VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTELIGENTNÍ AUTOPILOT ZALOŽENÝ NA AGENTNĚ ORIENTOVANÉM PROGRAMOVÁNÍ

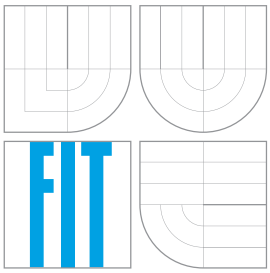
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADEK BURDA

Brno 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTELIGENTNÍ AUTOPILOT ZALOŽENÝ NA AGENTNĚ ORIENTOVANÉM PROGRAMOVÁNÍ

INTELLIGENT AUTOPILOT BASED ON AGENT-ORIENTED PROGRAMMING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADEK BURDA

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing FRANTIŠEK ZBOŘIL, Ph.D.

BRNO 2016

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Inteligentní autopilot založený na agentně orientovaném programování“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

ABSTRAKT

Práce se zabývá problematikou leteckých soubojů stíhacích letounů. Cílem práce je vytvoření chytrého autopilota s využitím agentního systému, který bude inteligentně provádět bojové manévry a taktiky v běžící simulaci. V první části práce bude představen teoretický základ leteckých soubojů - tedy zbraňové systémy, manévrování a taktiky letounů v leteckých soubojích 1 na 1, při přesile či oslabení 2 na 1, a nakonec souboje hromadné. Bude představeno agentní programování, vytvoření pravidel a jejich transformace do jazyka agentů. V druhé části bude vytvořeno grafické simulační prostředí, které bude postaveno na herním enginu JMonkey. Dále bude vytvořen agentní systém zastupující jednotlivá letadla v rámci simulace a dokumentován vlastní protokol pro síťovou komunikaci inteligentního chování s vytvořeným simulačním modelem.

KLÍČOVÁ SLOVA

Inteligentní autopilot, Multiagentní systém, BDI Agent, Letecký simulátor, Bojové manévry, Bojové taktiky, Stíhací letouny, Letecké souboje, Jason, JMonkey, Java

ABSTRACT

Thesis aims at fighter combat and maneuvring - so called Dogfighting. The purpose of this work is to create intelligent autopilot based on Agent system, eligible of executing in-air maneuvers and tactics in real-time simulation. In the first part, theoretical basis of air combat will be introduced, such as weapon systems, maneuvring and tactics in mutual combat 1 on 1, odds fight 2 on 1, and last but not least mass fights. Also agent programming will be introduced, as well as recognizing of agent rules and processes and its transformation to agent language. The second part describes building of a simple graphical simulation environment based on JMonkey game engine. Agent system maintaining every single aircraft within the simulation will be created and own network socket protocol for communication between intelligent behavior and simulation environment will be discussed and documented.

KEYWORDS

Intelligent autopilot, Multiagent system, BDI Agent, Flight simulator, Air-combat maneuvers, Air-combat tactics, Fighters, Dogfight, Jason, JMonkey, Java

BURDA, Radek *Inteligentní autopilot založený na agentně orientovaném programování*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta informačních technologií, Ústav inteligentních systémů, 2016. 44 s. Vedoucí práce byl Doc. Ing František Zbořil, Ph.D.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Doc. Ing. Františku Zbořilovi, Ph.D. za odborné vedení, konzultace, pozitivní přístup a podnětné návrhy k práci.

Brno ....

podpis autora

OBSAH

ÚVOD

Letecké souboje jsou podstatnou součástí válčení již od počátku letectví. Hlavně v Druhé Světové válce sehrály letecké bitvy dominantní roli při válečném vývoji, když se nebe nad Evropou a Pacifikem proměnilo v nejkrvavější arénu lidských dějin. V těchto dobách byla všechna letadla řízena pilotem, který seděl v kokpitu letadla, případně navigátorem nebo pomocným střelcem. V době dnešní, kdy letectví prošlo téměř stoletým technickým vývojem, se začínají vynořovat limitace lidského faktoru (reakční doba, špatné vyhodnocení, únava nebo přetížení, které je lidský organismus schopný zvládnout) a výhody faktoru strojového (inteligentní modely, rychlost výpočtů, velikost hardware). Ovládání letadel se přenechává inteligentním autopilotům, přičemž člověk kontroluje pouze správné fungování stroje.

V dopravním letectví je autopilot součástí přístrojové desky už minimálně půl století. Tento autopilot není inteligentní, umí pouze držet kurz, rychlost a výšku na základě vstupů od uživatele a navigačního systému. Naproti tomu autopiloty, které se objevují v leteckých bojových simulátorech, za inteligentní považovat můžeme. Bojový autopilot sám iniciativně upravuje rychlost, provádí letové manévry, útočí na cíl a drží se své skupiny. Pro implementaci takovéhoto chování se výborně hodí model Agentního systému, neboť BDI Agent sám o sobě je inteligentní entita, která je schopná fungovat v dynamickém prostředí, komunikovat s okolními agenty, upravovat svoje plány a cíle a iniciovat tak bojové manévry.

Práce je stukturována do několika tematicky odlišných celků. V první části práce bude představen koncept agentního programování, který v umělé inteligenci získává v posledních letech významně na popularitě. Bude popsán model BDI Agenta včetně jeho modelu výpočtu a praktického usuzování. Nakonec budou v rámci první kapitoly popsány základy programování v jazyce Jason, což je interpret dialektu agentního jazyka AgentSpeak.

Druhá část práce se zabývá teorií ohledně leteckých soubojů. Bude popsán vývoj strategií a taktik průběhů souboje od dob prvních leteckých soubojů až po současnost. Nedílnou součástí leteckých soubojů je bojová výzbroj, takže budou zběžně popsány zbraňové systémy nasazované v boji, jejich hlavní komponenty a aspekty správného použití. Teoretická část bude uzavřena popisem základních letových manévrů a popisem průběhu vzájemných soubojů.

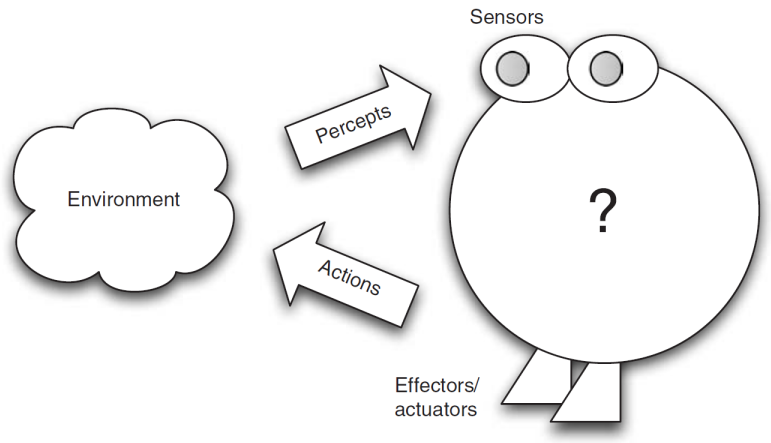
Poslední část práce je věnována návrhu a implementaci agentního chování a frameworku pro agentní letecké souboje. Bude popsán protokol pro síťovou komunikaci mezi agentním prostředím a serverem, stejně tak bude popsána interakce mezi agentním chováním a agentním prostředím na klientské straně aplikace.

1 AGENTNÍ PROGRAMOVÁNÍ

Myšlenka Agentně-orientovaného programování byla poprvé představena v roce 1993 [1]. Přestože agentně orientované programování je poměrně nový přístup, oblast multi-agentních systémů [2] se v posledním desetiletí stává cílem zájmů čím dál většího množsví výzkumných skupin. Výzkumníci, pohybující se kolem agentních systémů, se domnívají, že agentní přístup v následujících letech velmi výrazně ovlivní návrh a fungování výpočetních systému, neboť stále více systémů je situováno v dynamických, nepředvídatelných prostředích [4].

1.1 Představení Agenta

Pojmem Agent máme na mysli systém, který je *situován* v nějakém *prostředí*. Agenti jsou schopni vnímat své okolí (pomocí *senzorů*), mohou provádět *akce*, kterými (pomocí *efektorů*) mění nebo nějakým způsobem ovlivňují své okolí. Prostředí agenta může být fyzické (v případě robotů v reálném světě) nebo softwarové (v případě softwarově simulovaného prostředí).



Obr. 1.1: Agent vnímající okolí senzory a ovlivňující okolí efektory, převzato z [8].

Při vytváření agenta vyplívá na povrch zásadní otázka: jak *rozhodnout*, *co udělat*, na základě informací získaných ze senzorů? Rozhodování agenta je dosaženo sestavením *plánů*, které budou představeny dále v textu.

Vedle schopnosti vnímat podněty z okolí a provádět akce za účelem změny prostředí by měl agent být [9]:

Autonomní
Je důležité zmínit, co autonomie v rámci agentních systémů znamená, neboť pojem samotný vystihuje poměrně široké spektrum možností. Na jedné straně tohoto spektra se nachází například textový nebo tabulkový editor, který disponuje takřka nulovou autonomií. Vše, co se s aplikací děje, je řízeno námi (člověkem) - například vložení textu, kliknutí na prvek uživatelského rozhraní atd. Na druhé straně spektra se nacházíme my (uživatelé, lidé), kteřížto jsme zcela autonomními jednotkami. Přestože společnost nás v našem rozhodování částečně omezuje a směruje, můžeme se sami rozhodnout, čemu chceme věřit nebo co chceme dělat. Máme vlastní cíle a vlastní řád. Všechny tyto vlastnosti jsou přirozeně naše a nikdo nám je explicitně nediktuje. Autonomie v rámci agentů leží někde v rozcestí těchto dvou extrémů. Agentovi jsou *delegovány* cíle, on následně autonomně rozhoduje, jak se co nejlépe zachovat, aby těchto cílů dosáhl.Agent nemá tedy naprosto svobodnou vůli. Je omezen cíli, které jsou mu delegovány, a plány, které definují, jak cílů dosáhnout. Autonomie je dosaženo pomocí agentova nezávislého rozhodnutí, jak poskládat jednoduché plány do komplexních za účelem dosažení cíle.

Proaktivní
Proaktivností máme na mysli schopnost osvojit si cílem řízené chování. Pokud je agentovi delegován cíl, očekáváme, že se agent iniciativně bude snažit cíle dosáhnout.

Reaktivní
Být *reaktivní* znamená *reagovat* na změny prostředí. Ve skutečném světě nejde nic přesně podle plánu. Plány jsou často narušovány, ať už záměrně nebo neplánovaně, a my musíme na tyto změny adekvátně reagovat. Agent reaguje na změnu vytvořením alternativního plánu.

Sociální
Zde je interakce na sociální úrovni myšlena tak, že agenti jsou schopni *kooperovat* a *koordinovat* svou činnost s ostatními agenty za účelem dosažení cíle. Agenti jsou schopni mezi sebou sdílet své vědomosti na úrovni *představ*, *plánů* a *cílů*.

1.2 Agentně orientované programování

Agenty považujeme za *autonomní* entity, protože se dokážou sami starat o množinu svých vnitřních odpovědností. Agenti jsou také *interaktivní* entity, které jsou schopné zasílat a zpracovávat široké množství zpráv. Tyto zprávy mohou způsobit invokaci plánů nebo akcí, ale také informovat ostatní agenty o důležitých událostech, dotazovat se na informace

nebo odpovídat na dřívější dotazy. Protože agenti jsou autonomní entity, mohou komunikaci zahájit dle svého uvážení a na zprávu odpovědět libovolným způsobem [5].Je třeba si uvědomit, že agentně orientované programování je, stejně jako programování objektové nebo funkcionální, pouze jedním z mnoha přístupů strukturování programů. Použití agentního přístupu vede k větší decentralizovanosti programu. Podívejme se teď agenta jako objekt, tak jak ho známe z objektově orientovaného programování - objekt obsahuje atributy a metody, které jsou invokovány zasláním zprávy. Objekt samotný je pasivní entita a když chceme, aby proběhl výpočet, musíme zavolat konkrétní metodu přes rozhraní objektu. Oproti tomu na agenta můžeme nahlížet jako na aktivní objekt, kterému aplikační vývojář pouze vytvoří představy, plány a cíle, agenti pak sami provedou žádoucí výpočet.Padgam a Winikov nahlíží v [7] na agenty jako na snaživé a motivované zaměstnance. kdežto na objekty nahlíží jako na zaměstnance spolehlivé, ale pasivní. Pasivní zaměstnanec (objekt) za námi bude chodit s každým problémem, což nám jako managerům (programátorům) vytváří zbytečnou časovou režii. Naproti tomu motivovanému zaměstnanci (agentovi) stačí zadat cíl a on se sám postará o jeho splnění. Takového spolehlivého zaměstnance můžeme pak nechat pracovat i z domu, čímž přispějeme k decentralizaci systému.

1.3 BDI Agent

Model *belief - desire - intention* (BDI) vychází z modelu praktického usuzování člověka, jak ho definoval filosof Michael Bratman [11]. Samotný model BDI vznikl v polovině 80. let dvacátého století jako výzkumný projekt na Stanfordově výzkumném institutu[10]. Architektura BDI nahlíží na agenta tak, jakoby měl svůj vlastní stav mysli. Takže když mluvíme o systému *Belief - Desire - Intention*, myslíme tím opravdu logickou analogii *Představy - Přání - Záměry*.Jednotlivé struktury BDI jsou definovány následovně:

Beliefs - představy
Představy jsou informace o okolním světě, kterými agent disponuje. Tyto informace mohou být neaktuální, nepřesné nebo dokonce i lživé. Záleží pak pouze na agentovi, jak tyto svoje představy bude interpretovat.

Desires - přání
Přání značí cíle, kterých by agent mohl chtít dosáhnout. Fakt, že agent má nějaký cíl, nemusí znamenat, že bude chtít tohoto cíle dosáhnout. Klidně se může stát, že se dva různé cíle vzájemně vylučují. Přání jsou často označovány jako *možnosti* agenta.

Intentions - záměry
Záměry rozumíme zvolené způsoby, jak dosáhnout přání. Agent začíná s nějakým přiděleným cílem, následně se podívá na své možnosti, jak cíle dosáhnout, a nakonec vybere ty možnosti, které jsou v aktuální situaci *aplikovatelné*. Z takto zvolených možností se stávají

záměry. Záměry mohou mít hierarchicky uspořádané podcíle, které se na nejnižší úrovní skládají z atomických přímo proveditelných záměrů - akcí.

1.3.1 Praktické usuzování

Nyní už víme, že klíčovými datovými strukturami agentů jsou *představy*, *přání* a *záměry*. Jak ale agent na základě těchto informací provede správnou akci? Disciplína, která se věnuje takovémuto rozhodování, se nazývá *praktické usuzování*. Cílem praktického usuzování je ohodnocení relevantních možností, tedy zvážení pro a proti jednotlivých možností, přičemž všechny relevantní možnosti vycházejí z agentových přání/hodnot/starostí, kterým agent věří [12].Praktické usuzování člověka se skládá ze dvou hlavních procesů: *zvažování* (stanovení cílů, kterých chceme dosáhnout, t.j. stanovení záměrů); a *rozhodnutí plánu* (rozhodnutí posloupnosti akcí, abychom splnili dříve zvážené záměry). Stejně tak je tomu u BDI agenta. Jádrem takovéhoto rozhodovacího procesu je *plánovač* (Means-End Reasoner), který na základě **1) agentových přání, cílů a záměrů**, **2) agentově znalosti (představách) o stavu okolního prostředí** a **3) akcí dostupných agentovi**, sestaví *plán*. Plán je posloupnost akcí, kterou agent následuje, aby dosáhl svých záměrů.

1.3.2 Model výpočtu praktického BDI usuzování

Představme si, že chceme implementovat agenta, který bude schopný praktického usuzování tak, jak jsme zmínili v textu výše. Výsledný program by rozhodně musel obsahovat následující podprocesy:

1. Zaznamenání změn v okolním světě, upravení představ podle těchto změn.
2. Zvažování a následné rozhodnutí, jakých záměrů chce agent dosáhnout.
3. Použití plánovače pro sestavení plánu pro dosažení záměrů.
4. Provedení plánu.

Model výpočtu BDI agenta potom vypadá následovně:

```
// inicializace
initialize_state();
// hlavní smyčka
repeat
  // výpočet proveditelných možností
  options := option_generator(event_queue);
  // výběr vhodných možností
  selected_options := deliberate(options);
  // upravnení záměrů
  update_intentions(selected_options);
  // sestavení plánu
```

```
  means_end_reasoning();
  // provedení plánu
  execute();
  // obsloužení externích událostí
  get_new_external_events();
  // vyčištění provedených a neproveditelných záměrů
  drop_successful_attitudes();
  drop_impossible_attitudes();
end repeat
```

Kód 1.1: Řídící smyčka BDI agenta. Upraveno z [6, 17].

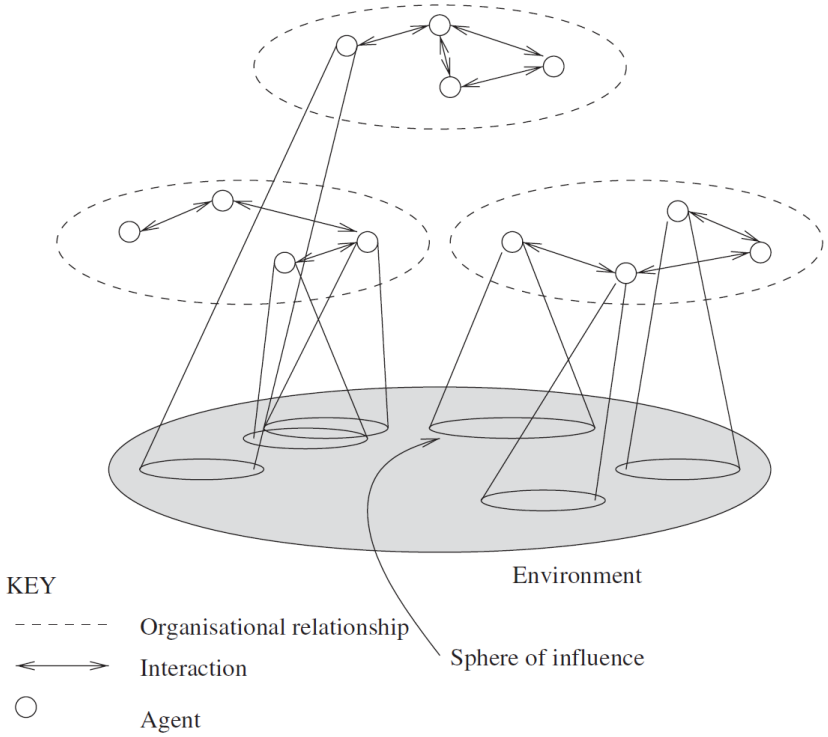
Na začátku každého cyklu generátor možností přečte frontu událostí a vrátí seznam agentových možností. Následně deliberator vybere vhodnou podmnožinu těchto možností a přidá ji do záměrů. Následně plánovač sestaví plán, který vede k dosažení záměrů. Nakonec interpret vyhodnotí, které záměry byly splněny a které už splnit nelze, a odebere je z odpovídajících struktur. Rozhodování agentů je řízeno BDI logikou. Pro matematickou definici BDI logiky zde čtenáře odkáži na literaturu [13] , která se této problematice dopodrobna věnuje.

1.4 Multiagentní systém

Doposud jsme se o agentech bavili jako o izolovaných jednotkách nacházejících se v prostředí. Ve skutečnosti je však systém, skládající se z pouze jednoho agenta, naprostá vzácnost. Ve většině případů se v prostředí s agenty nacházejí ostatní agenti, kteří tak dohromady vytváří multi-agentní systém [10]. Na obrázku 1.2 je znázorněna struktura multi-agentního systému. Ve spodní části je znázorněno celé prostředí, avšak každý agent má kolem sebe *sféru vlivu* na prostředí, se kterým nějak interaguje nebo ho dokáže částečně manipulovat. Nad prostředím se nacházejí samotní agenti, kteří mezi sebou mohou mít různé organizační vztahy (například mohou být dva agenti na stejné úrovni, kteří se budou zodpovídat agentovi v hierarchii nadřazenému nad nimi). Agenti sdílejí vzájemně informace o ostatních, i když v rámci prostředí nemusí mít kompletní přehled o ostatních agentech v systému.

1.5 AgentSpeak

Pro implementaci multi-agentního systému potřebujeme jazyk, který budou jednotliví agenti umět interpretovat. Mezi takovéto jazyky (platformy) patří například JADE, SARL nebo AgentSpeak. Pro účel práce jsem zvolil poslední zmíněný - Agent speak, konkrétně Jason, což je interpret dialektu agentního jazyka AgentSpeak. V následující kapitole budou představeny základy programování BDI agentů v tomto jazyce. Pro podrobnější informace



Obr. 1.2: Struktura multi-agentního systému [16].

---

o jazyce Agentspeak (Jason) zde čtenáře odkáži na literaturu [10], kde programování v Jasonu popisují přímo jeho tvůrci. Při představování nového jazyka se stalo zvykem podat čtenáři ukázkový příklad kódu *Hello world*, aby nabyl první dojem o jazyce, se kterým se potýká. Program helloworld v Jasonu vypadá následovně:

```
// initial beliefs
started.
// triggering plan execution
+started <- .print(''HelloWorld!'').
```

Definice agenta se skládá ze dvou částí: agentovy **počáteční představy** (potenciálně počáteční cíle) a agentovy **plány**. Agentovi je tedy hned při definici vytvořena představa *started*, tečka (‘**.**‘) zde zastupuje syntaktický oddělovač stejně jako středník v Javě nebo C. Druhý řádek značí samotný plán agenta. V tomto případě plán interpretujeme následovně: “Jakmile uvěříš, že *started*, vytiskni text ‘HelloWorld!’. ”

Plán v AgentSpeaku se skládá ze 3 částí. První částí je **spouštěcí podmínka**, kterou v našem případě zastupuje *+started*. Symbol ‘+’ v našem plánu značí “pokud nabydeš přesvědčení”, takže celá spouštěcí podmínka znamená “pokud nabydeš přesvědčení *started*”.

Před provedením plánu agent ověřuje **kontext plánu**. Nicméně v případě helloworld je kontext prázdý, takže plán je vždy proveditelný. Poslední částí je samotné **tělo plánu**. Za zmínku stojí ještě interní akce *.print()*, což vypadá jako představa, ve skutečnosti se ale jedná o *interní akci*. Interní akce začínají tečkou a na rozdíl od akcí externích nemění prostředí.

```
// syntaxe plánu složeného ze všech tří částí
událost: kontext <- tělo
```

Událost může vzniknout, jak jsme již zmínili, buď změnou představ nebo změnou cílů. Při změně pak rozlišujeme dvě operace: přidání (+) a odebrání (-) představy. Když nastane událost, která odpovídá hlavičce některého z plánů (a kontext je platný), říkáme, že plán je *relevantní*.

Kontext plánu udává, kdy je plán aplikovatelný. Z aplikovatelných plánů agent vybírá ten, který provede.

Tělo je posloupnost příkazů, které mají splnit příslušný cíl. Tělo může obsahovat posloupnost cílů i plánů. Pokud agent narazí na nějaký cíl, snaží se najít plány, které mu pomohou cíl uskutečnit.

Příklad plánu implementovaného v AgentSpeak:

```
    // konkrétní příklad plánu
+!drink : has_drinks(0) <- !get_drink; !drink.
```

Událost je na příkladu s pitím spuštěna přidáním praktického cíle *drink*. V případě **kontextu** agent věří, že nemá co pít. Pokud dojde k přidání cíle *drink* a představy *has_drinks(0)*, plán je aplikovatelný. **Tělo** pak obsahuje cíle, které říkají, že má agent získat pití a následně se napít.

2 TAKTIKA A MANÉVROVÁNÍ STÍHACÍCH LETADEL

Taktika stíhacích letadel se stala s prvním nasazením armádních letounů nedílnou součástí leteckých soubojů. Již v první světové válce vznikaly pravidla a předpisy, jak se ve vzduchu orientovat a jak vést letecký souboj. Nejznámějším takovým manuálem je soubor pravidel napsaný prvním německým leteckým esem Oswaldem Boelcke v roce 1916 - Dicta Boelcke[15].

2.1 Dicta Boelke

Dicta Boelke, aneb Boelckeho doktrína, je seznam 8-mi základních taktických pravidel pro vzdušný souboj, kterými se řídilo a dodnes řídí mnoho stíhacích pilotů.

1. Pokuste se zajistit si výhodu před samotným útokem.
Mezi hlavní výhody v První světové válce patřily rychlost, výška, moment překvapení a početní výhoda.

Rychlost: Pilot s rychlejším letadlem měl kontrolu nad průběhem souboje. Měl možnost přerušit bitvu a stáhnout se, pomalejší stroj ho pak nemohl dohnat. Pomalejší stroj se, v případě obrany, stáhnout naopak nemohl, musel setrvat do konce souboje.

Výška: Díky výškové výhodě nad nepřítelem mohl pilot rozhodnout kde a jak se bude vzdušný souboj odehrávat. Mohl naletět na svého protivníka ve vysoké rychlosti pro rychlý útok a následný únik. Pokud byla výhoda na straně nepřítele - například početní - pilot se mohl pokusit o útěk s markantním náskokem. I nejlepší stíhací letouny první světové války měly, v porovnání s pozdějšími modely, velmi malou stoupavost. Výška znamenala těžce získanou potenciální energii, kterou musel pilot dobře využívat.

Moment překvapení: Zahájení střelby ještě před tím, než je nepřítel připraven opětovat palbu, byl nejbezpečnější a preferovaný typ útoku. Většina vzdušných vítězství byla dosažena prvním průletem. Bez vševidoucích zařízení jako jsou radary, se mohl pilot k jeho protivníkovi přiblížit nepozorovaně s pomocí mraků, mlhy nebo dokonce protivníkových křídel a ocasního trupu. Zejména sluneční záře poskytovala osvědčenou skrýš.

Znalosti pilota: Znalost silných, slabých stránek, možností vlastního a nepřátelského letadla, byla kritická. Kdo byl rychlejší, kdo byl obratnější, kolik letadel se účastnilo souboje atd. Boelcke na své studenty apeloval, aby nezahajovali útok, dokud neměli dostatečně zajištěné poziční výhody [1]

2. Nikdy nezastavujte rozjetý útok.
Piloti začátečníci měli tendenci ze strachu přerušit útok a dát se na útěk. Tatko akce nevyhnutelně vystavila začátečníkův ocas zkušenému nepříteli, což pro nepřítele znamenalo

[1] Jeden z Boelckeho studentů, Manfred von Richthofen (více známý pod přezdívkou Rudý Baron), se toto pravidlo naučil velmi dobře a později se stal nejlepším esem První Světové války [16].

jednoduché vítězství.

3. Palte pouze na blízkou vzdálenost a to pouze pokud se nepřítel nachází ve Vašich mířidlech.
Častou chybou nezkušených pilotů byla palba na první nepřátelský stroj, který v dálce zahlédli. Střely vypálené ze vzdálenosti 1km měly minimální šanci zásahu. Zvuk střelby navíc často znamenal přerušení momentu překvapení, a tak měl nepřítel čas zareagovat patřičným obranným manévrem. Dalším důvodem pro zvýšení přesnosti střelby byla omezená kapacita munice. Většinou s sebou měl střelec pouze několik set nábojů. Takovéto množství vystačilo asi jen na dobu jedné minuty nepřetržité střelby.

4. Snažte se neustále mít nepřítele na očích.
Pravidlo "Mít nepřítele neustále na očích," se může zdát býti zjevným, ale je třeba ho zmínit. Ve zmatku a adrenalinu vzdušné bitvy bylo jednoduché ztratit cíl z očí. Správně by mělo pravidlo znít následovně: *Nikdy nepředpokládejte, že víte, kde Váš soupeř je nebo bude.*

5. Zásadou všech útoků je útok zezadu.
Pokud nepřítel letí při kulometné přestřelce křižmo přes pilotův zaměřovač, musí pilot počítat s rychlostí letadla, rychlostí letu kulek, s rychlostí střelby a s tím vším střílet patřičně před nepřítele. Přímé útoky - ať už zepředu nebo zezadu - potřebují minimální nebo dokonce žádné předsazení zaměřovače, pilot tím pádem může vystřílet větší dávky s větší přesností. Nicméně přímý útok zepředu vystavuje pilota kulometům nepřítele, útok zezadu je efektivnější a bezpečnější. Zadní kulomety byly zavedeny právě jako opatření v reakci na převážnou většinu útoků zezadu.

6. Pokud na Vás nepřítel nalétává, leťte proti útoku.
Toto pravidlo úzce souvisí s výše zmíněným pravidlem 5. Instinktivní chování nováčků bylo utíkat z boje boje, pokud na ně z výšky nalétává útočník. Útěk pak znamenal nastavení ocasního trupu nepříteli s katastrofickými následky. Pokud ale pilot jde proti útoku čelem, může nepřítele přinutit k obraně nebo přinejmenším hrát vyrovnaný souboj, i když by měl pilot stoupat a ztrácet rychlost. Tento tah znamená relativní vyšší rychlost přibližování letadel a tak menší časové okno pro protivníkovu palbu. Pokud navíc oba setrvají, na další útok se musí obracet a výšková výhoda mezi piloty se obrací.

7. Nad nepřátelským územím myslete na směr ústupu.
Pokud se pilot rozhodl uniknout od nepřátelské přesily nebo klesal s poškozeným letadlem, bylo zásadní, aby nemrhal časem a letěl správným směrem. Mnoho pilotů se zřítilo na nepřátelském území právě kvůli tomu, že byli zmatení a ztratili přehled o směru návratu.[2]

[2]Dnes má pilot na desce informace o pozici i směru, v První Světové válce se ale navigoval pouze podle vizuálních bodů na zemi nebo horizontu a polohy Slunce.

8. Rada pro letky: Je lepší útočit ve skupinách po čtyřech nebo po šesti. Pokud se souboj rozpadne do několika samostatných soubojů, je důležité, aby spřátelení piloti útočili na různá letadla.
V První Světové válce se vzdušné souboje většinou odehrávaly ve formátu 1 na 1. Známé esa jako Pegoud, Garros, Boelcke a Immelmann likvidovaly oblohu samostatně. S rostoucími počty letadel na obloze začaly některé letky cestovat spolu kvůli vzájemné ochraně. Létání ve skupině umožnilo vedoucímu letky se plně soustředit na útok na cíl, zatímco jeho podporující piloti ve formaci zastávali obranné činnosti. Později se týmová spolupráce stala klíčem k úspěchu a přežití vzdušných soubojů.

2.2 Zbraně stíhačů

Stíhací letouny existují z důvodu ničení nepřátelských stíhačů nebo bombardérů. Na samotné letadlo se tak dá nahlížet jako na prostředek, který má za úkol pouze doručit zbraň na správné místo ve správný čas. Přestože zbraně stíhačů prošly v průběhu let ohromným vývojem, každá má specifické požadavky pro úspěšné bojové nasazení. Mezi ně patří efektivní dostřel, míření, relativní pozice stíhače a cíle a mnoho dalších faktorů. Cílem manévrování a bojových taktik je splnění těchto požadavků pilotem a znemožnění téhož nepřátelskému pilotovi [16, str. 6].

Pro modelování bojových taktik je třeba modelovat zbraňové systémy, v následující kapitole budou tedy popsány 2 základní typy zbraní, jejich výhody a nevýhody.

2.2.1 Střelné zbraně

"The most important thing in fighting was shooting, next the various tactics in coming into a fight and last of all flying ability itself."

Lt. Colonel W. A. "Billy"Bishop, RAF
Vedoucí eso Královského letectva WW-I
72 Vítězství

Střelné zbraně byly až do nedávna nejpoužívanějším typem zbraně pro vzdušné souboje. Rozdělujeme 2 základní typy kulometů: pevné a pohyblivé. Pevné kulomety jsou nainstalovány stacionárně na přední stranu letadel a jejich zaměřování se provádí natočením celého letounu. Střílí se pak pouze dopředu. Pohyblivé kulomety (kulometné věže) jsou operovány člověkem (dnes už se o míření může starat stroj) a mohou být natáčeny do všech směrů kolem letounu.

Pevné dopředně mířící kulomety jsou vhodné pro malé manévrovatelné letadla. Snadno se upevňují a nevytváří za letu velký odpor, takže mají menší vliv na výkon. Pohyblivé zbraně většinou potřebují vlastního operátora navíc vedle pilota, což přidává na velikosti a hmotnosti letadla. Kvůli těmto důvodům se pevné kulomety staly dominantními pro

útočné stíhací letouny, zatímco pohyblivé sloužily pro obranu větším, méně obratným letounům, pro obranu.

Standardní vybavení stíhačů ve WW-I zahrnovalo 2 dopředu mířící pevné kulomety ráže 30mm často spojené se synchronizátorem, který umožňoval střílet přes vrtuli pohonu. Růst výkonu motorů ve WW-II umožnil letadlům více zbrojit. Začalo se používat více munice, větší projektily, vyšší dostřel a rychlost střel, vyšší kadence, explozivní náboje. Některé dvojice těchto faktorů spolu nepřímo souvisí - např. vyšší hmotnost náboje znamená menší dostřel a kadenci střelby. Hledá se optimální poměr mezi těmito faktory a optimální poměr se přesouvá k těžším nábojnicím. Další neznámou v rovnici je zranitelnost cíle - vyšší kadence menších střel znamená větší pravděpodobnost zásahu cíle, zatímco větší náboj znamená větší poškození.

V padesátých a šedesátých letech minulého století se přestávaly montovat kulomety do stíhačů. Armáda měla dojem, že kvůli vysokým rychlostem proudových stíhačů a těžkému obrnění nových bombardérů, se staly kulomety a kanóny zastaralými. Hodně stíhačů v této době nemělo střelné zbraně vůbec. Zbraňová sada stíhačů pro vzdušné souboje se skládala pouze z neřízených raket a potažmo raket řízených (které budou diskutovány v následující kapitole). Tento trend se obrátil v letech sedmdesátých, když bojové zkušenosti ukázaly přídavnou hodnotu kulometů a nedostatky některých pokročilých zbraní.[3]

"Suddenly you go into a steep turn. Your Mach drops off. The MiG turns with you, and you let him gradually creep up and outturn you. At the critical moment you reverse your turn. The hydraulic controls [F-86] work beautifully. The MiG [-15] cannot turn as readily as you and is slung out to the side. When you pop your speed brakes, the MiG flashes by you. Quickly closing the brakes, you slide onto his tail and hammer him with your 50's."

Colonel Harrison R. "Harry"Thyng, USAF
WW-II a Korejský Konflikt
10 Vítězství

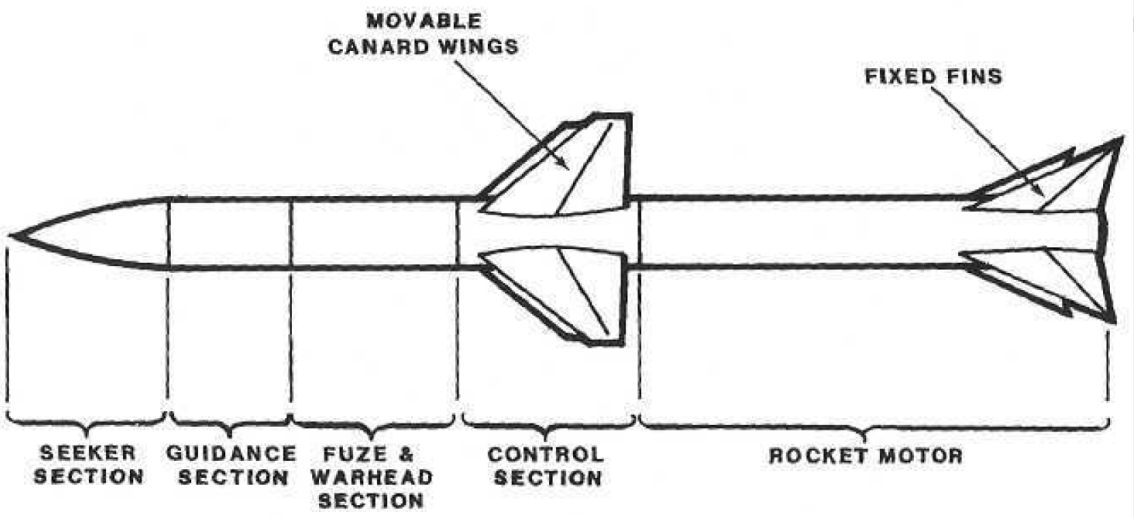
Tabulka 2.1 obsahuje statistiky střelných zbraní, které byly v průběhu let využívány americkým válečným letectvem. Výrazným ukazatelem technologického vývoje střelné zbraně je snižování hmotnosti projektilů a kadence střelby za minutu.

V tabulce je váha projektilu modelována jako faktor W_f. Destruktivní síla neexplozivních nábojnic se rovná kinetické energii - tedy jedna polovina násobku hmotnosti a druhé mocniny rychlosti. Pro výpočet přesné hodnoty energie by se měla použít rychlost nábojnice při nárazu na cíl, pro zjednodušení je ale použita rychlost při výstřelu z hlavně. Smrtelnost zbraně je počítána jako násobek destruktivní síly projektilu a počtu zásahů. F_L (Smrtelnost zbraně) slouží jako hodnota pro porovnání podobných zbraní různého kalibru a kadence. Hodnota F_L pro kanóny je značně podhodnocená, protože destruktivní síla kanónů vychází hlavně z jejich výbušných nábojnic.

[3]Rakety například nebylo možné odpálit, pokud bylo nepřátelské letadlo příliš blízko. Trosky by tak rozmetaly i letadlo pilota, který rakety vystřelil.

Type	Operational Date	Bullet Weight (lbs)	Rate of Fire (rounds/min)	Weight of Fire W_F (lbs/min)	Muzzle Velocity V_M (ft/sec)	Lethality FL (W_f x VA x ID^3)
Machine Guns						
.30-cal M2	1929	.02	1,200	25	2,600	1.7
.50-cal M2	1933	.10	800	81	2,810	6.4
.50-cal M3	1947	.10	1,200	121	2,840	9.8
Cannon						
20-mm M2	1941	.30	650	196	2,850	15.9
20-mm M3	1944	.30	800	241	2,750	18.2
20-mm M39	1953	.22	1,500	332	3,330	36.8
20-mm M61	1957	.22	6,000	1,330	3,300	144.8
37-mm M4	1941	1.34	135	181	2,000	7.2

Obr. 2.1: Vlastnosti kanónů a kulometů používaných ve stíhacích letadlech Americké armády. Převzato z [16]



Obr. 2.2: Obecné schéma komponent naváděné rakety

2.2.2 Naváděné rakety

Naváděnými raketami v kontextu leteckých soubojů se myslí takové rakety, které jsou schopné měnit svoji trajektorii letu podle manévrů zaměřeného cíle. Následující kapitola popisuje rakety vzduch-vzduch (AAMs), přičemž rakety typu země-vzduch (SAMs) fungují téměř stejně.

Na obrázku 2.2 je znázorněná naváděná raketa a její běžně používané subsystémy. Každá komponenta má určité limity, ve kterých funguje optimálně. Mimo tyto limity funguje se sníženou účinností nebo vůbec. Vyřazením jedné komponenty dojde tedy k vyřazení celého zbraňového systému [17].

Pohonná jednotka

obsahuje raketový motor a palivo. Motor definuje rychlost rakety, množství paliva znamená doletovou vzdálenost a velikost rakety. Obecně je cílem vytvořit raketu, která

má maximální rychlost a minimální hmotnost (tedy omezené množství paliva).

Řídicí jednotka
umožňuje raketě manévrovat na základě vstupů z naváděcí jednotky. Rakety jsou většinou řízeny aerodynamicky křidélky jako běžné letadlo, mohou ale mít dodatečné trysky, které napomáhají manévrovacím schopnostem. Pilot může při obraně raketu přinutit do takového manévru, který raketu vychýlí ze pronásledovacího kurzu a raketa vyčerpá palivo, než doletí k cíli.

Naváděcí jednotka
poskytuje vstupy řídicí jednotce, která manévruje raketu za účelem zasáhnutí cíle. Během letu rakety je z odpalovací platformy monitorována pozice rakety i cíle a tak upravována trajektorie letu rakety potřebná pro úspěšný zásah.

Proti cílům v pohybu se používají 3 typy naváděni: pasivní, semi-aktivni, aktivní. Nejjednodušší z nich - Pasivní navádění - používá pro navádění emise vycházející z cíle (zvuk, rádio, radar, teplo, světlo). Semi-aktivní naváděcí systémy používají odraz energie od cíle. Tato energie (radar nebo laser) je vyzařována ze zdroje, který není součástí rakety, ale je součástí odpalovacího zařízení. Při aktivním navádění raketa sama ozařuje a sleduje cíl.

Na obrázku 2.3 jsou znázorněny trajektorie letu naváděné rakety při rychlosti 1.5 násobku rychlosti letu cíle. Rakety jsou vystřeleny v čase 1, cíl letí po přímce konstantní rychlostí.

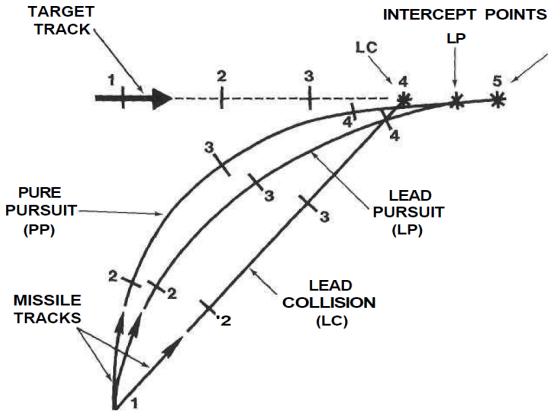
Raketa používající naváděcí trajektorii PP (Pure Pursuit - Přímé pronásledování) nastavuje svůj rychlostní vektor tak, aby pořád směroval na cíl. Výsledkem je let po zakřivené trajektorii a následné pronásledování letadla. Ke střetu rakety a cíle dojde v bodě 5.

Při LP (Lead Pursuit - Předsazené pronásledování) raketa míří před cíl ve směru jeho vektoru letu. Tato trajektorie skončí taky pronásledováním ocasu letadla, ke střetu dojde ale o pár okamžiků dříve - mezi body 4 a 5.

Nejefektivnější ze zmíněných trajektorií je LC (Lead Collison - Předsazená kolize), která je vyobrazena jako přímka se střetem v bodě 4.

Senzor
je zodpovědný za sledování cíle a poskytnutí nezbytných informací systému naváděcímu. Maximální vzdálenost zpozorování cíle senzorem znamená efektivní dosah naváděcího systému. Senzor vnímá cíl díky reflexním charakteristikám cíle, jako je velikost cíle, materiál povrchu, tvar a aspekt.

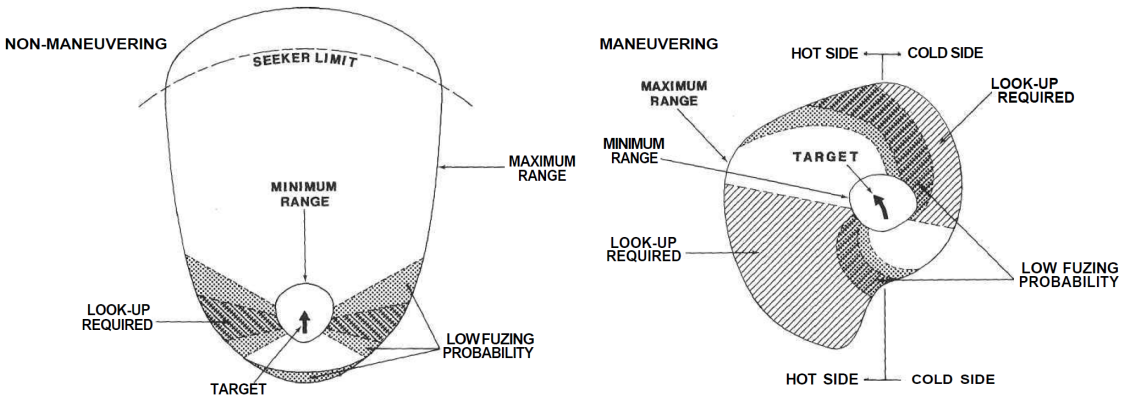
Nejběžněji používané senzory jsou senzory teplotní. Senzory jsou pak citlivé na teplo (infračervené záření) vycházející z trysek motoru. Takovéto senzory jsou limitovány šumem prostředí - teplo vycházející ze slunce, odrazy od vody a sněhu nebo mraků. Při vysokém šumu a nízkém výkonu motoru cíle pravděpodobnost střetu rakety a cíle výrazně klesá.



Obr. 2.3: Stíhací křivky naváděné rakety. Převzato z [16]

Dále se používají laserové a radarové senzory, které využívají odrazu signálu od cíle. Například radary mají problém rozpoznat letadlo letící nízko nad zemí, kvůli velkému šumu odrazů od povrchu Země.

Roznětka a bojová hlavice Účelem roznětky je zajištění detonace bojové hlavice, která způsobí maximální možné poškození cíle. Používají se roznětky kontaktní, časované, pravděpodobnostní nebo s manuální aktivací. Pokud roznětka odpálí hlavici ve špatný okamžik, je raketa neškodná. Roznětka také potřebuje po odpalu krátkou dobu pro inicializaci (může být aktivována zrychlením při odpalu), tím vzniká minimální vzdálenost, po kterou raketa není účinná.



Obr. 2.4: Běžná obálka naváděné střely. Vlevo cíl, který nemanévruje, vpravo manévrující cíl. Převzato z [16]

Každá raketa má svou obálku, ve které lze znázornit její efektivitu. Obrázek 2.4 takovouto obálku znázorňuje při pohledu z vrchu na cíl. Levé straně obrázku (cíl, který neuhýbá) je cíl znázorněn šipkou směřující směrem nahoru ve směru letu. Vnější obálka znázorňuje

maximální dolet rakety. Je zjevné, že z přední polo-sféry je maximální dostřel výrazně vyšší, neboť letadla se k sobě přibližují. V obálce je znázorněna oblast, ve které rozbuška pravděpodobně selže (LOW FUZING PROBABILITY). Oblast označená jako LOOK-UP REQUIRED znamená, že pilot musí vizuálně ověřovat cíl např. laserem, kvůli zmatení senzoru rakety a vysokému šumu prostředí.

Druhá strana obrázku zobrazuje obálku cíle, který provádí levotočivý manévr, zatímco raketa směřuje na cíl. Na obrázku je popsána horká strana a chladná strana pro znázornění směru otáčení. Manévrovací aerodynamická obálka maximálního dosahu je silně asymetrická, dosah na horké straně je daleko vyšší, než na straně chladné. Cíl letí naproti střele tak, aby jej trefila z levé strany, zároveň tak ulétává raketám, které by jej měly trefit ze strany pravé. Výběrem správného směru manévru může výrazně ovlivnit maximální dostřelovou obálku rakety.

2.3 Základní letové manévry

Základní letové manévry (BFMs) jsou stavebním kamenem leteckých taktik bojových stíhačů.

"No guts, no glory. If you are going to shoot him down, you have to get in there and mix it up with him."

Major Frederick C. "Boots"Blesse, USAF
Korejský Konflikt
10 Vítězství

V následujících kapitolách budou představeny konkrétní bojové manévry. Pro jejich správné vysvětlení je třeba představit několik výrazů vztahujících se k orientaci ve vzdušném prostoru leteckého souboje. Popsané výrazy jsou znázorněny na obrázku 2.5.

Letový kurz Zeměpisný kurz letadla ve stupních 0 - 360. 0 (360) značí sever.

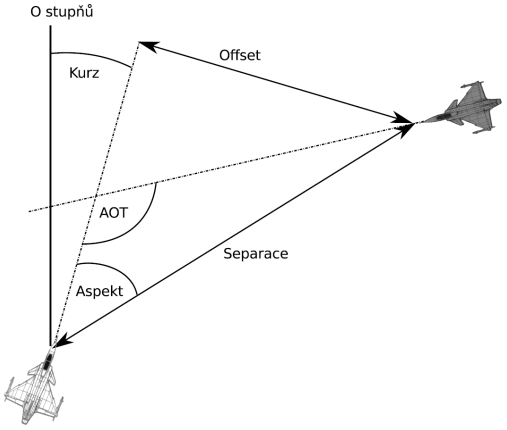
Vzájemný kurz V literatuře označován jako AOT - angle off the tail - znamená úhel, který svírají letové kurzy dvou letadel. Letadla letící proti sobě mají vzájemný kurz 180. Rozlišujeme kladný a záporný kurz (pravá a levá strana).

Aspekt Úhel, pod kterým vidí pilot letadlo druhé. Může být vyjádřen slovně (vlevo, vpravo, vpředu, vzadu) anebo jako hodina na pomyslném 12-ti hodinnovém ciferníku.

Separace Vzdálenost mezi dvěma letadly po pomyslné přímce.

Offset Vzdálenost letadla od osy letu druhého letadla.

Poloměr otáčky Poloměr kružnice opsané letadlem při otáčení v aktuální rychlosti.



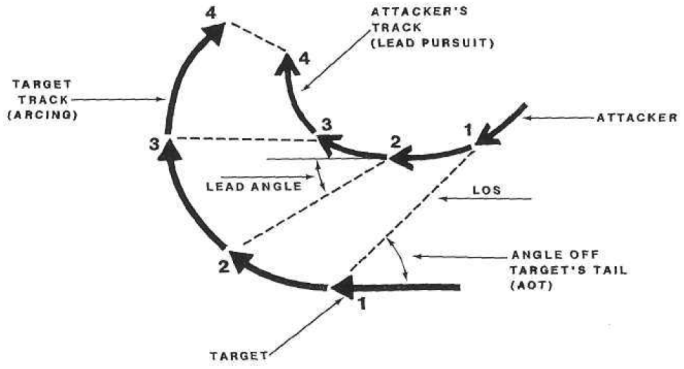
Obr. 2.5: Znázornění pojmů pro vzájemnou orientaci letadel ve vzduchu. Převzato z [17].

Stíhací křivky - Pursuit Curves

Stíhací křivka byla představena už ve vztahu k trajektoriím naváděných raket. Existují 3 typy - předsazená, přímá a zpožděná (lead, pure, lag). Liší se v tom, jestli rychlostní vektor útočícího letadla míří před, přesně na, anebo za protivníkovo letadlo.

Předsazená křivka

pronásledování se používá pro přiblížení k cíli. Díky kratší dráze oblouku předsazené křivky může útočník dokonce dostihnout rychleji letící letoun, pokud udržuje vysoké AOT. Obrázek 2.6 popisuje takovou situaci přiblížení po předsazené křivce.

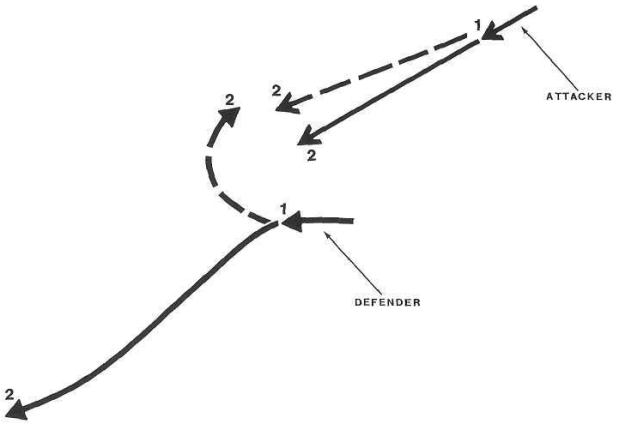


Obr. 2.6: Přiblížení k cíli po předsazené křivce.

Na obrázku 2.7 jsou znázorněny dvě možné obrany před útočníkem, který se pohybuje mimo dostřel zbraně. První možnost (plná čára) využije obránce s rychlejším letadlem. Obranný manévr je proveden snížením AOT na minimální možnou míru a nastavením motoru na maximální výkon. Obránce tak provede navýšení separace letadel za využití výhody rychlejšího letounu. Obránce pak může uletět nebo se otočit do vzájemného souboje, protože eliminoval poziční výhodu protivníka. Tato varianta však nemusí být vhodná

ve všech případech, obránce se může srovnáním AOT dostat do dostřelové obálky útočníka.

Druhou možností, pokud má obránce pomalejší letadlo, je přitažení oblouku otáčení směrem k útočníkovi (přerušovaná čára). Pokud je manévr proveden dobře, tedy letadla se potkají při vysokém AOT, poziční výhoda nepřítele je zmenšena nebo eliminována.

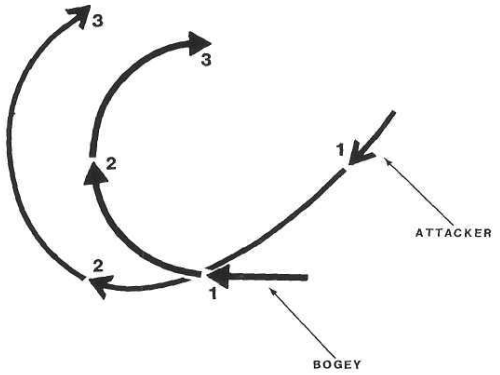


Obr. 2.7: Možná obrana proti pronásledování po předsazené křivce.

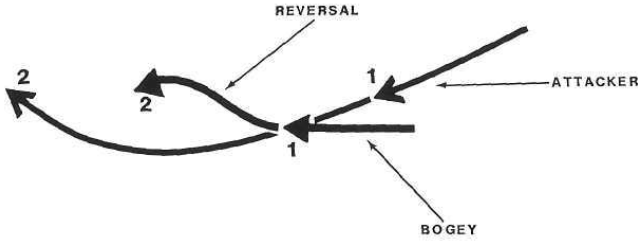
Přímá křivka
spočívá v udržování směru rychlosti přímo na nepřátelské letadlo. Poskytuje pomalé přibližování za nízkého AOT. Používá se pro dosažení střelné pozice - 6 hodin z pozice nepřítele.

Zpožděná křivka
je provedena směrováním letadla za letadlo nepřítele. Tato taktika se používá pro zpomalení nebo zastavení přibližování, pro nastavení správné vzdálenosti od cíle a snížení AOT. I rychlejší útočník se tak může dostat za pomalejšího nepřítele do optimální palebné pozice. Takovýto manévr je znázorněn na obrázku 2.8.

Efektivní obranou je změna rychlosti a směru oblouku. Takový manévr pak evokuje rapidní snížení vzdálenost a může se dokonce stát, že útočník se dostane do přední polo-sféry obránce, čímž si útočník a obránce vymění role. Obrácení oblouku je efektivní pouze proti stíhačům s naváděnými raketami, neboť obránce proletí kolem útočníka v minimální vzdálenosti a raketu tak není možné odpálit. Takový obranný manévr a přelet útočníka je znázorněn na obrázku 2.9.



Obr. 2.8: Přesun do palebné pozice po zpožděné křivce.



Obr. 2.9: "Přestřelení" oblouku po zpožděné křivce útočníkem.

Takovému přestřelení ze strany útočníka se dá zabránit provedením manévru, který oddálí útočníka od obránce. Manévr nazveme **Zpožděná stíhací otočka** (Lag-Pursuit Roll). Útočník nabere výšku, čímž zpomalí a přizpůsobí oblouk tak, aby se zařadil za nepřátelské letadlo. Znázorněno na obrázku 2.10. Variace takovéto otočky se používají pro snížení AOT a zaměření nepřítele do palebné obálky.

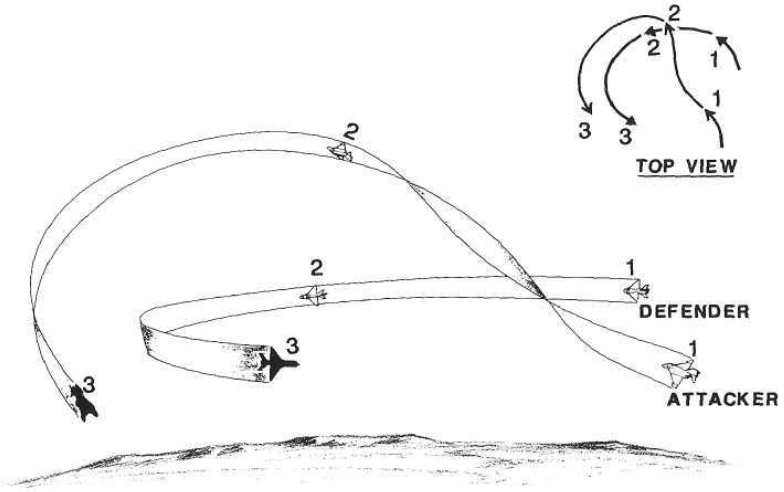
Vysoké Yo-Yo

je manévr podobný stíhací otočce. Používá se při středním AOT (30° až 60°), pokud má útočník srovnatelnou rychlost s obráncem, ale je přitom od obránce dostatečně daleko. Vysoké Yo-Yo je znázorněno na obrázku 2.11.

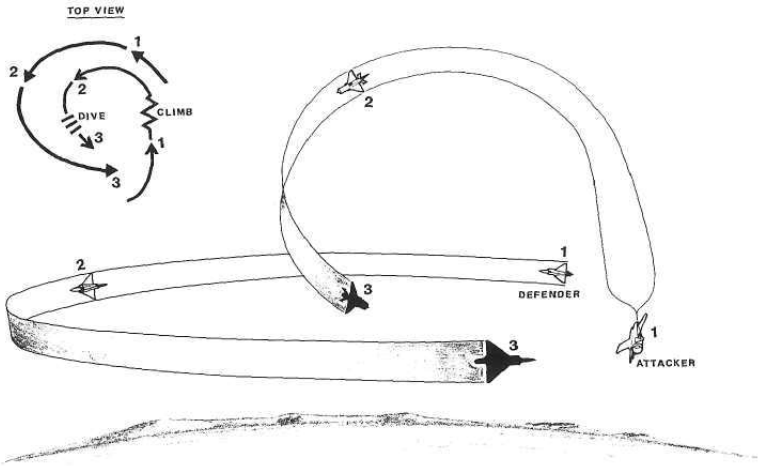
Útočník nejprve směřuje přímo na cíl a snižuje separaci. Když se začne přibližovat tak, že by mohl přestřelit, zvedne špičku a zahájí stoupání. Letadlo tak ztrácí rychlost a snižuje se rychlost přibližování. V další fázi je útočník nad cílem v jeho zadní polo-sféře. S výškovou výhodou může zahájit stíhání po předsazené, přímé nebo zpožděné křivce, aby se tak dostal do palebné pozice.

Nízké Yo-Yo

Zatímco výše zmíněné manévry spočívaly v odklonění směru od cíle a pomalém přibližování při snižování AOT, Nízké Yo-Yo spočívá v rychlém přiblížení a odklonění na stranu



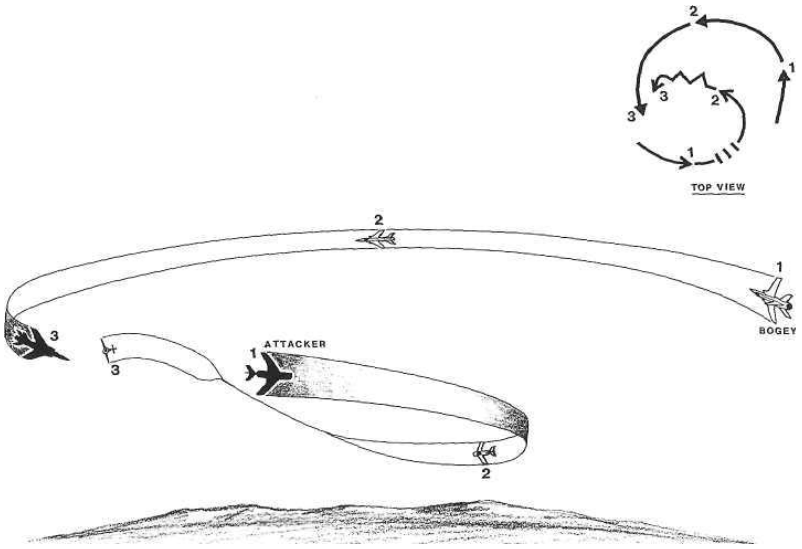
Obr. 2.10: Zpožděná stíhací otočka.



Obr. 2.11: Vysoké Yo-Yo.

oblouku.

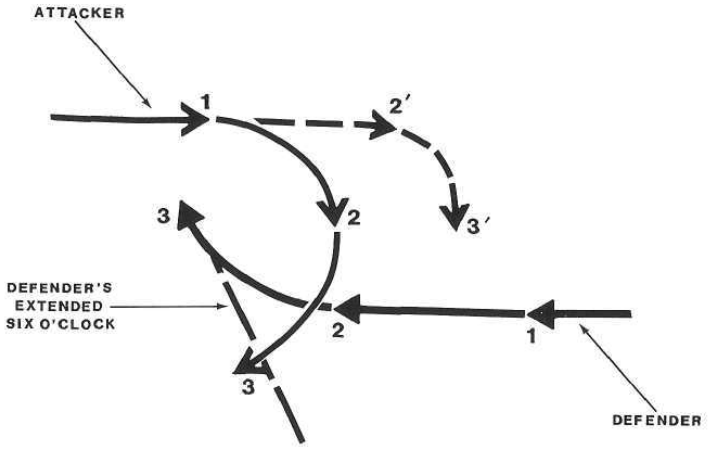
Takový manévr se používá typicky při dlouhém pronásledování po zpožděné křivce, kdy útočník nemá takovou obratnost, aby své mířidla rychle dostal na nepřítele, nebo by při provedení takového přitáhnutí výrazně ztratil rychlost. Na obrázku 2.12 je vyobrazena taková situace. Útočník nemá dostatečnou obratnost, aby se dostal protivníkovi za záda, tak zahajuje klesání, při kterém přitahuje oblouk. V bodu 2 má útočník výrazné vedení v křivkovém oblouku. Čím větší AOT na začátku manévru, tím větší musí být vedení, aby mohl útočník dostoupat a dostat se do palebné pozice.



Obr. 2.12: Nízké Yo-Yo.

Většinou se používá kombinace manévrů - **Nízké Yo-Yo** pro snížení vzdálenosti, následované **Vysokým Yo-Yo** nebo **Stíhací otočkou** pro získání palebné pozice. Avšak snaha získat v jednom manévru moc vzdálenosti dává obránci víc prostoru na strategii protiútoku.

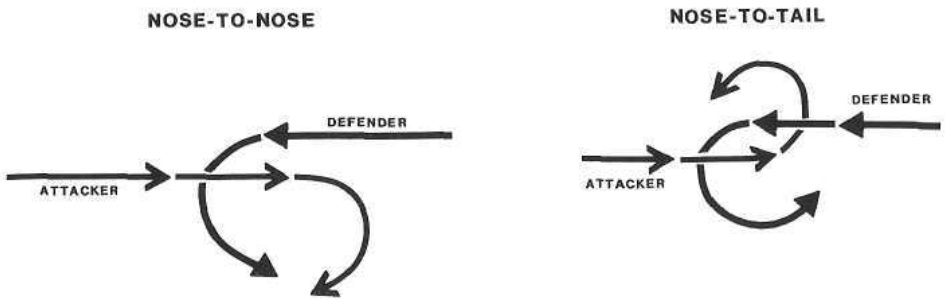
Předsazené otočení - Lead Turn Manévr začíná v momentě, kdy 2 letadla letí čelem proti sobě (aspek kolem 180°) s určitým offsetem. Útočník se začne otáčet dřív, než se letadla vzájemně minou, čímž získá poziční výhodu a dostane se do zadní polo-sféry obránce. Obrázek 2.13 zachycuje manévr **Předsazené otočení** - když se letadla míjí v bodě 2, útočník má značnou poziční (úhlovou) výhodu. Čárkované šipky značí trajektorii letu za předpokladu, že obě letadla začaly točit až poté, co se vzájemně minuly.



Obr. 2.13: Předsazené otočení

Typicky čím dříve je zahájen točivý oblouk, tím lepší bude mět útočník poziční výhodu. Pokud ale začne točit moc brzo, může se dostat do přední polo-sféry obránce. Tato poziční nevýhoda může mít v případě obránce se zbraněmi na krátký dostřel katastrofické důsledky.

Nose-to-Nose and Nose-to-Tail otáčení
jsou dva způsoby manévrování letadel, které se potkají v jejich předních čtvrto-sférách. Při jednom z nich se nepřátelé vzájemně točí za svými špičkami nebo naopak za svými konci2.14.

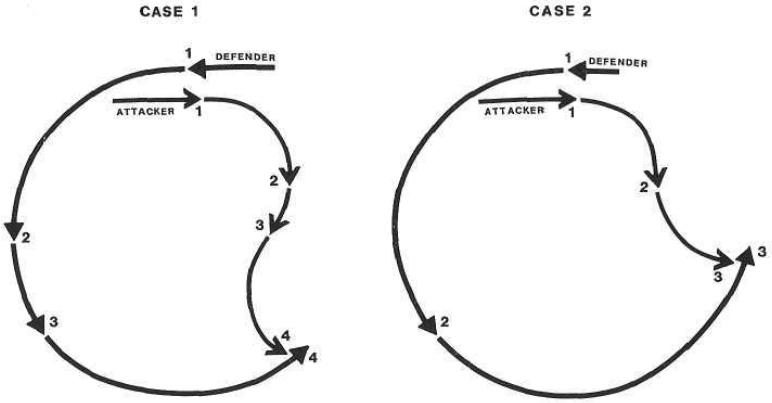


Obr. 2.14: Nose-to-Nose a Nose-to-Tail otáčení.

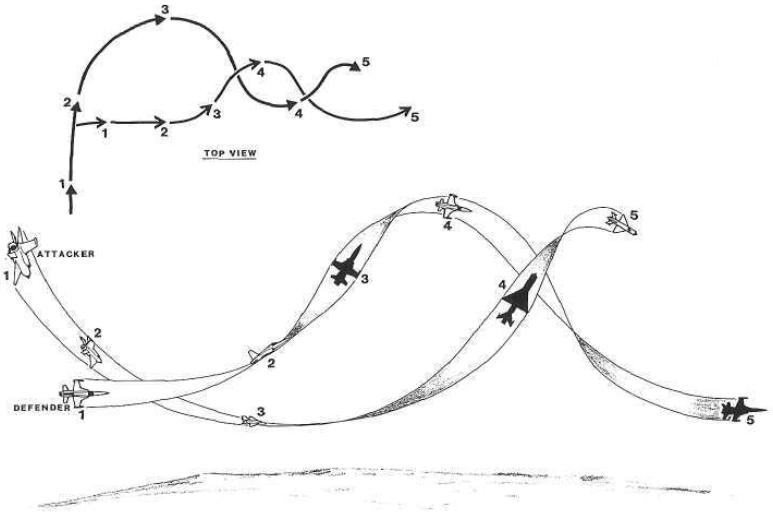
Při obou otáčení je zásadní rychlost a poloměr otáčky obou letadel. Útočník se snaží zkrátit oblouk otočení a dostat se tak protivníkovi za záda jako na obrázku 2.15. V prvním případě má útočník menší rychlost a lepší poloměr otáčky, takže dokáže zůstat v oblouku obránce a vytvořit separaci, kterou obránce přitáhnutím otáčky nedokáže eliminovat. Útočník pak využije separaci pro **Předsazené otočení** a dostane se tak do palebné pozice za záda obránce. V případě druhém je situace podobná. Avšak obránce s vyšším poloměrem otáčky má taky daleko vyšší rychlost. Tato rychlostní výhoda však není obránci k ničemu dobrá. Pokud útočník využil získanou separaci, má úhlovou výhodu a **Předsazeným otočením** se dostane opět do palebné pozice.

Při manévrování Nose-to-Tail je situace obdobná. Útočník se taky snaží zmenšit poloměr otáčky tak, aby vymanévroval protivníka a dostal se do palebné pozice. Opakováním těchto manévrů přechází souboj v **Nůžky**.

Vertikální nůžky
Zatímco Ploché (Horizontální) Nůžky se provádí v malých rychlostech ve stejné výšce, vertikální nůžky jsou manévr, který se provádí v rychlostech maximálních. Na obrázku 2.16 je znázorněn průběh takovéhoto manévru. V bodě 2 útočník nalétává na obránce. Obránce prudce strhne nahoru a začne nabírat výšku, čímž ztrácí rychlost pohybu dopředu a nutí útočníka, aby ho při další otáčce předletěl. Úspěch tohoto manévrů záleží na výkonu letadla a přesnosti pilota, který dokáže efektivně pracovat s energií letadla.



Obr. 2.15: Nose-to-Nose a vliv poloměru otáčky a rychlosti.



Obr. 2.16: Verikální nůžky.

Obranná Spirála

Obranná spirála jsou ve své podstatě velmi těsné nůžky, při kterém obě letadla letí ve vysoké rychlosti směrem dolů. Manévr nastane v okamžiku, kdy jeden ze stíhačů dohnal druhého (pomalejšího) a dostal se tak do palebné pozice 2.17.

Za účelem zvýšení AOT pro snížení šance střelby se pomalejší stíhač vrhne k zemi a za pomocí gravitace spustí nůžkový souboj, kde se jeden snaží přemanévrovat druhého. Manévr končí, když jeden z pilotů ukončí spirálu, nastaví plný tah a vystoupá nad nepřítele.

2.4 Vzájemný souboj 1 na 1

Při vzájemném souboji podobných letadel (parametry letadel se neliší o více než 10%) se snaží pilot nad soupeřem získat výhodu úhlovou anebo energetickou. Získání dobrého

Obr. 2.17: Obranná spirála.

Obr. 2.18: Vysoká manévrovatelnost vs vysoký výkon.

palebného úhlu a možnosti střelby je výhodné i za cenu ztráty energie. Taková možnost totiž může znamenat zničení nepřátelského letadla. Pokud chce pilot vyhrát energetický souboj (například chce získat výškovou výhodu), musí pořád dbát na to, aby se nedostal do protivníkovy dostřelové obálky.

Pro výběr zbraňové taktiky je zásadní znát protivníkovy zbraňové systémy. Je tedy rozdíl, pokud proti sobě má pilot pouze kulomety a kanóny, pouze naváděné rakety určené k vypouštění ze zadní čtvrto-sféry cíle, rakety pro plný aspekt nebo kombinace těchto zbraní.

Pokud se parametry letadel liší o více než 10%, jedná se o souboj různých letadel. Mezi tyto parametry patří hlavně aspekty energetické (stoupavost, zrychlení, maximální rychlost) a úhlové (manévrovatelnost). Existují samozřejmě další aspekty, které mohou mít vliv na souboj - schopnosti radarů, velikost a odolnost letadla, výhled pilota atd.

Vysoká manévrovatelnost vs vysoký výkon

V tomto typu souboje disponuje jeden pilot letadlem, které má nízké zrychlení a nízkou maximální rychlost, zato má superiorní obratné schopnosti, zatímco letadlo druhé má superiorní energetické schopnosti jako zrychlení, stoupavost a maximální rychlost. Úkolem pilota je exploitace zásadních soupeřových slabin a maximální utilizace výhod vlastních.

Pokud se podíváme na útok z pozice pilota se slabším výkonem, zato lepší manévrovatelností, měl by útok být prováděn následovně. Největším problémem pomalejšího letadla bude přiblížení k cíli. Pilot tak musí používat přiblížení po předsazené křivce. Měl by používat **Stíhací otočky**, **Vysoké a Nízké Yo-Yo** nebo **Horizonální nůžky** s využitím rotace **Nose-to-Nose**. Tak maximalizuje svůj zisk z výhody lepšího poloměru otáčky. Jelikož má nevýhodu ve stoupavosti, měl by souboj maximálně držet v horizontální úrovni. Pokud nepřítel, který má lepší schopnosti ve vertikálním souboji, zůstane v horizontálním souboji, je pouze otázkou času, než útočící pilot získá pozici střelby a vyhraje souboj.

Na straně druhé pilot s menší manévrovatelností a větším tahem motoru by se měl soustředit na výhru souboje energetického. Pokud se rozhodne útočit na pomalejší cíl, musí přistupovat po **Zpožděné křivce** a využívat energetickou (výškovou) výhodu - tedy používat vertikální a šikmé manévry.

Obranná spirála může být vhodný manévr pro letadlo s vyšším tahem, pokud je provedena rychle a korektně. Rychlejší letadlo tak může uletět letadlu pomalejšímu a získat výškovou poziční výhodu.

2.5 Souboje 2 na 1

"Never break your formation into less than two-ship elements. Stay in pairs. A man by himself is a liability, a two-ship team is an asset. If you are separated, join up immediately with other friendly airplanes."

Major Thomas B. "Tommy"McGuire, USAAF

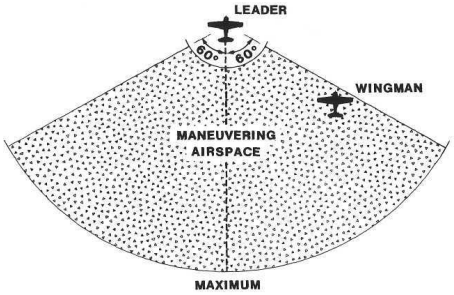
Krycí křídlo

V letecké jednotce skládající se ze dvou a více letadel je vždy jeden Leader, jehož hlavními prioritami jsou navigace, hledání nepřítele v přední polo-sféře, plánování útoku a iniciace manévrů. Hlídání své zadní polo-sféry je až jeho druhotnou prioritou. Od tohoto účelu má Leader svého Wingmana, který letí za vedoucím letky v aspektu asi 60° a stará se o zadní polo-sféru (Obrázek 2.19). Wingman si od vedoucího letadla drží typicky vzdálenost v řádu 70 - 100 metrů u moderních stíhačů. Kdyby se Leader náhle rozhodl rychle točit směrem k wingmanovi, pro předejití kolizi, letí wingman v nižší výšce.

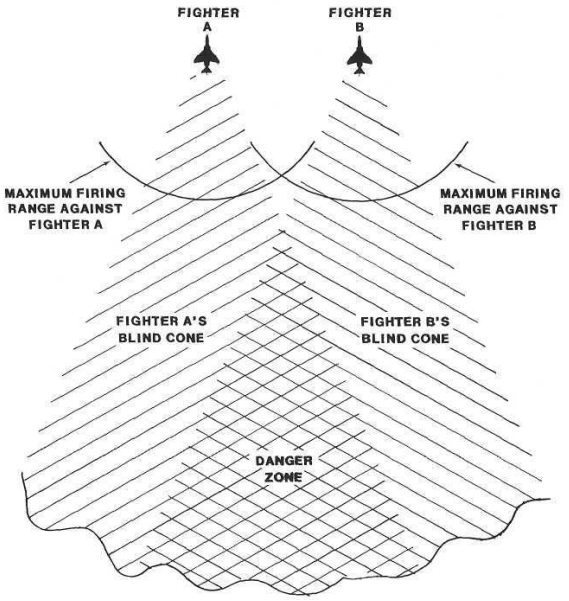
Útok iniciuje Leader tak, že začne souboj 1 na 1, zahájí přibližování a snižování AOT, aby mohl cíl sestřelit. Úkolem wingmana je vzdálit se, aby neomezoval útočné manévrování leadra. Musí zůstat na leadrových 6-ti hodinách, a poskytovat krytí pro případ, že by se nepřítel dostal do palebné pozice na leadera. Zpravidla čím větší má wingman odstup, tím větší má přehled o souboji a lepší možnost intervence v případě potřeby.

Společný útok

Jako metodu **Společný útok** označíme typ útoku, kdy každý z páru stíhačů podporuje toho druhého v samotném útoku, aniž by zůstali v útočící a pozorovací pozici, jak bylo zmíněno u taktiky **Krycí křídlo**. Tato doktrína povoluje stíhačům rozdělení a následné



Obr. 2.19: Znázornění pozice Leadera a Wingmana.



Obr. 2.20: Společný útok Leadera a Wingmana.

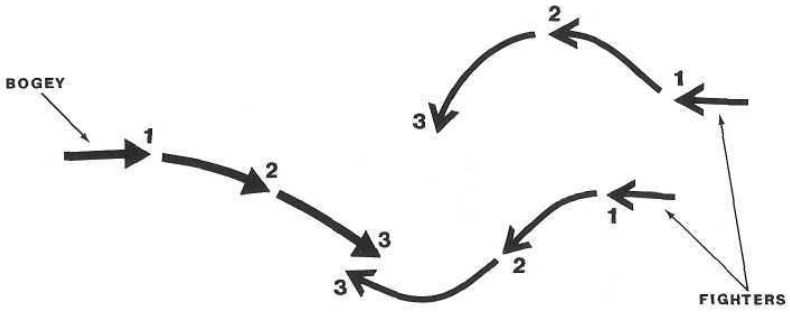
spojení kvůli provedení koordinovaných společných útoků. Pořád zde existuje pozice Leadera a Wingmana, jejich pozice se ale v průběhu souboje mohou měnit.

Při spatření nepřítele se pilot, který se nachází ve vhodnější pozici pro útok, stane leaderem a zahájí přibližování a útok. Wingman musí nastoupat a získat energetickou výhodu, přičemž musí udržovat adekvátní separaci, aby mohl provést následný manévr a připojit se do bitvy.

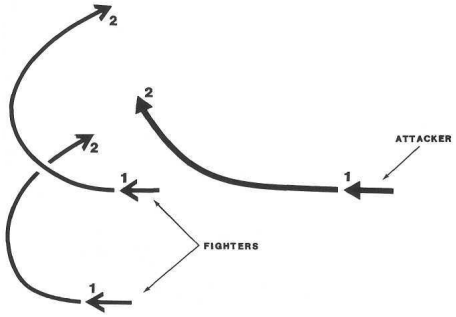
Většinou je zřejmé, kdo bude útočící pilot a kdo bude stoupající wingman, ale mohou nastat i případy, kdy lepší pozice pro útok jasná není. Takovým příkladem může být obrázek 2.21, ve kterém se útočící dvojice rozdělí a následně obklíčí nepřátelské letadlo. Na obrázku v bodě 2 letka - provádějící společný útok - přinutila nepřítele vybrat si letadlo, na které zaútočí. Jeho volba pak poskytla druhému letadlu offset a separaci, kterou využije na provedení manévru, kterým se dostane do zadní polo-sféry cíle.

Při obraně letí pilot a jeho wingman vedle sebe do té doby, než se útočník rozhodne zaútočit na jeden z dvou cílů. Úkolem pilota, na kterého není útočeno, je co nejdříve se manévrem dostat do útoku na nepřítele. Takováto situace je znázorněna na obrázku 2.22. Letadlo na pravé straně útočícího letadla se dostalo do ohrožení útoku. Ohrožený pilot zahájí obraný strhnutím doprava - směrem pryč od svého wingmana. Pokud nepřítel bude pokračovat v útoku, bude stíhán pilotem druhým, který se mu dostane do zad. Takovýto manévr nazveme poměrně výstižně - **Sendvič**.

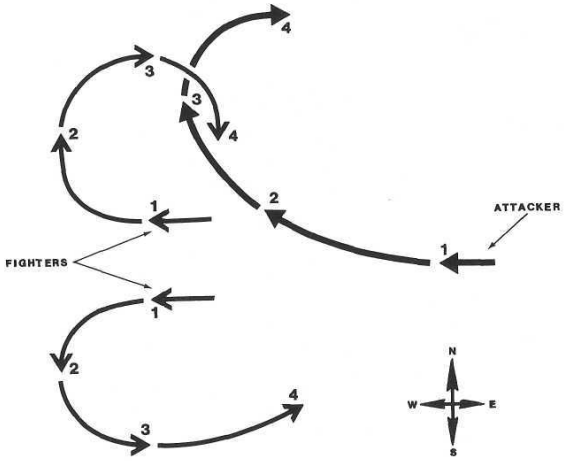
Sendvič je velmi účinný manévr, pokud je ohrožený pilot identifikován zavčasu. Identifikace je tím jednodušší, čím větší mají leader a wingman vzájemnou horizontální separaci. Pokud



Obr. 2.21: Obklíčení nepřátelského letadla.



Obr. 2.22: Manévr zvaný Sendvič.



Obr. 2.23: Obranné rozdělení.

se nepřítel nerozhodne zavčasu zaútočit na první nebo druhé letadlo, obránci provedou manévr **Obranné rozdělení** - znázorněno na obrázku 2.23. Souboj se pak přesouvá do roviny 1 na 1 a přechází v nůžky, zatímco podporující pilot se snaží dostat do pozice útoku ze zadní polo-sféry nepřítele.

3 IMPLEMENTACE MULTIAGENTNÍCH LETECKÝCH SOUBOJŮ

Pokud se díváme na leteckou simulaci jako na hru probíhající v reálném čase, multiagentní systém multiagentní systém se jeví ideální implementační platformou pro inteligentní pilotní chování. Agent, tedy pilot, má omezené znalosti o okolním světě, má vlastní zájmy, motivace a cíle.

Herní enginy obvykle nemají implementovanou podporu pro multiagentní systémy, jakožto reprezentaci chování NPC (Non playable characters), tudíž se nabízí 2 možnosti řešení: **1)** Vytvoření úplně nového enginu s nativně zabudovanou podporou pro agentní systémy, nebo **2)** spojení existujícího herního enginu s existující agentní platformou tak, aby se herní engine staral o simulační a grafickou část aplikace, zatímco agentní systém by obstarával inteligentní chování jednotlivých agentů, tedy odpovídajících pilotů.

V případě prvně zmíněného způsobu by běžící prostředí bylo rychlejší a efektivnější, ovšem objem práce a výzkumu pro vytvoření takovéhoto enginu by vydal na samotnou diplomovou práci. Druhý přístup je pro účel práce schůdnější, i když přináší obtíže v rámci synchronizace stavů datových entit, neboť se jedná o dva oddělené systémy. Přestože byly vytvořeny systémy kombinující MAS a herní enginy [20, 21, 22], zatím nenašly svou cestu mezi masové komerční projekty.

3.1 Použité nástroje

Pro implementaci projektu bylo třeba zvolit implementační platformu agentů, knihovny pro tvorbu vizualizačního prostředí a jazyk pro "slepení" celého projektu. Rozhodl jsem se pro níže uvedenou kombinaci nástrojů.

Jason
Architektura BDI je nejmodernějším přístupem k programování agentů, *Jason* [4] je platforma pro realizaci takovýchto agentů vytvořená v programovacím jazyce *Java*. Základem práce s *Jasonem* je vytvoření agentního prostředí v *Javě* a následné programování agentů v dialektu agentního programovacího jazyka *AgentSpeak* [3]. Jason je tedy konkrétně interpret dialektu AgentSpeaku, který za běhu dokáže komunikovat s vlastním prostředím, které jsem potřeboval pro implementaci leteckých soubojů.

JMonkey
JMonkey[18] je 3D herní engine postavený na jazyce Java, využívající knihovny JOGL (Java OpenGL) [19] pro vykreslování 3D grafiky. Engine je multiplatformní a zvolil jsem ho také kvůli tomu, že server a klient mohou sdílet části kódu (matematické operace, síťová komunikace) díky stejnému implementačnímu jazyku.

Pro spojení mezi javovým prostředím agentů na klientské straně a simulačním modelem

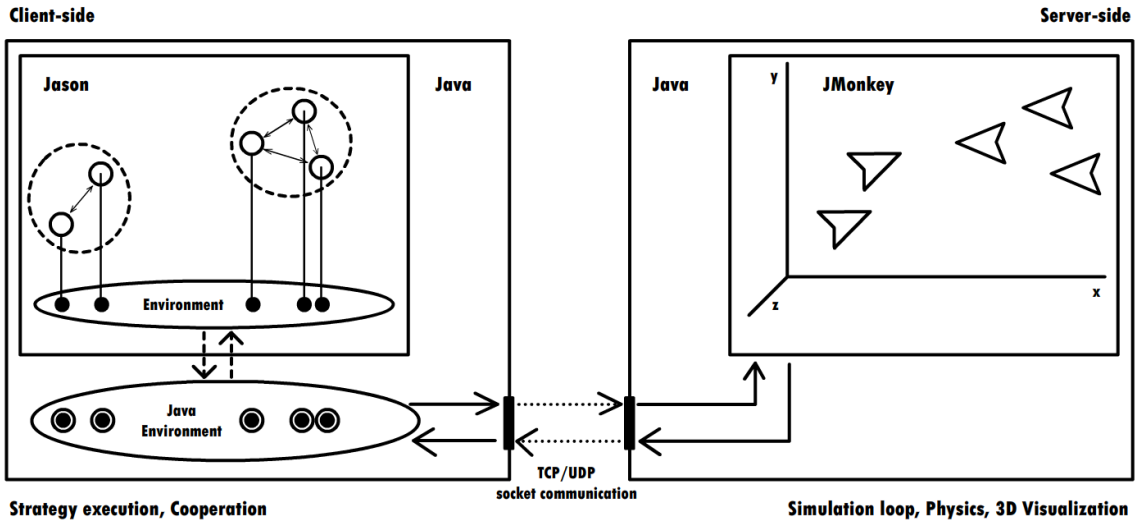
na straně serverové je využita socketová komunikace. Agenti mají pouze exekutivní a rozhodovací funkci, přičemž celá simulace probíhá na straně serveru. Model je tedy nachystaný na změnu simulačního modelu ze strany serveru (například v případě lepšího aerodynamického nebo zbraňového modelu simulace), za předpokladu dodržení požadavků na komunikační protokol mezi oběma stranami aplikace.

JSON
JSON jsem se rozhodl použít v rámci protokolu pro komunikaci ihned z několika důvodů: **1)** Jako serializační nástroj je dobře čitelný a poměrně úsporný, **2)** protože je moderní, má velkou podporu napříč programovacími jazyky.

3.2 Přehled architektury projektu

Cílem práce je vytvoření tří funkčních bloků a jejich následné propojení. Pokud se podíváme na aplikaci ze strany agentního chování, je třeba vytvořit samotné agentní chování na platformě Jason. Dále je nutné vytvořit vlastní agentní prostředí, přes které bude agent dostávat vjemy ze simulace a akcemi vytvářet požadavky na změnu prostředí. Poslední částí aplikace je vytvoření simulátoru, který bude aktualizovat agentovo prostředí a reagovat na akce vztažené agentem.

Na obrázku 3.1 je znázorněna navrhovaná architektura práce. Návrh byl vytvořen v rámci semestrálního projektu, avšak samotná implementace byla provedena až jako součást výsledné práce.



Obr. 3.1: Přehled architektury práce. Vlastní vizualizace dle [10].

3.3 Simulační model

Simulační model pro agenty představuje reálné prostředí, které má za cíl simulovat reálnou situaci a tedy reálné agentovo prostředí. Při tvorbě simulačního modelu bylo potřeba řešit několik aspektů modelu: **1)** Fyzikální model simulace, **2)** Výměnu a synchronizaci dat s agentním modelem **3)** Vizualizaci probíhající simulace

3.3.1 Fyzikální model simulace

Při návrhu simulačního modelu pro mě bylo důležité stanovit detailnost fyzikálního modelu simulace. Správnou aerodynamiku letu, do které je třeba zahrnout tah motoru, aerodynamický zdvih křídelní plochy, vítr, tření vzduchu a mnoho dalších faktorů, jsem do implementace simulačního modelu nezahrnul, neboť tvorba takovéhoto simulátoru by byla časově velmi náročná a ani není účelem práce. Pro účely adaptace agentního chování na simulační model jsem se rozhodl modelovat pouze poloměr otáčky letadla, tedy manévrovatelnost v závislosti na rychlosti agentního letounu. Fyzikální model počítá se dvěma typy zrychlení: **1)** lineární zrychlení a **2)** radiální zrychlení. Zatímco lineární zrychlení lze vypočítat jednoduše jako rozdíl velikostí rychlosti za jednotku času, radiální zrychlení je třeba počítat jako rozdíl normovaného vektoru rychlosti za jednotku času. Tohoto výpočtu jsem využil pro nastavení odpovídající manévrovatelnosti reálných letadel.

V literatuře [23, 24] se pro definici poloměru otáčky používá hodnota *standard rate turn*, též nazývaná jako *rate one turn* (ROT), ve které je velikost přetížení přímou funkcí náklonu. Pro výpočet manévrovatelnosti letounu jsem použil následující vzorce:

Výpočet přetížení působícího na pilota:

$$a = \frac{1}{\cos(bank\_angle)} \tag{3.1}$$

Na základě náklonu, aktuální rychlosti a gravitační konstanty lze spočítat poloměr otáčky.

$$turn\_rate = \frac{v^2}{g * \tan(bank\_angle)} \tag{3.2}$$

Pokud dosadíme do vzorce pro obvod kružnice a vydělíme rychlostí letadla, dostaneme úhlovou rychlost letounu.

$$perimeter = 2 * \pi * turn\_rate \tag{3.3}$$

$$turn\_time = \frac{perimeter}{v} \tag{3.4}$$

$$\omega = \frac{360}{turn\_time} \tag{3.5}$$

Při správném nastavení měřítek a konstant v rámci simulace odpovídá manévrovatelnost v simulátoru tabulkovým hodnotám manévrovatelnosti reálných letadel. Pro parametrizaci

a nastavení simulátoru jsem použil údaje z amerického letounu F-15 volně dostupné na wikipedii [1].

Náklon letadla	**Přetížení**
10°	1,02 g
15°	1,04 g
30°	1,15 g
45°	1,41 g
60°	2,0 g
70°	2,92 g
80°	5,76 g
83°	8,21 g
85°	11,47 g
88°	28,65 g

Tab. 3.1: Vliv náklonu na přetížení letadla.

Protože simulátor nepočítá aktuální náklon letadla, stanovil jsem náklon (*bank_angle* ve výpočtech) na konstantních 83°. Při takovém náklonu se letadlo dostává na hranici svých konstrukčních možností.

Rychlost $[km * h^{-}1]$	**Rychlost** $[Mach]$	**ROT** $[m]$	**Doba otáčky** $[s]$	$\omega[°/s]$
594	0,5	341	13	28
831	0,7	667	18	20
1188	1,0	1 363	26	14
1782	1,5	3 067	39	9
2376	2,0	5 452	52	7

Tab. 3.2: Vliv rychlosti na poloměr oblouku při konstantním náklonu 83°.

Pro výpočet přetížení naváděných raket je použit stejný výpočet, avšak parametr *bank_angle* je nastaven na hodnotu 86°, protože rakety vydrží konstrukčně daleko vyšší zrychlení. Rozbušky jsou konstruovány tak, aby vydržely zrychlení 20g [25].

[1] https://en.wikipedia.org/wiki/McDonnell_Douglas_F-15_Eagle#Specifications_.28F-15C.29

3.4 Protokol pro síťovou komunikaci

Pro synchronizaci prostředí na serveru a klientovi bylo třeba navrhnout vlastní síťový protokol. Jak již bylo zmíněno v sekci o použitých nástrojích, rozhodl jsem se pro serializaci a následnou výměnu dat použít JSON. Každá zpráva se skládá z identifikačního bytu na začátku toku, který značí typ zprávy, a řetězce s daty uloženými ve formátu **JSON** kódovány v **UTF-8**. Čísla jednotlivých zpráv jsou uložena ve třídě **Protocol.java**, která se nachází jak na serverové, tak na klientské straně programu.

```
// vytah identifikacnich cisel zprav ze tridy Protocol.java
public static final int END = -1;           // End message
public static final int START_INIT_REQ = 0; // Client initial message
public static final int START_INIT_ACC = 1; // Server initial confirm
public static final int DATA_REQ = 2;       // Cliend needs data
public static final int DATA_ACC = 3;       // Server META message
public static final int DATA_UPDATE = 4;    // Server sending update
public static final int ACTION_REQ = 5;     // Agent action required
public static final int ACTION_ACC = 6;     // Server action confirm
public static final int ACTION_UPDATE = 7;  // Agent sends action
```

3.4.1 Navázání spojení - Handshake

Začátkem každé komunikace je navázání spojení. Proto nejdřív server čeká na spojení na dané adrese a portu (imlicitně localhost na portu 9990). Client zasílá zprávu **START_INIT_REQ**, ve které zároveň zasílá parametry letecké simulace z agentního prostředí. Server následně odpovídá zprávou **START_INIT_ACC**. Tím je spojení navázáno a jsou spuštěny hlavní smyčky na serveru i klientu. Inicializační zpráva pro server má následující tvar:

```
 // priklad inicializacni zpravy pro server
{ "blue": 1, "red": 2, "separation": 800, "direction": 1 }
```

3.4.2 Komunikační cyklus

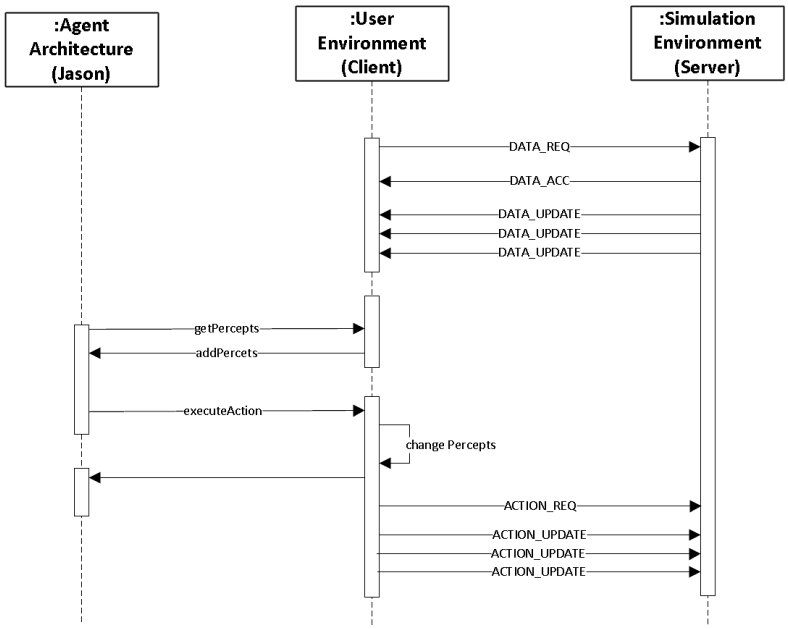
Po inicializaci serveru klientem se programy na obou stranách dostanou do hlavní smyčky, kde agentní prostředí vyžaduje od serveru aktualizace podle potřeby, přijímá je, a nakonec zasílá akce provedené agentem na prostředí. Jeden celý komunikační cyklus potom probíhá tak, jak je znázorněn na obrázku 3.2. Celá smyčka výpočtu se skládá ze 3 částí:

Agentní prostředí získá změny prostředí ze serveru
Klient (Javové prostředí na klientovi) zašle serveru zprávu **DATA_REQ**, kterou serveru

oznamuje, že chce od serveru poslat nová data. Server tento požadavek zpracuje a odpovídá zprávou **DATA_ACC**, kterou zároveň sdělí, kolik zpráv klientovi pošle. Následně klient přijímá zprávy **DATA_UPDATE**, které obsahují stavové data jednotlivých stíhačů a aktivních naváděných raket na serverové straně simulace. Jakmile klient přijme všechny data, je spuštěno agentní chování.

Provedení agentního chování
Přijatá data jsou transformována do informací relevantních pro agenta, následně jsou předány agentům jako nové vjemy z prostředí. Agent na základě získaných vjemů provede akce, kterými chce prostředí ovlivnit. Tyto změny jsou zapsány do odpovídajících struktur v javovém prostředí a v další fázi odeslány na server.

Agentní prostředí pošle na server požadavky na změnu
Klientské prostředí zašle serveru zprávu **ACTION_REQ**, server se následně nastaví do režimu přijímání akcí. Klient odesílá akce (**ACTION_UPDATE**) a server je zpracovává. Simulace běží dál do doby, než klient znovu zažádá o aktualizaci vjemů.



Obr. 3.2: Sekvenční diagram toku dat mezi serverem, agentním prostředím a agenty.

3.4.3 Obsah zpráv

V rámci synchronizace dat mezi klientem a serverem jsou zasílány 2 typy zpráv. Prvním typem zpráv jsou zprávy aktualizace ze simulačního modelu. Při aktualizaci dat jednotlivých letadel každá zpráva obsahuje název agenta, pozici v 3D prostoru v metrech, normalizovaný vektor rychlosti v metrech za sekundu, aktuální rychlost v jednotce Mach (násobek

rychlosti zvuku), aktuální nepřátelský cíl a příznak, zda může nebo nemůže agent střílet.

```
{ // zprava aktualizace letadla
"name": aero_1,
"locx": 1866.9705,  "locy": 48.834465, "locz": -1644.3248,
"velx": 0.59298307, "vely": -0.06615467, "velz": -0.80249274,
"speed": 1.2, "target": aero_2,
"fire_en": false, "alive": true }
```

Zpráva obsahující aktualizaci naváděné rakety obsahuje obdobné informace jako zpráva pro letadlo, neboť jsou na serverové i klientské straně vytvořeny děděním od letadla.

```
{ // zprava aktualizace navadene rakety
"name": aero_1_missile,
"locx": 1040.5052, "locy": 644.8987,  "locz": 924.2481,
"velx": 84.67193, "vely": 29.245943, "velz": 44.444813,
"speed": 2.0, "target": aero_2, "active": true}
```

Odlišný formát má zpráva akce agenta, která pouze udává naváděcí bod, kam má letadlo na straně serveru letět. Zpráva navíc obsahuje příznak **fire**, který udává, zda-li má letadlo na serverové straně vystřelit na nepřítele uloženého v poli **enemy**.

```
{ // zprava akce zaslana klientem
"name": aero_1, "enemy": aero_2,
"checkx": 2958.152, "checky": 500.81683, "checkz": -2056.682,
"fire": false }
```

3.5 Agentní chování

Při návrhu agentů do leteckého souboje jsem se snažil, aby agent přemýšlel jak člověk, a na základě toho provádět dekompozici problému na podproblémy. Člověk provádí manévr na základě pozice a vzdálenosti nepřítele, rychlosti, aspektu a aot. V jednu dobu může provádět takový manévr, který byl nejvýhodnější na začátku plánování manévru, ovšem v průběhu provádění změnil nepřítel pozici, takže člověk vyhodnotí situaci a začne provádět úplně jiný manévr, který ho má dostat do výhodnější pozice. Pro toto chování se hodí čistě reaktivní agent, který se rozhoduje při každé aktualizaci pouze na základě informací z prostředí o nepřátelském agentovi.

V rámci agentního chování bylo třeba se zabývat dvěma různými programy. Prvním z nich je vlastní prostředí pro agenty, které agentům aktualizuje vjemy na základě aktualizačních zpráv ze serveru, a provádí akce, kterými agenti ovlivňují toto své prostředí. Druhou částí bylo samotné chování agentů vytvořené v jazyce AgentSpeak.

3.5.1 Agentní prostředí

Vlastní prostředí bylo vytvořeno v javě jako potomek třídy Environment, která je implementována v knihovně Jasonu. Agentní prostředí má za úkol aktualizovat vjemy agentů a provádět akce, kterými agent prostředí ovlivňuje. Třída Environment poskytuje následující metody pro práci s vjemy agentů:

1. **addPercept(L)**: přidat literál L do globálního seznamu vjemů, tedy agenti uvěří, že **L**.
2. **addPercept(A,L)**: přidat literál L do seznamu vjemů, které jsou výhradní pro agenta **A**. Tedy pouze agent **A** uvěří, že **L**.
3. **removePercept(L)**: odstranit literál **L** z globálního seznamu vjemů.
4. **removePercept(A,L)**: odstranit literál **L** ze seznamu vjemů agenta **A**.
5. **clearPercepts()**: odstranit všechny vjemy z globálního seznamu vjemů.
6. **clearPercepts(A)**: odstranit všechny vjemy agenta **A**.

Použitím výše uvedených funkcí implementovaných v prostředí Jasonu, po aktualizaci dat ze serveru, jsou agentům v každém cyklu aktualizovány vjemy znázorněné v tabulce 3.3.

Vjem	**Interpretace vjemu**
tick(Counter)	Čítač kroků v prostředí
alive(Alive)	Booleovská hodnota udávající, zda je agent naživu
team(TeamNumber)	Tým agenta (1 nebo 2)
fire_enabled(FireEn)	Příznak značící zda může agent střílet
team(Agent, TeamNumber)	Tým odlišného agenta (agentů)
distance(Agent, Distance)	Vzdálenost k druhému agentovi
angle_off_tail(Agent, AoT)	AOT vzhledem ke druhému agentovi
aspect(Agent, Aspect)	Aspekt vzhledem ke druhému agentovi
reverse_aspect(Agent, RAsp)	Aspekt z pohledu druhého letadla
offset(Agent, Offset)	Offset vzhledem ke druhému agentovi

Tab. 3.3: Vjemy agenta aktualizované v jednom cyklu výpočtu

Na základě vjemů aktualizovaných prostředím se agent snaží splnit své cíle provedením aplikovatelných plánů. Výsledkem provedení plánu je předání akce do agentního prostředí. Implementované akce agenta jsou popsány v tabulce 3.4.
Tyto externí akce jsou předány do modelu, kde jsou transformovány na odpovídající funkce. Výsledkem provedení manévrové akce je výpočet dalšího kontrolního bodu, kterým se stíhač na serverové straně modelu bude snažit proletět. Kompletní výpočetní smyčka na straně agentního prostředí vypadá následovně:

Akce	Interpretace akce
lead_pursuit(Target)	Stíhací manévr předsazená křivka
pure_pursuit(Target)	Stíhací manévr přímá křivka
lag_pursuit(Target)	Stíhací manévr zpožděná křivka
high_yoyo(Target, Height)	Manévr vysoké Yo-Yo s výškou manévru
gain_offset(Target, Offset)	Získat určitý offset od cíle
defensive_spiral(Target, Height)	Manévr obranná spirála s hloubkou spirály
fire(Target)	Vystřelení naváděné rakety na cíl

Tab. 3.4: Seznam akcí dostupných agentům včetně jejich významu.

```
ziskej_nove_pozice_a_rychlosti_ze_serveru();
aktualizuj_vjemy_agentu();
upozorni_agenty_na_zmenu_prostredi();
proved_jeden_krok_agentniho_chovani();
zasli_agentovy_akce_na_server();
zabij_mrtve_agenty();
inkrementuj_citack_kroku();
pockej_cas_jednoho_kroku();
```

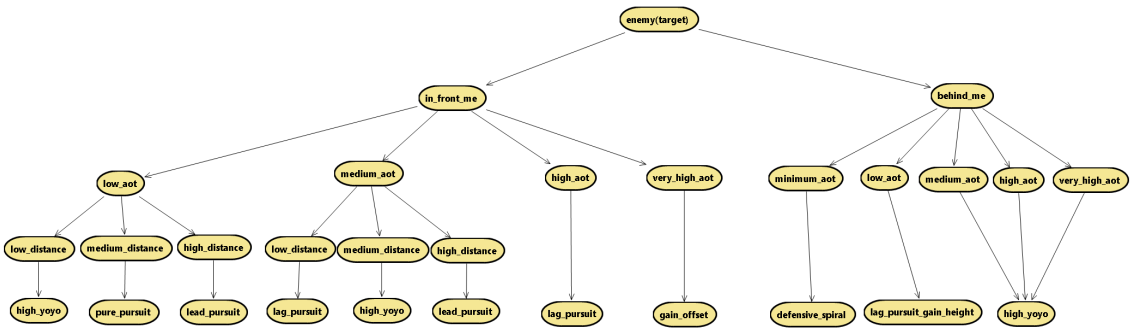
Čas jednoho kroku je definovaný v protokolu jako konstanta **SYNCHRO_STEP**. Pokud běží klient i server na lokální smyčce, je možné nastavit tento parametr na 50 - 100 ms. Konstanta SYNCHRO_STEP je pro agentní prostředí velice důležitá, protože na základě časové prodlevy je vypočítána kompenzace pozic na straně klienta.

3.5.2 Implementace agenta

Chování agenta je modelováno v Jasonu - dialektu agentního jazyka AgentSpeak. Agent pracuje s vjemy a akcemi prostředí popsanými v předchozí kapitole. Vstupním bodem do agentního programu je vjem **tick(Counter)**, který udává číslo kroku. Agent následně vybere nepřítele (pokud nemá) a provádí manévr, aby se dostal do palebné pozice.

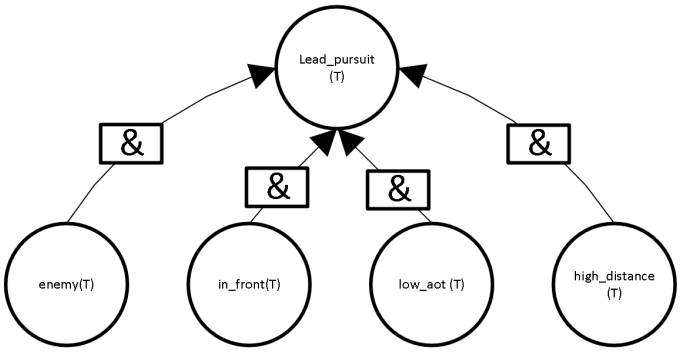
Výběr manévru spočívá v pozici nepřátelského stíhače. Chování agenta je čistě reaktivní, takže se rozhoduje v každém kroku podle pozice a směru vektoru rychlosti nepřítele. Implementované chování agenta lze znázornit obrázkem ve formě rozhodovacího stromu níže v textu 3.3.

Rozhodovací proces na obrázku 3.3 začíná v kořenovém uzlu **enemy(target)**. V druhé úrovni se agent rozhoduje, zda je nepřítel v přední nebo zadní polo-sféře. Následně se podle AOT a vzdálenosti nepřítele rozhoduje, jaký manévr použije. Manévry (potažmo



Obr. 3.3: Rozhodovací strom agenta při útoku na nepřítele.

akce) jsou v grafu znázorněny jako elementy na nejnižší úrovni. Zpětná cesta ke kořenovému elementu vytváří seznam nutných podmínek pro splnění kontextu plánu tak, jak je znázorněno na obrázku 3.4. Chování agenta podle výše uvedeného rozhodovacího stromu je v projektu k nalezení v souboru **lib_attack.asl** v adresáři se zdrojovými kódy klientské části aplikace.



Obr. 3.4: Plán **Lead_pursuit(T)** je proveden, pokud agent nabyde představ, že má nepřítele, nepřítel je před ním, nepřítel se nachází v malém AOT a ve velké vzdálenosti.

3.5.3 Implementace leteckých manévrů

Manévry jsou implementovány jako externí akce v agentním prostředí, které agent volá prostřednictvím Jason API. Externí akce jsou v prostředí obsluhovány metodou **executeAction()**, která jako vstupní parametry přijímá název agenta a strukturu akce, ze které jsou parsovány literály cíle agenta.

Při odevzdávání práce byly implementovány následující manévry (popsané v kapitole 2.3): **Předsazená křivka pronásledování** - agent posílá kontrolní bod, který směřuje před letadlo nepřítele. Akce **lead_pursuit** je parametrizovatelná velikostí předsazení, která je u agenta definována představou **pursuit_offset(PursOffset)**.

Přímá křivka pronásledování - je implementována jako pronásledování pozice nepřátelského agenta, v prostředí je akce identifikovaná jako **pure_pursuit**.

Zpožděná křivka pronásledování - opak předsazené křivky, velikost zpoždění je taky udána představou **pursuit_offset(PursOffset)**, je značená jako **lag_pursuit**.

Vysoké Yoyo - tento manévr, používaný pro získání výšky a stočení směrem k nepříteli, je implementován jako přímá křivka s výškovým posunutím. Výška "Yo-Ya" je dána představou **yoyo_height(YoyoHeight)**.

Získání offsetu - manévr získání offsetu předchází předsazenému otočení. Provádí se v soubojích, při kterých letí letadla čelem proti sobě. Velikost offsetu je vypočítaná z rychlosti letadla jako dvojnásobek poloměru otáčky.

Obranná spirála - jediný čistě obranný manévr Obranná spirála se používá, když se nepřítel nachází za zády agenta. Je parametrizována představou **spiral_height(SpHeight)**.

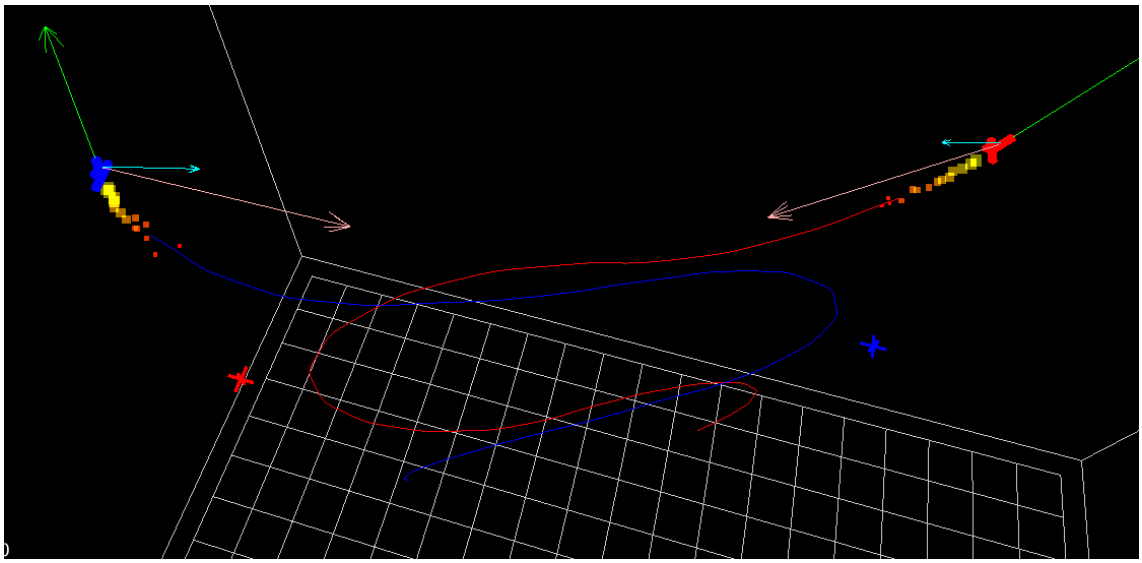
3.6 Vizualizace agentní simulace

Grafické zobrazení leteckého souboje bylo vytvořeno s využitím již zmíněného JMonkey enginu, který vykresluje grafiku pomocí javové implementace grafické knihovny OpenGl.

Server provádí krokovanou simulaci a zajišťuje výpočet fyziky (pohybu) letadel a raket. Všechny akce jsou iniciovány z agentního prostředí pomocí síťové komunikace. Grafický výstup simulátoru na straně serveru je znázorněn na obrázku 3.5 a 3.6.

Po spuštění čeká server na lokální smyčce na portu 9990 na inicializační zprávu od klienta, která obsahuje informace o počtu letadel dvou týmu a jejich rozmístění v rámci scény leteckého souboje. Pro práci v 3D prostoru je používána třída trojrozměrného vektoru **Vector3f**, která je nativně implementována v knihovnách JMonkey. Třída Vector3f byla přenesena i do agentního prostředí, aby nad ní mohly být počítány transformace a nové pozice v prostoru, které jsou klientem zasílány na server. Pokud se rozhodnete zasahovat do komunikace mezi serverem a klientem, je důležité si uvědomit, že **simulace na serveru používá jiné měřítko oproti agentnímu systému** (kvůli názornosti a jednoduchosti agentních výpočtů). Server počítá se 100* menšími hodnotami, které jsou při sestavování zpráv konvertovány do měřítka agentů.

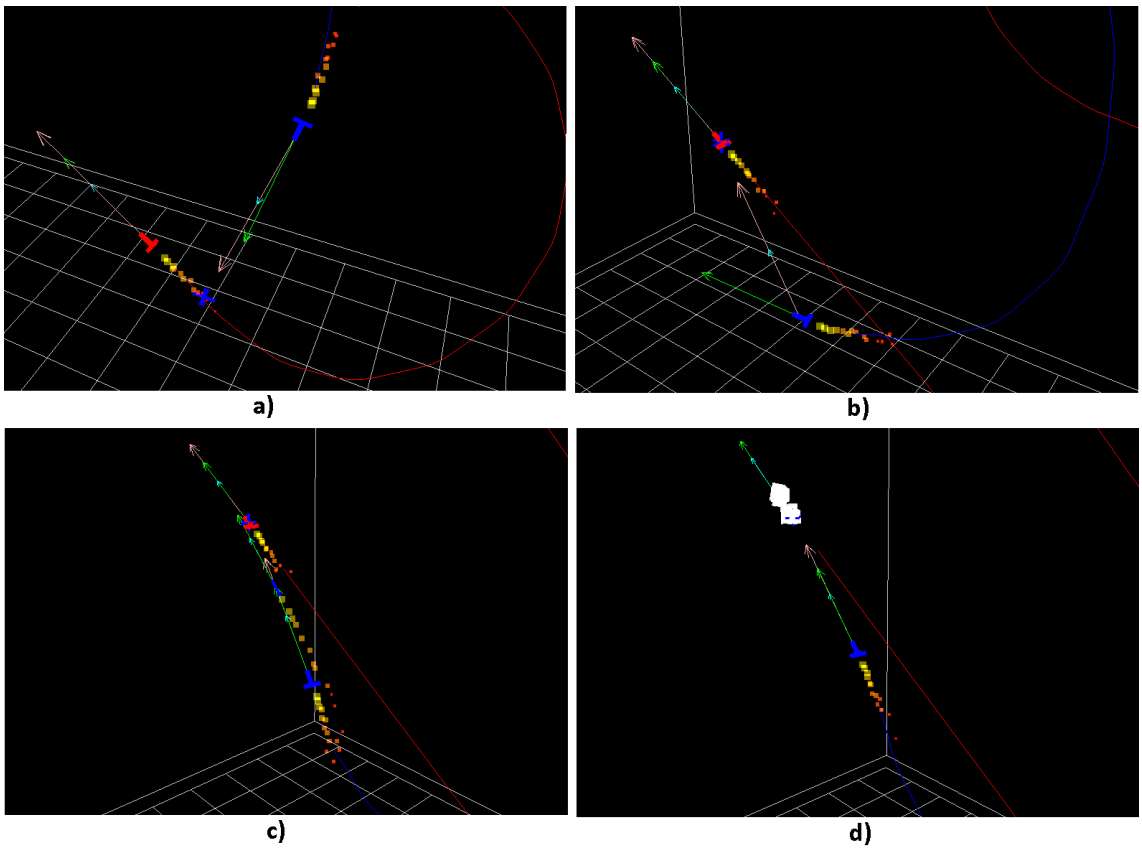
Taktéž je třeba počítat s různými jednotkami rychlostí posílaných přes síť - zatímco jednotlivé výpočty jsou prováděny v **jednotkách SI**, přes síť jsou rychlosti posílány v jednotkách **Mach**, protože člověk sedící v kokpitu stíhacího letadla počítá rychlost v jednotkách



Obr. 3.5: Na obrázku proti sobě bojují dva agenti s identickým chováním. Výsledkem snažení obou dostat se do útočné pozice, se manévrování dostalo do fáze vertikálně směrovaných nůžek. Zelené šipky značí vektor rychlosti letadla, modré šipky znázorňují aktuální zrychlení a hnědé šipky ukazují na kontrolní bod, ke kterému letadlo směřuje.

---

bližších lidskému uvažování. Všechny tyto konverze a poměry jsou uloženy v souboru **Protocol.java**, který je společný pro obě části síťové aplikace.



Obr. 3.6: V prvním pod-obrázku agent nalétává na nepřítele po zpožděné křivce, v druhém obrázku se již dostává do palebné pozice po přímé křivce, na třetím vypouští na cíl naváděnou raketu. Na posledním pod-obrázku dochází k sestřelení nepřítele.

4 ZÁVĚR

V kapitole první byly popsány letecké principy důležité pro úspěch v prvních leteckých konfliktech, přičemž na těchto principech staví všechny moderní strategie pro souboje novodobých stíhačů. Dále byly představeny zbraňové systémy a jejich historický vývoj, neboť jsou důležitou součástí modelování leteckých soubojů. V sekci Základní letové manévry byly představeny jednotlivé manévry, které pilot používá pro přiblížení do správné vzdálenosti, upravení palebného úhlu a nastavení se do útočné pozice, stejně jak manévry obranné, v případě že se nepřítel dostal do výhodnější pozice. V podkapitole Vzájemný souboj 1 na 1 byly popsány taktiky a způsoby, jak by měl pilot bojovat s využitím výhod svého letadla a nevýhod letadla nepřítele. Poslední podkapitola v rámci taktiky stíhacích letadel představila koncept letecké formace a létání ve skupinách. Byly popsány 2 taktiky v závislosti na situaci leteckého souboje a představeny ofenzivní i defenzivní manévry skupiny o dvou letadlech.

Druhá část práce se věnovala vytvoření agentního prostředí, agentního chování a jednoduchého simulátoru pro ověření chování agentů autopilotů. Součástí druhé částí bylo zdokumentování vlastního protokolu navrženého pro účely zabudování agentů do simulačního modelu. Implementační část práce dokumentuje vjemy přidávané agentům v jejich prostředí. Reaktivní agent reaguje na tyto vjemy prováděním relevantních plánů, které vedou k vyvolání externích akcí na agentovo prostředí. V práci jsou zmíněny pouze implementované akce a principy provádění, stejně tak jako princip vyhodnocování relevantních plánů pro agenta na základě pozice nepřítele. Agentní chování potom záleží na konkrétní implementaci agenta a jeho plánů na klientské straně projektu.

Simulátor implementuje fyziku manévrování na základě rychlosti letadla tak, jak je definováno v odborné literatuře o letectví, je však nachystaný na rozšíření fyziky o lineární zrychlení na základě hmotnosti a směru letu vůči gravitačnímu zrychlení a odstředivé zrychlení taktéž založené na hybnosti letadla. Mezi navazující práci patří tedy rozšíření fyzikálního modelu na straně simulátoru a eventuálně rozšíření zbraňového systému o různé druhy raket a střelné zbraně.

Simulátor lze také použít jako demonstrační aplikace pro studenty agentních systémů nebo jako nástroj pro ladění různých letových taktik implementovaných na straně agenta.

Vytvořené agentní chování si dobře počíná ve vzájemném souboji 1 na 1. Ovšem nekomunikuje a nespolupracuje s ostatními agenty, takže do navazující práce určitě patří vytvoření agentů, kteří kooperují a provádějí bojové souboje ve dvojicích. Simulátor je robustní i pro větší množství agentů, takže dalším projektem může být vytvoření chování pro celé letky a testování jejich vzájemných soubojů na vytvořeném simulátoru.

LITERATURA

[1] SHOHAM, Y. *Agent-oriented programming.* Artificial Intelligence, 60:51–92, 1993.

[2] WOOLDRIDGE. M. *An Introduction to MultiAgent Systems.* John Wiley & Sons, 2002.

[3] RAO S., Anand. *AgentSpeak(L): BDI agents speak out in a logical computable language.* In Walter Van de Velde and John Perram, editors, Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), 22–25 January, Eindhoven, The Netherlands, number 1038 in Lecture Notes in Artificial Intelligence, pages 42–55, London, 1996. Springer-Verlag.

[4] BORDINI, Rafael H.; Jomi, Hübner F. *Jason: A Java-based interpreter for an extended version of AgentSpeak.* Department of Computer Science. University of Durham, February 2007. Available online at: http://jason.sourceforge.net/wp/

[5] ODELL, J. *Objects and agents compared.* Journal of Object Technology, ročník 1, 2002: s. 41-53.

[6] RAO, Anand S., et al. *BDI agents: From theory to practice.* In: ICMAS. 1995. p. 312-319.

[7] PADGHAM, L.; WINIKOFF, M. *Developing Intelligent Agent Systems: A Practical Guide.* John Wiley & Sons, 2005.

[8] RUSSEL, S.; NORVIG, P. *Artificial Intelligence: a Modern Approach.* 3rd edn. Prentice Hall, Englewood Cliffs, NJ, 1999

[9] Wooldridge M, Jennings NR. *Intelligent agents: theory and practice. The Knowledge Engineering Review 10.* 1995

[10] BORDINI, Rafael H.; JOMI, Hübner F.; WOOLRIGDE, M. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology).* Wiley-Interscience, první vydání, 11 2007, ISBN 9780470029008.

[11] BRATMAN, ME. *Intention, Plans, and Practical Reason).* Harvard University Press, Cambridge, MA, 1987.

[12] BRATMAN, ME. *What is intention?* In Intentions in Communication (ed. Cohen PR, Morgan JL and Pollack ME). The MIT Press, Cambridge, MA, 1990.

[13] ZBOŘIL, F. *Plánování a komunikace v multiagentních systémech.* (Disertační práce). Brno: Vysoké učení technické v Brně, 2004. Dostupné online: http://www.fit.vutbr.cz/ zborilf/PhD/thesis.pdf

[14] WINIKOFF M, et al. *Declarative and procedural goals in intelligent agent systems.* In Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning, April, Toulouse.

[15] ENGLISH, D. *The Air Up There.* [str.62], ISBN: 0-07-141036-8.

[16] SHAW, R. L. *Fighter Combat: Tactics and Maneuvering.* Naval Institute Press, 6. vydání, 1985. ISBN: 9780870210594.

[17] ŠALBABA, V. *Multiagent simulation model for flight squadrons.* Brno, 2013.

[18] KUSTERER, R. *jMonkeyEngine 3.0 Beginner's Guide: A cross-platform game engine*

for adventurous Java developers. Packt Publishing Ltd, 2013.

[19] *JOGL: Java binding for the OpenGL API.* [online] http://jogamp.org/

[20] *JADE: JAVA Agent DEvelopment Framework.* [online] http://jade.tilab.com/doc/.

[21] G. A. Kaminka, M. M. Veloso, S. Schaffer, C. Sollitto, R. Adobbati, A. N. *GameBots: a flexible test bed for multiagent team research.* Communications of the ACM, vol. 45, no. 1, Jan. 2002.

[22] GEMROT, J., et al.. *Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents.* In: Agents for games and simulations. Springer Berlin Heidelberg, 2009. p. 1-15.

[23] ILLMAN, Paul E. *The pilot's handbook of aeronautical knowledge.* TAB books, 1995.

[24] COMMAND, Air Mobility. *Aeronautical Information Manual: Official Guide to Basic Flight Information and ATC Procedures, February 9, 2012.* As of March, 2012, 29.

[25] BLANCHARD, D. G. *A brief history of air-intercept missile 9 (Sidewinder).* In Proceedings. 1996.

A MANUÁL

Spuštění práce

Pro spuštění práce je třeba mít instalováno 64-bitové Java JDK ve verzi 1.8. Postup spuštění programu (převzato od Ing. Vojtěcha Šalbaby):

1. Spustit `Jason1.3.8\bin\jason.bat` - tak spustíte Jason IDE
2. Nastavit cestu Javu Home (menu *Plugins* → *Plugin Options* → *Jason* a nastavit cestu Java Home na: `C:\ProgramFiles\Java\jdk1.8.0_73\`)
3. Otevřít projekt (menu *File* → *Open* → `dogfight_agents_x64\dogfight_agents_x64.mas2j`)
4. Spustit projekt (menu *Plugins* → *Jason* → *Run Project*)

Pokud se Vám otevře dialogové okno se seznamem agentů, víte, že Vám Jason funguje. Dialogové okno se po chvíli samo zavře, protože vyprší timeout připojení na server.
Proto první spusťte aplikaci serveru a následně spusťte agentní chování.

Javovou aplikaci serveru naleznete v přiloženém CD v cestě
`Dogfight3D\dist\Mygame.jar`.

Server spustíte zadáním příkazu
`java -jar MyGame.jar`.

Agentní prostředí se připojí na server a po úspěšném handshakeu je simulace spuštěna. Před každým dalším spuštěním simulace je třeba restartovat serverovou aplikaci.

Změna spouštěných agentů

Soubor `src/dogfight_agents_x64.mas2j` definuje, jak bude agentní systém vypadat. Jasonovské prostředí přijímá 5 argumentů - první z nich je název agentního systému, ostatní parametry nastavují počet agentů na obou stranách týmu, vzájemnou počáteční separaci a eventuální poziční výhodu jednoho týmu nad druhým. Je třeba dodržet názvy a pořadí číslování agentů, protože prostředí komunikuje s agenty skrze jejich jména, stejně tak jak serverová strana aplikace používá jména aero0, aero1 atd.

Ovládání simulace

Na straně agentního prostředí je možné chování pozastavit například pro účely odladění chování. Na serverové straně je možné pozastavit simulaci stisknutím klávesy **mezerník**. V případě pozastavení je možné simulaci krokovat stisknutím klávesy **n**. Při pozastavení bude stále probíhat na pozadí komunikace s agentním prostředím, pouze se nebude měnit pozice a rychlost letadel. Dále je možné pohybem myši a klávesami **W,S,A,D,Q,Z**, pohybovat kamerou ve scéně.