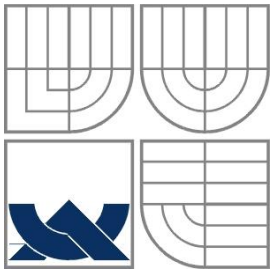
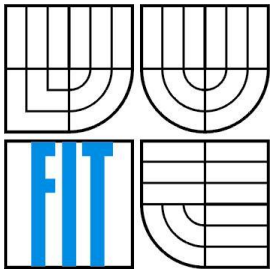


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# VIZUALIZACE ČINNOSTI MULTI-AGENTNÍHO SYSTÉMU POMOCÍ ENGINU UNITY

VIZUALIZATION OF MULTI-AGENT SYSTEM WITH USING ENGINE UNITY

BAKALÁŘSKÁ PRÁCE
BACHELOR‘S THESIS

AUTOR PRÁCE
AUTHOR

JAROSLAV PROKOP

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN SAMEK, Ph.D.

BRNO 2016

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav inteligentních systémů Akademický rok 2015/2016

# Zadání bakalářské práce

Řešitel: **Prokop Jaroslav**
Obor: Informační technologie
Téma: **Vizualizace činnosti multi-agentního systému pomocí enginu Unity**
**Vizualization of Multi-Agent System With Using Unity Engine**
Kategorie: Umělá inteligence

Pokyny:

1. Prostuduje problematiku simulace multi-agentních systému s pomocí nástroje Jason. Dále prostudujte a popište možnosti grafického vykreslovacího enginu Unity pro vizualizaci činnosti agentů v multi-agentním systému.
2. Navrhněte komunikační protokol mezi systémy Jason a Unity pro vizualizaci chování agentů v multi-agentním systému.
3. Vytvořte sadu základních funkcí v enginu Unity pro vizualizaci agentů a agentního prostředí. Tyto funkce budou dostupné s využitím vám navrženého protokolu.
4. Vytvořte sadu demonstračních ukázek v systému Jason pro demonstraci funkčnosti vaší aplikace.

Literatura:


- Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. *Programming Multi-Agent Systems in Agentspeak Using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons.
- MURRAY, Jeff W. *C# game programming cookbook for Unity 3D*. Boca Raton: CRC Press, 2014, xvii, 440 pages. ISBN 9781466581401.


Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Samek Jan, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Petr Hanáček
*vedoucí ústavu*

# Abstrakt

Cílem této práce je popsat základní vlastnosti multi-agentního systému Jason a herního enginu Unity3D. Následně se práce zabývá vytvořením vizualizační aplikace, s kterou bude přes síťovou komunikaci navazovat spojení aplikace napsaná ve frameworku Jason pro modelování multi-agentního prostředí. Ta bude zasílat požadavky na vizualizaci agentů a jejich prostředí na straně vizualizační aplikace.

# Abstract

The aim of this thesis is to describe the basic properties of multi-agent system Jason and Unity3D game engine. Subsequently, the thesis deals with creating visualization application which is going to communicate over a network connection with an application written in framework Jason for modeling of multi-agent environment. This sends an requests to visualization agents and their environment on the side of visualization application.



# Klíčová slova

Multi-agentní systém, Jason, Unity, Unity3D, vizualizátor, vizualizace, engine Unity, komunikace

# Keywords

Multi-agent system, Jason, Unity, Unity3D visualizer, visualization, engine Unity, Communication


# Citace

Prokop Jaroslav: Vizualizace činnosti multi-agentního systému pomocí enginu Unity, bakalářská práce, Brno, FIT VUT v Brně, 2016

# Vizualizace činnosti multi-agentního systému pomocí enginu Unity

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jana Samka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jaroslav Prokop
10. října 2015

## Poděkování

Tímto bych chtěl poděkovat vedoucímu své bakalářské práce, Ing. Janu Samkovi, Ph.D, za odborné vedení, věcné připomínky a poskytnutí ukázkové realizace projektu v Jason.

# Obsah

# Slovník pojmů:

Agent – aktivní prvek systému
Aktuátor – akční člen
Efektor – vykonavatel, realizátor
Engine – jádro, jádro programu
Frame – v překladu znamená *snímek*. Používá se v termínu *Frames per sekond* (snímky za sekundu)
Framework – softwarová struktura sloužící jako podpora při programování softwarových projektů.
Garbage Collector – způsob automatické správy paměti
Konečný automat – výpočetní model složený z několika stavů a z několika přechodů
Mesh renderer – vykreslovač sítě
Motion capture – proces nahrávání pohybu skutečného objektu a jeho převedení na digitální model
Multi-agentní – například prostředí, ve kterém se nachází více různých typů agentů
Mutiplatformní – určený pro víc různých platforem/systémů
Open Source – software, ke kterému je k dispozici zdrojový kód a je možné jej dále upravovat
Paket – blok dat přenášených v počítačových sítích
Placeholder – ukázková hodnota
Platforma – pracovní prostředí
Prefabrikát – předvytvořený objekt
Render – stará se o renderování
Renderování – tvorba reálného obrazu na základě počítačového modelu
Senzor – čidlo, snímač
Shader – vykreslovací řetězec, grafický řetězec
Sprite – dvourozměrný obrázek

# 1 Úvod

Multi-agentní systémy se hojně využívají v případech, kdy by realizace řešení v reálném světě byla příliš nákladná nebo velice časově náročná. Pomocí multi-agentních systémů jde například simulovat těžbu v dolech. V tomto případě je mnohem přínosnější, když je možnost vidět aktuální dění na vlastní oči. Toho bylo možné dosáhnout pomocí jazyku Java, v kterém se jednotlivé prvky systému vykreslovaly.

Unity3D je vcelku mladý herní engine, který si v poslední době získal velkou základnu uživatelů, kteří v něm vyvíjí svoje projekty. Většinou se jedná o hry, ale díky tomu, že je tento engine jednoduchý na ovládnutí a je v něm spousta prostoru na rozvoj jeho funkcí, tak je možné v něm provést takřka cokoli.

Hlavní motivací této práce je vytvořit aplikaci, která bude schopná vizualizovat multi-agentní prostředí a jeho agenty při práci. Nápad na vytvoření lepšího způsobu vizualizace agentů v jejich prostředí vzešel od vedoucího této bakalářské práce, který vyučuje předmět zabývající se multi-agentními systémy a prostředím Jason, v kterém se tito agenti a jejich prostředí realizují.

## 1.1 Obsah kapitol práce

V následující kapitole číslo 2 je popis frameworku Jason pro modelování multi-agentních prostředí a vysvětluje multi-agentní systémy s jejich základními komponentami a důležitými částmi. V kapitole 3 je popsaný herní engine Unity 3D a ve zkratce jsou zmíněny jeho možnosti. Kapitola 4 je věnovaná komunikačnímu protokolu Jason-Unity, kde je popsaný princip komunikace a zpracovávání jednotlivých zpráv. Další kapitola číslo 5 již pojednává o samotné výsledné aplikaci téhle práce. Jsou zde popsány použité nástroje pro její realizaci, její vzhled, architektura a její ovládání. Poslední kapitola je věnována ukázce výstupu z aplikace vizualizéru a testování správnosti výstupu/zobrazování multi-agentního prostředí ve zmíněné aplikaci vizualizéru.

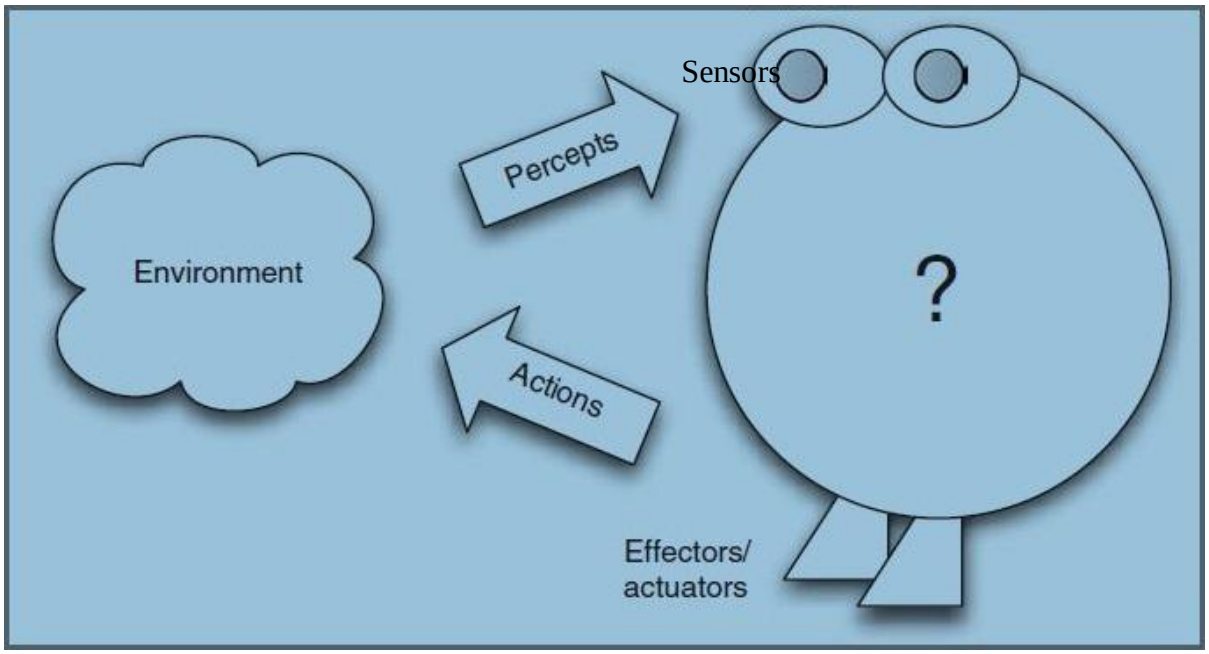
# 2 Jason

Jason je interpret rozšířené verze jazyka AgentSpeak a je psaný v jazyce Java. Implementuje operační sémantiky toho jazyka a poskytuje platformu pro vývoj multi-agentních systémů s mnoha uživateli s přizpůsobitelnými vylepšeními. Jason je dostupný jako Open Source a je distribuován pod GNU LGPL.

Vývojáři interpretu Jason jsou Jomi F. Hübner a Rafael H. Bordini, kteří stavěli na předchozí práci vypracované svými kolegy. [4]

## 2.1 Charakteristiky agentů

Agenty bereme jako systémy, které jsou umístěny v nějakém prostředí, které může být fyzické nebo programové. Tím je myšleno, že agenti jsou schopni vnímat jejich prostředí skrze senzory a mají skupinu akcí, které mohou vykonat skrze efektory a aktuátory za účelem modifikace jejich prostředí. Rozhodování co dělat je založeno na informacích obstaraných ze senzorů. [5]




Obrázek 2.1: Agent a prostředí. Převzato z [5].

Krom toho, že agent musí být umístěný v prostředí, tak očekáváme, že bude mít ještě jiné vlastnosti. Agenti by měli mít následující vlastnosti [5]:

- autonomie;
- proaktivita;
- reaktivita; a
- sociální schopnost.

### 2.1.1 Autonomie

Důležité je si uvědomit, že autonomie je hodně široké spektrum. Na jednom konci spektra máme počítačové programy, jako jsou konvenční textové editory a tabulky, které obsahují malou nebo

žádnou autonomii. Na druhém konci spektra jsme my. Lidé jsou kompletně autonomní. Můžeme si svobodně zvolit, v co budeme věřit a dělat si co chceme – každopádně společnost udržuje naši autonomii v mezích, aby nám zabránila dělat věci, které jsou nebezpečné nám či našemu okolí.

Ve zkratce autonomie znamená schopnost jednat nezávisle za účelem dosažení cílů, které jsme agentům určili. Tudíž, v nejmenším, autonomní agent tvoří nezávislá rozhodnutí o tom, jak dosáhnout delegovaných cílů – jejich rozhodnutí (tudíž i akce) jsou plně pod jejich kontrolou a nejsou řízeny ostatními. [5]

### 2.1.2 Proaktivita

Proaktivita je úroveň osobního zaměření a úsilí na realizaci stanovených cílů. Pokud byl agent delegován pro splnění určitého cíle, bude se od něj očekávat, že se bude pokoušet o jeho splnění. Proaktivita zcela ovládá pasivní agenty, které nikdy nezkusili cokoli udělat. I když většinou nebereme objekt v Javě jako agenta, tak může být takový objekt původně pasivní, dokud něco nezavolá metodu, která je v něm umístěna, například mu řekne, co má dělat. [5]

### 2.1.3 Reaktivita

Být reaktivní znamená odpovídat na změny v prostředí. V praxi to pak znamená, že pokud máme nějaký plán, který je narušen během průběhu, tak dokážeme vymyslet alternativní cestu k jeho splnění.

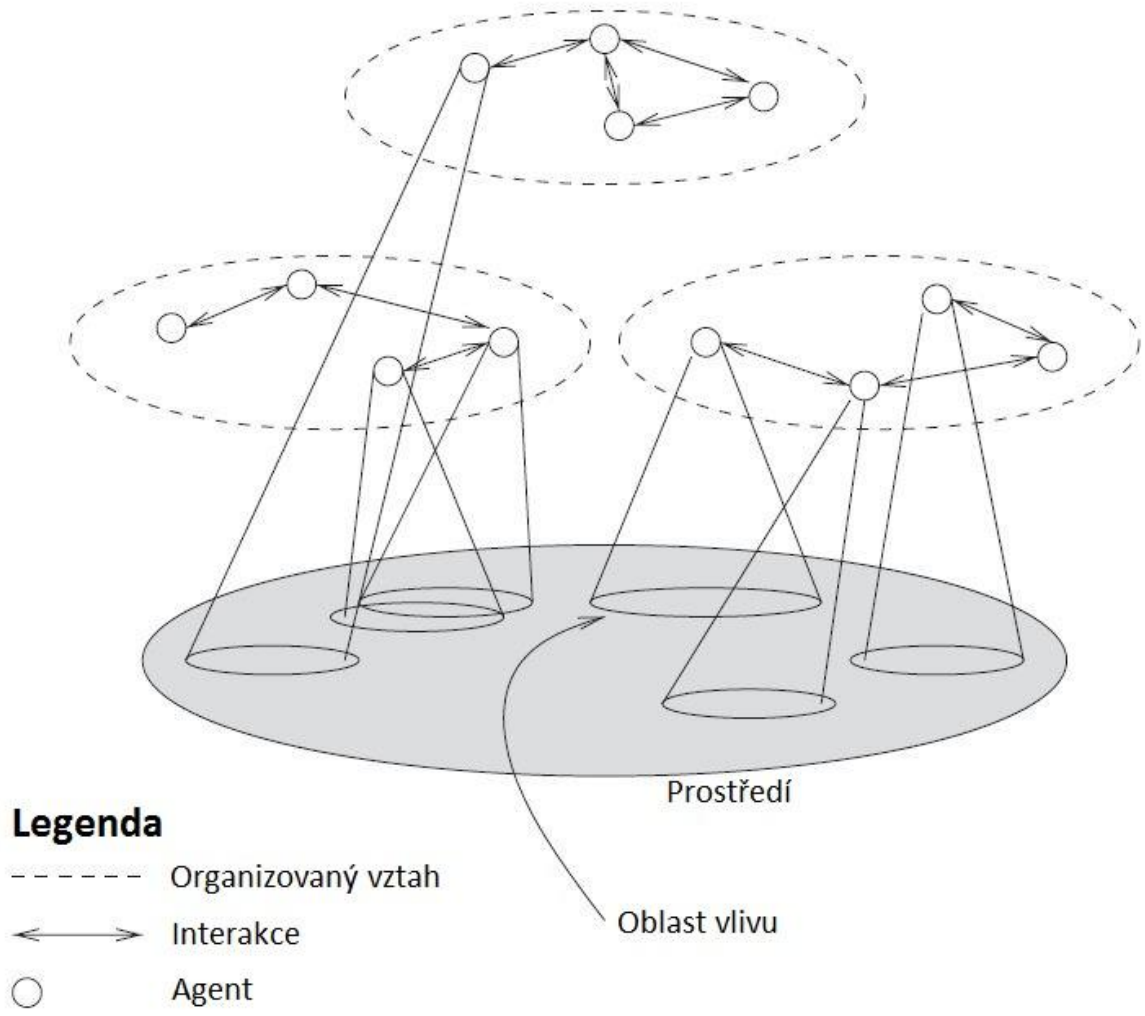
### 2.1.4 Sociální schopnost

Sociálními schopnostmi v našem případě myslíme schopnost koordinovat a spolupracovat s ostatními agenty za účelem dosažení našich cílů. Je užitečné mít agenty, kteří pouze nevyměňují byty informací mezi sebou, ale také umí komunikovat na uvědomělé úrovni. Chceme agenty, kteří jsou schopní komunikovat na základě přesvědčení, cílů a plánů mezi sebou. [5]

## 2.2 Multi-agentní systémy

V praxi jsou jedno-agentní systémy vzácné. Mnohem běžnější je případ, kdy agenti osídlí prostředí, které obsahuje jiné agenty, čímž vytvoří multi-agentní systém. Agenti tedy sdílí společné prostředí a každý agent má svoje „pole vlivu“ v tomto prostředí – to je část prostředí, kterou mohou kontrolovat nebo alespoň částečně ovlivňovat. Tyto oblasti vlivu agentů se mohou překrývat a to znamená, že nejsou obecně kontrolované části prostředí jedním agentem, ale vládne tam spíše spojené ovládání. Kvůli tomu je pro agenty mnohem obtížnější dosáhnout nějakého výsledku v prostředí, které požadují.

Nad prostředím stojí samotní agenti, kteří jsou součástí organisovaných vztahů jeden k druhému. Tihle agenti vědí jeden o druhém, i když se může stát, že agent nebude mít všechny znalosti o ostatních agentech v systému. [5]




Obrázek 2.2: Typická struktura multi-agentního systému. Převzato z [5].

## 2.3 Softwarový model BDI

BDI označuje zkratku pro belief-desire-intention (*přesvědčení-přání-zájem*). Tento softwarový model je založen na základě lidského chování, které bylo ujasněné filozofy, a byl vyvinutý pro programování inteligentních agentů. Původ modelu leží v teorii o lidském praktickém smýšlení vyvinuté filozofem jménem Michael Bratman, která se zaměřuje převážně na roli záměrů v praktickém uvažování. [5]

Zběžné charakterizování implementací představ, zdání a záměrů agenta a využívá těchto pojmů při řešení konkrétních problémů v programování agenta. V podstatě jde o poskytnutí mechanismu pro oddělení aktivity při výběru plánu z právě prováděných aktivních plánů. Jednotlivé plány jsou umístěny v knihovně nebo externí aplikaci. [6]

### 2.3.1 BDI Agenti

BDI agent je druh omezeného racionálního softwarového agenta, naplněného konkrétními duševními postoji. [6]

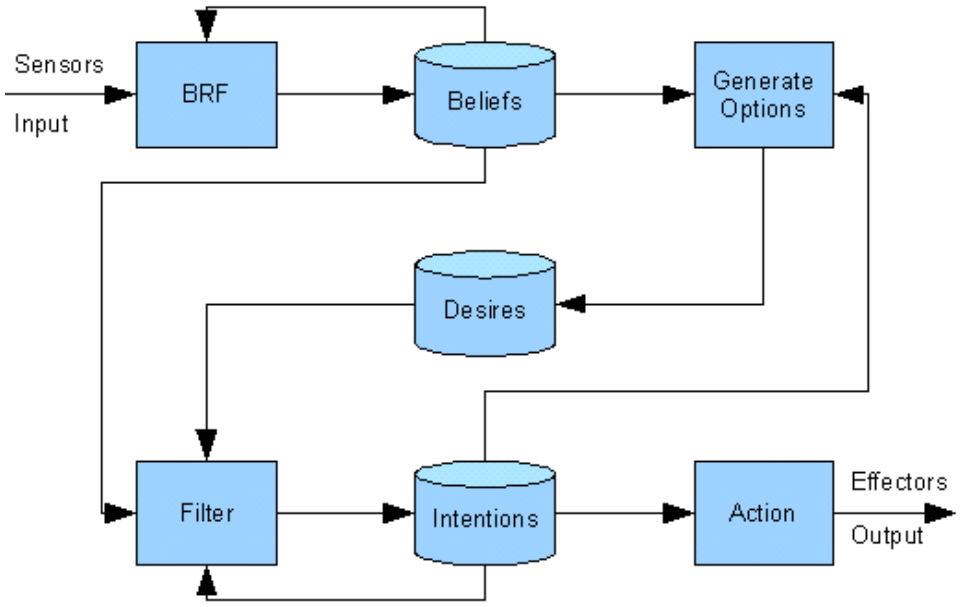
**Vlastnosti**

- Deliberativní – Uchovává symbolickou reprezentaci svého prostředí, případně další znalosti. Je řízen určitou vnitřní inteligencí.
- Řízený logikou –Jjeho akce jsou voleny podle logického kalkulu, oproti agentům zaměřeným na maximalizaci užitkové funkce.
- Racionální – Vykoná akce, které jsou v jeho nejlepším zájmu s ohledem na představy, které má o světě.
- Omezeně racionální – Nemá k dispozici neomezené informace a kognitivní schopnosti a na rozhodování nemá neomezený čas.
- Zaměřený na cíl – Snaží se dosáhnout splnění svých přání či cílů. [7]

**Architektura**

- Přesvědčení – Představují informovaný stav agenta - jeho přesvědčení o světě, ve kterém se vyskytuje (včetně sebe a ostatních agentů). Přesvědčení také zahrnují odvozovací pravidla, která umožňují vpřed řetězení, které vede k novým přesvědčením. Tyhle informace mohou být neaktuální nebo nepřesné. [5][6]
    - Beliefset (*sada přesvědčení*) – Jednotlivá přesvědčení jsou uložena v databázi nebo jinde, dle řešení implementace. [6]
- Přání – Představují motivační stav agenta. Reprezentují objekty nebo situace, kterých by měl agent dosáhnout nebo přivodit. To, že má agent nějaké přání, ještě neznamená, že podle něj jedná. Má pouze potencionální vliv na jeho chování. [5][6]
    - Cíle – Jsou to přání, která byla přebrána pro aktivní činnost agenta. Použití termínu „cílů“ přidává další omezení, kdy musí být množina aktivních přání konzistentní – nesmí si navzájem oponovat. [6]
- Záměry – Představují rozhodovací stav agenta – co se agent rozhodl dělat. Záměry jsou přání, pro které se agent do určité míry rozhodl. V implementovaném systému to znamená, že agent spustil plán.
    - Plány – Představují sekvenci akcí, které agent může provést za účelem dosažení jednoho nebo více záměrů. Plány mohou obsahovat jiné plány.
- Události – Jsou to spouštěče pro reaktivní činnosti agentem. Událost může změnit přesvědčení, spustit plány nebo upravit cíle. Mohou být generování externě a přijaty pomocí senzorů nebo integrované systémem. Navíc mohou být události generovány interně a vyvolat nezávislé aktualizace nebo plány činností. [6]

Důležitou součástí BDI agenta je **plánovač**. Ten na základě přání, záměrů a představ sestavuje plán, jak dosáhnout cílů. Plány však mohou být i dopředu implementovány a v průběhu vybírány ze seznamu plánů. [7]




Obrázek 2.3: Jednoduchá architektura BDI agenta. Převzato z [7].

### 2.3.2 Komunikace agentů

Každá akce, kterou agent vykoná, ovlivňuje stav jeho prostředí. V multiagentním systému se však v prostředí nachází i ostatní agenti, kteří se v tomto systému nachází. Z pohledu agenta je jiný agent pouze prvek v prostředí, na který může působit. Někdy může být proto agentovým záměrem změnit vnitřní (mentální) stav jiného agenta. [8]

Rozlišují se dva způsoby, jakými může agent měnit mentální stav jiného agenta:

- Nepřímé ovlivnění – Agent nepůsobí přímo na jiného agenta, ale mění stav jeho okolí tak, aby ten při kontaktu s tímto okolím změnil svůj postoj žádaným způsobem.
- Přímé ovlivnění – Agent působí na jiného agenta přímo a jediným možným způsobem tohoto ovlivnění je komunikace.

Důvod ke komunikaci s jiným agentem může být několik [8]:

- Dotazování – Pokud agent hledá nějakou informace a dotazuje se jiného agenta, o kterém věří, že může tuto informaci poskytnout.
- Hledání informace – Agenti společně hledají informaci, která není známá ani pro jednoho z nich.
- Přesvědčování – Agent se snaží přesvědčit jiného agenta, aby přijal nějaké jeho záměry
- Vyjednávání – Agenti vyjednávají podmínky o sdílení informací, při kterých všichni zúčastnění dosáhnou maximálního zisku.
- Porada – Hledání řešení pro problém, který se týká všech zúčastněných agentů.
- Eristický dialog – Agenti si vyměňují expresivně informace za účelem dosažení svých záměrů. Informace nejsou ani logickou podporou argumentu, ani vyjednáváním či dotazováním (například hádka).

Vlastní komunikace je proces, během kterého si dva nebo více agentů vyměňují informace ve formě základních zpráv. Každý zpráva má odesílatele, příjemce obsah a informaci o typu zprávy, který určuje její význam – otázka, nabídka, informování či zamítnutí. [8]

### 2.3.3 KQML

Knowladge Query and Manupulation Language (*Vědomostní Požadavek a Manipulační jazyk*) nebo také zkráceně KQML je jazyk a protokol pro komunikaci mezi softwarovými agenty a vědomostně založenými systémy. Byl vyvinut začátkem roku 1990 jako součást projektu pro sdílení znalostí Sharing Effort, který vyvíjela agentura ministerstva obrany Spojených států amerických, DARPA (*Defense Advanced Research Projects Agency*), založená roku 1958. [15] Projekt byl zaměřený na vývoj technik pro budování velkoplošných vědomostních základen, které by byly sdělitelné a znovupoužitelné. [26]

Zpráva v KQML je struktura se syntaxí podobnou jazyku LISP. Vyjadřuje jeden řečový akt, který se liší podle typu (dotaz, nabídka, otázka, souhlas, odmítnutí, atd.). Zpráva se skládá z identifikátoru definujícího, o jaký úkon se jedná, a následující jednotlivé prvky zprávy. Mezi typy komunikačních zpráv patří i takové, které nemají s vlastní komunikací moc společného. Vztahují se například k práci s databází (*Virtual Knowledge Base*, VKB) - příkazy *insert*/*delete* vloží/zruší znalosti ve VKB. Jiné typy zpráv se vztahují k síťové komunikaci. [8]

Formát zpráv a protokol KQML mohou být využity pro interakci s inteligentním systémem jak pomocí aplikace, tak i jiným inteligentním systémem. KQML nahradil FIPA-ACL(*Foundation for Intelligent Physical Agents - Agent Communication Language*). [26]

### 2.3.4 ACL

ACL bylo navrženo Nadací pro inteligentní fyzické agenty (FIPA). Je to navržený standardní jazyk pro komunikaci agentů. [15]

Struktura zprávy ACL je podobná struktuře zprávy KQML. V ACL jsou řečovými úkony pouze takové zprávy, jejichž typy odpovídají řečovým aktům tak, jak jsou obecně chápány. Proto pro každý z těchto aktů může existovat formální zápis vyjadřující jako efekt na komunikačního partnera.

Každý typ zpráv má ve standardu FIPA určenu podmínku použití a dopad na mentální stav příjemce, tzv. racionální efekt. Například při použití informační zprávy je podmínkou, kdy jeden agent věří svému tvrzení a nevěří, že by druhý agent měl povědomí o stavu platností této zprávě, o které jej míní informovat. Racionální efektem, který bude mít vliv na mentální stav informovaného agenta je skutečnost, že první agent bude věřit platnosti zprávy. [8]

## 2.4 Programovací jazyk agenta

Níže jsou popsané hlavní požadavky, které jsou pro AgentSpeak a Jason určeny ke splnění. [5]

- Jazyk by měl podporovat delegaci na úrovni cílů. Jak již bylo dříve zmíněno, jakmile zadáme agentovi nějaký cíl, tak tím nechceme agentovi dát přesný popis akcí, které má vykonat. Převážně s ním chceme komunikovat na úrovni cílů – měli bychom být schopní popsat naše cíle agentovi ve vyšším smyslu, nezávisle na přístupu k dosažení těchto cílů.
- Jazyk by měl poskytovat podporu pro cílené řešení problémů. Chceme, aby byli naši agenti schopni jednat tak, aby dosáhli námi určených cílů, soustavně se pokoušeli o jejich dosažení.
- Jazyk by měl vést sám sebe k produkci systému, které je vnímavý ke svému prostředí.
- Jazyk by měl přesně včlenit cílené a vnímavé chování.
- Jazyk by měl podporovat komunikaci a kooperaci na úrovni znalostí.

# 3 Unity 3D

Unity je multiplatformní herní engine vyvinutý společností Unity Technologies a je využívám pro vývoj videoher pro počítače, konzole, mobilní zařízení a webové stránky.

Podporované platformy: BlackBerry 10, Windows Phone 8, Windows, OS X, Android, iOS, Unity Web Player (zahrnuje Facebook), PlayStation 3, PlayStation 4, PlayStation Vita, Xbox 360, Xbox One, Wii U, Nintendo 3DS line, Wii. [2]

Unity je možné využít pro vytváření aplikací a her díky velkému množství dostupných nástrojů (animace, fyzika, optimalizace, zvuk, 2D a 3D grafika, GUI, aj.) a podporuje scriptování v jazycích JavaScript, C# nebo Boo. [3]

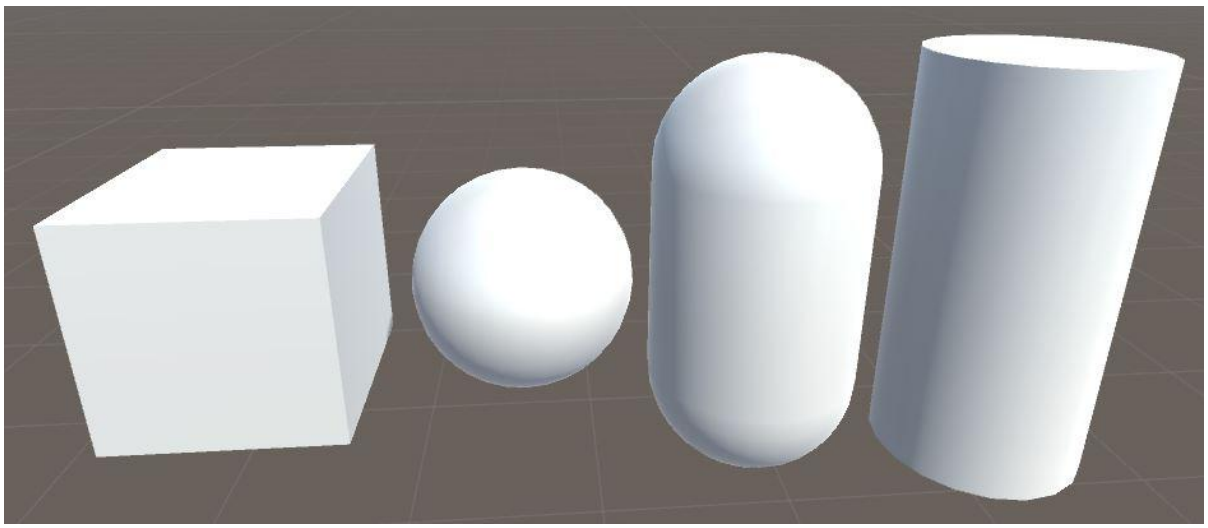
## 3.1 Grafika

Jedním z hlavních faktorů v enginu Unity je využívání právě grafiky, jak 3D, tak 2D, která je hlavní součástí této práce. Níže jsou popsány jednotlivé druhy zobrazování, třídimenzionální (3D) i dvojdimenzionální (2D). Následně je popsáno několik typů osvětlení scény, což má vliv na samotné vykreslování povrchu objektů a jejich textur, případně obrázků ve 2D. Nakonec je v krátkosti popsán vztah mezi materiálem a tónováním.

### 3.1.1 3D

V třídimenzionálním prostředí je možné vykreslovat materiály a textury na povrch objektů, což zapříčiní, že budou tvořit pevné prostředí, postavy a objekty, které budou tvořit virtuální svět. Kamera se obvykle může pohybovat volně po scéně, kde objekty vrhají ve světle realistické stíny. 3D scény jsou většinou referované v perspektivě, takže se objekty jeví větší, když se k nim kamera přiblíží.

3D modely se mohou do prostředí importovat z různých modelových programů. Unity dokáže přečíst data s příponou .FBX, .dae (Collada), .3DS, .dxf a .obj. Dále může z jiných aplikací převést do prostředí objekty pomocí konverze (například z programů Max, Maya, Blender, Cinema4D, Modo, Lightwave a Cheetah3D). Hlavní výhodou vkládání modelů je, že exportujeme pouze taková data, která chceme.



Obrázek 3.1: Ukázka základních 3D objektů v Unity.

Samotné Unity již má několik jednoduchých objektů v nabídce, které je možné využít. Mezi nejzákladnější 3D objekty patří krychle, koule, kapsle a válec.

### 3.1.2 2D

Přesto, že bylo Unity z počátku navrženo pouze pro aplikace v 3D prostředí, v pozdější verzi se dočkalo i možnosti vytvářet aplikace ve 2D, kdy se kamera přesune do pozice nad herní plochou a využívají se pouze osy „x“ a „y“ pro zobrazování herních objektů. Osa „z“ je tak využita pouze pro nastavení vzdálenosti kamery od herní plochy.

Vzhledem k tomu, že je Unity založeno na 3D, tak je možné využívat 3D objekty ve 2D prostředí. Pro tyto účely Unity poskytuje nástroje, které pomáhají vytvářet a spravovat Spritery.

**Sprite Creator -** Dovoluje vytvořit placeholder (*rezervovaný prostor)* pro projekt, takže se může pokračovat ve vývoji bez toho, aby se čekalo na grafiku.

**Sprite Editor -** Dovoluje extrahovat grafiku pro spriter z většího obrázku a upravit několik komponent obrázků do jedné velké textury. Může být použito, pokud chceme mít textury pro různé části modelu (ruce, nohy, kola, …) oddělené od hlavní textury.

**Sprite Renderer -** Využívá se při referování Spriterů namísto Mesh Renderer, který je pro 3D objekty. Je použitý pro zobrazování obrázků jako Spritery jak pro 2D tak pro 3D scény.

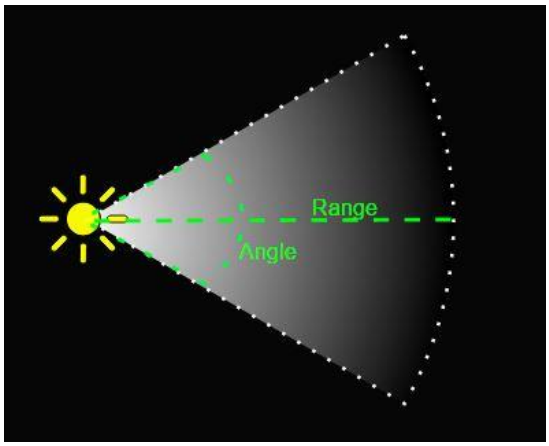
**Sprite Packer –** Je použitý pro optimalizování výkonu video paměti projektem. [16]

### 3.1.3 Osvětlení

Pokud má být správně vypočítané stínování 3D objektů, musí Unity znát intenzitu, směr a barvu světla na objektu. Všechny tyto vlastnosti jsou poskytovány světelnými objekty na scéně. Základní barva a intenzita jsou nastaveny stejně pro všechny světla, ale směr světla je závislý na typu použitelného světelného objektu. Také se může světlo vytrácet v závislosti na vzdálenosti od zdroje. [17]

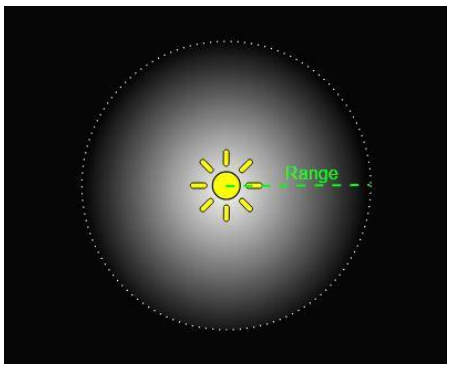
V Unity 3D jsou 4 typy světelných objektů:

**Point Lights** *(Bodové světla)* – Tento typ světla je umístěn na určité místo v prostoru a do všech stran emituje stejné množství světla. Intenzita světla se zmenšuje se vzdáleností od zdroje, kdy v určité vzdálenosti dosáhne nuly. [17]



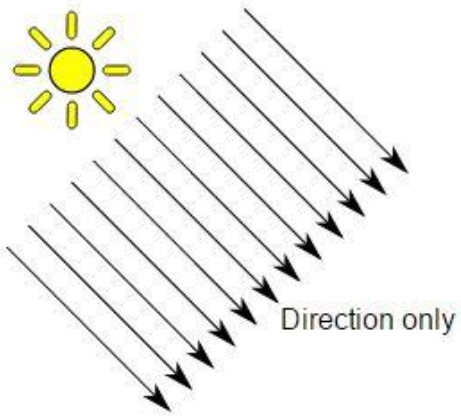

Obrázek 3.3 Ukázka Spot Light. Převzato z [17].

**Spot Lights** *(Místní světla)* – Stejně jako předešlý typ světel, i tyhle mají stanovené místo v prostoru a vzdálenost dosvitu. Navíc však mají úhel svitu, čímž vznikne kužel, kterým světlo putuje pouze v určeném směru. [17]




Obrázek 3.2: Ukázka Point Light. Převzato z [17].

**Directional Lights** *(Směrová světla)* – Směrová světla nemají definovanou pozici, proto může být světelný zdroj umístěn kamkoli na scéně. Všechny objekty jsou osvětleny, jako když je světlo stále ze stejné pozice. Vzdálenost světla od objektů není definována, takže síla osvětlení neklesá. [17]




Obrázek 3.4: Ukázka Direction Light. Převzato z [17].

**Area Lights** *(Prostorové světla)* – Prostorové světlo je definováno jako obdélník v prostoru. Vyzařuje světlo do všech stran, ale pouze z jedné strany obdélníku. Světlo opadá po definované vzdálenosti. Protože je světelná kalkulace velice náročná, není povolena za běhu aplikace a smí využívat pouze lightmaps (*světelné mapy*). [17]


Obrázek 3.5: Ukázka Area Light. Převzato z [17].

### 3.1.4 Materiály a tónování

V Unity je velmi blízký vztah mezi materiálem a tónováním. Materiál určuje jeden speciální tón k použití a zvolený tón určuje, které možnosti jsou k materiálu dostupné.

Pro většinu běžného vykreslování (čím se myslí postavy, scenérie, prostředí, pevné i transparentní objekty, tvrdé a měkké povrchy atd.) je Standard shader nejlepší možností. Je to velice přizpůsobitelný shader, který je schopný vykreslování mnoha povrchů velice realistickou cestou.

**Standart Shader** je vždy zvolený jako výchozí tón při vytváření nového materiálu. Standart Shader obsahuje již většinu vlastností tónování, jako například tvrdost, transparentnost, zrcadlení, aj.. Díky tomu je možné pouze zvolit libovolné vlastnosti a ty se již aplikují na materiál. Je tu také zahrnutý mnohem vyspělejší světelný model, který se nazývá Physically Based Shading *(Pohybově založené stínování)*. [18]

**Physically Based Shading** (PBS) simuluje interakci mezi materiály a světlem způsobem napodobování reality. Funguje nejlépe v situacích, kdy osvětlení a materiály potřebují být sehrány intuitivně a realisticky. Aby k tomu došlo, následuje principy fyziky včetně uchovávání energie, Fresnelův odraz a jak předměty pohlcují samy sebe mezi ostatními. [18]

## 3.2 Fyzika

Pro přesvědčivé fyzikální chování objektů ve hře je nutné mít správnou akceleraci a ovlivnitelnost kolizemi, gravitací a jinými silami. Pro tyto účely má Unity vestavěný fyzikální engine, který poskytuje komponenty schopné simulovat fyzikální chování za nás. Díky tomu, a troše skriptování, můžeme vytvořit dynamické objekty, které se chovají jako vozidla, stroje nebo kousky látky.

### 3.2.1 Rigidbody

Rigidbody (*tuhé těleso*) je hlavním komponentem pro aktivování fyzikálního chování u objektů. Pokud je k objektu připojený, okamžitě začne objekt reagovat na gravitaci. Pokud se navíc k objektu připojí nějaký Collider (*kolizní tvar*, viz dále), bude se pohybovat v závislosti na kolizích.

Vzhledem k tomu, že Rigidbody komponent přebírá pohyblivost objektu, ke kterému je připojený, tak by neměl měnit pozici pomocí skriptu, kde nastavíme jeho novou pozici přes Transform, ale měla by se na něj aplikovat síla, která bude vyhodnocena ve fyzikálním enginu a objekt se následně posune chtěným směrem.

Jakmile se objekt pohybuje, ať už se jedná o pohyb, rotaci nebo cokoli jiného, tak fyzikální engine předpokládá, že se jednou zastaví. Jakmile k tomu dojde, tak se rigidbody přesune do stavu spánku, kdy se nespotřebovává žádné zdroje, dokud není znovu ovlivněn kolizí nebo jinou fyzikální silou, která ho přivede do pohybu. [19]

### 3.2.2 Colliders

Pomocí Colliders (*kolizních tvarů*) definujeme tvar objektu pro účely fyzikálních kolizí. Collider je neviditelný a nemusí mít totožný tvar jako samotný tvar objektu a většinou je i mnohem efektivnější, pokud nemá úplně přesný tvar jako objekt, ale pouze se mu zhruba podobá. Čím je Collider jednodušší, tím je méně náročný na procesor a tudíž tolik nezatěžuje systém. Převážně díky nejjednodušších kolizních tvarů můžeme vytvořit vcelku podobný tvar, jaký má objekt a zároveň udržet nízkou náročnost na zdroje. [20]

Nejjednodušší kolizní body ve 3D:

- Box Collider (*krychlový kolizní tvar*),
- Sphere Collider (*kulovitý kolizní tvar*),
- Capsule Collider (*kapslový kolizní tvar*).

Kolizní body ve 2D:

- Box Collider 2D (*krychlový kolizní tvar 2D*),
- Circle Collider 2D (*kruhový kolizní tvar 2D*).

Mohou ovšem nastat případy, kdy ani kombinace jednoduchých kolizních tvarů nebude stačit. Ve 3D proto můžeme využít Mesh Collider (*síťový kolizní tvar*), který vytvoří přesný kolizní tvar objektu na základě síťové reprezentace jeho tvaru. [20]

### 3.2.3 Joint

Joint (*spoj*) je, díky které můžeme spoji dvě rigidbody dohromady nebo vytvořit fixní bod v prostoru. Většinou chceme tomuto spojení umožnit určitou svobodu pohybu, proto je několik různých typů spojů:

- Hinge Joint (*závěsný spoj*) – povoluje objektu rotaci po osách,
- Spring Joint (*pružinový spoj*) – ponechává objekty rozdělené, ale povoluje jim se od sebe lehce vzdálit.,
- Hinge Joint 2D (*závěsný spoj 2D*),
- Spring Joint 2D (*pružinový spoj 2D*).

Spoje mají také jiné možnosti, které se dají aktivovat. Jako například nastavit zničení spoje, pokud je na něj aplikovaná síla v určitém rozsahu. Některé spoje dovolují nastavit propojení hnací síly mezi objekty, čím je uvede všechny do pohybu automaticky. [21]

## 3.3 Skriptování

Jak bylo již v úvodu kapitoly zmíněno, tak Unity podporuje skriptování v jazycích JavaScript, C# a Boo. Unity využívá implementaci standardu Mono runtime pro skriptování, avšak stále má své postupy a techniky pro přístup k enginu ze skriptů. [1]

## 3.4 Animace

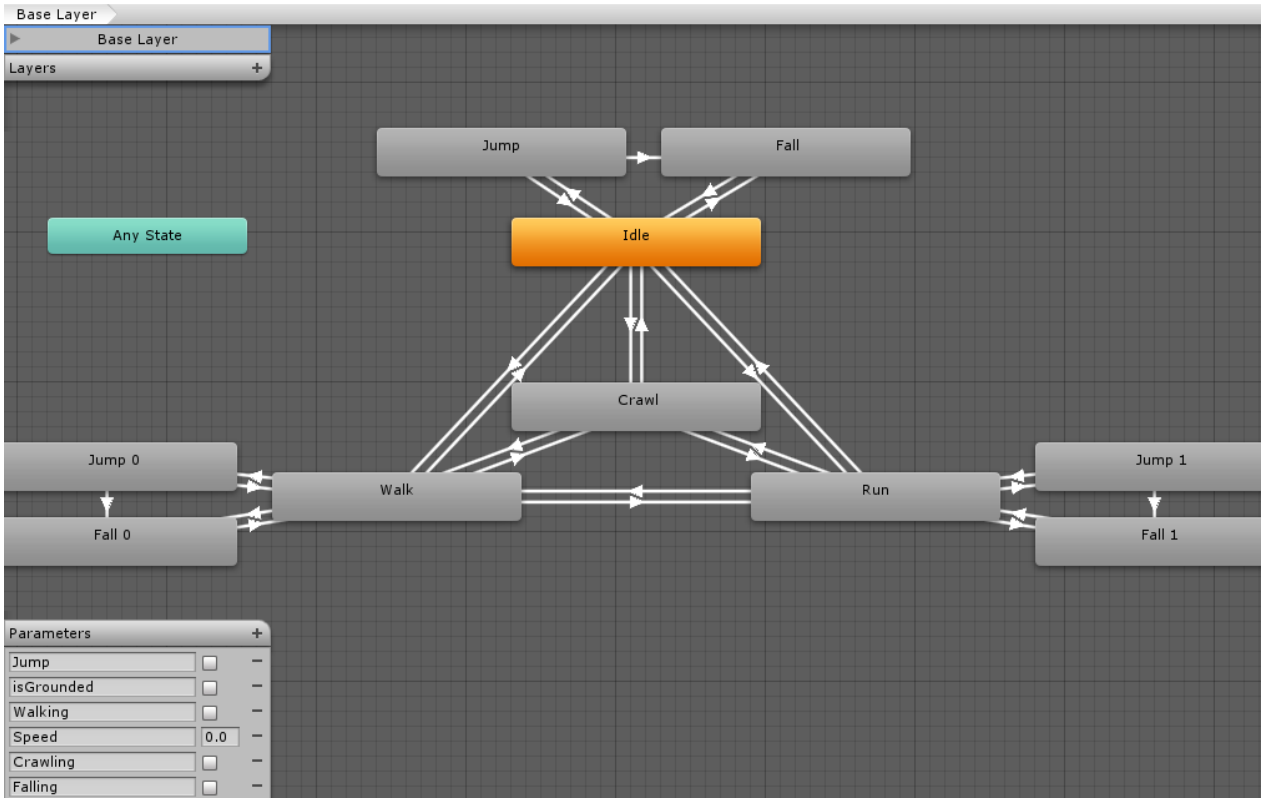
Unity má bohatý a sofistikovaný animační systém, který poskytuje:

- Jednoduchý průběh animací pro všechny elementy Unity, včetně objektů, postav a vlastností.
- Podporuje importované animované klipy a animace vytvořené vně Unity.
- Přesměrování animací humanoidů – schopnost aplikovat animace z jedné postavy na jinou.
- Zjednodušené pracovní postupy pro spojení animačních klipů.
- Pohodlný náhled na animační klipy, přechody a interakce mezi nimi. To umožní animátorům pracovat nezávisle na programátorech.
- Management komplexních interakcí mezi animacemi s vizuálním programovacím nástrojem.
- Animování různých částí těla s rozdílnou logikou.
- Vrstvení a maskovací funkce. [22]

Animační systém je založený na konceptu animačních klipů, které obsahují informaci o tom, jak mají určité objekty změnit svou pozici, rotaci nebo jiné vlastnosti během času. Každý klip může

být brát jako jedna lineární nahrávka. Animační klipy z externího zdroje jsou vytvořeny umělci nebo animátory pomocí 3D nástroje jako Max nebo Maya nebo jsou pořízeny pomocí motion capture studií či jiných zdrojů.

Animační klipy jsou organizovány do strukturovaného systému, který se nazývá Animator Controller. Ten se chová podobně jako konečný automat a hlídá, které klipy mají být zrovna přehrány, když by se měla změnit animace nebo splynout s jinou. Animator Controller může obsahovat jen dvě animace (například otevírání a zavírání dveří) nebo několik desítek animací (veškeré animace pro postavy, zvířata, aj.). [23]




Obrázek 3.6: Ukázka rozhraní Animátoru v Unity. Převzato z [27].

# 4 Komunikační protokol Jason-Unity

V rámci této práce byl vytvořen komunikační protokol mezi multi-agentním systémem Jason a herním enginem Unity.

Komunikace je zajištěna pomocí protokolu TCP/IP a je vedena oběma směry. Z prostředí Jason (klient) se odešle příkaz, který definuje akci, co se má vizualizovat na straně aplikace (serveru). Ta pak následně odpoví připojenému klientovi kódem, který reprezentuje, zda byla během provádění operace, zaregistrována jakákoli chyba.

## 4.1 Klient-server

Model klient-server je distribuovaná aplikační struktura, která rozděluje úkoly nebo pracovní zatížení mezi více poskytovatelů zdrojů nebo služeb, nazývané servery a žadatele služeb, nazývané klienti.

### 4.1.1 Klient

Klient je aplikace, která žádá o poskytnutí služeb server.

Vzhledem k tomu, že klient je v podstatě aplikace napsaná ve frameworku pro modelování multi-agentních prostředí a ta se stará o všechnu logiku, tak klientovy požadavky jsou všechny zaměřené na provedení určitých změn na straně serveru.

### 4.1.2 Server

Server je program, který přijímá požadavky a posílá zpět výsledky v odpovědích. Server se může obvykle vypořádat s více požadavky od několika žádajících klientů ve stejnou chvíli. Většina serverů čeká na požadavky na „dobře známém“ portu, takže jejich klienti vědí, který port mají použít pro nasměrování svých požadavků. [9]

V tomto případě se server stará o znázornění všech požadavků, které od klientů přicházejí. Postup je takový, že klient se připojí na server a zašle mu požadavek na vykreslení nového agenta. Server tento požadavek přijme, agenta vykreslí a následně klientovi odešle odpověď, zda se podařilo jeho požadavek zpracovat správně.

### 4.1.3 Komunikace klient-server

Klient i server jsou aplikační procesy, které komunikují přes síťové rozhraní. Může jít i o procesy běžící na stejném počítači. Základní činností klienta je posílání žádostí o nějakou síťovou službu. Server čeká na příchozí požadavky, přijímá je, zpracovává a posílá zpět odpovědi.

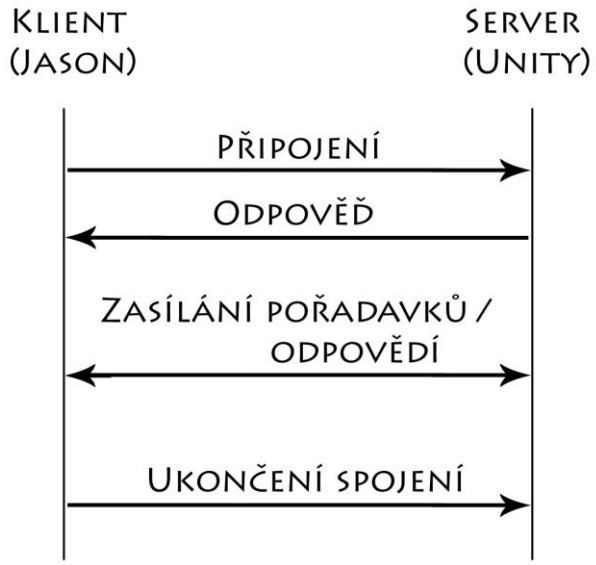
Důležité je, že zpracování požadavku probíhá výhradně na straně serveru. Klient pouze předává požadavek a zobrazuje odpověď. U modelu klient-server komunikaci obvykle začíná klient. Server čeká na dotazy a odpovídá je. Většinou umožňuje zpracování více požadavků najednou. Komunikace mezi serverem a klientem je popsána tzv. protokolem.

## 4.2 Komunikace Jason-Unity

Jak již bylo napsáno výše, tak komunikace je řešené pomocí protokolu TCP/IP, který zajišťuje bezztrátový přenos dat. Veškeré příkazy pro vizualizaci posílá aplikace v Jason. Po odeslání požadavku na stranu Unity se čeká na jeho zpracování. Po dokončení požadavku a případně následné vizualizaci na straně serveru (Unity), server odešle návratovou hodnotu, která indikuje, jestli byl požadavek vyřízen v pořádku nebo nastalo k nějakému problému při jeho zpracovávání.

Komunikace probíhá následovně:

1. Klient se připojí k serveru pomocí jeho IP adresy a portu.
2. Server odešle informaci o tom, jestli se klient připojil a je připravený přijímat od klienta požadavky.
3. Následně už může klient posílat příkazy ke zpracování. Na každý odeslaný příkaz přijde odpověď v podobě kódu, který reprezentuje stav zpracování. (0 – úspěch; jiný než 0 – chyba)
4. Po dokončení komunikace klient odešle požadavek pro ukončení spojení.




Obrázek 4.1: Ukázka průběhu komunikace mezi Jason a Unity.

## 4.3 Zpracování požadavku

Požadavky přijímá server, který je na straně Unity. Většina požadavků je čistě vizualizace objektu na obrazovce. K serveru se může připojit více klientů současně. Pokud se tak stane, pro každého klienta je vytvořen objekt s potřebnými třídami, který se stará o komunikaci s daným klientem. To znamená, že pokud se připojí na server 6 klientů, každý bude mít vlastní objekt, který bude zpracovávat pouze jeho požadavky. Pro všechny je k dispozici jeden interpret, který zpracovává požadavky tak, jak mu je postupně klienti vkládají.

Samotné zpracování požadavku na straně serveru má následující kroky:

1. Proces, který se stará o příjem požadavků od daného klienta dostane upozornění, že jsou na lince přítomné nové data.

2. Všechna přijatá data se uloží do paměti a ověří se, zda jsou všechny staženy.
    a. Pokud nejsou staženy celé (požadavek není ukončený znakem konce řádku „\n"), tak se přijaté data uloží do mezi-paměti a vrátí se na krok 2.
    b. Pokud jsou stažena všechna data, tak se pokračuje následujícím krokem.
3. Přijaté požadavky se odešlou ke zpracování, kde se rozdělí na jednotlivé příkazy a jejich parametry.
4. Takto upravený seznam se pošle do *commander (velitel)*, což je objekt na pozadí, který obstarává interpretaci příkazů, které přijdou od klientů a stará se o jejich interpretaci. Pokud při jejich zpracovávání dojde k nějaké chybě, zpracovávání se ukončí a klientovi se odešla informace v podobě chybového kódu o tom, k jaké chybě došlo.
5. Po zpracování požadavku se odešle klientovi informace o jeho správném zpracování.

# 5 Vizualizátor Unity

Hlavním produktem této práce je aplikace pro vizualizaci multi-agentního systému. V této kapitole se budu věnovat funkcím a zpracování vizualizátoru. Aplikace je navržena tak, aby zobrazovala 2D prostředí s agenty a různými objekty. Bylo by také možné vytvořit 3D vizualizaci.

## 5.1 Použité nástroje

V této části jsou uvedeny všechny použité nástroje k vytvoření aplikace, jejich popis a jaké uplatnění při tvorbě měly spolu s motivací, proč jsem si tyto nástroje vybral.

### 5.1.1 Unity 3D

Unity je detailněji popsáno výše v kapitole 3.

Pro vytvoření samotné aplikace pro vizualizaci bylo použito právě enginu Unity, který je přímo stvořený pro vytváření pro podobné úkony. Jelikož je to herní engine, tak již podporuje spoustu funkcí, které napomohly k realizaci vizualizátoru.

Celá síťová komunikace je psaná právě na straně Unity, spolu s celým systémem vizualizovaní agentů a prostředí pro jejich pohyb.

### 5.1.2 C#

C Sharp je vysokoúrovňový objektově orientovaný programovací jazyk vyvinutý firmou Microsoft spolu s .NET framework. Je založený na jazycích C++ a Java.

V C# neexistuje vícenásobná dědičnost – každá třída může být potomkem pouze jedné třídy. Třída ovšem může implementovat libovolný počet rozhraní. Neexistují tu globální metody ani proměnné, namísto toho jsou využity statické metody a proměnné veřejných tříd. Často se u objektově orientovaného programování používá zapouzdření atributů, kdy se k nim dá přistoupit pouze nepřímo a to pomocí dvou metod get (přístup) a set (změna). V jazyce C# se rozlišuje mezi velkými a malými písmeny – je case sensitive. Dále zprostředkovává uvolňování paměti pomocí tzv. garbage collector (volně přeloženo: *popelář, sběrač odpadků*). Ten zajišťuje, aby všechny nepropojené objekty (v aplikace na ně neexistuje jakýkoli odkaz) byly z paměti vymazány. [24]

Jazyk C# jsem využíval ve všech skriptech pro Unity. Je v něm naprogramovaná veškerá síťová komunikace na straně Unity aplikace pro vizualizaci a také ovládání aplikace a její vizualizovaní obdržených příkazů.

### 5.1.3 Jason

Celá 2. kapitola je věnovaná právě nástroji Jason, proto pouze uvedu jeho samotné využití pro následné řešení.

Jason jsem využil pro upravení projektu napsaného v tomto prostředí tak, aby zasílal požadavky pro zobrazení na spuštěnou unity aplikaci, která je následně vizualizuje. Byl tak vytvořen ukázkový příklad komunikace a bylo možné otestovat správnost řešení.

Vedoucí této bakalářské práce na fakultě Jason vyučuje a rád by měl k dispozici nějaké pokročilejší možnosti vizualizace činnosti agentů, které by mohl zapojit do výuky, proto jsem si zvolil právě tento multi-agentní systém.

### 5.1.4 Java

Je to objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems a prezentovala ho roku 1995. Je navržený pro podporu síťových aplikací. Java je také interpretovaný jazyk, což znamená, že se místo strojového kódu vytváří mezikód. Ten je nezávislý na architektuře počítače nebo zařízení, proto pak může program běžet na libovolném počítači nebo zařízení, což jej činní multiplatformní a přenositelným. Stejně jako C#, tak i Java využívá automatický garbage collector, takže se programátor nemusí starat o uvolňování paměti, pouze o zrušení všech odkazů na objekty. [25]

Jazyk Java byl využit pouze při vytváření testovacího a ukázkového projektu. Celá komunikace s vizualizační aplikací na straně Jason je tedy psána v jazyce Java.

### 5.1.5 Eclipse

Je to open source (*otevřený software*) vývojová platforma, který je určená pro programování v jazyce Java. Eclipse byl vytvořen vývojáři z Eclipse Foundation a jeho první vydání bylo již v roce 2001. [14]

### 5.1.6 Bitbucket

Bitbucket je webová služba, která zajišťuje podporu pro vývoj softwaru při používání verzovacích nástrojů Git a Mercuria. Poskytuje neomezené repozitáře na zálohování projektů jednotlivců či skupin do 5 členů zdarma.[11][13] Má jej na svědomí společnost s názvem Atlassian, která zajišťuje nástroje pro týmové projekty. [12]

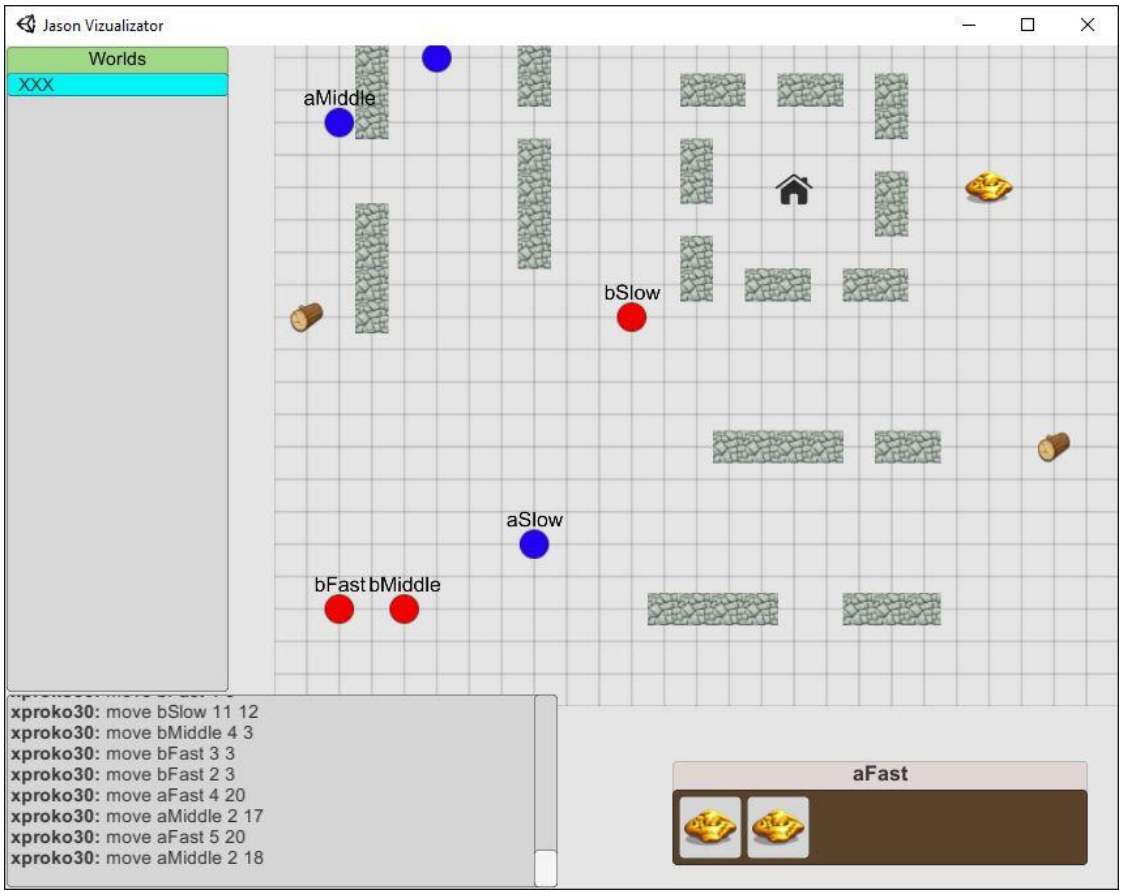
Již jsem několik vlastních projektů zálohoval na této doméně a díky kladným zkušenostem jsem si ji zvolil znovu.

### 5.1.7 SourceTree

Byl vytvořen společností Atlassian, stejně jako Bitbucket. Tato aplikace souvisí s předchozím nástrojem Bitbucket a zajišťuje uživatelské rozhraní pro zálohování dat do repozitáře.

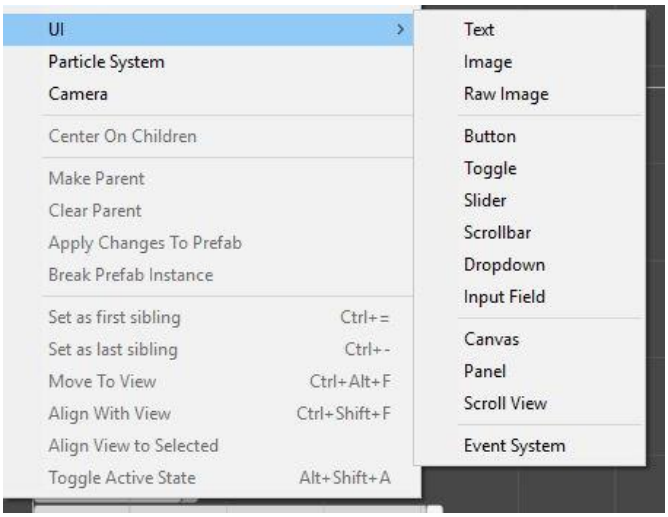
## 5.2 Uživatelské rozhraní

Uživatelské rozhraní jsem se snažil udržet jednoduché, aby nebylo ovládání aplikace příliš složité nebo zbytečně překombinované. Rozhraní se skládá z levého panelu, kde se zobrazuje seznam všech vytvořených světů. V dolní části obrazovky je log příchozích požadavků. Hlavní část pak tvoří samotné zobrazení světa, ve kterém jsou agenti. Inventář agenta je pak zobrazený v pravém dolním rohu.

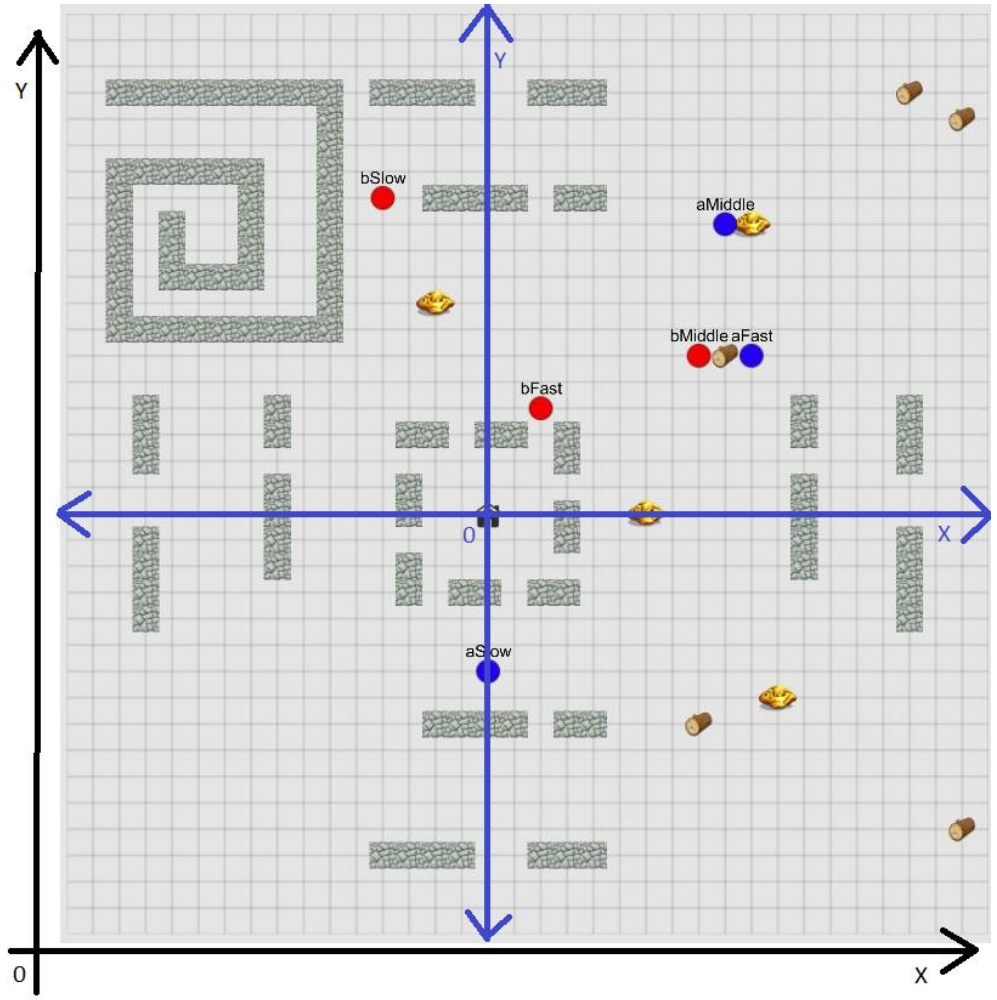



Obrázek 5.1: Okno aplikace pro vizualizaci.

Mřížka na pozadí, která zobrazuje jednotlivé body prostředí, je vytvořena pomocí jednoho objektu, na kterém je vykreslená textura mřížky. Velikost mřížky se odvíjí od samotné velikosti prostředí, které definuje uživatel příslušným příkazem. Je možné si vybrat ze dvou pozic, na kterých bude umístěn bod [0;0]. První pozicí je levý dolní roh a druhou je střed.




Obrázek 5.2: Seznam jednotlivých prvků rozhraní.




Obrázek 5.3: Dvě různá umístění bodu [0;0]

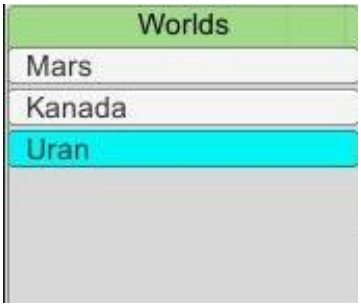
Rozhraní je vytvořeno pomocí vestavěného uživatelského rozhraní v Unity. To umožňuje brát základní prvky (text, panel, tlačítko, …), vložit je přímo na scénu a vytvořit díky nim požadované prvky rozhraní. Všechny prky jsou vykreslovány na plátno (*Canvas)* a řízeny pomocí událostního systému (*Event System*), který se stará o všechny uživatelovi vstupy a předává je dál.

K celkové funkčnosti uživatelského rozhraní bylo použito několik skriptů, které mají na starost vykreslování například dalších záložek světů nebo předměty v inventáři agenta.

### 5.2.1 Seznam světů

Seznam světů je panel, ve kterém jsou uspořádány všechny vytvořené světy. Po kliknutí na některý ze světů se předchozí svět schová a zobrazí se uživatelem vybraný svět spolu se všemi agenty, předměty v prostředí a překážkami.

Samotný seznam je neomezeně dlouhý a může tak být vytvořeno tolik světů, kolik budou klienti požadovat. V tomto ohledu také záleží na samotném stroji, na kterém aplikace běží, protože se může stát, že při zpracovávání více světů bude zapotřebí velkého množství výpočetních zdrojů.




Obrázek 5.3:Seznam světů v aplikaci.

Zobrazené světy jsou uloženy ve třídě *WorldsList*, která obsahuje seznam všech světů, které jsou pro zobrazování v rozhraní realizovány pomocí třídy *WorldListItem*.

Třída WorldList

V této třídě je seznam všech dostupných světů. Zajišťuje zobrazování jejich seznamu na postranním panelu rozhraní, také jejich přidávání do seznamu a mazání. Při smazání se stará o odstranění agentů z klientova seznamu všech vytvořených agentů a smazání všech předmětů v prostředí daného světa. Také odkazuje na třídu *WorldListItem* právě zobrazeného světa. V případě, smazání světa, který je právě zobrazený, tak se zobrazí první svět na seznamu všech světů.

Třída WorldListItem

Jednotlivé prvky v seznamu světů mají implementovanou tuhle třídu. Řeší se tu pouze dvě věci. Výběr světa a zneviditelnění všech objektů na scéně.

Při změně výběru světa ze seznamu se nejdřív změní označení světa. To znamená, že se změní barva pozadí vybraného prvku v seznamu světů, nastaví se vybraný svět uživatelem jako aktivní a následně se všechny objekty na scéně zneviditelní a zviditelní se pouze ty, které jsou ve vybraném světě.

Zneviditelnění objektů se provede tak, že se získají jejich rendery, texty a případně interaktivní oblasti (kolizní body, které jsou nastavené na interakci s uživatelem, jako je zobrazení inventáře agenta po kliknutí je agenta), které se následně vypnou. Tím se docílí toho, že všechny objekty jsou dále dostupné, i když nejsou zrovna viditelné.

### 5.2.2 Svět

Reprezentace světu v aplikaci je vyřešena pomocí třídy *World*. Protože všechny hlavní záležitosti ohledně zobrazování a zviditelňování objektů v prostředí jsou již řešeny přes *WorldList* a *WorldListItem* třídy. Pokud je nastavený *WorldLock* v souboru config.ini, tak může svět smazat pouze vlastník, tj. klient, který svět vytvořil.

Třída World

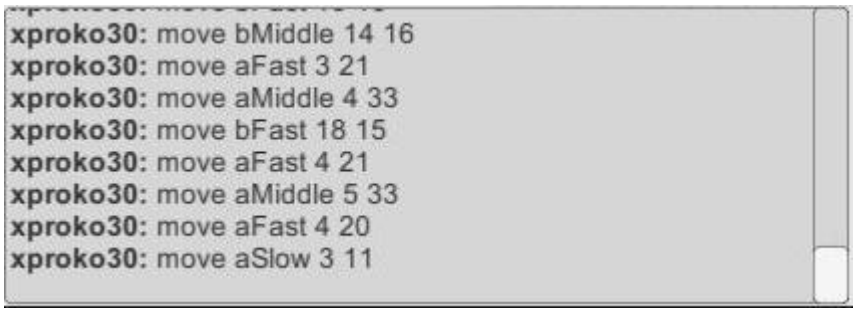
Obsahuje seznam agentů a seznam objektů v prostředí daného světa, které se využívají při jeho mazání. Tehdy je zapotřebí mít odkaz na jednotlivé předměty v prostředí spolu s agenty a jejich vlastníky (klienty), aby bylo možné jejich úplné smazání a klienti tak neodkazovali na agenty, kteří již neexistují. Také má v sobě odkaz na klienta, který objekt vytvořil.

### 5.2.3 Log událostí

Zaznamenává všechny příchozí požadavky a zobrazuje je v logu, který je umístěný v dolním levém rohu obrazovky. Bohužel při větším množství textu se stávalo, že byla aplikace velmi náročná na zdroje, protože Unity mělo tendence vykreslovat i text, který byl vidět. Proto je v logu místo pouze pro 4000 znaků, kdy předešlé příkazy, které přišly, jsou smazány, aby udělali víc prostoru. Samotné vypisování do logu je také vcelku náročné, obzvlášť s větším množstvím příchozích příkazů. V souboru config.ini je možné log vypnout. V logu je možné zobrazit Rich Text (*Formátovaný text*), který využívá Unity pro textové prvky ve svém UI rozhraní. [10]
Rich Text podporuje následující:

- tučný – vypíše text uvnitř značek tučně
  příklad: <b>tučně<b>

- kurzív – vypíše text uvnitř znaků v kurzívě
  příklad: kurzívou
- velikost – změní velikost textu
  příklad: <size=50>velký text</size>
- barva – změní barvu textu
  příklad: <color=#00ffffff>bílá</color>
  <color=red>červená</color>




Obrázek 5.4: Log událostí.

### 5.2.4 Inventář agenta

Agentův inventář zobrazuje předměty, které agent nasbíral. Inventář není zobrazený do chvíle, kdy se klikne na nějakého agenta. Pomocí tlačítka Escape se inventář opět skryje. V horní části je jméno označeného agenta a pod ním všechny předměty v jeho inventáři. Jednotlivé sloty na předměty se přidávají zároveň s předmětem. Okno se dle potřeby na výšku zvětšuje a je možné jej přetáhnout na libovolnou pozici na obrazovce.

Jednotlivé objekty inventáře jsou uloženy u agentů a při zobrazení okna se vždy vykreslují znovu.


Obrázek 5.5: Inventář agenta s obsahem.

Třída Inventory

Obsahuje funkce pro přidání slotu a předmětu do inventáře, které využívají agenti po kliknutí na jejich vizuální reprezentaci na scéně. Má na starosti zobrazování a skrytí inventáře na scéně. Kvůli tomu také sleduje, zda bylo stisknuto tlačítko Escape, aby se mohl zobrazený inventář skrýt.

Třída WindowDrag

Zajišťuje, aby okno, na které je tato třída aplikována, bylo možné po scéně přesouvat na libovolnou pozici.

## 5.3 Struktura

V této části popíšu jednotlivé prvky aplikace, které zajišťují její běh a vizualizaci, stejně tak jako samotné agenty a předměty, které se na scéně zobrazují. Všechny tyto prvky mají vlastní třídy, které udávají jejich vlastnosti a některé spolu úzce spolupracují.

V první části 5.2.1 je popsáno fungování samotné serveru. Následuje pak reprezentace klienta v 5.2.2, kde je popsáno jeho fungování. Dalším úsekem 5.2.3 je samotný interpret příkazů pro vizualizaci nazvaný Commander, který je jednou z nejdůležitějších částí aplikace. Nakonec jsou v 5.2.4 popsány objekty, které se na scéně aplikace mohou zobrazovat. U každého prvku je seznam tříd, které využívá a jejich popis funkčnosti.

### 5.3.1 Server

Nejdůležitější částí aplikace je serverová část, která se stará o všechny příchozí spojení. Samotný server je v aplikaci reprezentovaný jako objekt, který využívá třídu *ServerCommunication*. V případě, že je nové příchozí spojení, tak server vytvoří nov objekt klienta, kterého napojí na příchozí komunikaci a řeší tak pouze komunikaci se svým připojeným klientem.

Třída ServerCommunication

Jednoduchá třída, která se stará o příchozí spojení od klientů. Jakmile nějaké takové spojení detekuje, tak vytvoří samostatný objekt, který klienta v aplikaci reprezentuje a předá mu příchozí spojení na starosti.

Třída ParserIni

Obstarává extrahování dat ze souboru config.ini. V tomto souboru se nastavuje:

- IP adresa a port pro naslouchání příchozích spojení,
- timeout, který určuje čas, po kterém se neaktivní klienti odpojí od serveru,
- viditelnost událostního logu,
- výchozí hodnoty pro rozměr prostředí světa a umístění bodu [0;0],
- uzamknutí světů proti smazání (může je smazat pouze klient, který svět vytvořil).

### 5.3.2 Reprezentace klienta

Po připojení klienta k serveru je vytvořený objekt, který následně převezme spojení a zpracovává příchozí zprávy od klienta Jason a posílá odpovědi. Je to stejný způsob, jako použití funkce *fork()* v jazyce C, kdy se duplikuje proces serveru, který pak obsluhuje připojeného klienta, zatímco hlavní proces se vrátí k obsluze příchozích spojení.

Každý připojený klient může vytvářet nové světy a do jejich prostředí vkládat agenty či různé předměty nebo překážky. Agenti, které klient vloží do světa, jsou pak dostupní pouze danému klientovi. Ostatní připojení klienti je nemohou ovládat. Co se týče ostatních předmětů, které klienti vložili do prostředí, tak ty jsou dostupné pro všechny klienty a mohou být odstraněny kterýmkoli klientem, ať už předmět do prostředí vložil on či nikoli.

Třída Connection

Stará se o veškerou komunikaci s klientem. Po přijetí zprávy od klienta se pošlou dál ke zpracování do třídy *Parser*, kde se rozdělí do jednotlivých příkazů a parametrů. Ty se následně pošlou do třídy *Commander* k dalšímu zpracování.

V této třídě je také nastavený časovač pro timeout (*vypršení spojení*) spojení. Může se tady nastavit délka čekání na vypršení spojení nebo tuto funkci přímo zrušit. V případě, že nastane timeout, tak se agenti klienta smažou a klient se z aplikace odstraní. Navíc třída obsahuje unikání Id, které se používá při ověřování vlastníka (například při mazání světa).

Třída Parser

Má za úkol rozdělit příchozí zprávy od klienta do jednotlivých parametrů a jejich příkazů.

### 5.3.3 Commander

Tento objekt zajišťuje interpretaci a vizualizaci obdržených příkazů, což z něj dělá klíčový objekt v celé aplikaci spolu se serverem. Vizualizuje všechny objekty a zajišťuje výpis událostí do logu. Je úzce spojený s objekty klientů, kteří mu předávají přijaté příkazy k vizualizaci. Commander (nebo také česky „velitel“) je v celé aplikaci pouze jeden.

Třída Commander

Jedna z nejdůležitějších tříd v aplikaci. Má v sobě implementovány všechny metody a funkce pro vizualizaci objektů na scéně. Tato třída je poslední místo, kde se zpracují přijaté příkazy od klientů, které jsou sem předány z třídy Connection. Pokud nastane nějaká chyba při vykonávání příkazu, tak se odsud odesílají příslušné návratové kódy. Úzce spolupracuje s třídou Spawner.

Třída IngameConsole

Má na starosti zápis nového textu do logu. Metoda *Write* je volána s parametrem, který obsahuje zprávu, co má být zobrazena. Před výpisem se kontroluje, jestli celkový počet znaků v logu nepřekročil 4000 znaků. Pokud ano, tak se začnou první záznamy postupně mazat s každým dalším novým záznamem. Tahle podmínka byla implementována, protože prostředí Unity mělo v nejnovější verzi chybu, kdy při větším množství textu na scéně (i když nebyl přímo vidět) docházelo k úpadkům výkonnosti.

Třída Spawner

Zajišťuje přidání nového objektu na scénu. Jak agenta, tak jiného předmětu. Má v sobě seznam prefabrikátů agentů a seznam prefabrikátů předmětů, které jsou dostupné pro vložení na scénu.

### 5.3.4 Objekty prostředí

Jsou dva druhy objektů v prostředí, které jsou pro uživatele viditelné. To jsou agenti a ostatní objekty, jako překážky nebo předměty, s kterými mohou agenti nějakým způsobem interagovat.

**Agent**

Agent je přímo ovládaný klientem a pohybuje se ve vytvořeném prostředí. Každý agent může dostávat příkazy pouze od klienta, který ho vytvořil a musí mít unikátní jméno v rámci klienta. To

znamená, že nemohou být vytvoření dva agenti se stejným jménem pod jedním klientem, ale může se stát, že budou dva agenti se stejným jménem, ale každý bude ovládaný jiným klientem.

Třída Agent

Tahle třída v sobě nese základní informace o agentovi. Má v sobě umístěnou kontrolní hodnotu, zda je inventář agenta zobrazený, aby znovu zobrazil jeho obsah v případě, že agent získá do inventáře nový předmět. Také odkazuje na seznam agentů svého klienta a na svět, v kterém se agent nachází. Také v sobě obsahuje seznam všech předmětů, které má agent v inventáři.
Dále by tato třída mohla být rozšířena o vlastnosti agenta nebo jiné definice možností agenta. Například, jestli se má zobrazovat inventář, jakou rychlostí se má agent pohybovat, atp.

Třída AgentInventory

Zajišťuje vlastní agentův inventář. Obsahuje odkaz na okno inventáře, do kterého má předměty vykreslovat. Pokud je tato třída aplikovaná na agenta, tak po kliknutí na jeho reprezentaci na scéně se zobrazí okno inventáře s jeho jménem a předměty, které s sebou nese.

Třída AgentMove

Tato třída umožňuje agentovi využívat příkazy pro plynulý pohyb po scéně. To znamená, že se nebude přesouvat z bodu A do bodu B okamžitě, ale bude tam muset určitou rychlostí dojít.
Obsahuje dva typy přesunů:

1. V rámci časového úseku, kdy za určitou dobu dojde na zvolenou pozici.
2. Dle stanovené rychlosti, kde se počítá o jakou vzdálenost se má každý frame přesunout k nově zvolené pozici.

**Předmět**

Předmětů jsou dva druhy. Prvním jsou předměty, s kterými mohou agenti nějakým způsobem interagovat. Například rudy, které mohou sbírat a nijak jim nebrání v pohybu. Druhým typem jsou překážky, které agentům brání vstoupit na pozici, na které jsou umístěny. To mohou být například zdi nebo nějaké propasti atp. Všechny předměty jsou vkládány do jednoho seznamu ve světě, ke kterému má přístup každý. Proto taky musí mít unikátní název, aby je bylo možné vyhledat.

## 5.4 Ovládání

V této kapitole se zaměřím na popis ovládání samotné aplikace a způsoby, jakými se řídí její vizualizace prostředí multi-agentních systémů.

### 5.4.1 Spuštění a chod aplikace vizualizéru

Samotné spuštění aplikace je jednoduché. Před samotným spuštěním aplikace je dobré nastavit IP adresu a port, na kterém se bude naslouchat síťová komunikace a také nastavit timeout (*časový limit připojení*) pro klienty, kteří jsou na serveru připojení. Všechny tyto parametry se nastavují v souboru config.ini, který se nachází v kořeném adresáři aplikace. Následně stačí otevřít spustitelný soubor .exe, který je hned vedle souboru .ini a je hotovo. Po načtení okna se samo spustí naslouchání na IP adrese a portu, který je uvedený v souboru config.ini. Pokud načítání ze souboru config.ini selže, tak

je jako výchozí IP adresa nastavena na localhost (127.0.0.1) a port na 1991. Timeout je defaultně vypnutý.

K ovládání prostředí stačí klávesnice a myš. Pomocí kláves WASD, případně šipek, se ovládá pohyb kamery do stran po prostředí. Kolečkem myši se pak oddaluje a přibližuje kamera. Ostatní prvky jsou ovladatelné levým tlačítkem myši – to jsou například agenti, kdy klikneme na nějakého agenta, abychom si zobrazili jeho inventář.

### 5.4.2 Připojení ze strany Jason

Pro připojení ze strany klienta je třeba znát Ip adresu, na které běží vizualizační aplikace a port, na kterém sleduje komunikaci. Připojení je realizováno pomocí protokolu TCP/IP, v jazyku Java. Java je využitá na straně klienta, takže se připojí na server pomocí vytvoření nového objektu *Socket*, který se následně pokusí připojit na vloženou IP adresu a port. Po úspěšném spojení je již možné zasílat příkazy, které má vizualizér zpracovat.

**Demo příklady**

Veškerou komunikaci a odesílání příkazů zpracovává třída UnityEnv. Odesílá a přijímá zprávy. Také čeká na odpověď ze serveru, kdy do logu vždy vypíše, jestli byl příkaz správně zpracován.

Třída UnityEnv

Zajišťuje připojení k serveru a obsahuje všechny doposud implementované příkazy, které umí vizualizační server zpracovat. Umí odesílat i přijímat zprávy. Čeká na odpověď serveru a jeho zprávu o zpracování vypisuje do logu. Jednotlivé příkazy je možné volat přímo z agenta v multi-agentním prostředí Jason. Pokud ještě nebyl vytvořený svět, tak jej vytvoří automaticky, jakmile je svět v některém příkazu vyžadován. Takovým způsobem vytvořený svět má výchozími hodnoty pro velikost a orientaci, které jsou na serveru nastavené.

**Gold-miner (AgsProject)**

Bylo zapotřebí vytvořit balíček unity, který má v sobě všechny dostupné příkazy již implementovány a stačí je tedy zavolat se správnými parametry. Po jejich spuštění klient odešle příslušný příkaz, který je pak interpretován a vizualizován. Balíček je složený ze čtyř základních tříd:

- UnityAgent,
- UnityConnector,
- UnityWorld,
- UnityWorldView.

Třída UnityAgent

Obsahuje popis agenta v Unity. Nese v sobě jeho základní údaje, jako je pozice v prostředí, počet předmětů v inventáři nebo jméno. Vždy, když je vytvořený agent na straně klienta, je vytvořená i jeho „kopie“ pro Unity.

Třída UnityConnector

Tato třída má v sobě implementovány všechny dostupné příkazy, které aplikace vizualizéru podporuje a zajišťuje síťové spojení klienta se serverem.

Třída UnityWord

Drží v sobě popis vytvořeného světa pro vizualizér. Pokud se vytvoří prostředí na straně klienta, musí se vytvořit i jeho reprezentace pro server. Nese v sobě informaci o výšce a šířce světa, ve kterém se agenti pohybují, a jméno světa.

Třída UnityWorldView

Zde je obsažena všechna logika. Zajišťuje správný postup při odesílání požadavků na server. Zajišťuje kontrolu nad duplicitou požadavků při vytváření objektů v prostředí. Tato třída úzce spolupracuje s třídou UnityConnector, protože tady se rozhodne, zda bude poslán požadavek na vizualizaci nebo ne.

### 5.4.3 Podporované příkazy

V této podkapitole se zaměřím na popis jednotlivých příkazů, které aplikace přijímá od klienta na straně Jason a jejich funkčnost. Všechny příkazy mohou být posílány malý i velkým písmem a jednotlivé parametry musí být odděleny buď mezerou, nebo středníkem.

**Name clientname**

(př.: name xproko30)
Změní zobrazené jméno (*clientname*) klienta v logu událostí. Každý klient má u svého spojení proměnnou, která má v sobě jako výchozí hodnotu uloženou IP adresu, z které se klient připojil. Pokud není tento příkaz zadán, zobrazuje se v logu pouze tato IP adresa.
Parametry:

- Clientname – textový řetězec, který definuje jméno připojeného klienta, které se bude zobrazovat v logu událostí na straně vizualizéru.

**AddAgent worldname prefab ID X Y**

(př.: addagent Earth Agent01 superman 1 2)
Vytvoří agenta s unikátním jménem (*ID*), ve zvoleném světě (*worldname*), který bude mít vzhled dle vybraného prefabrikátu (*prefab*) na zmíněných souřadnicích XY. Agent musí mít unikátní ID v rámci klienta a musí být vložen do světa, který je již vytvořený.
Parametry:

- Worldname – textový řetězec, který určuje již vytvořený svět v aplikaci vizualizéru. Pokud tento svět neexistuje, příkaz je ukončen chybou.
- Prefab – textový řetězec, určuje vzhled agenta. Může být použitý pouze takový, který je obsažený v aplikaci. Jejich seznam je zmíněný níže. Pokud neexistuje, ukončí se zpracování příkazu chybou.
- ID – textový řetězec, který určuje jméno agenta. Musí být jedinečný v rámci agentů klienta. Později se ID využívá k nalezení požadovaného agenta.
- X – celé číslo, určuje místo na ose X, kam se má agent do prostředí vložit.
- Y – celé číslo, určuje místo na ose Y, kam se má agent do prostředí vložit.

Tento příkaz je velmi důležitý, protože do multi-agentního prostředí vloží nového agenta, který bude pod kontrolou klienta, jež příkaz zaslal. Měla by se dodržovat jiná pravidla s výběrem vzhledu agenta nebo jeho jména, aby bylo možné rozlišit agenty od jednotlivých klientů. Například udržet všechny agenty od jednoho klienta vzhledově stejné.

**AddItem worldname prefab ID X Y**

(př.: additem Earth Ore01 saphire 1 3)
Vloží předmět s jedinečným jménem (*ID*), který má vzhled vybraného prefabrikátu (*prefab*), do prostředí zvoleného světa (*worldname*) na zmíněné souřadnice XY.
Pomocí prefabrikátu se volí, jaký vzhled předmětu. Tímto se také vybere, jestli se zobrazí překážka nebo jiný předmět. Každý předmět musí mít unikátní jméno ve zvoleném světě.
Parametry:

- Worldname – textový řetězec, který určuje již vytvořený svět v aplikaci vizualizéru. Pokud tento svět neexistuje, příkaz je ukončen chybou.
- Prefab – textový řetězec, určuje vzhled předmětu. Může být použitý pouze takový, který je obsažený v aplikaci. Jejich seznam je zmíněný níže. Pokud neexistuje, ukončí se zpracování příkazu chybou.
- ID – textový řetězec, který určuje jméno předmětu. Musí být jedinečný v rámci předmětů ve vybraném světě. Později je využitý k nalezení předmětu a jeho případnému odebrání.
- X – celé číslo, určuje místo na ose X, kam se má předmět do prostředí vložit.
- Y – celé číslo, určuje místo na ose Y, kam se má předmět do prostředí vložit.

Tento příkaz zajistí, aby se do prostředí vkládaly nové předměty. Mohou to být předměty, s kterými mohou agenti nějakým způsobem interagovat nebo překážky, které budou agentům bránit v průchodu. To z tohoto příkazu dělá velmi důležitý a nepostradatelný.

**Move ID X Y**

(př.: move superman 1 3)
Přesune agenta (*ID*) v nulovém čase na novou pozici XY. Příkaz může být použit pouze na agenty, které klient vlastní.
Parametry:

- ID – textový řetězec, určuje, který agent se má přesunout na novou pozici.
- X – celé číslo, určuje místo na ose X, kam se má agent přesunout.
- Y – celé číslo, určuje místo na ose Y, kam se má agent přesunout.

Využití mohou být různá. Pokud chceme upravit pozici agenta, například pokud byl agent zničen, můžeme ho přesunout zpět do startovní lokace. Může se také používat pro méně náročné přesouvání agentů po prostředí.

**Movet ID X Y time**

(př.: movet superman 1 3 1000)
Přesune agenta (*ID*) na novou pozici XY, během zvoleného času v milisekundách (*time*). Příkaz může být použit pouze na agenty, které klient vlastní.
Parametry:

- ID – textový řetězec, určuje, který agent se má přesunout na novou pozici.
- X – celé číslo, určuje místo na ose X, kam se má agent přesunout.
- Y – celé číslo, určuje místo na ose Y, kam se má agent přesunout.
- Time – celé číslo, určuje čas v milisekundách, během kterého se má agent přesunout na novou pozici.

Pokud budeme chtít, aby agentův pohyb po prostředí byl plynulý, ale víme, za jakou dobu se má na novou pozici přesunout, může být použit tento příkaz, kdy se agent během stanového času přesouvá na svou novou pozici.

**Moves ID X Y speed**

(př.: moves superman 1 3 5)
Přesune agenta (*ID*) na novou pozici XY, zvolenou rychlostí (*speed*). Příkaz může být použit pouze na agenty, které klient vlastní.
Parametry:

- ID – textový řetězec, určuje, který agent se má přesunout na novou pozici.
- X – celé číslo, určuje místo na ose X, kam se má agent přesunout.
- Y – celé číslo, určuje místo na ose Y, kam se má agent přesunout.
- Speed – celé číslo, určuje konstantní rychlost, jakou se má agent pohybovat na novou pozici.

Pokud máme stanovené konstantní rychlosti agentů a požadujeme plynulí pohyb po prostředí, tak se tento příkaz může využít snadno. Rychlost zde určuje o jak velkou vzdálenost se agent posunu každý snímek. Výpočet téhle vzdálenosti je následující:

*rychlost * čas potřebný na zpracování předchozího snímku.*

Takto se určí vzdálenost jednoho kroku (posunu) pro aktuální zpracovávaný snímek obrazovky.

**DestroyAgent ID**

(př.: destroyagent superman)
Odstraní agenta (*ID*) z prostředí. Příkaz může být použit pouze na agenty, které klient vlastní.
Parametry:

- ID – textový řetězec, určuje, který agent má být odebrán.

Příkaz se využívá převážně, když je třeba odstranit agenta ze scény. Použití může být například ve chvíli, kdy byl agent odstraněn na základě nějaké události nebo jiným agentem. Po použití tohoto příkazu je agent odstraněn trvale a pokud jej chceme zpět, musíme ho znovu vytvořit pomocí příkazu *AddAgent*.

**DestroyItem worldname ID**

(př.: destroyitem Earth saphire)
Odstraní předmět (*ID*) z prostředí zvoleného světa (*worldname*).
Parametry:

- Worldname – textový řetězec, určuje svět, z jehož prostředí má být předmět odebrán.
- ID – textový řetězec, definuje jméno předmětu, který se má odebrat. Bylo zadáno při jeho vytvoření.

Velmi užitečný příkaz, který má své uplatnění v níže uvedených ukázkách. Využije se například, když agent zničí nebo sebere nějaký předmět ze svého prostředí a ten musí být odstraněn.

**InvAdd prefab ID**

(př.: invadd Ore01 superman)
Přidá prefabrikát (*prefab*) předmětu ze seznamu všech dostupných objektů do prostředí, do inventáře zvoleného agenta (*ID*). Příkaz může být použit pouze na agenty, které klient vlastní.
Parametry:

- Prefab – určuje vzhled předmětu, který se má přidat do inventáře agenta. Mohou být použity pouze předměty, které jsou obsažené v aplikaci.
- ID – textový řetězec, je to unikátní jméno agenta, kterému má být vložen předmět do inventáře.

V případě, že mají agenti inventář, jako v níže uveden ukázce těžení. Může agentovi vložit do inventáře jakýkoli předmět, který je možné vložit i do samotného prostředí.

**InvRemove prefab ID**

(př.: invremove Ore01 superman)
Odstraní prefabrikát (*prefab*) předmětu z inventáře zvoleného agenta (*ID*). Příkaz může být použit pouze na agenty, které klient vlastní.
Parametry:

- Prefab – určuje předmět, který se má odebrat z inventáře agenta.
- ID – textový řetězec, je to unikátní jméno agenta, kterému má být odebrán předmět z inventáře.

Odstranění předmětu z inventáře tímto příkazem je velmi úzce spjato s příkazem předchozím. Jeden bez druhého by byly zbytečné. V případě, že mají agenti inventář, který mohou vyprázdnit, je tento příkaz užitečný.

**Grid x y orientation**

(př.: grid 10 10 0)
Nastaví velikost mřížky (*XY*), která definuje velikost prostředí.
Parametry:

- X – celé číslo, určuje velikost prostředí podél osy X.
- Y – celé číslo, určuje velikost prostředí podél osy Y.
- Orientation – celé číslo, určuje pozici, na které je umístěný bod [0;0].
    - 1 – bod [0;0] levý dolní roh
    - 0 – bod [0;0] střed mřížky

Velikost prostředí je ovlivněna velikostí mřížky, která jej definuje. Je možné tento příkaz využít pro změnu velikosti prostředí i při běhu simulace, ne pouze na jejím začátku. Každé prostředí může být jinak rozsáhlé, proto je tento příkaz nezbytností.

**CreateWorld worldname**

(př.: createworld earth)
Vytvoří svět se zvoleným unikátním jménem (*worldname*) a výchozí velikostí prostředí 10x10.
Parametry:

- Worldname – textový řetězec, definuje jméno světa. Musí být unikátní, pokud je již vytvořený svět s daným jménem, tak se vrací chyba.

Vytvoření samotného světa, nebo také prostředí pro agenty, zajišťuje tento příkaz. Díky němu je možné vytvořit několik různých světů, v kterých se budou nezávisle na ostatních světech pohybovat agenti a budou v něm různé předměty. Díky němu může více klientů provádět akce na různých světech nebo již vytvořených světech, do kterých jednoduše vloží nové agenty či předměty.

**CreateWorld worldname x y orientation**

(př.: createworld earth 15 15 0)
Vytvoří svět se zvoleným unikátním jménem (*worldname*) a nastaví velikost mřížky (*XY*), která definuje velikost prostředí. *Orientation* nastaví umístění bodu 0;0, viz příkaz Grid výše. Je možné vytvořit svět přímo tímto příkazem nebo zkombinovat předchozí vytvoření světa a příkaz Grid.
Parametry:

- Worldname – textový řetězec, definuje jméno světa. Musí být unikátní, pokud je již vytvořený svět s daným jménem, tak se vrací chyba.
- X – celé číslo, určuje velikost prostředí podél osy X.
- Y – celé číslo, určuje velikost prostředí podél osy Y.
- Orientation – celé číslo, určuje pozici, na které je umístěný bod [0;0].
    - 1 – bod [0;0] levý dolní roh
    - 0 – bod [0;0] střed mřížky

Stejně jako předchozí příkaz, jen kombinuje i možnosti příkazu *Grid*, který určuje velikost prostředí nastavením velikosti mřížky. Díky tomu je možné dvě věci zvládnout pomocí jednoho příkazu.

**DestroyWorld worldname**

(př.: destroyworld earth)
Odstraní svět (*worldname*) a všechny objekty spolu s agenty v něm.
Parametry:
- Worldname – textový řetězec, určuje svět, který se má odebrat, podle jeho unikátního jména.

Při vytváření světa je také nutné mít možnost jej odstranit. Příkaz odstraní z aplikace celý svět se všemi agenty a předměty v jeho prostředí. Agenti jsou odebráni i ze seznamu vlastněných agentů klientů, kteří v daném světě agenty měli.
Pokud byl smazán svět klientem, který nebyl jeho vlastníkem (daný svět nevytvořil), odešle se vlastníkovi světa zpráva o jeho smazání. Zpráva má formát (bez uvozovek): „*300: World <jméno světa> has been deleted.“*

**Close**

(př.: close)
Uzavře spojení se serverem a vymaže všechny agenty, kteří byli pod správou odpojeného klienta. Pokud není nastavený timeout na straně serveru, tak je tento příkaz nutné použít vždy, než se klient ukončí, jinak se stane, že agenti klienta zůstanou v prostředí do doby, než je svět se svým prostředím smazán.

**Stručný přehled použitelných příkazů**

| Příkaz | Parametry | Stručný popis |
|---|---|---|
| **addagent** | jméno světa [řetězec] agent [řetězec] jméno agenta [řetězec] x [číslo] y [číslo] | Přidá nového agenta na zvolenou pozici. |
| **additem** | jméno světa [řetězec] předmět [řetězec] jméno předmětu [řetězec] x [číslo] y [číslo] | Přidá nový předmět na zvolenou pozici do prostředí světa. |
| **close** | | Ukončí spojení se serverem. |
| **createworld** | jméno světa [řetězec] | Vytvoří nový svět. |
| | jméno světa [řetězec] x [číslo] y [číslo] orientace [číslo] | Vytvoří nový svět s vlastní mřížkou. |
| **destroyagent** | jméno agenta [řetězec] | Zničí agenta. |
| **destroyitem** | jméno světa [řetězec] předmět [řetězec] | Zničí předmět ve světě. |
| **destroyworld** | jméno světa [řetězec] | Odstraní svět. |
| **grid** | jméno světa [řetězec] x [číslo] y [číslo] orientace [číslo] | Nastaví velikost a orientaci mřížky. |
| **invadd** | předmět [řetězec] jméno agenta [řetězec] | Přidá předmět do inventáře. |
| **invremove** | předmět [řetězec] jméno agenta [řetězec] | Odstraní předmět z inventáře. |
| **move** | jméno agenta [řetězec] x [číslo] y [číslo] | Teleportuje agenta na pozici XY. |
| **moves** | jméno agenta [řetězec] x [číslo] y [číslo] rychlost [číslo] | Přesune agenta konstantní rychlostí na pozici XY. |
| **movet** | jméno agenta [řetězec] x [číslo] y [číslo] čas [číslo] | Přesune agenta během určitého času na pozici XY. |
| **name** | jméno [řetězec] | Nastaví jméno klienta. |

### 5.4.4 Návratové kódy

Protože se zpracovává spousta přijatých příkazů, musí se také kontrolovat, jestli byly zpracovány správně. Kvůli tomu se klientovi zasílá odpověď s informací, zda byl poslední požadavek zpracován správně. To zajišťuje několik kódových označení:

0 – vše proběhlo v pořádku,
1 – při zpracování nastala nedefinovaná chyba,
2 – neznámý příkaz,
3 – zadané ID není unikátní,
4 – neexistující prefabrikát,
5 – nesprávný počet parametrů příkazu,
6 – neexistující svět,
7 – klient není vlastník.

### 5.4.5 Seznam dostupných objektů (prefabrikátů)

Tady je zmíněný seznam všech dostupných objektů v aplikaci - jak agentů, tak předmětů. Dle níže zmíněných názvů se dají využívat v příkazech, které jsou k tomu určeny. Na obrázku 5.6 níže jsou všichni agenti a předmět uspořádáni z leva doprava dle zmíněných seznamů.

Agenti:

- Devil
- Mario
- AngryBird
- YellowAgent
- GreenAgent
- PinkAgent
- RedAgent
- BlueAgent

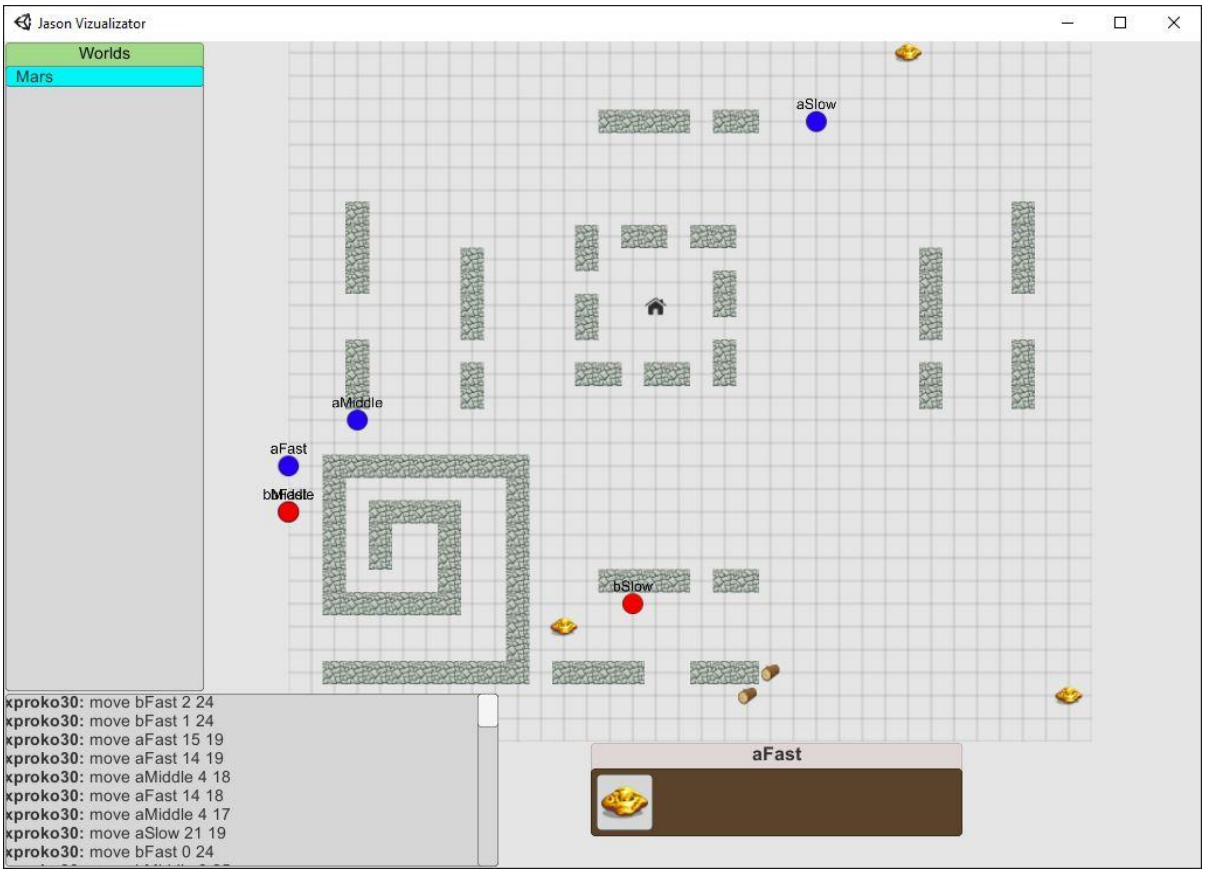
Předměty:

- StoneWall
- Wood
- Saphire
- Gold
- Iron
- Platinum
- Home

Obrázek 5.6: Agenti a objekty pro vložení do prostředí.

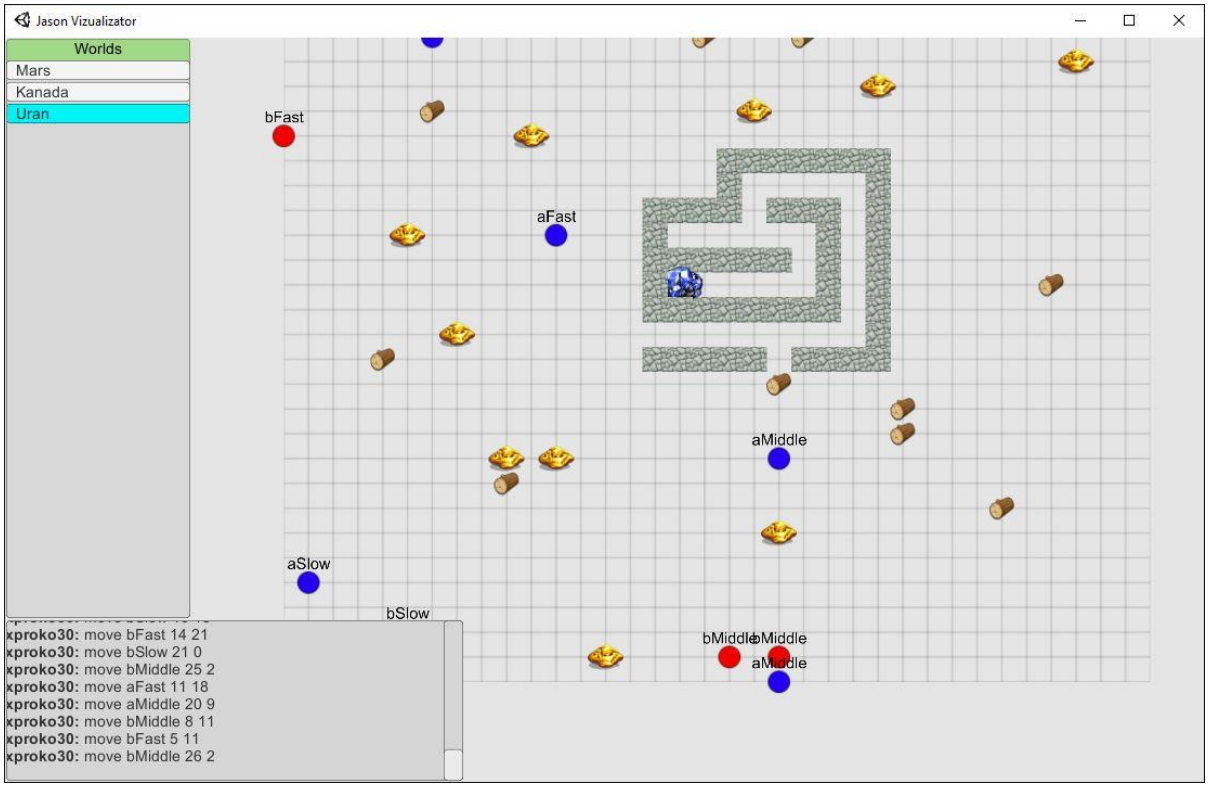
# 6 Ukázka výstupu aplikace

Hlavním výstupem aplikace je samotné vykreslení prostředí spolu s agenty, kteří jsou v něm umístěni. K tomu také vypisuje jednotlivé přijaté příkazy od každého klienta, případně IP adresy, kterou má klient přidělenou. Na následujícím obrázku 6.1 je vidět jak vypadá okno spuštěné aplikace s připojeným klientem, který posílá požadavky na vizualizaci do světa Mars. Obrázek je pořízený z ukázkového projektu, kde mají agenti dvou různých týmů sesbírat jednotlivé suroviny zlata a dřeva a odnést je do depa, které je uprostřed mapy, po které se pohybují.




Obrázek 6.1: Ukázka vizualizace agentů od připojeného klienta.

Jednotlivý agenti mohou mít různé barvy, případně i nějakou texturu. V aplikaci je několik různých textur pro agenty, které mohou být použity a několik různých textur pro suroviny, které se mohou po prostředí vyskytovat. Seznamy všech agentů a objektů, které se mohou do prostředí vložit, jsou v předchozí kapitole a na obrázku 5.6 výše.




Obrázek 6.2: Ukázka okna aplikace s více světy.

## 6.1 Sada ukázkových příkladů

V rámci testování aplikace bylo vytvořeno několik demonstrativních příkladů, kde každý z nich představoval použití několika příkazů implementovaných na straně serveru.

- Demo 01 – Agenti a předměty
  - Ukázka vložení všech typů agentů a předmětů do prostředí.
    Použité příkazy: addagent, additem, movet
- Demo 02 – Stavitel
  - Ukázka pohybu agenta s postupný vkládáním zdi.
    Použité příkazy: addagent, auditem, destroyagent, destroyitem
- Demo 03 – Multi-světy
  - Ukázka více světů současně s předměty v nich a jejich následné mazání.
    Použité příkazy: createworld, additem, destroyworld
- Demo 04 – Inventář
  - Vkládání věcí do inventáře agenta a jejich následné odebrání.
    Použité příkazy: addagent, invadd, invremove
- Demo 05 – Pohyb agenta
  - Různé druhy přesunů agenta. Teleport, přesun v čase a přesun s rychlostí.
    Použité příkazy: addagent, move, movet, moves
- Demo 06 – Mřížka
  - Ukázka předefinování mřížky v prostředí.
    Použité příkazy: grid

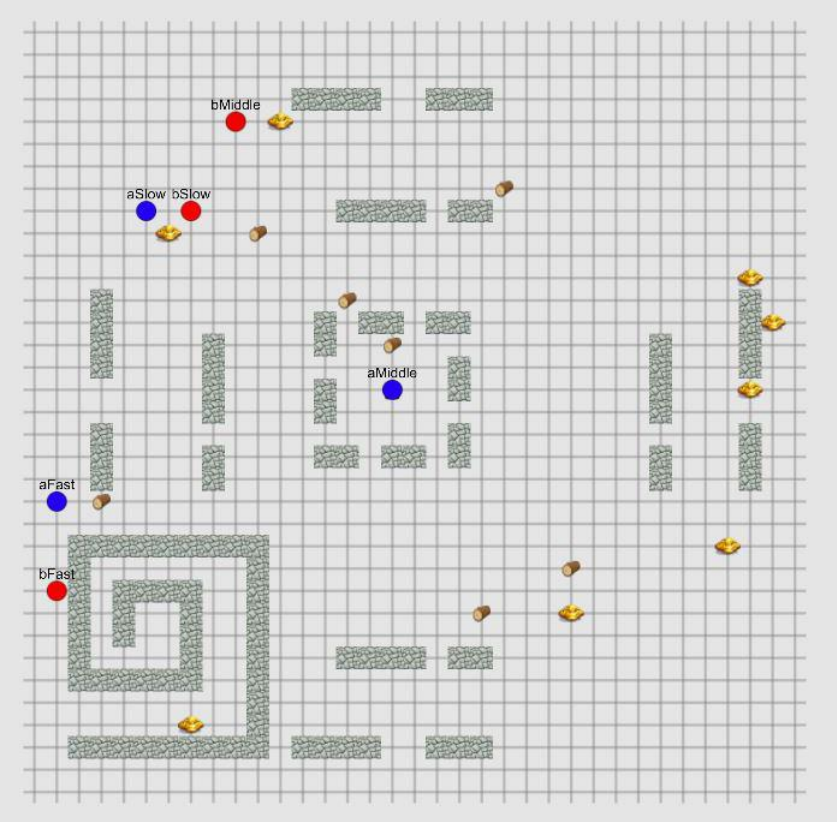
## 6.2 Ukázka z projektu Gold-miners

Projekt byl vyhotoven v rámci výuky předmětu Agentní a multiagentní systémy na fakultě informatiky. Autory projektu jsou Jan Horáček a František Zbořil. Samotné řešení projektu, které zahrnovalo chování agentů, zajistil vedoucí této práce.

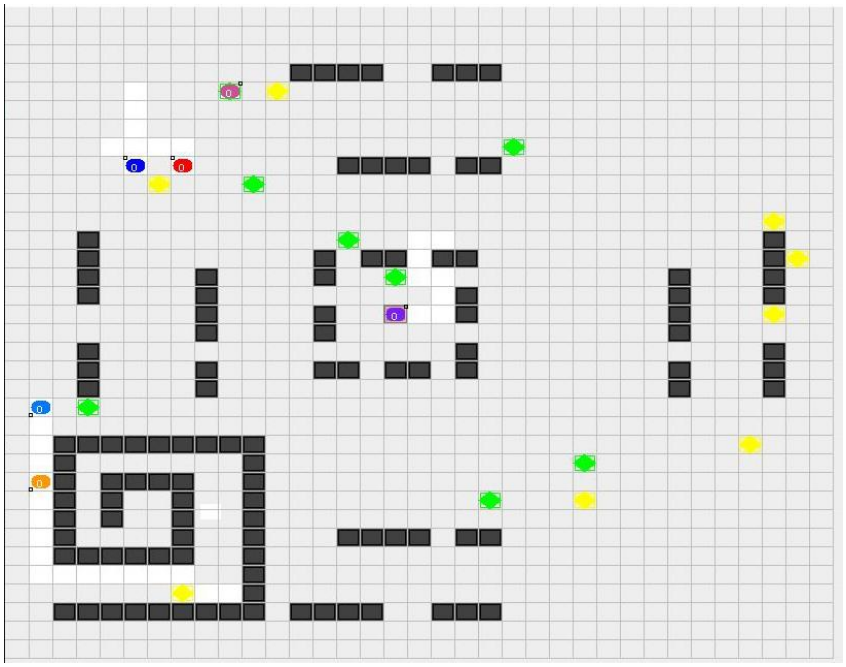
Projekt je již mnohem komplexnější a bylo tak možné otestovat aplikaci v mnohem náročnějším prostředí. V projektu je k dispozici šest různých prostředí, v kterých se agenti mohou pohybovat. Některé mají v prostředí překážky, jiné zase ne. Bylo tak možné otestovat, jak se bude aplikace chovat při větším počtu objektů na scéně. Navíc sám podporuje možnost vizualizace prostředí pomocí jazyku Java. Díky tomu jsem byl schopný porovnat výstup z vizualizéru s reálným výstupem z ukázkového projektu.

Pro otestování jsem se rozhodl pro prostředí číslo 2, protože obsahuje víc objektů, které pro agenty činí překážky. Znázornění předmětů na straně agenta je taková, že dřevo je zeleně, zlato žlutě, překážka černě a agenti mají odstíny modré a červené dle toho, jak rychlý jsou. Na straně serveru jsou pojmenovaní a samotné zlato, dřevo a překážka mají vlastní texturu.

Na obrázcích 6.3 a 6.4 níže je vidět začátek simulace a její vizualizace na straně serveru (Unity) a klienta (Jason). Navíc na straně klienta je vidět, že agenti zanechávají „stopu“ v místě, kde se nacházeli. Patrný rozdíl je vidět hned na začátku v tom, že samotná mřížka na straně vizualizéru zobrazuje body v průsečících přímek, namísto v čtvercových polích jako je tomu u klienta Jason. Když se zaměříme na rozmístění objektů po scéně, tak jsou všechny na správných pozicích.

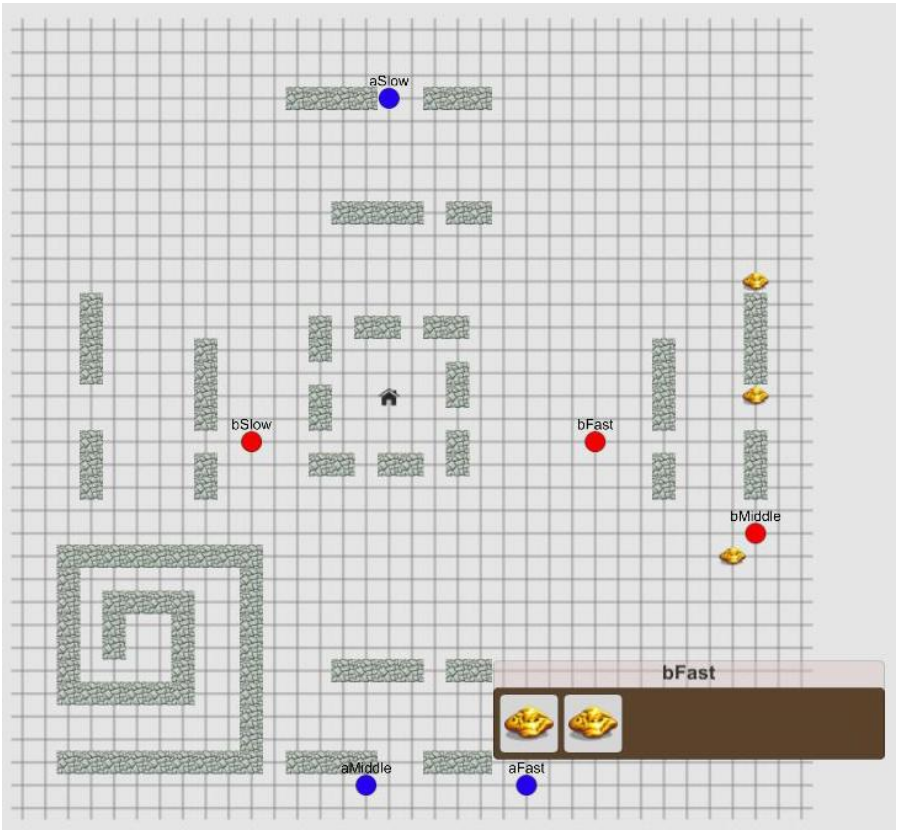



Obrázek 6.3: Začátek simulace na straně serveru.

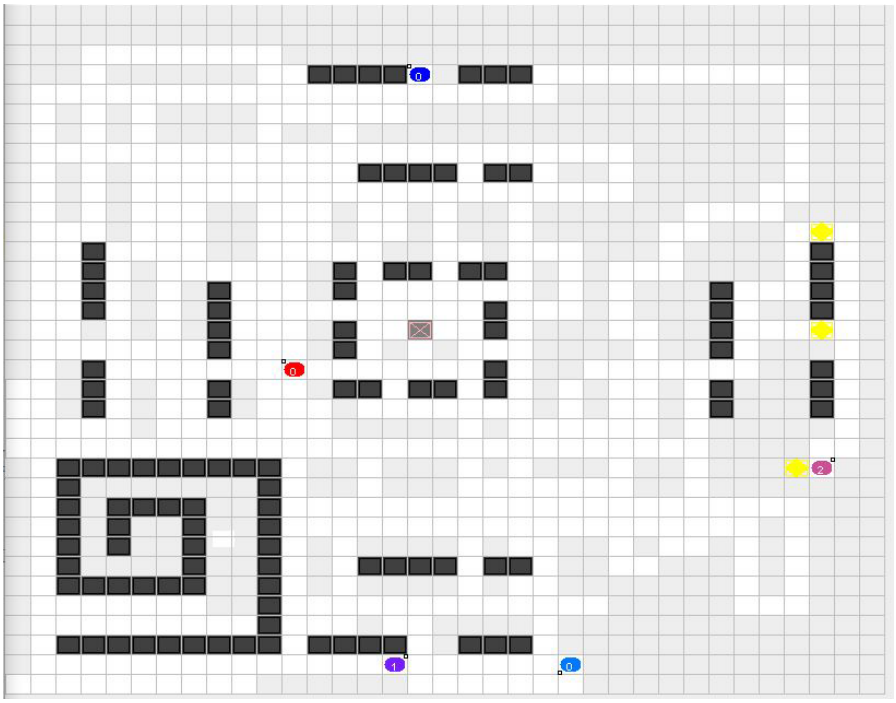


Obrázek 6.4: Začátek simulace na straně klienta.

Na následujících obrázcích 6.5 a 6.6 jsou zachyceny přibližně v půlce simulace, kdy agenti již sesbírali většinu zdrojů z prostředí. Ne obrázku 6.5 je vidět inventář agenta bFast, v kterém jsou obsaženy dvě zlaté rudy. Na obrázku 6.6 tento agent ale není vidět, protože při zachytávání obrazovky nebyl v danou chvíli vykreslený.

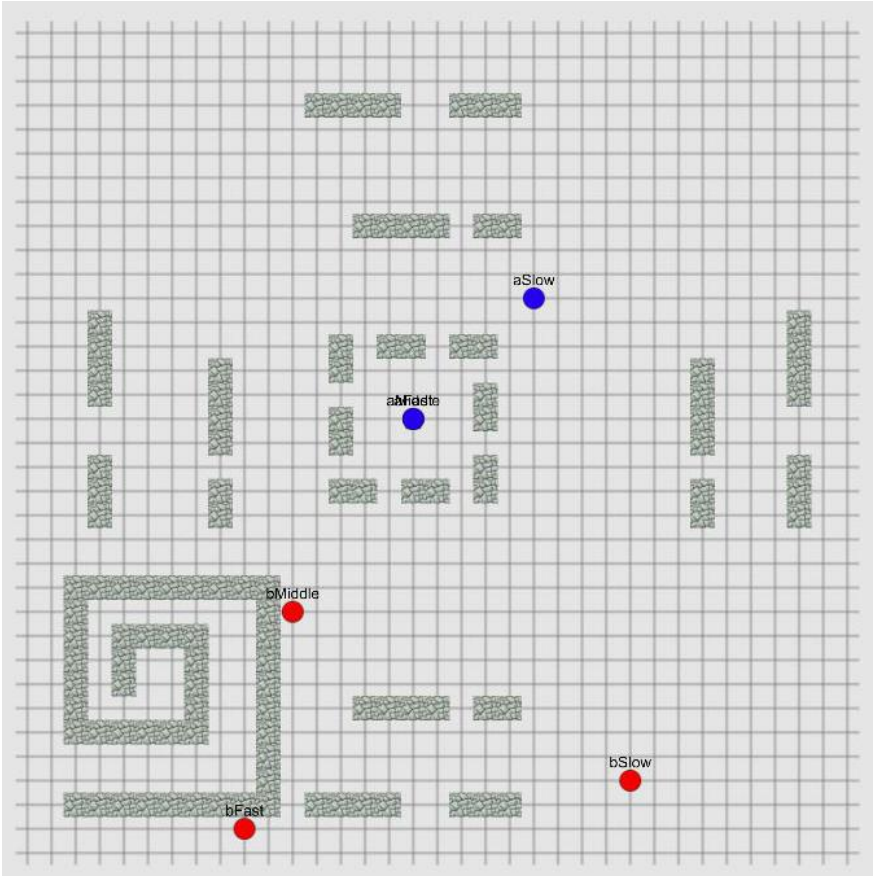



Obrázek 6.5: Průběh simulace na straně serveru.

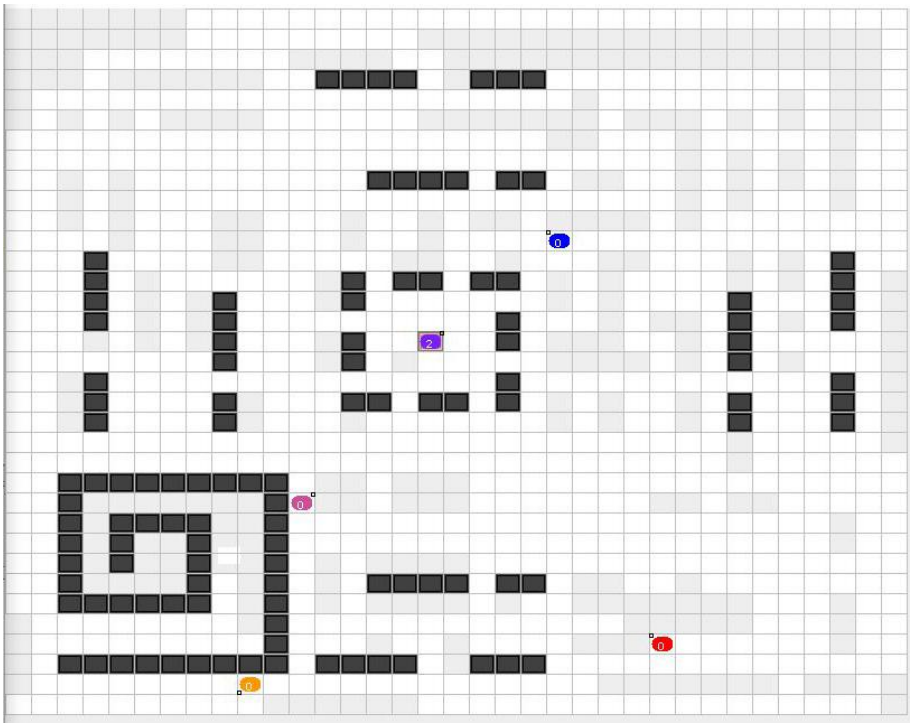


Obrázek 6.6: Průběh simulace na straně klienta.

Poslední dva obrázky 6.7 a 6.8 zachycují konec simulace, kdy se dva modří agenti vrátili zpět do depa, kam odnesly poslední získané suroviny a tím ukončili simulaci. Všechny suroviny jsou sesbírané na obou stranách a agenti stojí na stejných pozicích. To znamená, že vizualizace byla dokončena správně.




Obrázek 6.7: Konec simulace na straně serveru.



Obrázek 6.8: Konec simulace na straně klienta.

# 7 Závěr

V rámci této práce byla vytvořena síťová aplikace pro vizualizaci multi-agentních prostředí se základní sadou příkazů pro její využití.

Výsledná aplikace je schopná vizualizovat agenty několika připojených klientů současně ve více prostředích v danou chvíli. Díky tomu je možné ji využívat pro vizualizaci projektů psaných v multi-agentních prostředích a upadá tak nutnost vizualizace přímo na straně klienta, kde by bylo nutné vykreslovat všechny prováděné události. Protože je aplikace založená na síťové komunikaci, tak je možné, aby byla umístěna na jiném stroji, který by byl určený k vizualizaci, a tím se také snížila spotřeba výpočetních zdrojů na straně klienta. Při vývoji jsem se snažil aplikaci navrhnout tak, aby byla snadno ovladatelná a proto stačí nastavit informace o spojení v jednom souboru a zapnout aplikaci, která je během okamžiku připravená přijímat nové spojení od klientů a zpracovat jejich požadavky.

Pro potřeby komunikace byla vytvořena sada základních příkazů, které dokážou vizualizovat většinu projektů, které by měly být vizualizovány.

## 7.1 Možnosti rozšíření

V rámci téhle práce je stále mnoho možností pro rozšíření aplikace a pro její samotné zdokonalování. V této podkapitole je proto zmíněno několik nápadů pro její vylepšení do budoucna.

**Administrace na straně serveru**

Administrativní pravomoci na straně serveru v tuhle chvíli chybí a nebylo by na škodu, kdyby bylo možné mazat jednotlivé světy, pokud by jich bylo již příliš nepoužívaných nebo vytvářet nová na straně serveru spolu možností vložení jednotlivých předmětů do prostředí.

**Víc informací o klientech a agentech**

Zatím není moc možností, jak vidět, kolik klientů je připojeno, jaké mají k dispozici agenty nebo které světy vytvořili. Tento seznam by byl také dobrý pro lepší orientaci po scéně, kdy by stačilo kliknout na příslušného agenta a objevily by se jeho statistiky. Například inventář, vlastník a jiné statistiky.

**Nastavení vlastností agentů**

Ve třídě Agent by bylo možné vytvořit víc vlastností, jako třeba dohled agenta, jeho rychlost či velikost inventáře aj. Poté by se přes nový příkaz dalo nastavit, jaké vlastnosti bude agent mít a jaké informace by pak bylo relevantní zobrazovat. Díky tomu by pak bylo možné implementovat následující rozšíření.

**Posílání dat klientovi o svém prostředí**

Agenti by mohli zkoumat své prostředí v určitém osobním dosahu, který by bylo možné realizovat pomocí fog of war (*válečná mlha – prostředí je celé za mlhou, kdy agenti mají určitý dohled, díky kterému je jejich okolí jasně vidět*). Agent by pak posílal informace o předmětech v jeho blízkém okolí klientovi, ten by data zpracoval a poslal další příkazy. Takto by bylo možné vytvořit vlastní

prostředí, do kterého by se vložili agenti od více klientů a ti spolu navzájem komunikovali, soupeřili nebo spolupracovali.

**Možnost 3D vizualizace**

Aplikaci by bylo možné předělat na verzi pro 3D. To by následně umožňovalo vytvořit prostředí s různorodým terénem.

**Vlastní agenti/objekty**

Tohoto by se dalo dosáhnout pomocí XML souborů, které by nemělo být těžké definovat a následně je v aplikaci zpracovat a na jejich základě vytvořit požadované předměty, agenty, překážky nebo jiné objekty, které by se daly vizualizovat v požadovaném prostředí. V tuhle chvíli je nutné vytvářet nové objekty přímo v editoru Unity a vytvářet vždy nový spustitelný soubor.

# 8 Reference


[1] Scripting. In: *Unity: Documentation* [online]. Copenhagen: Unity Technologies, 2015 [cit. 2016-04-20]. Dostupné z: http://docs.unity3d.com/Manual/ScriptingSection.html

[2] Unity (game engine). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2015-10-27]. Dostupné z: https://en.wikipedia.org/wiki/Unity_(game_engine)

[3] Unity: Engine Features. *Unity 3D* [online]. 2015 [cit. 2015-10-28]. Dostupné z: https://unity3d.com/unity/engine-features

[4] *Jason: a Java-based interpreter for an extended version of AgentSpeak* [online]. 2015 [cit. 2015-11-15]. Dostupné z: http://jason.sourceforge.net/wp/

[5] BORDINI, Rafael H, Jomi Fred HÜBNER a Michael J WOOLDRIDGE. Programming multi-agent systems in AgentSpeak using Jason. 1. Hoboken, NJ: J. Wiley, 2007, xv, 273 p. ISBN 978-0-470-02900-8.

[6] Belief–desire–intention software model. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-04-04]. Dostupné z: https://en.wikipedia.org/wiki/Belief%E2%80%93desire%E2%80%93intention_software_model

[7] Softwarový model Belief-Desire-Intention. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-04-04]. Dostupné z: https://cs.wikipedia.org/wiki/Softwarov%C3%BD_model_Belief-Desire-Intention

[8] ZBOŘIL, František. *Plánování a komunikace v multiagentních systémech* [online]. Brno, 2004 [cit. 2016-04-05]. Dostupné z: http://www.fit.vutbr.cz/~zborilf/PhD/thesis.pdf. Disertační práce. Vysoké učení technické v Brně. Vedoucí práce Doc. Dr. Ing. Petr Hanáček.

[9] MURHAMMER, Martin W a Eamon MURPHY. *TCP/IP tutorial and technical overview* [online]. 6th ed. Upper Saddle River, N.J.: Prentice Hall PTR, c1998 [cit. 2016-04-06]. ISBN 01-302-0130-8. Dostupné z: http://www.redbooks.ibm.com/pubs/pdfs/redbooks/gg243376.pdf

[10] Unity Manual: Rich Text. In: *Unity3D* [online]. Unity Technologies [cit. 2016-04-15]. Dostupné z: http://docs.unity3d.com/Manual/StyledText.html

[11] NOEHR, Jesper. *Atlassia* [online]. Atlassia, 2010 [cit. 2016-04-19]. Dostupné z: https://www.atlassian.com/

[12] NOEHR, Jesper. *Atlassia* [online]. Sydney: Atlassia, 2002 [cit. 2016-04-19]. Dostupné z: https://www.atlassian.com/

[13] Bitbucket. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-04-19]. Dostupné z: https://cs.wikipedia.org/wiki/Bitbucket

[14] Eclipse (software). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-04-20]. Dostupné z: https://en.wikipedia.org/wiki/Eclipse_(software)

[15] Knowledge Query and Manipulation Language. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-04-21]. Dostupné z: https://en.wikipedia.org/wiki/DARPA

[16] Sprite Packer. In: *Unity: Documentation* [online]. Copenhagen: Unity Technologies, 2015 [cit. 2016-04-20]. Dostupné z: http://docs.unity3d.com/Manual/SpritePacker.html

[17] Lighting Overview. In: *Unity: Documentation*[online]. Copenhagen: Unity Technologies, 2015 [cit. 2016-04-20]. Dostupné z: http://docs.unity3d.com/Manual/Lighting.html

[18] Standard Shader. In: *Unity: Documentation*[online]. Copenhagen: Unity Technologies, 2015 [cit. 2016-04-20]. Dostupné z: http://docs.unity3d.com/Manual/shader-StandardShader.html

[19] Rigidbodies. In: *Unity: Documentation* [online]. Copenhagen: Unity Technologies, 2015 [cit. 2016-04-20]. Dostupné z: http://docs.unity3d.com/Manual/RigidbodiesOverview.html

[20] Colliders. In: *Unity: Documentation* [online]. Copenhagen: Unity Technologies, 2015 [cit. 2016-04-20]. Dostupné z: http://docs.unity3d.com/Manual/CollidersOverview.html

[21] Joints. In: *Unity: Documentation* [online]. Copenhagen: Unity Technologies, 2015 [cit. 2016-04-20]. Dostupné z: http://docs.unity3d.com/Manual/Joints.html

[22] Animation System Overview. In: *Unity: Documentation*[online]. Copenhagen: Unity Technologies, 2015 [cit. 2016-04-20]. Dostupné z: http://docs.unity3d.com/Manual/AnimationOverview.html

[23] Animator Controllers. In: *Unity: Documentation* [online]. Copenhagen: Unity Technologies, 2015 [cit. 2016-04-20]. Dostupné z: http://docs.unity3d.com/Manual/AnimatorControllers.html

[24] C Sharp (programming language). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-04-20]. Dostupné z: https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[25] Java (programming language). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-04-20]. Dostupné z: https://en.wikipedia.org/wiki/Java_(programming_language)

[26] Knowledge Query and Manipulation Language. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2016 [cit. 2016-04-21]. Dostupné z: https://en.wikipedia.org/wiki/Knowledge_Query_and_Manipulation_Language

[27] Mecanim - transition from run to jump. In: *Unity* [online]. [cit. 2016-05-02]. Dostupné z: http://answers.unity3d.com/questions/512831/mecanim-transition-from-run-to-jump.html

# Příloha A

# Obsah CD

Soubory na CD jsou uspořádány ve stromové struktuře a každý podstrom obsahuje README soubor s popisem.

- Jason vizualizátor – obsahuje soubory k Unity serveru pro vizualizaci multi-agentních prostředí.
    - AplikaceWin – zde je spustitelný soubor vizualizační aplikace pro Windows.
    - Projekt – obsahuje všechny soubory potřebné pro otevření projektu v prostředí Unity.
- Nástroje – nástroje potřebné pro spuštění aplikace a otevření projektu spolu s licencemi.
- Text – obsahuje text práce.
- Ukázky – ukázky pro demonstraci vizualizace.
    - AgsProject – projekt Gold-miners.
    - Demonstrace – sada ukázkových příkladů.

## Příloha B

# Instalace a nastavení potřebných nástrojů

V této příloze je uvedený postup pro nainstalování potřebných nástrojů pro spuštění jednotlivých částí práce, které se nachází na přiloženém CD.

## B.1 Unity

Aplikace vizualizéru je určená pro platformu Windows. Je možné ji spustit na čisté instalaci systému, takže nepotřebuje nic navíc ke svému chodu. Pro otevření samotného projektu pro další vývoj, je nutná instalace prostředí Unity. Pro správnou instalaci je nutné mít operační systém Windows 7 nebo novější.

Instalace Unity:

1) Spustit instalátor Unity, který je umístěný na přiloženém CD ve složce Nástroje. Zde spustit soubor UnitySetup32-5.3.4f1.exe a provést instalaci.
2) V průvodci instalace bude na výběr z dvou nástrojů pro nainstalování – Unity a MonoDevelop. MonoDevelop je to prostředí pro psaní skriptů. Pokud nepoužíváte vlastní software, je dobré jej nainstalovat.
3) Pokračovat dál, dokud se nezahájí instalace.
4) Po dokončení instalace můžete zapnout Unity. Před samotným spuštěním po Vás bude chtít přihlašovací údaje. Pokud nemáte založený účet, tak se zaregistrujte. V textu nad rámečky pro e-mail a heslo je červeně odkaz na registrační formulář pod slovy „create one“. Na něj klikněte, vyplňte potřebné údaje a potvrďte registraci.
5) Po dokončení registrace vyplňte do Unity e-mail a heslo a přihlaste se. Na další stránce přepněte na Unity Personal Edition a dejte „Next“.
6) Vyskočí potvrzení, že používáte Unity pouze pro osobní použití. Dejte poslední možnost „I don’t use Unity in a Professional capacity“ a klikněte na „Next“. Po zaktivování licence dejte na poslední stránce „Start Using Unity“.
7) Zvolně nahoře vpravo „Open“ a najeďte do složky s projektem. *Jason vizualizátor\Projekt* a klikněte na „Vybrat složku“. Vše je hotov, projekt se otevře.

## B.2 Jason

Pro spuštění demonstrativních přívkladů je nutné nainstalovat multi-agentní prostředí Jason. Jsou dvě možnosti, jak spustit multi-agentní prostředí Jason. Spouštění přímo z jEdit, který je obsažený v Jason nebo přes prostředí Eclipse s přidaným pluginem pro prostředí Jason.

Jason:

1) Nejdřív bude nutné nainstalovat Java SE Development Kit, pokud se na Vašem počítači nenachází. Ve složce s nástroji otevřete složku Java RE a spusťte soubor jdk-8u91-windows-i586.exe, proklikejte se přes instalátor tlačítkem „Next“ a počkejte, až se instalace dokončí.

2) Ve složce s nástroji otevřete složku Jason. Zde najdete archiv zip. Ten stačí rozbalit na Vámi vyhovující místo. Například na disk C.
3) Po rozbalení najeďte do složky, kterou jste rozbalili – Jason-1.4.2 - a zde spusťte soubor Jason.exe. Prostředí by se mělo otevřít.
4) Ještě než začnete cokoli dělat, otevřete v horní liště Plugins -> Plugins Options -> Jason a tam nastavte správnou adresu v Java Home. Je to adresář, kam jste nainstaloval Java SDK. Standardně to je *C:\Program Files\Java\jdk1.8.0_9* a změny uložte tlačítkem „Apply“ nebo „Ok“.
5) Nyní můžete otevřít ukázkové příklady. Stačí nahoře v liště zmáčknout na File -> Open a ve složce ukázky si zvolit projekt. Vždy se otevírá soubor s příponou „.mas2j“.
6) Po načtení projektu jej spustíte dole vpravo zelenou šipkou „Run MAS“

Eclipse:

1) Stejně jako v předchozím postupu, i tady bude nutné nainstalovat Java SE Development Kit, pokud se nenachází ve vašem systému. Ve složce s nástroji otevřete složku Java RE a spusťte soubor jdk-8u91-windows-i586.exe, proklikejte se přes instalátor tlačítkem „Next“ a počkejte, až se instalace dokončí.
2) Ve složce Nástroje najeďte do adresáře Eclipse. Zde spusťte soubor eclipse-inst-win32.exe a zahajte instalaci. Zde vyberte hned první položku „Eclipse IDE for Java Developers“, nastavte místo, kam chcete aplikaci nainstalovat, dejte „Install“ a potvrďte licenční podmínky.
3) Po dokončení instalace spusťte Eclipse tlačítkem „Launch“. Bude po Vás požadována nastavení workspace, to je místo, kam se budou ukládat soubory k projektům. Nastavte vlastní nebo nechte výchozí a dejte „Ok“.
4) V horní liště klikněte na Help -> Install New Software. V otevřeném okně dejte „Add“. Do první kolonky napište *jasonide* a do druhé vložte odkaz *http://jason.sourceforge.net/eclipseplugin/juno/* a dejte „Ok“.
5) V okně by se Vám měl objevit jasonide a po rozkliknutí by pod ním mělo být Jasonide_feature. Oboje označte a dejte „Next“. V dalším okně znovu „Next“ a v následujícím potvrdit licenční podmínky a kliknout na „Finish“. Během instalace pluginu vyskočí varování o používání neověřeného obsahu. Stačí potvrdit „Ok“ a instalace bude pokračovat. Plugin pro Jason se stáhne a nainstaluje. Po jeho nainstalování bude Eclipse vyžadovat restartovat, dejte „Yes“.
6) Po načtení dejte File -> Import a vyberte Jason Projekt a dejte „Next“.
7) Zde dejte „Browse“ a najeďte na složku s ukázkou a dejte „Ok“, projekt by měl být v okně vybraný. Pak dejte „Finish“. Během toho pravděpodobně vyskočí hláška, jestli chcete použít prostředí pro Jason, to potvrďte tlačítkem „Ok“.
8) Pokud projekt nevidíte po levé straně, tak dejte Window -> Show View -> Other -> General vyberte Project Explorer a dejte „Ok“.
9) Teď už stačí jen dvakrát poklepat na soubor s příponou .mas2j nebo jej přetáhnout do prostředního okna. Projekt se spustí zeleným tlačítkem v horní liště „Run Jason Application“.
    a) Je možné, že po importu bude Eclipse vypisovat dvě chyby na projekt. Projekt půjde spustit po kliknutí pravým tlačítkem na soubor .mas2j v levém panelu a kliknutí na „Run Jason Application“.

Oprava chyb importu:

1) Ve složce s nástroji otevřete složku Jason. Zde najdete archiv zip. Ten stačí rozbalit na Vámi vyhovující místo. Například na disk C.
2) Pravým tlačítkem klikne na projekt v levém panelu, tam se najede do Build Path -> Configurate Build Path a v záložce Libraries se přidá externí jar „Add External JARs“.
3) Tam přidejte knihovnu z rozbalené složky s Jason. Pokud jste rozbalil archiv s Jason na C, tak bude cesta taková *C:\Jason-1.4.2\lib*. Zde najděte soubor jason.jar a dejte „Otevřít“ a „Ok“.
4) Následně v souboru UnityEnv.java upravte první řádek na *package java.env;* a uložte.
5) Projekt se ovšem stále bude muset pouštět stejně, jak bylo popsáno v dodatku a) návodu pro Eclipse.

Vyhnutí se chybám importu:

1) File -> New -> Other a vybrat Jason Project.
2) „Project name“ nastavit na stejné, jako je jméno souboru s příponou .mas2j v požadované ukázce bez přípony (např. demo05). Kliknout na „Finish“.
3) Pravým tlačítkem kliknout na nově vytvořený projekt v levé liště a dát Import -> General -> File System.
4) Nahoře vpravo zmáčknout na „Browse“, najet na složku požadované ukázky a dát „Ok“.
5) Napravo se objeví obsah složky. Označit tam pouze soubor s příponou .mas2j.
6) Rozkliknout nalevo projekt a zaškrtnout políčko u složky src.
7) V options nastavit „Overwrite existing resources without warning“ a zmáčknout na „Finish“.
8) Klasicky už pouze poklikat na soubor s příponou .mas2j v levém postraním panelu s projekty a nahoře pomocí tlačítka „Run Jason Application“ spustit ukázku.