ModelMMORPG - D2.1. - Test-bed

Schatten, Markus; Okreša Đurić, Bogdan; Maliković, Marko

Other document types / Ostale vrste dokumenata

Large-Scale Multi-Agent Modeling of Massively Multi-Player On-line Role-Playing Games

Deliverable D2.1. - Test-bed

This project was funded by the Croatian Science Foundation

Principal investigator:

Markus Schatten

2015.

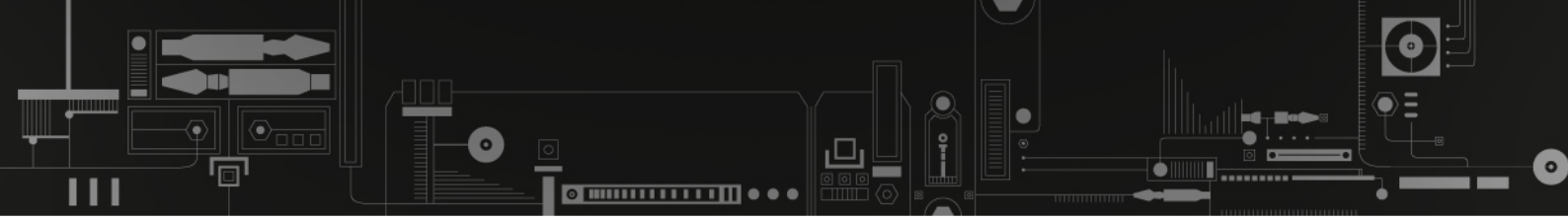
PUBLISHED BY ARTIFICIAL INTELLIGENCE LABORATORY,
FACULTY OF ORGANIZATION AND INFORMATICS, UNIVERSITY OF ZAGREB

HTTP://AI.FOI.HR/MODELMMORPG

Technical Report No. AIL2015002 – First release, May 2015

Document compiled by: Markus Schatten, Bogdan Okreša Đurić, Marko Maliković with inputs from other project team members

This work has been supported in full by the Croatian Science Foundation under the project number 8537.

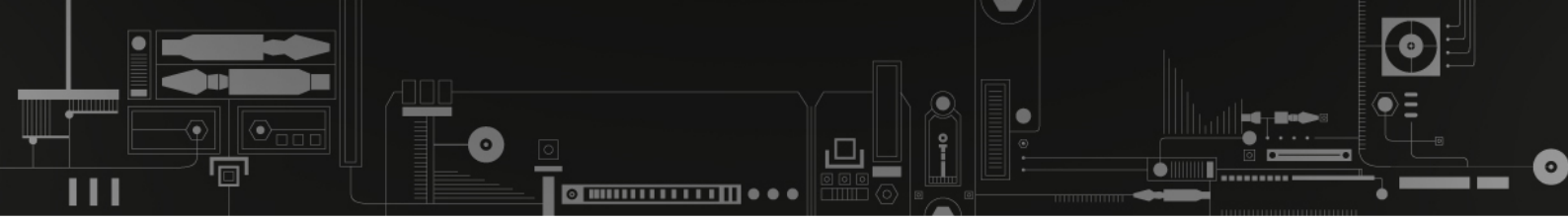


hrzz
Hrvatska zaklada za znanost

foi
SVEUČILIŠTE U ZAGREBU
FAKULTET
ORGANIZACIJE I
INFORMATIKE
VARAŽDIN

lab

Contents



hrzz
Hrvatska zaklada za znanost

foi
SVEUČILIŠTE U ZAGREBU
FAKULTET
ORGANIZACIJE I
INFORMATIKE
VARAŽDIN

lab

1. Project Description

1.1 Abstract

Massively multi-player on-line role playing games (MMORPGs) give us the opportunity to study two important aspects of computing: (1) large-scale virtual social interaction of people (players) and (2) the design, development and coordination of large-scale distributed artificial inteligence (AI). A common denominator for both aspects are the methods used to study them: social interaction can be descibed and simulated using agent-based models (ABM social science perspective) whilst distributed AI is commonly modelled in terms of multi-agent systems (MAS computer science perspective).

The important question to ask in both perspectives is how do agents organize in order to perform their tasks and reach their objectives? Project ModelMMORPG (Large-Scale Multi-Agent Modelling of Massively On-Line Role-Playing Games) will employ a combined empirical and theoretical approach towards finding the answer to this question.

From the empirical side, we shall study the human behaviour on a number of venues across various gaming servers in order to find most suitable structures, cultures, processes, strategies and dynamics employed by most successful player communities. From the theoretical side, we shall test a multitude of organizational architectures from organization theory in various MMORPG settings, and compare them with methods found in empirical research.

Our research is therefore aimed towards enriching the organizational design methods for the development of MMORPG to foster the development of self-organizing and adaptable networks of large-scale multi-agent systems.

With this in mind, our main goals are:

1. To identify and formalize adequate organizational design methods for developing LSMAS in MMORPGs.
2. To couple them with real-life and future scenarios from industry.
3. To provide open and accessible tools, which will allow for design, development, implementation, control, simulation and maintenance of LSMAS in MMORPG

1.2 Introduction

Role-playing video or computer games (commonly referred to as only role-playing games or RPGs) are a game genre in which the player controls the actions of some protagonist (or potentially several party members) in a world which is well defined [5]. A massively multi-player on-line game (MMOG) is a (computer) game that supports a great number of players playing on-line simultaneously causing or even fostering interaction among them [3]. Massively multi-player on-line role playing games are thus a mixture of these two genres allowing players to control the action of their protagonist (avatar) by interacting with a potentially large user-base on-line [4].

The global market for MMO games is growing rapidly with 2011 $\approx$ 8.5 billion €, 2012 $\approx$ 10.2 Bn€, 2013 $\approx$ 11.7 Bn€ and 2014 $\approx$ 15.0 Bn€ [1, 2]. While the economic importance of MMORPGs is obvious, another aspect is of equal importance: it allows us to investigate two aspects of large-scale computing - (1) social interaction of (large numbers of) players through a computing platform as well as (2) the design and implementation large-scale distributed artificial intelligence (in form of non-player characters – NPCs, mobs – various monsters to be fought, as well as AI players – bots). Both aspects can and should be studied using agent-based methods, the former by ABM (a social science perspective) and the latter by MAS (a computer science perspective), whilst the important question to ask in both perspectives is: how do agents organize in order to perform their tasks and reach their objectives?

In the following deliverable we will present an overview of the ModelMMORPG (Large-Scale Multi-Agent Modeling of Massively On-Line Role-Playing Games) project that will employ a combined empirical and theoretical approach towards finding the answer to this question. From the empirical side, we shall study the human behavior on a number of venues across various gaming servers in order to find most suitable organizational structures, cultures, processes, strategies and dynamics employed by most successful player communities. From the theoretical side, we shall test a multitude of organizational architectures from organization theory in various MMORPG settings, and compare them with methods found in empirical research. The research is therefore aimed towards enriching the organizational design methods for the development of MMORPGs and to understand the undelying principles of self-organizing and adaptable networks of large-scale multi-agent systems.

MMORPGs have a number of different subgenres, but a usual setting is that a protagonist is placed into a world in which he interacts with various NPCs and mobs which give out tasks (quests) that it has to solve to be able to buy better equipment, learn new skills like magic and similar, or proceed to higher levels. In ModelMMORPG we have chosen The Mana World (TMW)[1] MMORPG to conduct our research. The reasons for selection were: (a) it is open source (GPL licensed) allowing us to modify code and add additional functionality, (b) it has a supportive community, (c) it supports a number of interaction techniques which can be studied (e.g. trade among players, IRC based chat, organizing teams called parties, social network functions e.g. friends, enemies, parties etc.), (d) it is a (more or less) finished game featuring lots of quests that can be analyzed.

In order to answer our outlined questions, we firstly designed a special quest in which players ought to organize their activities in order to solve it. The quest is designed during a 3-day brainstorming session. Later on, during a data collection phase we will allow players to play the game in order to collect their behavioral data during a period of one month. Additionally to real players, we will add AI players that shall play alongside humans in order to provide a baseline for a future (organized) version of AI players. After data collection, the data will be analyzed using social network analysis (SNA) and natural language processing (NLP) techniques in order to find patterns of organizational behavior among successful players. After these patterns have

[1] See `http://themanaworld.org` for details.

been formalized, we will try to use them for building organized agents (AI players) and repeat the experiment in order to see if such agents behave better or worse then the initial non-organized version. In this deliverable we will particularly report on the quest design and implementation as well as the (initial) AI player implementation.

1.3 Team Members

Markus Schatten (Principal investigator)
Head of Artificial Intelligence Laboratory
markus.schatten@foi.hr
+385 42 390891

Joaquim Belo Lopes Filipe
Departament of Systems and Informatics, Escola Superior de Tecnologia de Setúbal
joaquim.filipe@estsetubal.ips.pt
+351 265 790 040

Nikola Ivković
Department of Computing and Technology
nikola.ivkovic@foi.hr
+385 42 390872

Mladen Konecki
Department of Theoretical and Applied Foundations of Information Sciences
mladen.konecki@foi.hr
+385 42 390873

Mario Konecki
Department of Theoretical and Applied Foundations of Information Sciences
mario.konecki@foi.hr
+385 42 390834

Robert Kudelić
Department of Theoretical and Applied Foundations of Information Sciences
robert.kudelic@foi.hr
+385 42 390852

Ivan Magdalenić
Department of Computing and Technology
ivan.magdalenic@foi.hr
+385 42 390872

Marko Maliković
Faculty of Humanities and Social Sciences Rijeka
marko@ffri.hr
+385-(0)51-265-765

Bogdan Okreša Đurić
Artificial Intelligence Laboratory
dokresa@foi.hr
+385 42 390891

Jurica Ševa
Department of Theoretical and Applied Foundations of Information Sciences
jseva@foi.hr
+385 42 390873

Pietro Terna
Università di Torino
pietro.terna@unito.it
+39 011 6706067

Igor Tomičić
Department of Computing and Technology
igor.tomicic@foi.hr
+385 42 390869

2. Quest Design

In order to define an actual quest that will push players to collaborate and compete with each other a 3-day brainstorming session was organized. Participants were the members of the ModelMMORPG project as well as a number of interested students and faculty from the Faculty of organization and informatics where the meeting has been held. During the session a number of creative group problem solving techniques was employed (including idea writing, mindmapping and the 6 thinking hats technique) in order to get valuable results. The brainstorming session has yielded the following backstory:

We begin at Anaces. Swept clean by the Earthquake, knows in local tales as Day zero, the land laid barren. A group of villagers went to Anaces to search for the remainings of their culture and the remaining of their loved ones. They never returned from their quest. A second group of villagers, supported by two boats of warriors, conducted the second search.

Only one of the warriors, the strongest of them called Knecki, has returned barely alive, strapped of his weapons and his armor and wout open wounds and burned skin. As he gained strength, helped by the healers, he started remembering the details of what happened. The memories first came in his sleep. He saw huge cracks as the ground under his feet turned into rubble and as the fiery water started flowing over the barren landscape. His screams were silenced by the roars coming from the depths of the cracks and a huge shadow, fast as lighting, was cast upon him. As he gained full memory he told the story: dragons, let loose from their undercover caves where they were locked by the Gods of the ancient times, started covering the sky limits as they flew out of their dungeons. A great threat has come upon the whole Mana world.

The old story goes: you can only fight fire with fire. Ankonno, the old hermit cast out of the village many winters ago, proposed the solution. In order to fight the dragons the one thing most precious to dragons has to be obtained, the dragon egg. For the dragon egg to hatch special skills and magic are required. He proposed: "I will show you the knowledge in exchange for my private training arena where you can send your children to learn." And that is how dragon hunters were created.

The main idea behind the defined quest (given out by the arch wizard NPC to a party leading player) is that there are a number of dragon nests all around the Mana World. Every 24 ± 1 hours

a dragon lays an egg on a random location. A party of players needs to provide three people (an archer, a warrior and a mage) to transport the egg to their castle. The three players need to stay in line-of-sight during the whole transport or the egg cracks. In order to hatch the egg they need to bring it to a special NPC (the egg hermit) together with a magic hatching potion. The hatching potion, on the other hand, is mixed by another NPC (Pauline the Witch) which needs certain ingredients to mix it. If the players do not hatch the egg within 24 hours, it goes bad. When the dragon is hatched it becomes the pet of the party's castle. It also allows party members to summon a dragon during their fights.[1]

With such a quest we try to foster two types of organizational behavior: (1) coordination of party members during the acquiring of the egg, its transportation as well as gathering of ingredients for the hatching potion, but also (2) competitiveness as well as trade as the different parties try to increase their power, so producing social effects, with coordination, emergence of hierarchies and of specialization. The second type of behavior will be additionally fostered through a social website designed to spread around data about party results and statistics so most successful parties are seen by others which might make them more competitive.

[1] A detailed quest design document is available at `http://ai.foi.hr/modelmmorpg/ModelMMORPG-QuestDesignFinal.pdf`

3. Quest Implementation

3.1 Introduction

The most important part of this quest's implementation lies in careful modeling of the involved NPCs. TMW has an integrated scripting language that enables such programming that most of the actions and interactions of the game with the player should be realized through interaction with NPCs, while smaller parts can be done using e.g. item or gear scripts. Interaction between an NPC and the player is defined in scripts developed especially for TMW. As such, they are well suited for NPC modeling, but have their constraints and it has been proven to be unexpectedly hard to implement new elements to be used along side those which already exist. This feature is visible the most when trying to develop some advance aspects of the quest, e.g. that it is essential to have three players carrying the egg.

NPC characters in TMW are modeled as simple reflexive agents having a preprogrammed response to the state of environment, i.e. a simple reflex without reasoning. This is achieved by defining agents as finite state automata. Reactions of NPCs are governed by variables connected to the player character. The NPCs developed especially for this quest are mostly affected by a variable `$@DragonEggQuestState` whose value indicates current state of the quest. Example of a simple quest NPC indicates the way the mentioned variable is used, and change of their behavior accordingly. The example code below depicts one such NPC, which conveys some knowledge of the world, if the player has not finished the quest (`$@DragonEggQuestState` < 7), or congratulates the player on a successfully finished quest (`$@DragonEggQuestState` == 7).

```
001-2.gat,92,30,0|script|Green Wizard#3|357{
if ( $@DragonEggQuestState == 7 ) goto L_QuestDone;
mes "[Green Wizard]";
mes "\"Whenever I need potions I go to a witch,
       they mix the finest of all.\"";
goto L_Close;
L_QuestDone:
       mes "[Green Wizard]";
       mes "\"You really did hatch that dragon,
```

```
                I can't believe it!\"";
        goto L_Close;
L_Close:
        close;}
```

The code above contains basic NPC construction template. Other than the main script, every NPC is defined by four mandatory elements: NPC's position (given by map name, location X, location Y and facing direction), type of the code that follows (which is always *script* for NPCs), name of the NPC, and NPC definition ID which references graphic and other elements of the NPC which are needed for the NPC to be represented in the game.

The most challenging NPCs needed for this quest are the Arch-Wizard and the Egg Hermit. Although they contain mostly basic interaction with the player, their states are interdependent and have to be synchronized according to the value of the quest variable mentioned earlier.

The Arch-Wizard This is the only NPC who can give out our quest to any player. As such, this is the only NPC which must be able to recognize a party leader, since only party leaders should be able to coordinate the quest. This is a challenge because there is no defined element of the script language to perform such a check on a player character. Custom implementation of the needed function proved to be rather difficult given the curent state of the implemented system. The rest of the functionality needed by the Arch-Wizard is rather simple transitioning between available states.

The Egg Hermit Some problems arose with how Egg Hermit NPC should function since the Egg item a player character brings to the Hermit should not be older than 24 hours. Such functionality is not natively implemented in the game, so the Egg item could not expire or go bad after certain amount of time. One of the solutions to this problem lies in modeling behavior of the Hermit NPC. When a player arrives to the Hermit carrying the Egg item, the Hermit checks values of variables concerning Egg pickup time and Egg laying time against current server time and using a number of rules reaches a conclusion on whether the egg is still fresh or it went bad.

The Egg problem The most complex problem, as of yet, directly concerning NPC scripts, was how to implement the quest requirement that an Egg item is to be laid every 24 ± 1 hours, expiring after 24 hours. To further complicate the problem, there are several sites in TMW where the Egg can spawn. Only one of these predesignated sites should be chosen to spawn the Egg item, on random, hence players could not predict where exactly the Egg will be spawned. After some thought this challenge was solved using several new NPCs. Since NPCs can interact with each other, an NPC called Central Egg Authority was created. This NPC coordinates four more NPCs who act as a kind of a nest.

Central Egg Authority One NPC is used as a central hub to control spawning of the Egg item. This NPC keeps timer in check, decides on the Egg-spawning place, spawns the Egg and updates values of the Egg-related timing variables. This NPC will have no immediate interaction with the player character, but will have control over the environment.

Egg Collectors Four NPCs are located on diverse remote locations throughout TMW, and are made visible and active, or invisible and thus inactive, by the Central Egg Authority. If an Egg Collector is active, it has the Egg, and the fastest player can get the Egg. In addition to ease communication with player characters, these NPCs perform several checks so as to decide if this particular player character is eligible to receive the Egg item.

Although the quest is not yet finished, a lot of the visible work has been done. Some of the mentioned challenges await, e.g. party leader identification, and mandatory Egg-transporting trio. We have to conduct static map instancing as well, even though it is not natively supported by TMW engine, but a satisfactory improvised solution is being developed. Even though the concepts are set, much of the code is subject to change.

The following chapter is covering the process of Dragon Egg quest implementation in The Mana World 2D open source MMORPG. Development of the content will be covered as follows:

The development process is going to portray all the needed resources, ranging from used file details to used code snippets to The Mana World (TMW) screenshots. Complete code developed for this purpose is attached at the end of this document, under RRRRRRRR, and will be duly referenced to. Addresses of all the folders used in the development process are stated using their relative location inside the default `tmwAthena` folder, subfolder `tmwa-server-data` e.g. file containing party information located at:

`/home/{active_username}/tmwAthena/tmwa-server-data/world/save/`

will be referenced as follows:

`/world/save/.`

3.2 Developing NPCs

In general, Non-Player Characters (NPCs) are defined in two aspects: graphical and behavioural. Developed on the client side, and used by the ManaPlus client run by a user, graphical aspect of an NPC is stored in the following files:

- `.xml` file at `/client-data/npcs/`
- `_include.xml` at `/client-data/npcs/`
- `.xml` file at `/client-data/graphics/sprites/npcs/`

Behavioural aspect of an NPC is defined by a script file located in a folder named according to a map where this given NPC is located, and must referenced in an include file of the map, as follows:

- `.xml` file at `/world/map/npc`
- `_include.xml` at `/world/map/npc`

The mentioned files will be described in detail for every NPC developed.

3.2.1 The Egg Hermit

The Egg Hermit NPC is the most important NPC in the quest described, for it helps hatch the dragon egg. Egg hatching takes place only after a player brings a certain number of defined items (1 Dragon Egg, and 1 Hatching Potion). Egg Hermit is the only NPC capable of hatching a dragon egg, therefore the quest cannot be completed successfully without communicating with the Hermit. Stationary and spawned on a single point on a predetermined map, Hermit is connected to a certain map.

Files

A number of files are needed for this NPC to be developed. In particular, these are described and detailed as follows.

hermit.txt (/world/map/npc/013-1/)

Script containing Hermit NPC's behaviour code, governing his location, sprite used, NPC name, and interaction possibilities of the NPC. Hermit is located on map 013-1, coordinates (111,69), the NPC is named Hermit, and uses sprite with ID 116. All the named elements are shown in listing 3.1 below.

Listing 3.1: Initial settings of Hermit NPC

```
013-1.gat,111,69,0|script|Hermit|116{...}
```

Script execution following interaction initiation is determined by a series of conditional statements depending on a special variable `DragonEggQuestState`. The script jumps to the label corresponding to a certain variable value, as shown in listing 3.2.

Listing 3.2: Conditional script execution

```
if (DragonEggQuestState == 1) goto L_St1;
if (DragonEggQuestState == 2) goto L_St2;
if (DragonEggQuestState == 3) goto L_St3;
if (DragonEggQuestState == 4) goto L_St4;
if (DragonEggQuestState == 5) goto L_St5;
if (DragonEggQuestState == 6) goto L_St5;
if (DragonEggQuestState == 7) goto L_St7;
goto L_End;
```

Since Hermit is the last NPC to have anything to do with the egg item, script analyzes time when the egg spawned/was laid, time of pickup and the time of current communication session beginning. According to the mentioned timestamps, the NPC will or will not allow further communication. This behaviour is noted in listing 3.3. If user has the egg, and the egg is bad (not given in within 24 hours of laying), the user will be informed, and the egg will be deleted from user inventory.

Listing 3.3: Conditions for egg-health determination

```
//check against pickup time
if (countitem(6006) &&
DragonEggPickupTime < $@DragonEggLaidTime || ((DragonEggPickupTime -
    ↪ $@DragonEggLaidTime) > 86400)) goto L_BadEgg;

//check against current time
if (countitem(6006) &&
gettimetick(2) < $@DragonEggLaidTime || ((gettimetick(2) -
    ↪ $@DragonEggLaidTime) > 86400)) goto L_BadEgg;
```

Once the egg is removed and the user is notified, quest of the state changes accordingly, based on possession of the Hatching potion item, as shown in listing 3.4.

Listing 3.4: In case of a bad egg Hermit will delete it from player's inventory and inform them

```
L_BadEgg:
       mes "This egg you have is bad, and you should feel bad. No amount of
           ↪  hatching potion will help you now.";
```

```
        mes "Let me dispose of it for you. These can be hazardous for the
            ↪ environment.\"";
        delitem 6006, 1;
        //if the Player has the potion
        if (countitem(6606) && DragonEggQuestState<5) set
            ↪ DragonEggQuestState, 3;
        if (countitem(6606) == 0 && DragonEggQuestState < 5) set
            ↪ DragonEggQuestState, 1;
        goto L_End;
```

Before further communicating with the player, Hermit checks if the given player has started the quest. Based on this data, Hermit either continues interaction, or simply ends it, as defined in listing 3.5 below.

Listing 3.5: Hermit will not communicate with just anyone

```
if (DragonEggQuestState == 0) goto L_St0;

L_St0:
        mes "Never mind,I got my work to do anyway. Get outta my way, don't
            ↪ you see I'm working?\"";
        menu
                "What are you doing?", L_Next,
                "O..K... (you back away slowly)", L_End;
L_Next:
        mes "[Ankonno the Hermit]";
        mes "\"I'm trying to hatch this duckling egg, but I'm out of
            ↪ hatching potion. You don't seem like somebody who'd have it,
            ↪ so go away.\"";
        goto L_End;
```

Hermit NPC takes care of the rule of three implemented in quest, stating that, when player finds the egg, two other players have to be their companions until the egg is returned to the Hermit. Checking process is performed in defined intervals of five seconds, as seen in listing 3.6. The functions involved in checking if these helpers are always near the player are discussed in chapter YYYYYYYYYYYY.

Listing 3.6: When the egg is collected timer is initialized and every 5 seconds the needed function is called unless checking is needed no more

```
OnEggPickup:
        //start timer when this event is initiated
        initnpctimer;
        end;

OnTimer5000:
        //stop checking when checking not needed
        if (DragonEggQuestState == 0 || $@DragonEggState == 0)
        goto L_StopCheck;
        callfunc "CheckingHelpers";
        end;
```

```
L_StopCheck:
	//stop timer and checking
	stopnpctimer;
	set MissingHelpWarning, 0;
	end;
```

_import.txt (/world/map/npc/013-1/)

The above mentioned Hermit script must be included in a map-wide file where all the NPCs present on the given map are catalogued, written as is shown in listing 3.7.

Listing 3.7: Every NPC script file must be included in order to be takein in account when server starts

```
npc: npc/013-1/hermit.txt
```

npc116.xml (/client-data/npcs/)

Visual representation of a given NPC is governed by files in `client-data` folder. Hermit NPC has ID `116` (line 3 in listing 3.8), it uses sprite as stated in line 4, enhanced by items referenced in lines 5-8.

Listing 3.8: Hermit is visually represented using sprite stated in line 4 enriched with items referenced in lines 5-8

```
<?xml version="1.0"?>
<npcs>
	<npc id="116">
		<sprite>model/male.xml|#53202b,7f4f45,9e6a43,d09459,fcd3a1,
		    ↪ fefffc;#2f312f,727471,a4a6a3,dad5f5;#3a3a3a,d5d5d5</
		    ↪ sprite>
		<sprite>equipment/chest/robe-male.xml|#201818,521800,7b5541,
		    ↪ ac9962,c5b176</sprite>
		<sprite variant="6">equipment/override/adult-outfits.xml</
		    ↪ sprite>
		<sprite variant="1">equipment/override/adult-head.xml</sprite
		    ↪ >
		<sprite variant="2">equipment/override/adult-head.xml</sprite
		    ↪ >
	</npc>
</npcs>
```

_include.xml (/client-data/npcs/)

The Hermit NPC visual definition must be included in a file where apll NPC visual definitions are catalogued, written as is shown in listing 3.9.

Listing 3.9: Hermit NPC must be included if it is to be used

```
<?xml version="1.0"?>
<npcs>
	...
	<include name="npcs/npc116.xml"/>
	...
</npcs>
```

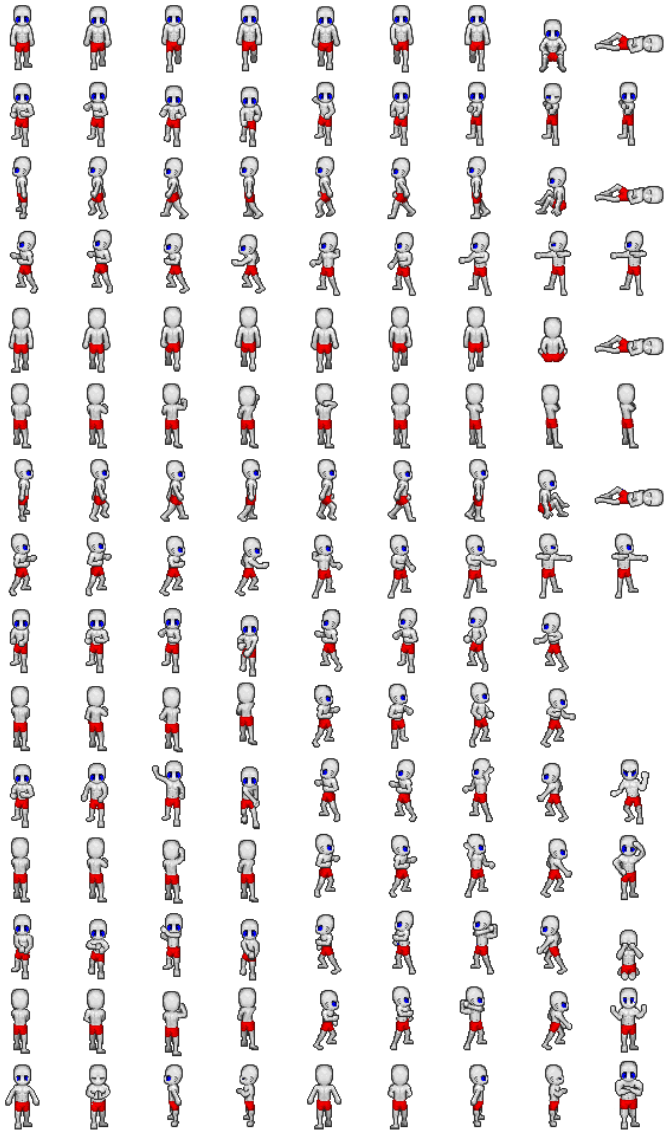


Figure 3.1: Sprite sheet of Talponian male sprite used by the Hermit NPC

male.xml (/client-data/graphics/sprites/model/)

Sprite used by Hermit NPC is based on a set of graphics `Talponian_male.png` (line 3 in listing 3.10, depicted in figure 3.1). Each graphic of the sprite is of size 64×64 pixels.

Listing 3.10: Sprite used by Hermit NPC is based on the image in line 3

```
<?xml version="1.0"?>
<sprite variants="135" variant_offset="1">
       <imageset name="base" src="graphics/sprites/model/Talponian_male.png
           ↪ |W;B;R" width="64" height="64"/>
       <include file="model/Talponian_male.xml"/>
</sprite>
```

3.2.2 The Arch-Wizard

The only NPC where a player can start their Quest for the Dragon Egg, the Arch-Wizard has a highly important role in formal sense. Arch-Wizard is only accessible to players who are members of a party. Once started, the Arch-Wizard provides players with advice, while Hermit NPC is responsible for cquest completion. The final step though, consisting of learning the dragon-summoning spell, is conducted by the Arch-Wizard. Once the egg is hatched by the Hermit, Arch-Wizard is willing to share the spell.

Files

Certain files are needed for this NPC to be developed. In particular, these are described as follows.

wizards.txt (/world/map/npc/001-2/)

One Arch-Wizard is initialized in every party-dependent map on specific location $(104, 27)$, is `named Arch-Wizard#[party map ID]` and is represented with sprite number 354.

Listing 3.11: Initial settings of Arch-Wizard NPC

```
0-101.gat,104,27,0|script|Arch-Wizard#101|354
{...}
```

The first condition Arch-Wizard must obey is that he will initiate conversation only with players who belong in a party. This condition is defined in line 3 in listing 3.12 below. Should the given player be suitable for conversation with the Arch-WIzard, second level of initial conditions is checked, similar to the Hermit NCP, testing state of the quest and behaving appropriately. Through the quest, before the final step, the Arch-Wizard is a very reactive agent. Only near the end of script, while interacting with a player, the Arch-Wizard summons a dragon in party common room (line 6), but before doing so, checks if there are any dragons alive in the said area, and kills them (line 5). Therefore, only one friendly dragon can live in party common room at any given time (except if the other dragons are summoned using a spell). When a player learns how to summon a dragon, their party ID is saved as a local (player-bound) variable, used by an attempt of party-wide summoning ability described beginning on page 33.

Listing 3.12: Arch-Wizard code excerpt focusing on initial conditioning and monster summoning and killing monsters in an area

```
0-101.gat,104,27,0|script|Arch-Wizard#101|354
{
       if (getcharid(1)) goto L_Quest;
       ...
       killmonster "0-101.gat", "All";
       monster "0-101.gat", 113, 79, "Dragon", 1134, 1;
       set DragonSummonParty, getcharid(1);
       ...
}
```

_import.txt (/world/map/npc/0-102/)

Arch-Wizard script file mus be, just like any NPC script file, included in the `_import.txt` file using code as follows in listing 3.13.

Listing 3.13: Every NPC file must be included in `_import.txt` file to be included in the game

```
npc: npc/0-102/wizards.txt
```

npc354.xml (/client-data/npcs/)
Arch-Wizard NPC by definition from line 1 in listing 3.12 uses NPC definition ID 354 for visualization purposes. This definition, in turn, uses sprite stated in line 3 of listing 3.14.

Listing 3.14: Arch-Wizard is visually represented with sprite named in line 3

```
<npcs>
	<npc id="354">
		<sprite>npcs/sittingsage.xml</sprite>
	</npc>
</npcs>
```

sittingsage.xml (/client-data/graphics/sprites/npcs/)
Sprite used by Arch-Wizard is based on a graphic `sittingsage.png` where elements are sized 40×49 pixels, as is written in line 2 of listing 3.15. Some of the frames of such imageset are turned into a simple animation effect (lines 4 to 18)

Listing 3.15: Animation for visual representation of Arch-Wizard is built using frames of imageset based on an image file from line 2

```
<sprite>
	<imageset name="base" src="graphics/sprites/npcs/sittingsage.png"
	    ↪ width="40" height="49"/>
	<action name="stand" imageset="base">
		<animation direction="default">
			<!-- First Line -->
			<frame index="0" delay="140"/>
			<frame index="1" delay="140"/>
			<frame index="2" delay="160"/>
			<!-- Goto Second Line -->
			<frame index="1" delay="90"/>
			<!-- Second Line -->
			<frame index="3" delay="160"/>
			<frame index="4" delay="140"/>
			<frame index="5" delay="160"/>
			<!-- Goto First Line -->
			<frame index="4" delay="80"/>
			<frame index="1" delay="100"/>
		</animation>
	</action>
</sprite>
```

3.2.3 Egg dealing set of NPCs

The problem of how to bring the Egg into the game world deserves some attention. It was not a straightforward implementation. While the Egg item needed a small number of simple code lines, implementing the egg-dealing system went through a number of changes before the final stage was achieved, which is in use in the game.

While the initial idea was having the Egg item lying around on the game floor, ready for a player to pick it up, the idea was abandoned shortly because of the game dynamics and the nature of dropped items and items found on ground - the problem is that they disappear after a specific amount of time.

The idea developed into an NPC giving out the Egg item. Such a solution provides many benefits, including scripting possibilities, allowing for better manipulation and player interaction. Further requirements, such as The rule of three (described in detail on p. 30), provided solid argument in favour of the NPC solution.

Since the Egg item can be encountered in randomly selected on of three possible locations in Mana World, such a requirement provided another challenge. The best solution was found in three distinct NPCs using almost the same script code, but appearing in different locations. Every time a location was randomly chosen, all the three NPCs are made invisible, and only the appropriate one is made visible again. This system assured location mystery even to the Game Masters, and enabled rather complex player interaction through using NPCs.

In order to coordinate these three Egg-dealing NPCs, one central character was needed. This was achieved using a hidden NPC which cannot be encountered anywhere in the game, but is active since the initiation of the NPC at server start. This NPC coordinates appearances and disappearances of the NPCs dealing Egg item, the interval between two Egg item appearances (fresh Egg item creation) and the location where the Egg item will be available next.

3.2.3.1 Calimero

Acting as the coordinator of the egg-dealing NPCs, this character acts according to the built-in timer which goes off every 24, 25 or 26 h, chosen randomly. When the time is right, Calimero randomly chooses a suitable location (out of the given three), kills alive dragons on all three locations, summons ten dragons on the chosen location, proclaims the egg as valid, allows the egg-dealing NPCs to hand out the egg, and informs all players of the event using a global announcement. In the end, Calimero restarts his timer, and waits for the new perfect moment.

hermit.txt (/world/map/npc/013-1/)

When initialized by server, this NPC immediately starts it's timer from 0, and randomly decides on the amount of time it is going to wait before doing anything. At the moment of initialization, first Egg item is placed in the world. Afterwards, only when the set timing is right, Egg item placement occurs. This process is shown in listing 3.16 below.

Listing 3.16: On initialization (line 13) NPC timer is started (line 14) and the first Egg item is placed (line 16) while subsequent Egg item placement is done according to the set intervals (lines 3-11)

```
001-2.gat,112,40,0|script|#Calimero|127
{
       OnTimer86400000:
              if ( $@HatcheryInterval == 0) goto L_LayEgg;
                     end;
       OnTimer86430000:
              if ( $@HatcheryInterval == 1) goto L_LayEgg;
                     end;
       OnTimer86460000:
              if ( $@HatcheryInterval == 2) goto L_LayEgg;
                     end;
       ...
       OnInit:
              initnpctimer;
              set $@HatcheryInterval, rand(3);
              goto L_LayEgg;
}
```

Upon jumping to L_LayEgg label in the script, NPC chooses the desired location, warps the appropriate Egg Collector NPC on a random spot on the location-based map. the players are informed of the new Egg item via the global announcement. and new monsters are spawned where the Egg item is going to be placed. The process is defined by code in listing 3.17 below.

Listing 3.17: Location is decided on random (line 2) and all Egg Collector NPCs are disabled (lines 3-5) before an NPC is newly positioned (lines 7-9) and the players are informed of the new Egg item (lines 14-16) while dragons are killed and spawned in the chosen area (lines 18-24) and Egg item creation time is memorised (line 26) and the NPC timer is restarted (line 28)

```
L_LayEgg:
       set $@location, rand(3);
       disablenpc "Egg Collector#S";
       disablenpc "Egg Collector#N";
       disablenpc "Egg Collector#W";

       if ($@location == 0) npcareawarp 30, 25, 125, 125, 0, "Egg Collector
          ↪ #S";
       if ($@location == 1) npcareawarp 20, 78, 85, 170, 0, "Egg Collector#
          ↪ N";
       if ($@location == 2) npcareawarp 20, 25, 90, 100, 0, "Egg Collector#
          ↪ W";

       set $@CanGiveDragonEgg, 1;

       if ($@location == 0) announce "A dragon has been sighted surrounded
          ↪ by snakes!", 0;
       if ($@location == 1) announce "A dragon has been sighted amidst the
          ↪ snow!", 0;
       if ($@location == 2) announce "A dragon has been sighted in the
          ↪ swamp!", 0;

       killmonster "006-3", "All";
       killmonster "047-3", "All";
       killmonster "026-1", "All";

       if ($@location == 0) areamonster "006-3", 30, 25, 125, 125, "Dragon
          ↪ ", 1137, 10;
       if ($@location == 1) areamonster "047-3", 20, 78, 85, 170, "Dragon",
          ↪  1136, 10;
       if ($@location == 2) areamonster "026-1", 20, 25, 90, 100, "Dragon",
          ↪  1138, 10;

       set $@DragonEggLaidTime, gettimetick(2);
       set $@HatcheryInterval, rand(3);
       setnpctimer 0;
       end;
```

Calimero NPC has no visual representation in the game world. Since there is no need for player interaction with this NPC, it has been coded using sprite ID 127 (line 1 in listing 3.16) meaning no visuals are used to represent this NPC and it is not clickable, thus no player-initiated interaction is

possible.

Naming mechanism used by The Mana World allows for NPCs with no in-game name visible, even though such an NPC can be referenced by name. This feature is achieved by using # symbol. Thus this NPC is named `#Calimero`, and can be referenced as such from inside the script, but has no visible name in the game.

3.2.3.2 Collectors

Three Egg Collector NPCs exist, each in one of the three designated areas. When initialized, they are all disabled and need to be enabled by the Calimero NPC.

hermit.txt (/world/map/npc/013-1/)

Upon initiated player contact the NPC checks if the player started the Quest for the Dragon Egg (line 3 of listing 3.18), and acts accordingly, teleporting the player to a set location in the Mana World if the player did not start the quest (line 13 in listing 3.18). Should the player who started interaction already own an Egg item, this NPC removes the old Egg item from the player's inventory (line 17 in listing 3.18) and gives them a new one (line 23 in listing 3.18). Upon giving the Egg item to the player, time is saved (line 25 in listing 3.18) and it further Egg item dealing is disabled (line 24 in listing 3.18). This mechanics was used in order to achieve uniqueness of the Egg item, i.e. only one usable Egg item can exist at any given time. If a player takes the Egg item from the Egg Collector NPC, the NPC is disabled in 100 miliseconds, preventing any other players to interact with it until new Egg items is ready to be dealt (line 36 in listing 3.18).

Listing 3.18: Location is decided on random (line 2) and all Egg Collector NPCs are disabled (lines 3-5) before an NPC is newly positioned (lines 7-9) and the players are informed of the new Egg item (lines 14-16) while dragons are killed and spawned in the chosen area (lines 18-24) and Egg item creation time is memorised (line 26) and the NPC timer is restarted (line 28)

```
006-3.gat,80,80,0|script|Egg Collector#S|358
{
        if (DragonEggQuestState < 1) goto L_NoQuest;
        mes "[Strange Person]";
        mes "\"Take it! Take it!!\"";
        if (countitem(6006) >= 1) goto L_DeleteDragonEgg;
        if (countitem(6006) < 1 && $@CanGiveDragonEgg == 1) goto
            ↪ L_GetDragonEgg;
        goto L_Close;

        L_NoQuest:
                mes "[Strange Person]";
                mes "\"You're not the right one!! Begone!\"";
                warp 001-1, 54, 38;
                goto L_Close;

        L_DeleteDragonEgg:
                delitem 6006, 1;
                if (countitem(6006) >= 1) goto L_DeleteDragonEgg;
                if ( $@CanGiveDragonEgg == 1) goto L_GetDragonEgg;
                goto L_Close;

        L_GetDragonEgg:
                getitem 6006, 1;
```

```
                set $@CanGiveDragonEgg, 0;
                set DragonEggPickupTime, gettimetick(2);
                announce "Somebody picked up the dragon egg, and they are on
                    ↪ their way to hatching it!", 0;
                specialeffect FX_MAGIC_BLUE_TELEPORT;
                initnpctimer;
                callfunc "NamingHelpers";
                goto L_Close;

        L_Close:
                close;

        OnTimer100:
                disablenpc "Egg Collector#S";
                end;
}
```

When a successfully player takes the Egg item from the Egg Collector NPC, the player must name two players to help him carry the Egg item. This process described on page 30 is started by line 29 in listing 3.18.

3.2.4 Pauline the Witch

One of the numerous Witches living in the Mana World, Pauline the Witch is located in the castle of Tulimshar, along with the Wizard Council. She demands the user to possess the knowledge of various herbs and potion ingredients. Building on this predefined behaviour is the script governing her Hatching Potion dealing policy. Player must bring some ingredients to be able to receive the Hatching Potion in return. This NPC is defined by a quite straightforward script based on simple reactive behaviour.

Files

pauline.txt (/world/map/npc/001-2/)

Pauline already exists in the game and has a set interaction with a player. Additional code was used to create the behaviour needed. Simple player communication informing the given player of the ingredients needed for the Hatching Potion (line 3 in listing 3.19), then checking whether the given player carries the needed ingredients with him (lines 16-26 in listing 3.19) and then finally giving them the potion (lines 34-46 in listing 3.19)..

Listing 3.19: Pauline the Witch NPC is an example of a slightly more complicated reactive NPC since the interaction is based only on status of the given player character

```
L_Hatching:
        mes "[Pauline]";
        mes "\"A hatching potions you say? Yes, I know how to mix one. I can
            ↪  mix it for you, but I need quite a lot of ingredients:
            ↪ different herbs like Alizarin Herb, Gamboge Herb, Cobalt Herb,
            ↪  and Mauve Herb. Then I need White Fur, a Mountain Snake
            ↪ Tongue, a Bat Wing, and a Pink Antenna. I also need some
            ↪ Leather Gloves to cover my hands.\"";
        set @pauline_state, 5;
        callsub S_Update_Mask;
        goto L_Close;
```

```
L_GetPotion:
       mes "[Pauline]";
       mes "Did you bring the ingredients for the hatching potion?";
       menu
              "Yes", L_HaveIngredients,
              "No", L_Close,
              "Can you tell me the list again?", L_Hatching;

L_HaveIngredients:
       mes "The witch looks in your backback";
       if (countitem("AlizarinHerb") < 1) goto L_NotEnough;
       if (countitem("GambogeHerb") < 1) goto L_NotEnough;
       if (countitem("CobaltHerb") < 1) goto L_NotEnough;
       if (countitem("MauveHerb") < 1) goto L_NotEnough;
       if (countitem("WhiteFur") < 1) goto L_NotEnough;
       if (countitem("MountainSnakeTongue") < 1) goto L_NotEnough;
       if (countitem("BatWing") < 1) goto L_NotEnough;
       if (countitem("PinkAntenna") < 1) goto L_NotEnough;
       if (countitem("LeatherGloves") < 1) goto L_NotEnough;
       next;
       mes "[Pauline]";
       mes "\"Let's see... Yes you have all I need! Do you want me to mix
          ↪ the potion for you?\"";
       menu
              "Yes", L_MixPotion,
              "No", L_Close;

L_MixPotion:
       delitem "AlizarinHerb", 1;
       delitem "GambogeHerb", 1;
       delitem "CobaltHerb", 1;
       delitem "MauveHerb", 1;
       delitem "WhiteFur", 1;
       delitem "MountainSnakeTongue", 1;
       delitem "BatWing", 1;
       delitem "PinkAntenna", 1;
       delitem "LeatherGloves", 1;
       mes "The witch puts on the leather gloves and mixes the ingredients
          ↪ in a jar. ";
       mes "[Pauline]";
       getitem "HatchingPotion", 1;
       mes "Here you go!";
       close;
```

3.2.5 Supportive NPCs

3.2.5.1 Council of Wizards

Consisting of several Wizards, this set of NPCs is used to guide the player in the right direction, i.e. to inform them of the general aspects of the Quest for the Dragon Egg. Simply reactive, Wizards

conduct highly constrained interaction with the player, depending only on the fact whether the quest is started by the player.

wizards.txt (/world/map/npc/0-101/)
For every party-specific map defined there is one file containing Arch-Wizard and Council of Wizards definitions. The Council consists of NPCs modeled as almost the simplest form of a reactive agent, where they have fixed interaction with a player, governed only by the fact whether the given player started the quest already or not.

Listing 3.20: Counfil of Wizards provides players with valuable information about the quest e.g. Purple Wizard informs about the possible whereabouts of the dragon while Green Wizard helps with information on the Hatcing Potion item

```
0-101.gat,92,30,0|script|Green Wizard#101|357
{
	if ( DragonEggQuestState == 7 ) goto L_QuestDone;
	mes "[Green Wizard]";
	mes "\"Whenever I need potions I go to a witch, they mix the finest
	    ↪ of all.\"";
	goto L_Close;

	L_QuestDone:
		mes "[Green Wizard]";
		mes "\"You really did hatch that dragon, I can't believe it
		    ↪ !\"";
		goto L_Close;

	L_Close:
		close;
}

0-101.gat,99,32,0|script|Purple Wizard#101|358
{
	if ( DragonEggQuestState == 7 ) goto L_QuestDone;
	mes "I heard that there have been dragon sightings all around,
	    ↪ frequently in the freezing north!";
	goto L_Close;

	L_QuestDone:
		mes "You really did hatch that dragon, I can't";
		mes "believe it!";
		goto L_Close;

	L_Close:
		close;
}
```

3.2.5.2 Desert Mana Seed

Located in every party's specific room, this NPC is used to share dragon invocation spell knowledge with other party members.

wizards.txt (/world/map/npc/0-101/)

With it's script contained in the same file as the Arch-Wizard and the Council of Wizards, there is one Mana Seed definition for every party-specific map. Desert Mana Seed NPC is the main medium in the process of party-wide dragon summoning spell propagation (described in further detail on page 33. When a player touches this Mana Seed NPC a special in-game function is called which runs a custom made event `#DragonSpell#101::OnSet` (line 9 in listing 3.21) with a delay of five hundred miliseconds.

Listing 3.21: The most important part of the Dragon Mana Seed NPC is usage of the `areatimer` function which runs event `#DragonSpell#101::OnSet` on map `0-101` in a set area after a delay of 500 ms

```
0-101.gat,104,19,0|script|Desert Mana Seed#101|166
{
...
L_Propagate:
mes "(a whisper fills the room)";
mes "\"By your will, oh benevolent Master of Dragons, your dragon will
    ↪ answer calls from all those now present.";
next;
mes "Remember: Sharing is caring!\"";
areatimer "0-101", 94, 17, 114, 37, 500, "#DragonSpell#101::OnSet";
goto L_Close;
...
}
```

3.2.5.3 Portrait of Alyva

Inspired by stories of Harry Potter, and the castle setting, entrance to the party-specific maps can be achieved by interacting with Portrait of Alyva NPC. This transition is implemented most efficiently using the NPC because of the level of interaction it requires. Since party-specific maps are bound to party ID, dynamic map name had to be referenced by warp function which is used for instant transportation of players from one point in the Mana World to another.

wizards.txt (/world/map/npc/001-2/)

Defined as a simple portal-like NPC, script of Portrait of Alyva is automatically run when a player enters an area of 2 units from the NPC. With no direct interaction needed, the given player is automatically teleported to the party-specific map, if applicable (line 4 in listing 3.22). Should the given player belong to no party, they are informed of it, and nothing else happens (lines 9-11 in listing 3.22).

Listing 3.22: Portrait of Alyva is not represented visually in the game world and there is no interaction with a player but the script is run automatically upon player's entry in an area and the player is informed of an action if there is one

```
001-2.gat,86,72,0|script|Portrait of Alyva|400,2,2
{
        if (getcharid(1) == 0) goto L_NoWarp;
        warp "0-" + getcharid(1), 85, 75;
        message strcharinfo(0), ":woosh!: sudenly, you are in your party's
            ↪ castle";
        close;
```

```
        L_NoWarp:
                mes "[Lady Alyva Laguia]";
                mes "I cannot let you enter the common room of a party
                    ↪ withouth you being in a party.";
                close;
}
```

Portrait of Alyva NPC uses sprite ID 400 since the NPC is clickable and can be engaged in interaction with a player, but does not have a defined visual representation in the game, rather acts as a simple clickable surface.

3.3 Developing Items

3.3.1 Dragon Egg

The Dragon Egg item is the main item of the quest. The implementation was accomplished using a preexistent Snake Egg sprite and the recolour function.

Dissimilar techniques were considered before the decision was settled on the current solution. The implemented solution was chosen for the sake of simplicity and code quantity.

Files

A number of files is needed for this item to be developed. In particular, these are described and detailed as follows.

item_db_use.txt (/world/map/db/)

Each defined item must be included in a list in one of the files containing item definitions. One such file, containing definitions of usable items, i.e. items which cause some effect upon being used, is `item_db_use.txt`. Each line in the file defines one item. Dragon Egg item is described as shown in listing 3.23.

Listing 3.23: Dragon Egg item is defined with characteristics as stated in `item_db_use.txt` file (here line 2) with value descriptions given in line 1

```
//ID, Name,    Label,   Type, Price, Sell, Weight, ATK, DEF, Range, Mbonus,
    ↪  Slot, Gender, Loc, wLV, eLV, View, {UseScript},    {EquipScript}
6006, DragonEgg, Dragon Egg, 2,  999999, 1,  10,   0,  0,  0,   0,   0,  2,
    ↪    0,  80,  50,  0,  {itemheal 15, 0;}, {}
```

item6006_DragonEgg.xml (/client-data/items/usable/)

While options considered in game mechanics are given in the aforementioned file, visual representation of the item, type and description (line 8 in listing 3.24) are defined in file `item6006_DragonEgg.xml`.

Listing 3.24: Dragon Egg item is visually based on Snake Egg item but recoloured and given a new ID and new status values and new description (line 8)

```
<items>
        <its:rules xmlns:its="http://www.w3.org/2005/11/its" version="1.0">
                <its:translateRule selector="//item/@name" translate="yes"/>
                <its:translateRule selector="//item/@description" translate="
                    ↪ yes"/>
                <its:translateRule selector="//item/@effect" translate="yes
                    ↪ "/>
                <its:translateRule selector="//*" translate="no"/>
```

```
        </its:rules>
        <item id="6006" image="use/food/snake-egg.png|W:#9fd485;R:#108D16;G
            ↪ :#FF3315" name="Dragon Egg" description="Stolen from a dragon,
            ↪  this delicate egg will go bad in at most a day." type="
            ↪ usable" hp="15" weight="10000"/>
</items>
```

3.3.2 Hatching Potion

Hatcing potion is developed using a sprite already existing in the game and customizing it. Sole purpose of the potion is to give it to the Hermit along with the Dragon Egg item. Reuired by quest definition, the mentioned combination hatches the egg and allows the summoning spell to be learned.

Files

A number of files is needed for this item to be developed. In particular, these are described and detailed as follows.

item_db_use.txt (/world/map/db/)

Analogous to the definition of any other item, Hatchin Potion item has in-game behaviour defined in list of usable items (listing 3.25).

Listing 3.25: Hatching Potion item is defined with characteristics as stated in `item_db_use.txt` file (here line 2) with value descriptions given in line 1

```
//ID, Name,      Label,      Type, Price, Sell, Weight, ATK, DEF, Range,
    ↪ Mbonus, Slot, Gender, Loc, wLV, eLV, View, {UseScript},    {
    ↪ EquipScript}
6606, HatchingPotion, Hatching Potion, 2,  6660,  1,  10,   0,  0,  0,   0,
    ↪    0,  2,   0,  80,  50,  0,  {itemheal 15, 0;}, {}
```

item6606_HatchingPotion.xml (/client-data/items/usable/)

Same as for the Dragon Egg item, visual representation of the item, type and description (line 8 in listing 3.26) are defined in file `item6606_HatchingPotion.xml`.

Listing 3.26: Dragon Egg item is visually based on Snake Egg item but recoloured and given a new ID and new status values and new description (line 8)

```
<items>
        <its:rules xmlns:its="http://www.w3.org/2005/11/its" version="1.0">
                <its:translateRule selector="//item/@name" translate="yes"/>
                <its:translateRule selector="//item/@description" translate="
                    ↪ yes"/>
                <its:translateRule selector="//item/@effect" translate="yes
                    ↪ "/>
                <its:translateRule selector="//*" translate="no"/>
        </its:rules>
        <item id="6606" image="use/potions/c.png|G:#383D26,ffffff" name="
            ↪ Hatching potion" description="Hatching potion. For hatching
            ↪ eggs." type="usable" hp="800" weight="1500" tag="Potions"/>
</items>
```

3.4 Developing the Dragon

Dragon monster implementation proved to be the greatest graphical challenge of the quest implementation. Graphic style of the monster included in the game should match the guidelines for creature development[1], should be a representative pixel art graphic, and should have sprite animations for the supported actions (moving, attacking, standing still).

Once the appropriate sprite was found online, it had to be redesigned to fit the limited sprite usage mechanics of the game. After some work aligning all the frames contained in a sprite, the image was ready for the game.

Several situations exist where a dragon is used:

- wild dragon roaming the location where the Egg item can be obtained;
- cooperative dragon which can be summoned;
- passive dragon which is summoned in party-specific map once the quest is solved.

Files

A number of files is needed for this monster to be developed. In particular, these are described and detailed as follows.

mob_db_over_100.txt (/world/map/db/)

Every monster (mob) present in game is listed in one of the following files according to presumed level of the monster:

- `mob_db_over_0_19.txt`
- `mob_db_over_20_39.txt`
- `mob_db_over_40_59.txt`
- `mob_db_over_60_79.txt`
- `mob_db_over_80_99.txt`
- `mob_db_over_100.txt`

Several similar variations of the dragon monster are implemented, as they appear in variuos parts of the map. One specific rendition of the dragon monster is implemented for summoning purposes, and one variation is created to be summoned in a party-specific map by the Arch-Wizard NPC upon quest completion. All these varieties of the dragon monster are shown in listing , where excerpt of the referenced file is given.

Listing 3.27: Dragon monster variations are set up under various IDs with varying names and slightly different stats and changing drop items and their drop-rates

```
1135, Dragon,  Dragon, 150, 30000, 0, 0, 9, 1, 100, 500, 60, 8, 50, 20, 50,
   ↪  60, 20, 50, 6, 1, 1, 9, 83, 143, 300, 1872, 672, 480, 539, 1000,
   ↪ 518, 400, 533, 150, 521, 70, 522, 1, 5115, 3000, 754, 2000, 0, 0, 0,
   ↪  0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 30
1136, DragonIce,  DragonIce, 150, 30000, 0, 0, 9, 1, 100, 500, 60, 8, 50,
   ↪ 20, 50, 60, 20, 50, 6, 1, 1, 9, 83, 143, 300, 1872, 672, 480, 539,
   ↪ 1000, 518, 400, 533, 150, 521, 70, 522, 1, 5115, 3000, 754, 2000, 0,
   ↪  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 30
1137, DragonSand,  DragonSand, 150, 30000, 0, 0, 9, 1, 100, 500, 60, 8, 50,
   ↪  20, 50, 60, 20, 50, 6, 1, 1, 9, 83, 143, 300, 1872, 672, 480, 539,
   ↪ 1000, 518, 400, 533, 150, 521, 70, 522, 1, 5115, 3000, 754, 2000, 0,
   ↪  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 30
1138, DragonForest,  DragonForest, 150, 30000, 0, 0, 9, 1, 100, 500, 60, 8,
   ↪  50, 20, 50, 60, 20, 50, 6, 1, 1, 9, 83, 143, 300, 1872, 672, 480,
   ↪ 539, 1000, 518, 400, 533, 150, 521, 70, 522, 1, 5115, 3000, 754,
   ↪ 2000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 30
```

[1] Available at the official The Mana World site: https://www.themanaworld.org/index.php/Dev:Monstersets

```
//ID, Name, Jname, LV, HP, SP, EXP, JEXP, Range1, ATK1, ATK2, DEF, MDEF,
    ↪ STR, AGI, VIT, INT, DEX, LUK, Range2, Range3, Scale, Race, Element,
    ↪ Mode, Speed, Adelay, Amotion,Dmotion,Drop1id,Drop1per,Drop2id,Drop2%,
    ↪  Drop3id,Drop3%, Drop4id,Drop4%, Drop5id,Drop5%, Drop6id,Drop6%,
    ↪ Drop7id,Drop7%, Drop8id,Drop8%, Item1, Item2, MEXP, ExpPer, MVP1id,
    ↪ MVP1per,MVP2id, MVP2per,MVP3id, MVP3per,mutationcount,
    ↪ mutationstrength
```

It is observable from the listing 3.27 above that some data changes from line to line. As described by line 5 of the mentioned listing, several items may be dropped by a dragon, defined by their drop-rate (probability of them dropping after killing the monster specified).

The most distinct feature of these five variations of the dragon monster is their behaviour. Behaviour of every monster is defined by their respective `Mode` value[2]. Wild dragons and the summoned dragon have `Mode` value `143` meaning they are moving looting assistive aggressive attacking monsters, while dragon monster summoned by the Arch-Wizard have `Mode` value `131` meaning they are moving attacking looters.

dragonG.xml (/client-data/graphics/sprites/monsters/)

The greatest challenge of visual nature was how to visually represent a dragon. Solution came in form of a dragon sprite sheet of interesting dragon forms, but it had to be redesigned and specially reformatted, so it would be usable by mechanics of The Mana World. the final image used for dragon sprite is shown as figure 3.2 (p. 28). Sprite definition using the mentioned picture is given as listing 3.28. Animations for the dragon monster consist of walking (lines 11-18 in listing 3.28), standing (lines 3-10 in listing 3.28), attacking (lines 25-42 in listing 3.28) and dying (lines 19-24 in listing 3.28) animations. All of these are defined by starting and ending frame of the given sprite sheet, which consists of a number of frames of fixed width and height. Each action can be specially oriented, covering monster movement, as in lines 12 and 15 of listing 3.28, where different animations are defined to be used depending on whether dragon is moving left or right.

Listing 3.28: Dragon monster variations are set up under various IDs with varying names and slightly different stats and changing drop items and their drop-rates

```
<sprite>
	<imageset name="base" src="graphics/sprites/monsters/dragonG.png|W"
	    ↪ width="240" height="184"/>
	<action name="stand" imageset="base">
		<animation direction="left">
			<sequence start="87" end="81" delay="175"/>
		</animation>
		<animation direction="right">
			<sequence start="0" end="6" delay="175"/>
		</animation>
	</action>
	<action name="walk" imageset="base">
		<animation direction="left">
			<sequence start="98" end="88" delay="175"/>
		</animation>
		<animation direction="right">
			<sequence start="11" end="21" delay="175"/>
```

[2]More on Mode value and how it influences monster behaviour is available at the official The Mana World web site: https://www.themanaworld.org/index.php/Dev:Monster_Design

```
            </animation>
        </action>
        <action name="dead" imageset="base">
            <animation>
                <sequence start="22" end="29" delay="175"/>
                <frame index="30" delay="3000"/>
            </animation>
        </action>
        <action name="attack" imageset="base">
            <animation direction="down">
                <sequence start="66" end="75" delay="75"/>
                <end/>
            </animation>
            <animation direction="left">
                <sequence start="142" end="134" delay="75"/>
                <end/>
            </animation>
            <animation direction="up">
                <sequence start="44" end="52" delay="75"/>
                <end/>
            </animation>
            <animation direction="right">
                <sequence start="55" end="63" delay="75"/>
                <end/>
            </animation>
        </action>
</sprite>
```

monster1138_DragonForest.xml (/client-data/monsters/)

Dragon sprite picture is rendered in greyscale because it makes a perfect basis for using recolouring effect built-in the game sprite definitions. Since three dragon monster variations were envisioned, it is usitable for them to be differentiated by colour. Therefore a green, a blue and a brown variation was created. Each of these dragon monster variations was defined separately. Just as shown in listing 3.28 where in-game mechanics details for dragon monsters was shown, listing below demonstrates how each dragon monster variation is defined visually. Forest dragon monster uses sprite mentioned earlier, but shaded green (line 3 in listing 3.29), and uses sounds for events of death, injury and spawning.

Listing 3.29: Each variation of the dragon moster is differentiated by it's couloured visualization e.g. Forest Dragon monster is shaded green (colour hex code `#7cdb7a` in line 3)

```
<monsters offset="0">
        <monster id="1138" name="DragonForest">
            <sprite>monsters/dragonG.xml|#7cdb7a</sprite>
            <sound event="die">monsters/dragon/dragon-die.ogg</sound>
            <sound event="hurt">monsters/dragon/dragon-hit.ogg</sound>
            <sound event="spawn">monsters/dragon/dragon-spawn.ogg</sound>
        </monster>
</monsters>
```

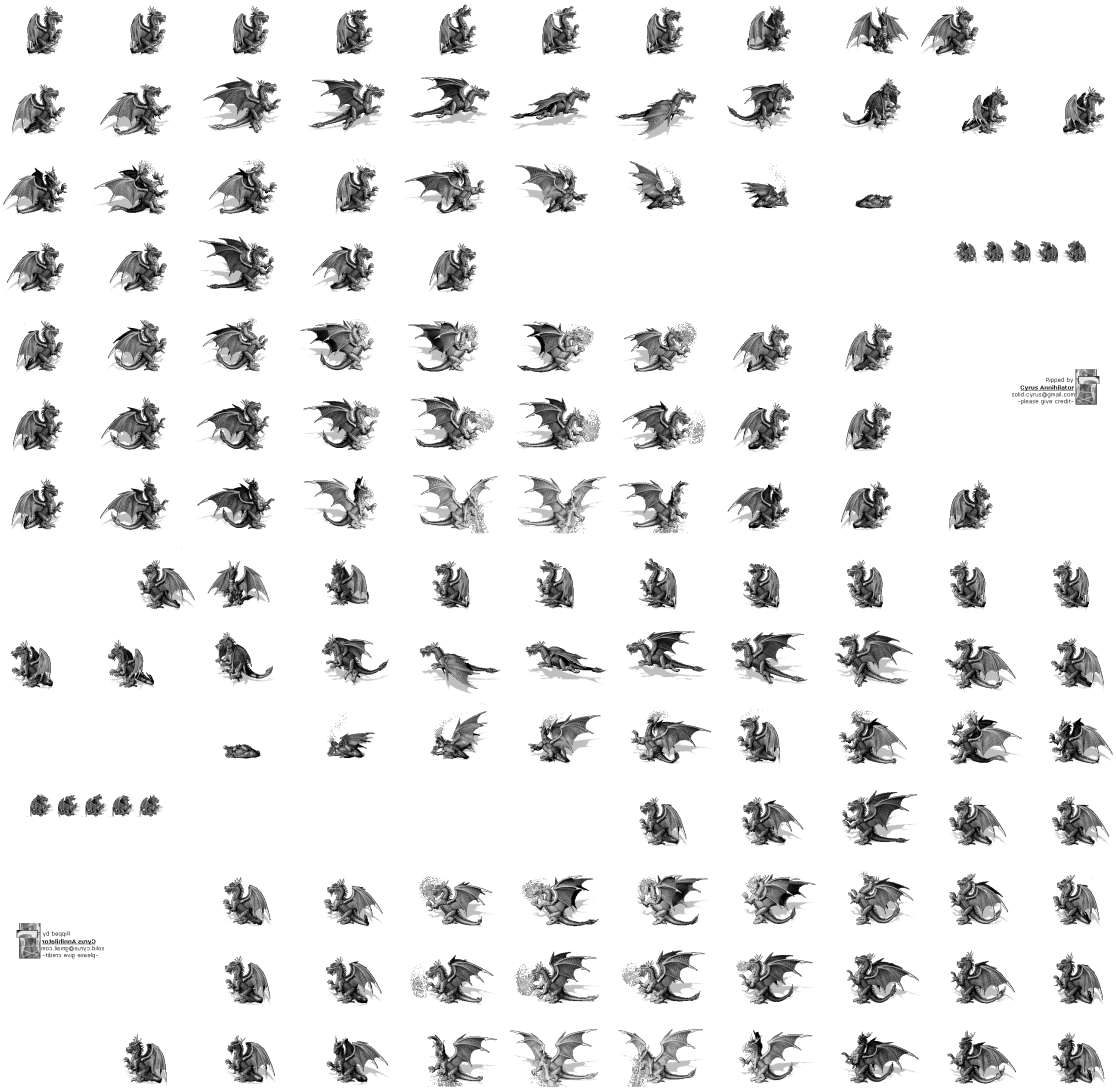


Figure 3.2: Greyscale dragon sprite image

Two other dragon monster variations are coloured as follows: Ice Dragon using colour `#92edf4`, and Sand Dragon using colour `#e8c76c`, while sprite for the sumoned dragon is in full colour, rather than simple recolour of a grayscale graphic.

3.5 Spell Development

Spell development is a rather straightforward process with spell definition mainly contained in one single file in form of a function. The challenge consisted of making the spell available party-wide, but contained in party only, i.e. a player cannot cast the spell if they leave the party where the quest was solved. More on this on page 33. Every spell in the Mana World is referenced by the appropriate incantation, and can be restrained by various conditions.

File

The whole spell definition is contained in one file described below.

magic-level2.sex (/world/map/conf/)

Spell developed for use in this quest uses incantation #dovahkiin (line 1 in listing 3.30), and can be cast only after the quest is finished (lines 7-23 in listing 3.30). The effect of casting this spell is a summoned cooperative dragon (lines 28-34 in listing 3.30) which is going to disappear 15 seconds after being summoned (line 34 in listing 3.30).

Listing 3.30: Each variation of the dragon moster is deverse by it's couloured visualization e.g. Forest Dragon monster is shaded green (colour hex code `#7cdb7a` in line 3)

```
(SPELL (LOCAL) summon-dragon "#dovahkiin" ()
       (=>
              (GUARD
                     (MANA 100)
                     (CASTTIME 5000))
              (EFFECT
                     (SCRIPT "{
                            set @CanSummonDragon, 0;
                            if(getcharid(1) == DragonSummonParty &&
                                ↪ DragonSummonParty) goto L_ChkTime;
                            set DragonSummonParty, 0;
                            end;
                     L_ChkTime:
                            if ((gettimetick(2)-DragonSummonTime) > 30)
                                ↪ goto L_Cast;
                            end;
                     L_Cast:
                            set @CanSummonDragon, 1;
                            set DragonSummonTime, gettimetick(2);
                            end;
                     }")
                     (IF (<
                            (script_int caster "@CanSummonDragon")
                            1)
                            (ABORT))
                     (CALL default_effect)
                     (sfx location SFX_SUMMON_START 0)
```

```
                    (WAIT 500)
                    (sfx location SFX_SUMMON_FIRE 0)
                    (spawn
                           (rbox location 2)
                           caster
                           1134
                           2
                           1
                           15000))))
```

Crucial part of the spell are lines 7-23 in listing 3.30, for it allows for party-constrained spell casting ability and introduces spell cooldown time. Before being cast, timing is checked against the last time spell was cast (lines 13 and 17 in listing 3.30). In case spell cooldown time passed, variable `DragonSummonParty` must have value identical to ID of the party the given player is a member of (line 9 in listing 3.30). Spell casting is aborted if variable `@CanSummonDragon` has value less then 1, i.e. if casting this spell is not permitted at the moment for the given player (lines 20-23 in listing 3.30).

3.6 Map Creation

Custom map instancing is not supported by the game engine, making it impossible to dynamically create party-specific maps. Since party-specific areas are supposedly the best places where parties can achieve private trade transactions and where members of the same party can hang out together, it was necessary to solve this challenge. The solution was found in creation of a large number of maps, but restraining access to a specific map to members of a specific party only. The matching was done using party ID and map ID.

As the desired map was created, all instances of map ID were changed into a placeholder text ready to be replaced automatically when new party-specific maps are created. Similar process was performed with NPC definitions for the NPCs which are placed on party-specific maps. Automatized party-specific map creation is realized using a simple Python script and several prepared files, as described above.

Maps were customized using Tiled[3], a free, easy to use and flexible tile map editor suggested by The Mana World team.

3.7 Ancillary Events

3.7.1 The Rule of Three

In order to induce cooperative behaviour amongst players, constraint was introduced concerning the Dragon Egg item. As defined during quest describing brainstorming session, three players are needed for the Egg item to be carried successfully anywhere. Since it is vital for the Egg item to be carried to the Hermit NPC (section 3.2.1, part 3.2, of this document), the Egg item must be carried according to the set constraint and it is not possible to pass it by. Used mechanics of realizing the constraint mentioned is as follows: upon receiving the Egg item, player must name two of their helpers, afterwards their positions are checked regularly and compared to the current position of the given player. The entire mechanics is described in detail below.

3.7.1.1 Naming Helpers

When a player starts communication with the Collector NPC which gives out Dragon Egg items, Collector NPC disappears, and the player must name two helpers who will accompany them on

[3]More information on Tiled: http://www.mapeditor.org/

their way to the Hermit NPC. Helpers are named one by one, and are identified by their names.

Every string (name) input by the player is verified on four levels:

- the player is notified if empty string was given, and prompted to try again (line 5, listing 3.31);
- online presence and general existence of named player character is verified, and the player is notified in case an offline or non-existent character name was given (line 7, listing 3.31);
- should the player input their character name, they are prompted to input new string, and are informed of the problem (cannot name themselves) (line 6, listing 3.31);
- party of the named player character is checked as well, for only members of the same party can be helpers in this quest (line 8, listing 3.31).

In case first helper was named successfully, the same process is performed for the second one. Only after both the helpers were named, is the player who initiated contact, given the Egg item. Code performing helper naming process for the first helper is shown in listing 3.31 below.

Interesting problem-inducing aspect of solution used, and presented in listing 3.31 below, are variable types and their constraints considering input data type. Even though it is nowhere stated clearly in developer's reference, The Mana World does not allow input to any variable type to be performed. Two main variable types are identified in general: integer (no suffix after variable name) and string ($ sign as suffix to variable name), e.g. `DragonChosenOne` is an integer variable and `DragonChosenOne$` is a string variable. Scope of a variable is defined by using prefix characters as follows:[4]

- @ - a temporary variable attached to a player, reset when the player logs out or when the server restarts;
- $ - a global permanent variable, stored by the map-server;
- $@ - a global temporary variable;
- # - a permanent account-based variable.

Problems arose because string variables can only be variables of a specific scope, i.e. string variables must have either prefix $ (permanent global variable) or $@ (temporary global variable). In the beginning this constraint caused problems because of the intended quest mechanics, but was solved introducing uniqueness constraint, i.e. only one Egg item is valid at any given time, and only one player can pick up the Egg item from a Collector NPC at any given quest period. Therefore, variables containing quest-related data can be global variables, as they are changed every time the Egg item is picked, and only one set of those is available at any given time.

Listing 3.31: Player helpers are named one by one and proposed character names are verified against several necessary constraints

```
L_ChooseHelper1:
       mes "Name the first player to help you:";
       input $@DragonHelper1$;
       set $DragonHelper1$, $@DragonHelper1$;
       if ($DragonHelper1$ == "") goto L_ChooseHelperConfirm1;
       if (getcharid(3, $DragonHelper1$) == $DragonOneID) goto
           ↪ L_WrongHelper1;
       if (isloggedin(getcharid(3, $DragonHelper1$)) == 0) goto
           ↪ L_HelperDoesNotExist1;
       if (getcharid(1, $DragonHelper1$) != getcharid(1,$DragonOneName$))
           ↪ goto L_WrongParty1;
       goto L_ChooseHelper2;
```

[4]Taken from The Mana World official web-site, available here.

```
L_ChooseHelperConfirm1:
       mes "[Ankonno the Hermit]";
       mes "If it happens you cannot elect two players to help you, you can
          ↪ give up now.";
       menu
              "Continue.", L_ChooseHelper1,
              "I give up.", L_PlayerGaveUp;

L_HelperDoesNotExist1:
       mes "[Ankonno the Hermit]";
       mes "This player " + $DragonHelper1$ + " seems offline or does not
          ↪ exist.";
       set $DragonHelper1$, "";
       next;
       goto L_ChooseHelperConfirm1;

L_WrongHelper1:
       mes "[Ankonno the Hermit]";
       mes "You can't name yourself.";
       next;
       goto L_ChooseHelperConfirm1;

L_WrongParty1:
       mes "[Ankonno the Hermit]";
       mes "Your helpers must be from your party.";
       next;
       goto L_ChooseHelperConfirm1;
```

3.7.1.2 Helper Location Validation

Once both helpers are named and verified to be real online payer characters, position validation process is initiated, and performed regularly every five seconds. Location verification consists of reading and storing location of the given player (lines 4-6, listing 3.32), and comparing it to locations of both of the helper characters (lines 8-12, listing 3.32). In case at least one of the helping player characters is outside of a 20×20 tiles virtual square with given player being the centre, they are warned of the violation (line 25, listing 3.32). Should the state repeat again in five seconds (during the next location verification), the Egg item gets broken and useless (line 22, listing 3.32).

Used both in function used for location validation and function for naming helper characters, is function `attachrid`. Every script initiated by a player character is passed a so-called RID (account ID of a character which caused the code to execute). This RID is used to access character-based data, to be used in scripts such as functions, scripts using timers, clock-based script activation, etc. Some scripts are run automatically, without a player initiating them. These scripts require usage of `attachrid` function, which attaches RID of a named player to the script. RID is removed from a script using `detachrid` function. In listing 3.32 below both of these functions are used, because data of several characters is needed for the whole script to be successful. At line 3 player who received the egg is attached to the script. Their location data is saved (lines 4-6), and first helper is attached to the script (line 8). If RID attaching is not possible, the referenced player character must be missing from the game or is not online. Location validation of the named player character is performed at line 9, where it is checked if they are located at the given map, in area defined by given coordinates. If positive, the other helper character is attached to the script (line 11) and their

location is validated (line 12). In the end, player who initiated the script is attached back to it, and new location validation period is started.

Listing 3.32: Player helpers are named one by one and proposed character names are verified against several necessary constraints

```
function|script|CheckingHelpers{
        detachrid;
        if (!(attachrid($DragonOneID))) end;
        set $DragonChosenMap$, getmap();
        set $DragonChosenX, getx();
        set $DragonChosenY, gety();
        detachrid;
        if (!(attachrid($DragonHelp1ID))) goto L_HelpMissing;
        if (!(isin($DragonChosenMap$, $DragonChosenX-10, $DragonChosenY-10,
            ↪ $DragonChosenX+10, $DragonChosenY+10))) goto L_HelpMissing;
        detachrid;
        if (!(attachrid($DragonHelp2ID))) goto L_HelpMissing;
        if (!(isin($DragonChosenMap$, $DragonChosenX-10, $DragonChosenY-10,
            ↪ $DragonChosenX+10, $DragonChosenY+10))) goto L_HelpMissing;
        detachrid;
        if (!(attachrid($DragonOneID))) end;
        set MissingHelpWarning, 0;
        donpcevent "Hermit::OnEggPickup";
        end;

        L_HelpMissing:
                detachrid;
                if (!(attachrid($DragonOneID))) end;
                if (MissingHelpWarning == 1) goto L_BreakEgg;
                if (MissingHelpWarning == 0) set MissingHelpWarning, 1;
                mes $DragonHelper1$ + " and " + $DragonHelper2$ + " are to
                    ↪ blame.";
                mes "The egg almost broke. Take more care about your helpers!
                    ↪  It might break if handled badly again.";
                message $DragonOneName$, "The egg almost broke!";
                donpcevent "Hermit::OnEggPickup";
                close;

        ...
}
```

3.7.2 Party-Wide Spell Propagation

By initial design of the quest in question, ability to summon a dragon was unlocked for all the members of a party, once leader of the party solved the quest. Detailed analysis of game mechanics proved such a concept to be cumbersome to implement and against logics of the given game.

With some help of The Mana World community, a different spell propagation system was developed. Since no party leader can be identified without heavy code adjustment, any member of the party can start and finish the quest, as long as the same character starts and finished the quest. Mechanism for spell casting constraint consists of spell incantation and special requirements such

as skill, magic school and skill or magic school levels. Although good enough for constraining spell casting for a single player, no mechanism is there which would allow a group of players to be able to cast a single spell.

Solution to this problem lies in script which can be run upon spell casting, but before the spell is cast entirely. Script was developed which verifies whether a player character is allowed to cast dragon summoning spell. Such a script was already mentioned in listing 3.30 on page 29, and is extracted in listing 3.33. Script embedded in spell definition works in unison with parts of scripts of two NPCs, i.e. with script of Arch-Wizard (p. 14, listing 3.34 below) and Desert Mana Seed (p. 20, repeated in listing 3.35); and one simple unbound script (listing 3.36).

The moment a player successfully finishes the quest, a variable is created attached to the given player, containing ID of the party the given player is a member of (line 8 of listing 3.34). This ensures the whole spell propagations system is bound to a particular party, i.e. to the party member of which was the player who solved the quest. If the given player, who solved the quest, wants to share spell casting ability with other players of their party, they must interact with Desert Mana Seed. Upon said interaction, area event occurs (line 7 of listing 3.35) and triggering execution of script #DragonSpell on all players present in predefined area. Since Desert Mana Seed is located in party-specific map, only members of a certain party can be present in the room. The triggered script sets variable value (line 5 of listing 3.36), similar to the process by the Arch-Wizard, to all the present player characters. This variable is crucial in spell-embedded script, as it is compared to the ID of a party the casting player is member at the moment of casting the spell (line 3 of listing 3.33). Further constraint is cooldown effect of the spell, where no less then 30 seconds must pass between two castings of the spell by the same caster. If all the conditions are met, caster can summon a dragon.

Listing 3.33: Script which is an integral part of spell propagation system is a part of the complete spell definition (seen in full in listing 3.30)

```
(SCRIPT "{
        set @CanSummonDragon, 0;
        if(getcharid(1) == DragonSummonParty && DragonSummonParty) goto
            ↪ L_ChkTime;
        set DragonSummonParty, 0;
        end;
        L_ChkTime:
                if ((gettimetick(2)-DragonSummonTime) > 30) goto L_Cast;
                end;
        L_Cast:
                set @CanSummonDragon, 1;
                set DragonSummonTime, gettimetick(2);
                end;
}")
```

Listing 3.34: Part of Arch-Wizard script working towards spell propagation system

```
L_St6:
        ...
        killmonster "0-102.gat", "All";
        monster "0-102.gat", 113, 79, "Dragon", 1134, 1;
        set DragonSummonParty, getcharid(1);
        set DragonEggQuestState, 7;
        goto L_End;
```

Listing 3.35: Part of Desert Mana Seed script working towards spell propagation system

```
...
L_Propagate:
        mes "(a whisper fills the room)";
        mes "\"By your will, oh benevolent Master of Dragons, your dragon
            ↪ will answer calls from all those now present.";
        next;
        mes "Remember: Sharing is caring!\"";
        areatimer "0-102", 94, 17, 114, 37, 500, "#DragonSpell#102::OnSet";
        goto L_Close;
...
```

Listing 3.36: Special script referenced by Desert Mana Seed

```
0-102.gat,104,20,0|script|#DragonSpell#102|127
{
        end;
        OnSet:
                set DragonSummonParty, getcharid(1);
                specialeffect FX_MAGIC_BLUE_TELEPORT;
                end;
}
```

4. AI Player Implementation

In this chapter we describe the formalization of the planning system for AI players that we have developed and which will play TMW together with real players. The planning system is formalized in SWI Prolog.[1] Our agents receive quests from various NPC's, so our system, based on our formal description of quests and on agent's knowledge and/or communication with other agents, generates an action plan that will lead it to achieve the goal (if that is possible given the current state of knowledge and state of the agent).

The state of the system (the world) is described with a number of predicates. There are groups of predicates that describe:

1. Positions of objects on maps (agents, a variety of items, NPCs, mobs, entrances and exits from rooms and maps, etc)
   - Positions of objects are stored in the predicate `location(o,m,x,y)` which has four parameters: *o* (name of the object), *m* (map) and *x* and *y* (coordinates of the object on map *m*). In this way we take care of agent's passages from one map to another. An agent gets knowledge about a map after he steps on it;
   - Instances of predicate *location* contain all the objects on the maps and from there we know which fields are free and which the agents can move towards;
2. Attributes of agents (level, weapons, clothes, shield, various collected items etc);
3. Attributes of monsters (weapons, shield);
4. Attributes of weapon (power);
5. Attributes of shields (hardness) and others.

Quests given out by NPCs can include simple missions like collecting items or talking to several NPCs, but also more complicated things like solving puzzles or winning a boss fight. Some quests have requirements like having completed another quest before or being at a certain level or above (the level can be either required or just recommended). Also, some quests have costs like items or money. Every quest will reward the agent with something. The rewards can be experience points (EXP), money, items, equipment, daily points, boss points, skills, magic spells or something else. Note that EXP rewards will grant the total amount, meaning they don't get cut off when reaching a

[1] See `http://www.swi-prolog.org` for details.

new level, so an agent might also raise more than a single level while completing a quest. Besides characteristics listed before, each quest has: a starting location, redoables and prerequisites.

As we will see, these characteristics will be of special importance in our formalization of quests. In this paper the quest Vincent from the set of the so-called Newbie Quests will serve as an example. All quests from the set Newbie Quests must be performed in a precisely defined order, and the quest Vincent is just one of them.

The starting location of quest Vincent is in Tulimshar (the only town of Tonori Continent) and given out by NPC Vincent. Required level for performing this quest is 1 but the quest is recommended for level 20. This quest is not redoable, thus an agent can perform it just once. Prerequisites of this quest are several other Newbie Quests. Once it is completed, the agent receives the reward of 1000 GPs, but performing of this quest will cost the agent 10 items (Bug Legs in this case which already have been or have to be collected).

To perform this quest the agent has to follow these instructions: *From the southern exit of Tulimshar, head east along the road to the end. When the road ends going east, head north. You will find Vincent on your left, by the well. Talk to Vincent, and he will ask you to collect 10 Bug Legs for his action figure. Go and collect 10 Bug Legs. Talk to Vincent one last time and give him the 10 Bug Legs you collected to complete the quest.* Additional important information for this quest is that Bug Legs are dropped in certain percentages by some mobs, for example, Maggot (4%), Cave Maggot (4%), Scorpion (7%), etc.

Below we show a detailed formalization of quest Vincent in SWI Prolog. All quests can be decomposed to basic actions. Therefore, we will (in addition to formalization of the quest) show the formalization of certain actions which are used in quest Vincent.

So, formalization of quest Vincent is mainly a set of basic actions lined up in required order and looks as follows:

```
do_quest(NPC,A,vincent) :-
    location(vincent,M,XVT,YVT),
    level(A,La), La>=1,
    done_quest(A,bernard),
    done_quest(A,mikhail),
    done_quest(A,sarah),
    waiting_quest(NPC,A,vincent),
    \+ done_quest(A,vincent),
    money(A,Ma),
    bug_legs(A,BL),
    retract(money(A,Ma)), NewMa is Ma+1000, assert(money(A,NewMa)),
    walk_to_location(A,M,XVT,YVT),
    assert(plan(talk(A,vincent,'Hi Vincent!'))),
    assert(plan(talk(vincent,A,'Give me 10 Bug Legs for my action figure!')
        ↪ )),
    collect_bug_legs(A,10),
    walk_to_location(A,M,XVT,YVT),
    assert(plan(talk(A,vincent,'I collected 10 Bug Legs. I give it to you
        ↪ !'))),
    ((BL>10 -> NewBL is BL-10,assert(bug_legs(A,NewBL)));assert(bug_legs(A
        ↪ ,0))),
    retract(waiting_quest(NPC,A,vincent))
    ->
    assert(done_quest(A,vincent));
    assert(failed_quest(NPC,A,vincent)).
```

The specification of action `walk_to_location` which means "Agent A on the map M walk on the location (X,Y)" called by `do_quest(NPC,A,vincent)` is a recursive rule and is implemented as follows:

```
walk_to_location(A,M,X,Y) :-
     location(A,M,Xa,Ya),
    (Xa < X, NewXa is Xa+1 ->
         retract(location(A,M,Xa,Ya)),
         assert(location(A,M,NewXa,Ya)),
         assert(move(A,M,Xa,Ya,M,NewXa,Ya)),
         assert(plan(move(A,M,Xa,Ya,M,NewXa,Ya))),
         walk_to_location(A,M,X,Y);
     Xa > X, NewXa is Xa-1 ->
         retract(location(A,M,Xa,Ya)),
         assert(location(A,M,NewXa,Ya)),
         assert(move(A,M,Xa,Ya,M,NewXa,Ya)),
         assert(plan(move(A,M,Xa,Ya,M,NewXa,Ya))),
         walk_to_location(A,M,X,Y);
     Ya < Y, NewYa is Ya+1 ->
         retract(location(A,M,Xa,Ya)),
         assert(location(A,M,Xa,NewYa)),
         assert(move(A,M,Xa,Ya,Xa,NewYa)),
         assert(plan(move(A,M,Xa,Ya,M,Xa,NewYa))),
         walk_to_location(A,M,X,Y);
     Ya > Y, NewYa is Ya-1 ->
         retract(location(A,M,Xa,Ya)),
         assert(location(A,M,Xa,NewYa)),
         assert(move(A,M,Xa,Ya,M,Xa,NewYa)),
         assert(plan(move(A,M,Xa,Ya,M,Xa,NewYa))),
         walk_to_location(A,M,X,Y);
      Xa == X, Ya == Y -> true).
```

As we can see, apart from the recursive action `walk_to_location`, from quest Vincent the recursive action `collect_bug_legs(A,N)` by which the agent collects N Bug Legs (if the agent already has some Bug Legs then he will collect just as much as he needs to N) is called as well. The formalization of this action is as follows:

```
collect_bug_legs(A,N) :-
    bug_legs(A,BL),
    (BL<N ->
    location(bl,M,Xbl,Ybl),
    walk_to_location(A,M,Xbl,Ybl),
    assert(plan(talk(A,A,'Here is one Bug Leg :)'))),
    retract(location(bl,M,Xbl,Ybl)),
    retract(bug_legs(A,BL)),
    NewBL is BL+1, assert(bug_legs(A,NewBL)),
    collect_bug_legs(A,N));
    retract(bug_legs(A,BL)),true.
```

Again we can see that the action `collect_bug_legs` calls the `walk_to_location` action.

In general, once we start an instance of command `do_quest(npc,agent,quest)` its re-

sult is a Prolog list of instances of basic actions. Specifically, after running the command `do_quest(npc,a,vincent)` which tells that agent *a* got quest instructions of an NPC, a list of actions which ought to solve this quest are generated.

The agent is going to solve a quest if he has met all preconditions for that quest (for example the required level). If it gets the quest for which it has no conditions to resolve, then it remains on its mind, and it is going to solve something else. Therefore, quests that agents get, but have not started to resolve are recorded in our system as instances of predicate `waiting_quest(npc,agent,quest)`. In addition, there is a possibility of failure during solving the quest. A failure to resolve the quest is recorder by an instance of the `failed_quest(npc,agent,quest)` predicate.

We have implemented the algorithm to select a quest which the agent will try to solve first. This algorithm is based on the so-called significance of the quest for the agent. Significance of the quest for an agent is given by instance of predicate `quest_sign(agent,quest,sign)`. In order that our system knows which quest it should firstly begin to solve, we have introduced a `quest_no(agent,quest,no)` predicate, where the number `no` indicates the ordinal number (priority) of the quest. Priority is calculated based on the above properties of quests and is recalculated when necessary. Thus, quests belonging to an agent *A* are sorted by priority, and sorting is conducted according to the following recursive rule:

```
sort_quests(A) :-
    waiting_quest(NPC1,A,Q1),
    waiting_quest(NPC2,A,Q2),
    quest_sign(A,Q1,QS1),
    quest_sign(A,Q2,QS2),
    quest_no(NPC1,A,Q1,QN1),
    quest_no(NPC2,A,Q2,QN2),
    QS1>QS2,
    QN1>QN2
    ->
    retract(quest_no(NPC1,A,Q1,QN1)),
    retract(quest_no(NPC2,A,Q2,QN2)),
    assert(quest_no(NPC1,A,Q1,QN2)),
    assert(quest_no(NPC2,A,Q2,QN1)),
    sort_quests(A);
    true.
```

If the agent has no waiting quests or cannot solve any waiting quest for some reason, then its goal is to explore the map hoping to find a new NPC that will give him a new quest. For this purpose we have created the rule by which the agent walks on a random location on the map it is currently located on:

```
random_walk(A) :-
    location(A,M,Xa,Ya),
    randomX(M,RLX),
    randomY(M,RLY),
    walk_to_location(A,M,RLX,RLY).
```

whereby the actions `randomX(M,RLX)` and `randomY(M,RLY)` are implemented to yield random coordinates (RLX, RLY) on map M. Using the mentioned rules we achieve that we must not "tell" the agent which quest should be solved or what to do at all. It knows it according to the following Prolog rule:

```
do(A) :-
```

```
    sort_quests(A),
    quest_no(NPC,A,Q,1)
    ->
    do_quest(NPC,A,Q);
    random_walk(A).
```

The rule `do(A)` says that agent *A* will first sort the quests according to importance. Then, if there is a quest on waiting with the highest priority (priority 1) the agent should go to solve this quest. Otherwise the agent should perform a random walk.

The overall control loop of our agent is formalized mostly in accordance with the overall control structure of a practical reasoning agent given in [6]. Namely, in our system we are focused on reasoning directed towards actions and not only on theoretical reasoning which affects only agent's beliefs about the world. In our formalization the agent continuously:

1. Observes the environment and changes in that environment. Changes can occur in:
   - location of certain objects, characters, monsters, items, etc;
   - attributes of agents (level, arm, clothes, the number of certain items that they have collected, skills, etc);
   - attributes of quests (quest is waiting to be performed, priority, done or failed, etc);
2. Deliberates to decide what intention to achieve. Deliberation is done by firstly determining the available options and then by filtering. Options of an agent are quests (or eventually a random walk around the map until it finds a new NPC that will give it a new quest). Thus, deliberation is actually deciding which quest will the agent try to perform next. In order to select the most appealing between competing options, an agent uses filtering. In our case, the filtering is actually the sorting of quests in order of priority.
3. Creating a plan to achieve the selected intention, that is to solve the selected quest. Namely, when the agent in step 2 chooses a quest to perform, it is necessary to determine the way in which the quest can be performed.
4. Executes the plan.

As in other similar planning systems, an important question occurs: How often (and when) should an agent re-observe the environment and reconsider its intentions? Given that in TMW these processes are not computationally cheap, it is necessary to reconsider as rarely as possible. On the other hand, during this time the environment changes, possibly rendering its newly formed intentions irrelevant. It is obvious that we have to find an optimum. Of course, the optimum is contained in this kind of system behavior: the agent has to reconsider his intentions if and only if afterwards the intentions will have to be changed. Our plan is to achieve this optimum by observing the behavior of our agents once our TMW system will be so developed that it can be tested.

4.1 Automated TMW Agent

In order to be able to use an AI player as developed in the previous sections, a connection between the Prolog system and the actual Mana World game server had to be established. To do so we implemented a lower-level interface in Python[2] that acts similar as the official ManaPlus client[3]. The lower-level interface implements the protocol and most important packages given at the official Mana World wiki[4]. It allows one to programatically login to the server, change directions, move to a particular destination, pickup items, communicate with NPCs and other players. The following listing provides an example of a session to the server:

[2] See `http://www.python.org`

[3] See `http://manaplus.org/`

[4] See `https://wiki.themanaworld.org/index.php/Dev:TmwAthena_Packets`

```
c = Connection( SERVER, PORT, USERNAME, PASSWORD, CHARACTER )
c.login()
time.sleep( 1 )
c.pb.go()
while not c.pb.hasNew():
    time.sleep( 0.1 )

c.setDirection( 2 )
time.sleep( 1 )
c.whisper( 'username', 'message' )
time.sleep( 1 )
c.sit()
time.sleep( 1.2 )
c.stand()
time.sleep( 1 )
c.setDestination( 90, 90, 1 )
time.sleep( 0.2 )
c.pickUp( 754 )
m1 = c.pb.getNew( ( 'SMSG_NPC_MESSAGE', 'SMSG_PLAYER_CHAT' ) )
m2 = c.pb.getNew( ( 'SMSG_NPC_MESSAGE', 'SMSG_PLAYER_WARP' ) )
if m2.type != 'SMSG_PLAYER_WARP':
    m3 = c.pb.getNew( 'SMSG_NPC_CHOICE' )
    c.NPCChoose( m1.NPCid, 1 )
    time.sleep( 1 )
    c.NPCNextDialog( m1.NPCid )
c.quit()
```

In this example session, the interface is firstly connected to the server by instancing a Connection object and calling the login method. The pb attribute of the connection is the package buffer which accepts all incoming messages from the server. It's method go starts listening on the given connection. The method hasNew returns True if there have been new messages. The setDirection method of the Connection object sets the direction of the character on the map. Likewise the whisper method sends a private message to a given player. The methods sit and stand let the character sit down or stand up. The setDestination method sets the destination of the character on the current map and lets the caracter move to the position. The pickup method lets the character pick-up a given item with given item ID. The package buffer's method getNew awaits a given packet based on its type. It blocks the execution until a package is received. The NPCChoose and NPCNextDialog methods of the Connection object lets the character interact with NPCs (choosing an option and going to the next NPC dialog).

5. Data Collection & Logging

5.1 Custom TMW Client Implementation

In order to allow detailed behaviour logging of players, besides the existing server logs, a custom TMW client has been implemented that acts as a wrapper around the official ManaWorld client. The client allows automated registration, login and starting of the embedded ManaWorld client. In the back, the client connects to a custom implemented server backend and sends necessary logging data including various forms of communication and in-game social network data. Figure 5.1 shows an example login screen from the implemented client.

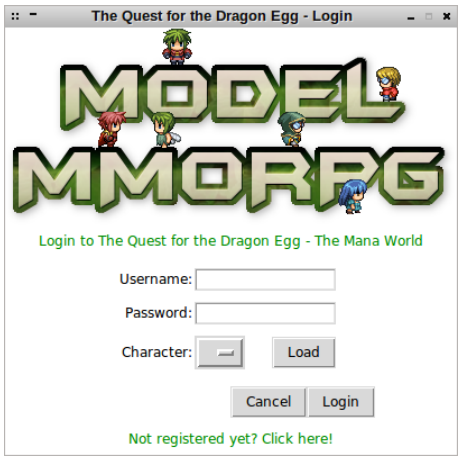


Figure 5.1: Custom TMW client's login interface

The custom client has been implemented in Python and packaged for Windows, Linux and MacOS compatibility, and is available for download (with installation instructions) on the social website of the project.[1].

[1] See `http://dragon.foi.hr/?page_id=1667`

5.2 Server Backend

The server backend consists of a custom web application developed in Python and a database storage developed in PostgreSQL[2]. The initial version of the database storage was developed in MongoDB[3] but due to a cumbersome implementation and problems with logging, a decision was made to switch to PostgreSQL, which also allows storage of JSON format.

The custom backend application does not only store data from clients, but also provides login and security facilities, statistical user data for the social website as well as administration.

Here should be mentioned that a great part of the client as well as the backend has been implemented by an exceptional student of the Faculty of organization and Informatics Lovro Predovan.

[2] See `http://www.postgresql.org/`
[3] See `https://www.mongodb.org/`

6. Infrastructure

6.1 Server

Besides software that has been developed for the particular test-bed, a server machine (HP ProLiant ML350 Gen9, E5-2609v3, P440ar/2GB FBWC, 4x1GbE, 1x16GB, 8-SFF HP, 2x300GB, DVD-RW, 1x500W HP NVIDIA Quadro K4200 GPU Module 1 KOM HP ML350 Gen9 Graphic Card Support Kit) has been acquired to host the game server, the data logging backend as well as a social website. The server has machine been properly marked as requested by the Croatian Science Foundation as seen on figure 6.1.

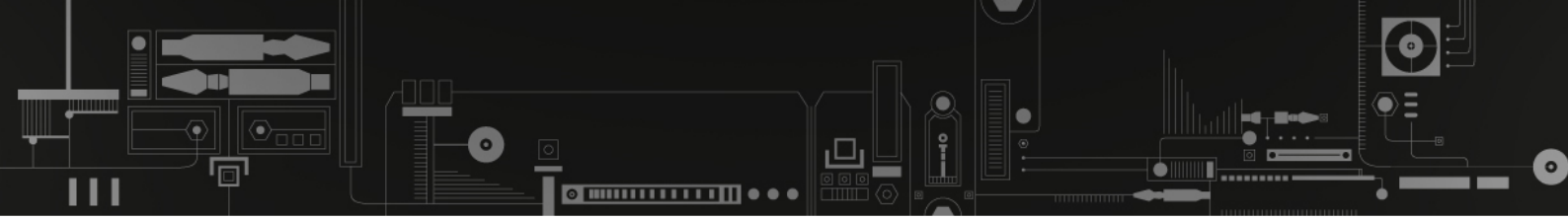
Figure 6.1: HP ProLiant ML350 Gen9 acquired for the test-bed

6.2 Social Website

In order to attract as many players as possible, a social website was implemented based around the developed dragon egg quest (see figure 6.2).

Figure 6.2: Social Website developed for the test-bed quest

Besides information about the project it features a user forum, chat, links to social networking profiles of the project (Facebook, Twitter, Google+ and YouTube) and the official Mana World Wiki, statistical information about current players and parties as well as a download section with the custom project game client.



hrzz
Hrvatska zaklada za znanost

foi
SVEUČILIŠTE U ZAGREBU
FAKULTET
ORGANIZACIJE I
INFORMATIKE
VARAŽDIN

lab

Bibliography

[1] Global Collect. *THE GLOBAL MMO GAMES MARKET: Payments, Intelligence and Trends*. 2014. URL: `http://www.globalcollect.com/the-global-mmo-games-market` (cited on page 2).

[2] NewZoo. *The Global MMO Market; Sizing and Seizing Opportunities*. 2012. URL: `http://www.newzoo.com/infographics/the-global-mmo-market-sizing-and-seizing-opportunities/` (cited on page 2).

[3] Wikipedia Contributors. *Massively multiplayer online game — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-February-2015]. 2015. URL: `http://en.wikipedia.org/w/index.php?title=Massively_multiplayer_online_game&oldid=646153954` (cited on page 2).

[4] Wikipedia Contributors. *Massively multiplayer online role-playing game — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-February-2015]. 2015. URL: `http://en.wikipedia.org/w/index.php?title=Massively_multiplayer_online_role-playing_game&oldid=642752380` (cited on page 2).

[5] Wikipedia Contributors. *Role-playing video game — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-February-2015]. 2015. URL: `http://en.wikipedia.org/w/index.php?title=Role-playing_video_game&oldid=642919862` (cited on page 2).

[6] Michael Wooldridge. *An introduction to multiagent systems*. Wiley. com, 2008 (cited on page 41).