá zejména pro podnikové aplikace, kdy v případě velké databáze či vysoké asymptotické složitosti výpočtu by mohl mít uživatel pocit, že se aplikace zasekla, neboť po určitou dobu nereaguje. Taková situace se řeší nastartováním UI vlákna, které informuje uživatele pomocí nějakého indikátoru průběhu o dosavadním pokroku a uživatel ví, že musí čekat.

V případě hry „Vzdušný souboj stíhačů“ tak složité výpočty nejsou nutné, ovšem tato UI vlákna jsem použil pro simulaci chování umělého hráče, případně pro dočasné zobrazení některých popup oken tak, aby nebylo třeba je manuálně potvrzovat.

UI thread se ve Vaadinu vytvoří klasicky, jako potomek třídy *Thread*, u kterého přetížíme metodu *run*. V aplikaci jsem vytvořil 4 typy specifických threadů pro různé herní situace:

ActionThread simuluje „přemýšlení“ mezi akcemi umělého hráče tak, aby tah vypadal lidsky a pro protihráče byl lépe čitelný.

RollDiceThread simuluje čekání mezi koncem tahu lidského hráče a začátkem tahu umělé inteligence, který začíná hodem kostkou.

ShootThread simuluje hod mincí a následnou akci (sestřel zaměřovačem nebo nic). Užit pro oba typy hráčů.

StuckedPopupThread zobrazuje a následně zavírá jednoduchý popup, který zobrazuje zprávu o nemožnosti dalšího tahu, poté je hra přepnuta na protihráče.

4.4.3 Návrhový vzor - Null Object pattern

Null Object Pattern je návrhový vzor, který má odstínit programátora od nutnosti psát kontroly na null object a obecně řešit problémy s null referencí tím, že se k reálné třídě objektu vytvoří další třída se stejným rodičem, která reaguje na volané metody neutrálním způsobem. Tato třída nahradí null referenci, instance tedy už není buď null nebo objekt, ale objekt neutrální, nebo objekt reálný. [17]

Stejný princip uvažování jsem použil u třídy *EmptyBox*, která je součástí herního plánu. Nenahrazuje sice null referenci, ale výrazně zjednodušila tvorbu sítě políček, neboť díky ní šla vytvořit jako jednoduchá obdélníková síť. Třída *EmptyBox* nemá nastavený žádný vzhled a na volané metody odpovídá neutrálně, čímž umožňuje ovládání pohybu a uživatelských akcí uvnitř sítě daleko snazším způsobem, než kdybych tento návrhový vzor nepoužil.

4.5 Umělá inteligence

Umělá inteligence je tvořena obecným systémem, který postupně vyhodnocuje z aktuální situace všechny možné kombinace možností, ty poté vyhodnotí a nejlepší variantu tahu vrátí zpět do vrstvy Presenter pro další zpracování. Samotné vyhodnocování tahu je dáno vnitřní třídou *PossibleMove*, kde je tah obodován statistickou ohodnocovací funkcí dle několika kritérií.

4.5.1 Obecný systém umělé inteligence

Celý obecný systém umělé inteligence pro možnost hry jednoho hráče proti počítači znázorňuje hlavní metoda třídy *PlayerPC*, kterou představuje veřejná metoda *computeMoves()* (úryvek kódu 1).

```
@Override
public List<PCAction> computeMoves(VssGame game) {
  this.game = game;

  ...
  // nastaveni pomocnych promennych z promenne game
  ...

  List<List<Coordinates>> allJetsCombs =
        computeCombsOfJets();
  possibleMoves = computePossibleMoves(allJetsCombs);
  computeAndEvaluateAllMoves();
  sortPossibleMoves();
  checkBestOption();
  printBestOption(); //debugging purposes
  return fillPCActions();
}
```

Úryvek kódu 1: Hlavní metoda umělé inteligence *computeMoves()*

Tuto metodu volá přímo Presenter, který jí předá instanci třídy *VssGame*, která obsahuje celou aktuální situaci na hrací ploše. Cílem celé metody je vrátit seznam akcí umělého hráče, které se posléze provedou v UI a upraví i aktuální situaci ve vrstvě Model.

Po nastavení pomocných proměnných pro jednodušší práci s instancí *game* dochází k samotnému vyhodnocování možných tahů. Tahem je míněn celkový tah, tedy soubor 1-3 akcí stíhaček. Vyhodnocování tahu probíhá v těchto krocích:

1. Nejprve v metodě *computeCombsOfJets()* vypočítám všechny možné kombinace z vlastních letadel.

2. Každé takové kombinaci stíhaček poté v metodě *computePossibleMoves()* přiřadím všechny možné hody kostkami a tím z nich vytvořím pár stíhačky a hodu. Tyto páry jsou většinou ve trojicích.

3. Každému takovému páru v metodě *computeAndEvaluateAllMoves()* vypočítám všechny uskutečnitelné akce dle pozice stíhačky a daného hodu. Dále každou takovou akci vyhodnotím (více v sekci 4.5.2).

4. Vyhodnocené akce poté po vytvořených trojicích sečtu a všechny trojice akcí (tedy jednotlivé tahy) seřadím od nejlepších po nejhorší v metodě *sortPossibleMoves()*.

5. Metoda *checkBestOption()* vezme vybranou nejlepší variantu tahu a provede poslední kontroly (např. zda nekončí tah dvou stíhaček na stejném místě). Pokud kontrolou neprojde, vezme se další nejlepší varianta a opět se zkontroluje. To trvá tak dlouho, dokud metoda nenalezne vyhovující variantu tahu.

6. Poté je konečně z tohoto tahu udělán v metodě *fillPCActions()* soubor jednotlivých akcí, které jsou poté předány zpět vrstvě Presenter, která zajistí jejich vykonání.

4.5.2 Hodnocení jednotlivé akce

Ohodnocení každé možné jednotlivé akce se děje pomocí statistické funkce. Tato funkce postupně vypočítává dílčí priority jednotlivé akce, aby nakonec pro dvojici stíhačka a hod kostkou určila optimální akci. Samotné vyhodnocení probíhá v metodě *computePriority()*, viz úryvek kódu2 (pro lepší čitelnost bez logů) .

```
private Integer computePriority(Coordinates possCoord) {
  Integer priority = 0;
  if (game.getBoxes().get(possCoord).getArea() == ON_SKY) {
    priority += computeKillEnemy(possCoord);
    priority += computeMoveTowardsEnemy(possCoord);
    priority += computeAgressionFactor();
    priority += computeDefendDirect(possCoord);
    priority += computeDefendAim(possCoord);
    priority += computeRunAway(possCoord);
    priority += takeOffBonus();
  } else {
    priority += computeTakeOff(possCoord);
  }
    return priority;
}
```

Úryvek kódu 2: Vyhodnocení jednotlivé akce v metodě *computePriority()*

Postupné dílčí priority jsou vyhodnocovány zvlášť pro stíhačky, které skončí akci na bojových políčkách a zvlášť pro ty, které ji skončí na vzletových políčkách. Cílové souřadnice představuje proměnná *possCord*. Protože hlavní prioritou hry je její zábavnost, je umělá inteligence nastavena tak, aby byla agresivnější, než by dle ideální strategie být měla.

Pro **vzletová políčka** je systém jednoduchý - čím výše se stíhačka dostane, tím větší prioritu pohyb má. Tím zajišťujeme co nejrychlejší přísun posil do boje.

U **bojových políček** je systém složitější:

1. Umělá inteligence nejdříve vypočítá možnost útoku na cílové souřadnici v metodě *computeKillEnemy()*. Vysokou prioritu získá, pokud lze nepřítele přímo zničit, poloviční bonus, pokud ho dostane do zaměřovače a má poloviční šanci na zničení.

2. Aby umělá inteligence preferovala spíš pohyb k nepříteli než únik, dostává bonus při akci, která znamená přiblížení. To je vyhodnoceno ve funkci *computeMoveTowardsEnemy()*.

3. Předchozí akce jsou ze své podstaty akce agresivní. Protože při početní převaze si může dovolit být agresivní více, je dosavadní dosažená priorita ještě navýšena, pokud má více stíhaček než soupeř, což se děje ve funkci *computeAgressionFactor()*.

4. Nesmíme zapomínat ani na obranu. Základní chybou v této hře je vystavit své letadlo do **přímé palebné linie**, tedy takové pozice, kde může být v příštím tahu soupeřem automaticky zničeno (červené a oranžové pozice na obrázku 4.3). Proto je v metodě *computeDefendDirect()* tato možnost zkoumána, a pokud by letadlo při této akci skončilo v takové pozici, je dán vysoký negativní postih.

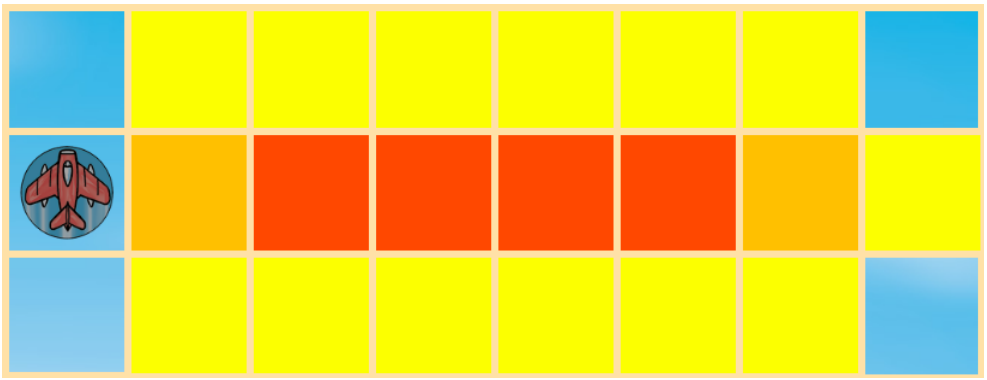
5. Menší negativní postih je přidělen v metodě *computeDefendAim()*, která zkoumá cílovou polohu letadla z hlediska **nepřímé palebné linie** (žluté pozice na obrázku 4.3). To je taková, kdy by soupeř mohl zaútočit zaměřovačem, tedy pouze nepřímo, s poloviční pravděpodobností sestřelu.

6. Předposledním bodem je metoda *computeRunAway()*. Ta ovlivňuje prioritu akce v situaci, kdy má umělá inteligence na bojových políčkách již pouze jedno letadlo, ale na vzletových ještě několik letadel čeká. V tu chvíli dostává vysokou prioritu tah, ve kterém osamocené letadlo utíká ohrožení, aby počkalo na posily.

7. Metoda *takeOffBonus()* ovlivňuje situaci, kdy se ze vzletových políček může stíhačka dostat na bojová. Takový tah je ohodnocen pozitivním bonusem, neboť cílem vzletové strategie je co nejrychlejší start.

Výše všech dílčích priorit je dána přednastavenými konstantami, případně jejich kombinací s aktuální situací (např. rozdíl vlastních a cizích letadel v metodě *computeAgressionFactor()*).

Výše konstant byla určena dle uživatelského testování s přihlédnutím k hratelnosti. Důležitou součástí byly i jednoduché úvahy ohledně pravděpodobnosti útoku a obrany dle polohy vlastního i nepřátelského letadla.



Obrázek 4.3: Pozice vůči nepřátelskému letadlu

Tyto úvahy jsou znázorněny v obrázku 4.3, ve kterém vidíme barevně odlišené pozice vůči nepřátelskému červenému letadlu vlevo. Pokud bychom se tedy posunuli na jednu z barevných pozic, v příštím tahu by nepřátelské letadlo mohlo zaútočit.

Červená pozice znamená útok jak přímo, tak i nepřímo z obou stran. Je to tedy nejnebezpečnější pozice, do které by se hráč neměl nikdy dostat.

Oranžová pozice je útok jak přímo, tak i nepřímo z jedné strany. Pravděpodobnost sestřelu je nižší než v předchozím případě, ale stále vysoká.

Žlutá pozice znamená útok nepřímo z jedné strany. Pravděpodobnost sestřelu je výrazně nižší, takové riziko lze tedy podstoupit ve správné situaci.

4.6 Testování aplikace

Dle testovací strategie naplánované v sekci 3.7 jsem prakticky všechno testování pokryl manuálně, napsal jsem jen několik unit testů pro obecnou komponentu *RandomGenerator*.

Manuální testování jsem prováděl zejména při testování umělé inteligence, vypomáhal jsem si širokým logováním a občasným ad hoc mockem konkrétní situace. Také jsem hru testoval na různých prohlížečích, a to s úspěchem.

Vzhledem k průběhu vývoje považuji manuální testování za dostatečné, neboť jsem v nepříliš dlouhém čase dokázal celou aplikaci vyladit do funkční podoby a jsem si jist, že psaní jednotkových či jiných testů by celý vývoj významně protáhlo.

4.7 Možná budoucí vylepšení

S nadsázkou lze říci, že žádná aplikace není nikdy zcela hotová, vždy lze nalézt prostor pro zlepšení stávajícího stavu. Na virtuální podobě hry „Vzdušný souboj stíhačů“ lze jistě zapracovat v mnoha směrech, některé bych zde rád nastínil.

4.7.1 Profil hráče a přihlášení

Nyní je každý hráč anonymní entitou. Pokud bychom implementovali možnost založení profilu a následného přihlášení, otevřelo by to možnosti dalším vylepšením.

4.7.2 Síťová verze hry

První zcela zřejmým vylepšením by byla možnost hry s člověkem přes internet. Tento způsob hry by mohl mít několik forem.

První z nich by byla hra s přítelem po předchozí domluvě, kdy by bylo možné založit privátní hru a k ní se přihlásit. Řešení si představuji buď pomocí zaslání kódu ke hře externím způsobem (Skype, Facebook), nebo možností pozvání uživatele v případě implementace přihlášení (viz 4.7.1).

Druhou a pravděpodobně jednodušší formou by byla hra přes internet s náhodným soupeřem.

I síťová verze hry by přinesla další možnosti:

- Implementace chatu pro dorozumívání hráčů.
- Různé žebříčky nebo ligy, kde by se mohly hrát soutěžní zápasy.

4.7.3 Responzivita i pro mobily

Současná podoba hracího plánu je pro mobilní telefony příliš velká. Umím si ovšem představit zmenšenou verzi s menším počtem políček i stíhaček a kratším startem, která by se dala hrát i na mobilním telefonu.

4.7.4 Automatické testování a testovací strategie

Jak vidno ze sekce 4.6, automatické testování je celou kompletní oblastí, která zatím zůstala nepokryta.

Představuji si tvorbu všech potřebných unit testů, vytváření mock objektů, integračních testů, situačních testů pro umělou inteligenci, testování UI a tes-

tovací strategie pro případ přidávání funkcionalit. Prioritou by samozřejmě byla automatizace celého testování.

4.7.5 Zlepšování umělé inteligence

Prakticky nekončící možnosti rozšiřování se ukrývají ve zlepšování umělé inteligence.

Lze ji zlepšovat:

- funkcionálně – tedy psaním nového kódu, kdy zejména součinnost jednotlivých akcí stíhaček by mohla být vylepšena,
- balancováním stávajícího chování, tedy optimalizací vah jednotlivých priorit.

Také by bylo možno vytvořit několik typů hráče, který by si uživatel mohl vybrat za soupeře.

Umělá inteligence by se mohla lišit dle obtížnosti (Lehký, Těžký), tak i dle stylu hry (Agresivní, Zbabělec, Normální).

Hra je také připravena na možnost vložení i jiné než stávající umělé inteligence.

Kapitola 5

Uživatelská příručka

Virtuální podobu hry „Vzdušný souboj stíhačů“ jsem se snažil implementovat dle zásad uživatelské přívětivosti, proto věřím, že je ovládání intuitivní. Přístup k aplikaci je veřejný, protože celá hra byla nasazena na aplikační server dle zadání a obsahuje 2 obrazovky (hlavní menu a herní plochu).

5.1 Spuštění hry

Stačí mít zařízení s internetovým prohlížečem a samozřejmě připojením. Pro spuštění hry do prohlížeče zadejte adresu `https://portal.uspin.cz/planes/` a zobrazí se hlavní menu celé aplikace.

5.2 Hlavní menu

V hlavním menu je 5 možných uživatelských akcí (viz obrázek 3.6):

1. Tlačítko **„Hra proti PC“** otevírá obrazovku „Herní plocha“ s umělou inteligencí.
2. Tlačítko **„1 vs. 1“** otevírá obrazovku „Herní plocha“ proti člověku.
3. Tlačítko **„Pravidla“** otevírá popup s podrobnými pravidly hry.
4. Tlačítko **„Autorství“** otevírá popup se seznamem autorů.
5. Nenápadné tlačítko **„ztlumit/obnovit zvuky“** uprostřed dole ovládá zvuky.

Na pozadí celé obrazovky hraje ve smyčce „bojová hudba“, která se po načtení stránky automaticky zapne. Protože tuto vlastnost *autoplay* prohlížeče blokují, je nutné ji manuálně zapnout v adresní řádce prohlížeče.

5.3 Herní plocha

Po otevření herní plochy jsou ovládací prvky jednoho z hráčů zašedlé a nereagující, ten je neaktivní, zatímco aktivní hráč je má jasně barevné.

Každý tah se odehrává v následujících krocích:

1. Aktivní hráč stiskne tlačítko „Do boje“, které spouští hod kostkou. Kostky jsou vrženy a tlačítko se stane neaktivní.
2. Na kostkách se zobrazí hozené hodnoty. Klikne na kostku, o jejíž hodnotu se chce posunout, a ta se zvětší na znamení toho, že je vybraná.
3. Poté se přesune kurzorem nad stíhačku, kterou chce posunout. Zobrazí se možné akce stíhačky v políčkách herního plánu.
4. Na jednu z akcí klepne a stíhačka se přemístí, případně ještě zaútočí, pokud je to možné.
5. Vybere další kostku a postup opakuje, dokud mu nedojdou kostky nebo stíhačky, případně nenastane konec hry.
6. Pokud hra pokračuje, hráči se přepnou a druhý hráč může začít celou sekvenci znovu.

Také zde jsou použity zvuky, ale protože jsou spouštěny až na přímou akci uživatele, není třeba je zvlášť v prohlížeči povolovat.

Na této obrazovce lze také otevřít popup **„Nastavení“** tlačítkem vpravo dole. Ten obsahuje možnosti návratu do hlavního menu, ztlumení/obnovení zvuků, zobrazení popupu s pravidly a návrat do hry.

Závěr

V práci jsem se zabýval analýzou, návrhem a implementací deskové hry „Vzdušný souboj stíhačů“ do virtuální formy, konkrétně do podoby webové aplikace.

Cíle bakalářské práce jsem nejen plně splnil, ale dle mého názoru i překonal, neboť jsem musel ke hře samotné vymyslet i pravidla taková, aby hru ozvláštnila a učinila konkurenceschopnou. Dle těchto pravidel jsem poté celou hru analyzoval a poté i implementoval.

V rešeršní části jsem úspěšně prozkoumal historii deskových her v souvislosti s technologií a našel několik existujících řešení, které mi posloužily jako inspirace mého vlastního. Tato existující řešení jsem prozkoumal jak ze stránky architektonické, tak i samotné realizace vnitřní logiky hry pomocí UML diagramu tříd.

V praktické části jsem nejdříve rozhodl o základních vlastnostech celé aplikace a poté hru implementoval.

Technologicky byla aplikace realizována pomocí frameworku Vaadin jako tenký klient a byl dodržen architektonický vzor MVP. Přicházející novinkou, kterou jsem se rozhodl využít, byla technologie Progressive Web Apps, která hru přibližuje nativním aplikacím.

V implementaci byla jistě nejzajímavější částí tvorba umělé inteligence, která je nyní důstojným protivníkem lidského hráče. Velice záludnou, ale nakonec úspěšně zvládnutou, překážkou byla nutnost vymyslet z vlastních pravidel i nejlepší strategii pro hru. Tuto strategii jsem poté s drobnými úpravami pro lepší hratelnost implementoval umělé inteligenci. Také nasimulovat vizuální hru umělého hráče mělo svá úskalí, která se ovšem podařilo překonat.

Hra je veřejně přístupná na adrese `https://portal.uspin.cz/planes/`, úspěšně se tedy podařilo hru nasadit na aplikační server a zpřístupnit ji pro každého.

Nezapomněl jsem ani na navržení možností budoucího rozvoje, z nichž jako nejzásadnější vidím možnost hry přes internet s jiným hráčem.

Celou aplikaci považuji za povedené převedení deskové hry do virtuálního prostředí a doufám, že přinese uživatelům nejméně tolik zábavy, kolik přinesla mně při samotné tvorbě.

Literatura

[1] UHLÍŘOVÁ, M.: *Šachová hra v historii inteligentních strojů - android Turek vs. počítač Deep Blue*. Bakalářská práce, Masarykova univerzita, Filozofická fakulta, Brno, květen 2012, [cit. 2019-11-24]. Dostupné z: https://theses.cz/id/ji3fiy/

[2] WILLIAMS, A.: *History of Digital Games: Developments in Art, Design and Interaction*. CRC Press, 2017, ISBN 9781138885554.

[3] UNIVERSITY OF ALBERTA: Scientists Solve Checkers. *Science Daily*, červenec 2007, [cit. 2019-11-25]. Dostupné z: https://www.sciencedaily.com/releases/2007/07/070719143517.htm

[4] DICKSON, B.: All the important games artificial intelligence has conquered. *TechTalks*, červenec 2018, [cit. 2019-11-24]. Dostupné z: https://bdtechtalks.com/2018/07/02/ai-plays-chess-go-poker-video-games/

[5] GERHÁT, R.: *Vývoj internetových aplikací typu klient-server s využitím moderních přístupů*. Diplomová práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, květen 2013, [cit. 2019-11-25]. Dostupné z: https://theses.cz/id/raq0cm/

[6] WASHBURN, Michael, Jr.: An Overview of Web App Architectures. *Michael Washburn Jr.*, srpen 2017, [cit. 2019-11-25]. Dostupné z: https://michaelwashburnjr.com/an-analysis-of-web-app-architecture/

[7] JIRSA, R.: Vytvoření webové hry za pomocí MVC architektury [online]. [online], květen 2015, [cit. 2019-11-25]. Dostupné z: https://theses.cz/id/pmlsq9/

[8] TICHÝ, L.: *Tahová strategická hra v internetovém prohlížeči*. Diplomová práce, Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované in-

formatiky, Zlín, červen 2011, [cit. 2019-11-25]. Dostupné z: https://digilib.k.utb.cz/handle/10563/18245

[9] POŠVÁŘ, J.: Webová aplikace – internetová online hra Bitevní pole. [online], prosinec 2012, [cit. 2019-11-25]. Dostupné z: https://docplayer.cz/23602752-Vysoka-skola-polytechnicka-jihlava-webova-aplikace-internetova-hra-bitevni-pole.html

[10] KOZAROVÁ, L.: Čínská dáma [online]. [online], květen 2019 [cit. 2019-11-25], [cit. 2019-11-25]. Dostupné z: https://dspace.vsb.cz/handle/10084/136082

[11] TECH Tutorials: Design Chess game. *Tech Tutorials*, červenec 2018, [cit. 2019-12-8]. Dostupné z: https://www.tech693.com/2018/07/design-chess-game.html

[12] DVCO-XX: Checkers. *GenMyModel*, listopad 2017, [cit. 2019-12-8]. Dostupné z: https://app.genmymodel.com/api/repository/dvco-xx/Checkers

[13] BUCHALCEVOVÁ, A.; PAVLÍČKOVÁ, J.; PAVLÍČEK, L.: *Základy softwarového inženýrství : materiály ke cvičení*. Vysoká škola ekonomická, 2007, ISBN 987-80-245-1270-9, 222 s.

[14] ROUSE, M.: Java Platform, Enterprise Edition (Java EE). *The Server Side*, březen 2017, [cit. 2019-12-1]. Dostupné z: https://www.theserverside.com/definition/J2EE-Java-2-Platform-Enterprise-Edition

[15] BARTÍK, M.: Proč a jak psát progresivní webové aplikace. *Ackee*, leden 2017, [cit. 2019-1-12]. Dostupné z: https://www.ackee.cz/blog/proc-a-jak-psat-progresivni-webove-aplikace/

[16] VAADIN team: *Book of Vaadin - Vaadin 14 edition*. Vaadin Ltd, 2019.

[17] PECINOVSKÝ, R.: *Návrhové vzory*. Brno, Computer Press, 2007, ISBN 978-80-251-1582-4, 527 s.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

UI User interface

XML Extensible markup language

UML Unified Modeling Language

AI Artificial intelligence

MVC Model-View-Controller

MVP Model-View-Presenter

CSS Cascading Style Sheets

JS Javascript

JEE Java Enterprise Edition

PWA Progressive Web Apps

CRUD Create, Read, Update, Delete

HTML Hypertext Markup Language

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF