Jul 2005
DTU

Role playing game development

Created by:
Robert Marzeta
Special course from
Game Development
Under supervision of
Michael Rose

1. Introduction

Role playing, in essence, is putting yourself in someone else's place, and reacting to situations the way you think that person would react. Sometimes role playing is compared to theater, but it's not like you'd see in a regular play on a stage. Your role has not been predefined, and your actions not predetermined by a script. Role playing allows you to behave freely, being able to change yourself and surrounding world allowing you to identify with that imaginary world.

From the moment I thought about completing this project, I knew that I cannot fulfill every aspect I have in my head. Making a good RPG (Role PlayinG) game requires years of work and combined efforts of graphic and sound artists, coders and the most important the director which has an inspiring idea - as this is what matters the most. My idea was to create a tool to allow non-programming directors realize their inspiration.

I have made an application which not only creates a world but also allows changing it in a big way. The universality was in fact the most challenging task as there are always difficulties with creating ‘general aimed’ application which fits all.

2. Vocabulary

For a start, let’s introduce some definitions used in the further description.

A person who created the application (wrote it in Java) is called: ‘**Programmer**’.

A person who is able to create his own type of game (through configure files) is called **GM** (Game Master).

A configured application which can be played is called: “**Game**”.

Finally, a person who is going to play a Game is called: “**Player**”.

3. Guidelines

The first plan was to make a typical RPG game, with predefined plot, graphics, and system. After examining enormous variety of other RPG games it was hard to find an original idea. Moreover, it is impossible to compete with commercial applications. The solution was to allow others create rather than just play. The new task appeared much more difficult, but yet - more challenging. About half time spent for the project was consumed by thinking what to implement and in the and, following guidelines have been followed.

3.1 Universality

The main idea was to allow future ‘directors’ to build a scenario according to their vision and try (or give somebody else to try) if it works the way they imagined. Making a scenario includes creating own graphics, maps, monsters, items and be able to control actions on the imaginary world.

As it is impossible to predict every unique idea that might come to other heads, an effort has been made to introduce as much as possible. Still, there has to be a defined, unfortunately inflexible, system backbone which organizes the very base of actions. The key of effective usage is to examine what the application offers and find a compromise. Almost any inconvenience can be handled by proper replacement.

As an example– instead of killing a player with a meteorite (which is NOT supported) that falls unexpectedly from the sky – create a deadly invisible monster called ‘meteorite’ that waits in certain location to attack the Player.

Suggestion - whenever the application does not completely allow doing something, the idea can be displayed in a message window, allowing the unaware player to identify with a situation.

It is also only a possibility that the world should resemble imaginary, fantasy creation with dragons, elves and magic potions. There is no problem in designing a today situation, with city, streets, citizens, baseball bats and villains waiting in dark alleys.

3.2 Simplicity

Another aim was to make game configuration as simple as possible. This is the reason of using simple XML, with no misleading attributes (every value is a separate element) and without fancy scripting involved. It has to be easy to understand how to set things right and similar methods should be used through the entire configuration.

The game should also be resistant to wrong input data. Though, there are no mechanisms checking the proper XML syntax, so it lies on GM head to do it right. The only advice is to use XML editors (like oXygen or Altova's XMLSPY) which check for such errors.

The application uses a default values whenever it is impossible to get the specified one. Mostly, the default value equals 0 (for value parameters) and empty string (for string parameters). Wrongly described image will result in no displaying the object, though an object might exist in the game making all normal actions (ex. creature). It is GM’s task to spot and correct any undesirable effects of wrongly configured application (like invisible monsters, of weightless items).

As for Player comfort, application possibilities are clearly described in the instruction and every action is intuitive. It has to be also similar to other popular RPG games, offering level-ups, saving or shops. Whole game is based on a common windows kind of display. There are also lots of hint messages appearing when holding a mouse cursor above some labels.

3.3 Diversity

There is nothing more boring from a game that can be predicted. One of the aims was to design a system that could surprise even the GM. It has been accomplished by introducing probability based values.

Such parameters use a special way to describe values by simulating dice throwing (original and most commonly used RPG system). It means that: ’3d4+2’ weird expression commands to sum results of throwing three times a dice that has 4 sides (possible values are: 1, 2, 3, 4) and add 2 generating in the and a value from 5 to14.

The additional value is not necessary to exist or can be negative: ‘1d3 – 4’ (result from -3 to -1). The algorithm handles unnecessary spaces expression. Constant values must take a form constructed from two factors, while the first does not contain‘d’ symbol, for example: ‘0 – 3’: results always in ‘-3’ or ‘0 +1’: results in ‘-1’. The first factor (even though if it is 0) is necessary for proper execution of the algorithm.

Thing to consider is that some value are generated many times basing on the given string (for example counting a weapon damage every time it is being used). Other are generated only one, when an object is created (like the weight or price or the stats modification).

4. Game logic (how it works)

All classes are packed in rpg.jar file allowing easy execution from a java runtime environment

```
java –jar rpg.jar
```

Without going into details- description how the game logic reacts and what actions it supports.

4.1 Initialization

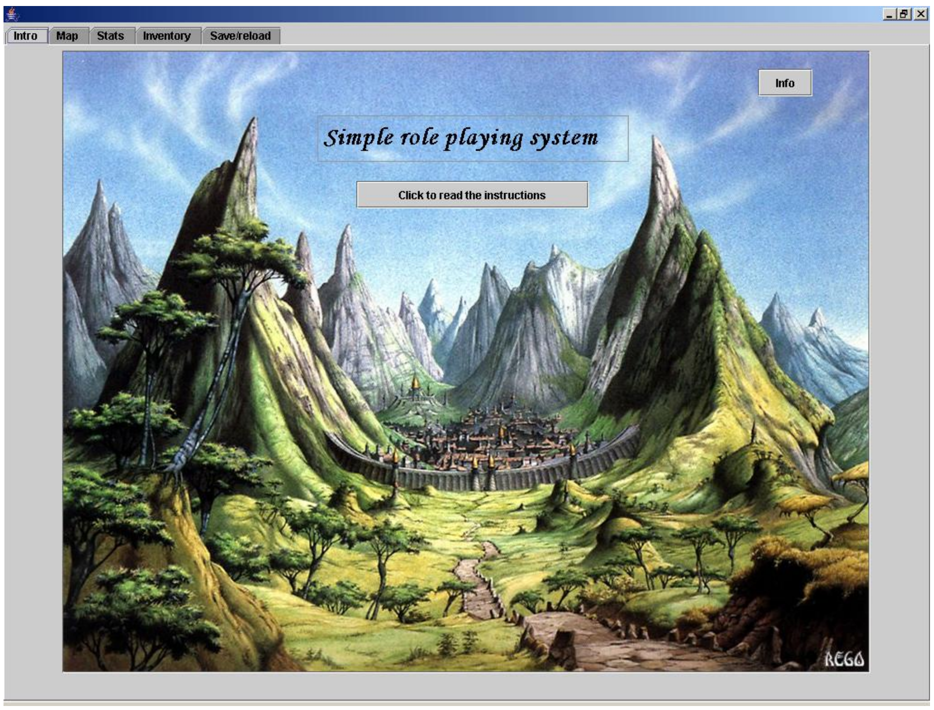
Firstly, the configuration files are loaded and created objects are stored in memory. Player is always described by the very first creature that appears in the "creatures.xml" file. Player is unique in the game and will appear only once; other creatures can be used many times.

4.2 Screen and views

Screen is divided into views which allow controlling the Game.

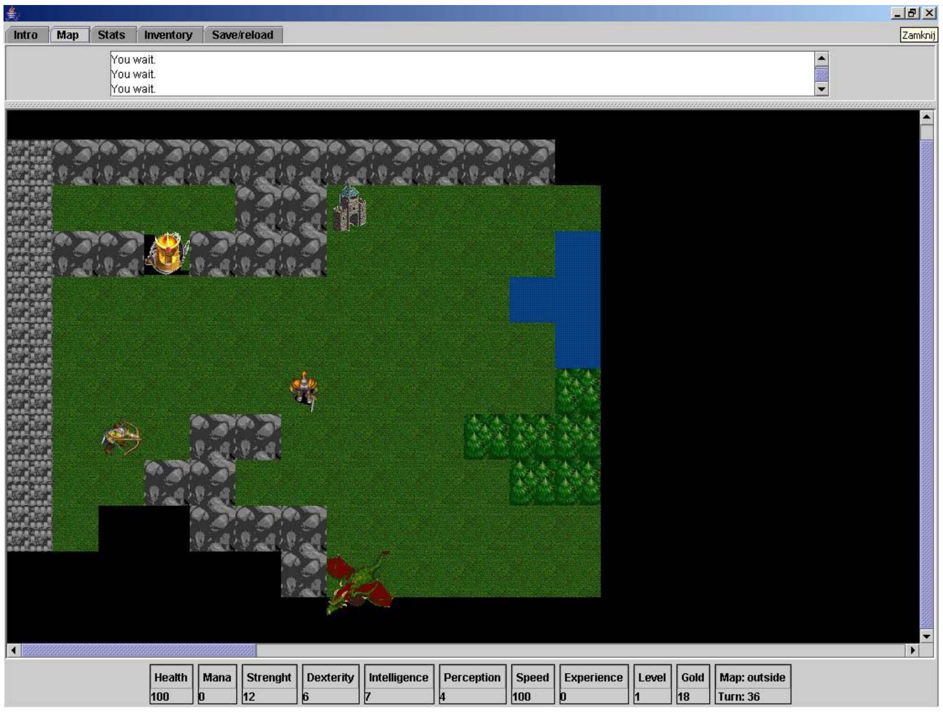
Intro view

First window shows introduction picture and some description of game controls. There is also a button showing formal information.



Map view

By clicking at a map tab (at the top of the window) map appears:



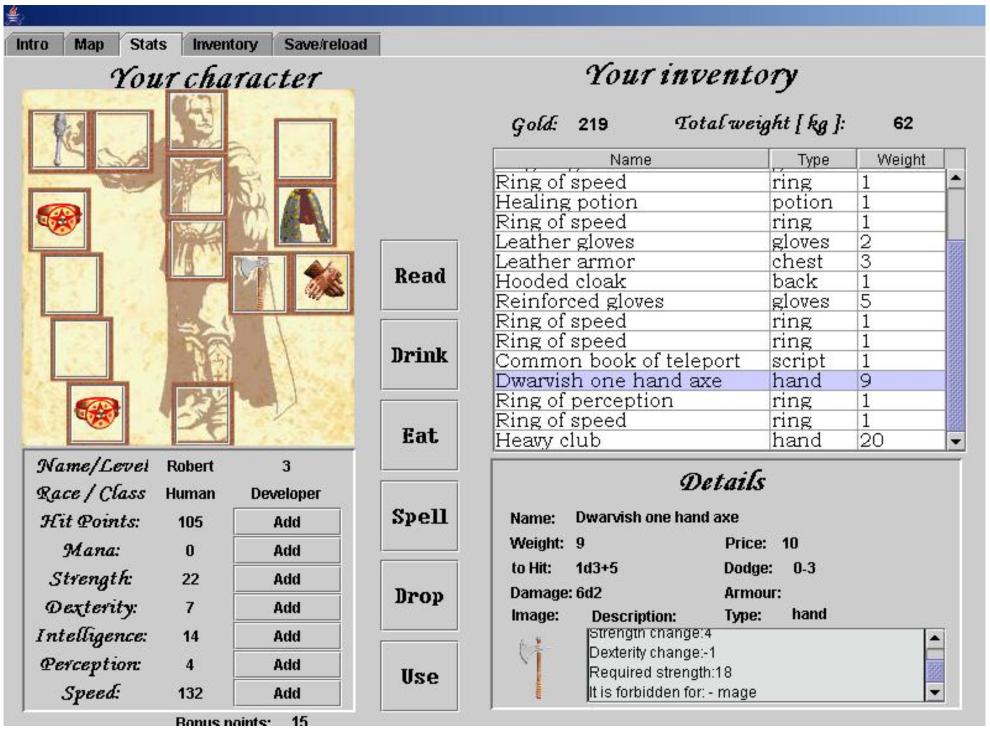
```
You wait.
You wait.
You wait.
```

Health	Mana	Strenght	Dexterity	Intelligence	Perception	Speed	Experience	Level	Gold	Map: outside
100	0	12	6	7	4	100	0	1	18	Turn: 36

Map actions will be explained thoroughly below.

Character view

Next tabs display character window which manages Player's statistics and controls possessed item (equipping, using, and dropping).



Name/Level	Robert	3
Race / Class	Human	Developer
Hit Points:	105	Add
Mana:	0	Add
Strength:	22	Add
Dexterity:	7	Add
Intelligence:	14	Add
Perception:	4	Add
Speed:	132	Add

Bonus points: 15

Gold: 219 Total weight [kg]: 62

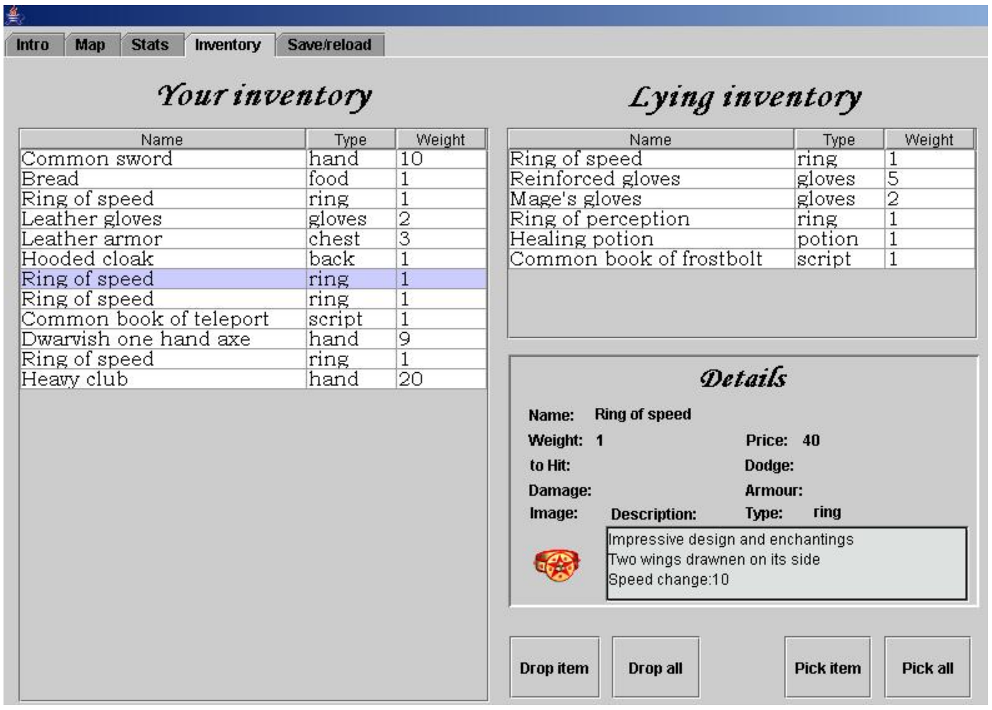
Name	Type	Weight
Ring of speed	ring	1
Healing potion	potion	1
Ring of speed	ring	1
Leather gloves	gloves	2
Leather armor	chest	3
Hooded cloak	back	1
Reinforced gloves	gloves	5
Ring of speed	ring	1
Ring of speed	ring	1
Common book of teleport	script	1
Dwarvish one hand axe	hand	9
Ring of perception	ring	1
Ring of speed	ring	1
Heavy club	hand	20

Name: Dwarvish one hand axe
Weight: 9 Price: 10
to Hit: 1d3+5 Dodge: 0-3
Damage: 6d2 Armour:
Image: Description: Type: hand

```
Strength change:4
Dexterity change:-1
Required strength:18
It is forbidden for: - mage
```

Inventory view

Inventory window is used to manage item picking and dropping.



Your inventory

Name	Type	Weight
Common sword	hand	10
Bread	food	1
Ring of speed	ring	1
Leather gloves	gloves	2
Leather armor	chest	3
Hooded cloak	back	1
Ring of speed	ring	1
Ring of speed	ring	1
Common book of teleport	script	1
Dwarvish one hand axe	hand	9
Ring of speed	ring	1
Heavy club	hand	20

Lying inventory

Name	Type	Weight
Ring of speed	ring	1
Reinforced gloves	gloves	5
Mage's gloves	gloves	2
Ring of perception	ring	1
Healing potion	potion	1
Common book of frostbolt	script	1

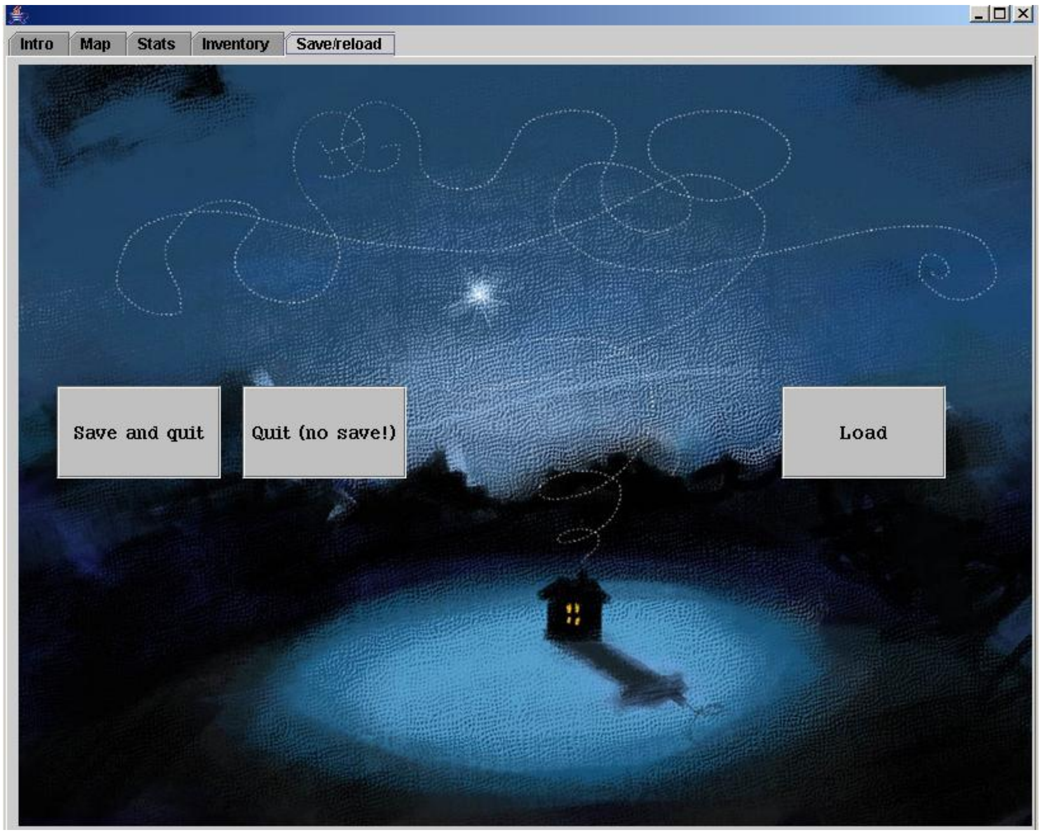
Details

Name: Ring of speed
Weight: 1 Price: 40
to Hit: Dodge:
Damage: Armour:
Image: Description: Type: ring

Impressive design and enchantings
Two wings drawnen on its side
Speed change:10

Quit/Save view

Allows saving and loading Player’s progress.



4.3 Keys

Keyboard actions are focused on the map canvas. In case focus moves to another element, just click on the map again. Main keys lie on the numeric keypad and so:

- ‘1’ moves down-left
- ‘2’ moves down
- ‘3’ moves down-right
- ‘4’ moves left
- ‘5’ waits a turn
- ‘6’ moves right
- ‘7’ moves up-left
- ‘8’ moves up
- ‘9’ moves up-right

Additionally: ‘>’ key is used to enter passages (field with a name “teleport”)
‘p’ picks the first item beneath player

4.4 Turns

Game time is divided into turns. Every turn starts with user action, following by all map creatures’ movement and finally checks and executes events. Amount of actions that other creatures can make depends of their speed (compared with players speed). It means that faster creatures can react more times in response to player action (ex. attack). They also move before slower ones. The speed mechanism is based on collecting turn points and executing a new action whenever turn points amount is higher than players speed.

4.5 Movement

It is possible to move in all the directions if:

- they are not occupied by another creature
- it is possible to walk there (for example not a wall or river)
- they are not out of map boundaries

Creatures move in random directions apart from those that are hostile and can sense Player presence. Those - try to follow (catch) Player and attack.

Some fields provide special possibility to move between maps. Those fields are called “teleport” but have the same role as stairs, gates, etc. Entering such a field is possible by pressing ‘>’ key. Creatures can not use teleports.

4.6 Visibility

Player perception has two aspects. On one hand, it influences vision, on the other – hearing. The idea was that hearing is more sensitive, but affects only movement (creatures).
Hearing detects all creatures at a distance lower than player’s perception. Distance is simply number of fields between to places.
Vision is more sophisticated. The algorithm uses recurrence to find which places are visible to the player. Checking begins with the player and checks all the surroundings. Then all the surroundings of the surroundings are checked, etc…, until player’s perception range ends. Obstacles that have visibility equal 0 block the line of sight but sometimes it is possible to see what is behind them through neighboring (visible) places.
For simplicity, other creatures use only hearing to seek and follow the player (if he gets in their perception range).
Note: In the next step, this was to be corrected as well as creature and player hiding/stealth feature. If place’s visibility is lower than 100 (%) a chance should be possible not to see creature standing there. This could also be improved by creature’s abilities (like stealth) or states (invisibility, size).
Hint: it is possible to create fields that are crossable, but have no visibility. They could also look like wall (image) what gives a nice way to build illusion walls.

4.7 Attack

Attack is triggered by the same keys as movement. They select the direction Player can move and attack (if the place is occupied by another creature). When a player tries to attack a peaceful being, a warning appears to confirm the choice. The effects of such foul behavior have not been developed yet. An attacked creature becomes of course hostile. A hostile creature will attack Player in its next turn when it stands right next to it.
All computed fighting factors involve checking and summing referring equipped items features. Some fighting professions get a damage bonus:

- warrior: +10
- thief: +5

Some strong races also get such bonuses also:

- dragon: +10
- orc: +5

Hard skinned races (like dragons) get also an armor bonus +10.

Attack can be divided in two phases:

- The attacker checks its ‘toHit’ ability and compares with defenders ‘dodge’ ability to indicate whether the hit reached the target (toHit > dodge). The ability is summed up from: ‘toHit’ and dexterity. By default – creatures have only ‘toHit’ value greater than 0. They do not use items they carry (*it was also next to correct*). All calculations involve possibility by throwing the dices.
- if ‘toHit’ is greater than ‘dodge’, the attacker counts its ‘damage’ (adding strength) ability to count how much damage it causes. Sefenders ‘defense’ ability, which absorbs part of the blow, subtracts part (or whole) of the damage. As an indication, a creature that has been successfully hit will have a blood stain on it for one turn.

4.8 Dying

Death is a natural effect of every fight. A creature (also Player!) dies when its health drops below (or equal) 0. When creature dies, (some times) it drops items and amount of experience is being added to Players experience amount. Experience is needed for getting to higher levels and improving Player stats. Some races (like humans) get an experience bonus +10% as they are able to learn faster than other – long lived races (ex. elves). When Player dies, game ends. There are different methods how to avoid dying but only GM should know them from a start. For Player – only learning possibility is to know the game.

4.9 Waiting

Instead of moving or attacking – player can simply skip a round. It can become useful when there is a peaceful creature standing in a narrow corridor, blocking it completely. Instead of killing it, it is possible to wait until it leaves by itself.

4.10 Picking items

Items appear randomly on the map and are being dropped by killed creatures. Player can pick them up after moving at their location and pressing ‘p’ (like Pick). Instead of picking one item at a time, it is possible to switch to inventory view and pick all of them or only the chosen one (after viewing it). Player has a limited capacity (load) of carried items which equals strength multiplied by 10 and cannot pick any new items after reaching this maximum burden.

4.11 Equipping

Only equipped items can influence player’s attributes. Some of them can be really powerful and usually have strict minimal requirements (about statistics). This is the reason to be wise in choosing items and adjusting statistics. Certain types of items fit only to certain places on the body.

4.12 Using, drinking, eating, reading

There are items that can be used in another way to increase Player stats. There share one common feature – they disappear after beeing used once. Scrolls (books, manuscripts) can be read, food – eaten, potions – drunk and finally tools – used. All of them can help Player in his journey.

4.13 Talking

After clicking an item that is next to player, it is possible to talk with it. It pays off to do it as sometimes such information reveals valuable item locations or secret passages.

4.14 Saving, loading, quitting

Most self respected games take care about player’s need to function in normal life and support saving and loading. The application contains Quit View allowing choosing between:

- quitting without saving
- quitting after saving
- loading saved game

5. Configuration files

There are five XML files, lying in the directory: “conf”, which store all game data. Each of them contains clear and simple database resembling structure. All datafields are XML elements. The application does not check proper XML syntax, so GM has to handle that by his own. The simplest way is to use XML editors (look - point 2.3).

5.1 Items.xml

It stores all available items. The root node name is: “<items>”. Every child item element node has to be named: “item” and contain specific information about an item:

```
<items>
        <item>
                <name> … </name>
                .
                .
        </item>
.
.
<items>
```

Item parameters

Possible item elements can be dived into ‘String’ and ‘Dice’ defined parameters.

String type parameters:

- name – simply names the item
- type – it has nothing to do with items name and features, but only with the place they can be equipped or way they can be used. Only equipped items influence player stats.

  Possible types are:

  - hand - primarily held and used in hands (weapon, shield)
  - chest - armors, robes, tunics, dresses, shirts, etc
  - ring - all kinds of rings
  - gloves- on palm
  - shoes - on feet
  - head - helmets, caps, hoods, etc

- back - cloaks, capes
- belt - different belts
- shoulders – shoulder armor like bracers
- ranged – ranged weapons: bows, crossbows, slings, darts, shurikens
- potion- all kinds of potions (possible drinking)
- book - all kinds of books, scrolls, manuals (possible reading)
- food - can be eaten (bread, meat, candy)
- scroll - can be read (books, manuscripts, scrolls)
- tool - can be used (bandages)

➢ filename - path and name of items image

➢ description - information about an item

➢ material - describes the type of material the item is made from: metal, wood, glass, brass, cloth, stone, … . *At this point of game development is has no meaning.*

Value type parameters:

<u>Item features:</u>

➢ weight - how heavy item is (free choice of units) smallest step is 1

➢ price - items cost (free choice of units) smallest step is 1

<u>Fighting usage (dice expression):</u>

➢ dodge -describes ability to avoid enemies hit

➢ armor -describes ability to weaken enemies hit

➢ toHit -describes ability to hit an enemy hit

➢ damage -describes ability to cause damage to enemy hit

<u>Following stats modifiers temporarily change player statistics (dice expression):</u>

➢ strength

➢ dexterity

➢ intelligence

➢ perception

➢ speed

➢ health

➢ mana

Describing minimal requirements to equip such item (dice expression).

- minStrength
- minDexterity
- minIntelligence
- minPerception
- minSpeed
- minLevel

Value parameters are described using dice expression which will make them randomized. Most of the random dice parameters (like weight, price, stats modifiers, requirements) stabilize (take a constant value) after item creation. Others are being counted with every item usage (fighting usage).

Scrolls, potions, food, tools disappear after using, giving their features to the player.
There are also unfinished entries in items description which indicate future plans:

- "forbiddenFor" node was supposed to forbid to equip according to players race/profession

5.2 Creatures.xml

Parent root element is named: "creatures" and every creature element is named: "creature". The first creature describes player and will not appear twice in the game.
Possible item elements can be dived into 'String' and 'Dice' defined parameters.

Creature parameters

String type parameters:

- name - the name of a unique creature
- race - race
- profession - profession
- filenamePrefix- the path and beginning of creature image file. Every creature should have 8 different images for 8 different world directions: ex: ogre_left.gif, ogre_right.gif, ogre_up.gif, ogre_down.gif, ogre_upleft.gif, ogre_downleft.gif,

ogre_upright.gif, ogre_downright.gif. For simplicity, only the beginning (and path) is passed: “ogre”. Also – file named “ogre.gif” will be used in case any other file is corrupt so it is advisable to add it also.

➢ description	- creatures description
➢ type	- this feature is also not introduced now, but it classifies creatures to groups, like: beasts, humanoids, undeads, deamons. Different groups could be influenced in different ways.
➢ attitude	- describes the will to attack the player: “hostile” creatures always attack, “neutral” and “peaceful” do not. Default is “hostile”
➢ move	- describes creatures movement, only “stay” parameter forbids the creature to move. There are plans to introduce other states like: guard, patrol, follow.
➢ map	- specifies the map the creature appears, default is “all” (all maps)
➢ locationx	- specify the exact X coordinates location the creature appears
➢ locationy	- specify the exact Y coordinates location the creature appears
➢ dialogs	- adds possible questions and answers for the creature, every subchild contains a pair of sentences. Conversation starts when a near by creature is clicked (on the map).

Value type parameters (dice expression):

➢ experience	- amount of experience for killing the creature
➢ money	- amount of money a creature possesses
➢ speed	- influences how often the creature has its turn
➢ health	- amount of Hit Point creature has
➢ mana	- amount of mana points (not used)
➢ toHit	- chance to hit opponent
➢ damage	- damage the hit opponent takes
➢ dodge	- chance to avoid opponents hit
➢ armor	- amount of blocked damage

Values below describe creature features. Now – they influence only the player.

- strength
- dexterity
- intelligence

- perception

- Items - Declaring new subchildren of element 'items' will give a creature item from a start. Item can be selected by their name:

```
<item>
        <itemName>Common sword</itemName>
</item>
```

or by randomly specifying their type (or 'all' – for unspecified)

```
<item>
        <itemName>random</itemName>
        <class>all</class>
</items>
```

5.3 Events.xml

Events can be used to build a proper adventure atmosphere, climate and control the plot. They are triggered by turn number, map location or experience amount. The parent element is named: "events" and following, every event is an element called "event".

Event parameters

Parameters can be divided in to groups: conditions (which specify on what condition an event will occur) and actions (what effects an event will give).

Conditions:

- turn - event is executed only when present turn is equal or above the value
- experience- event is executed only when players experience is equal or above
- map - event is executed only when present map equals the value
- locationx - specifies X coordinated
- locationy - specifies Y coordinated
- repeat - describes if the event should be used again (same conditions) or should it be erased from the list of waiting events (for example: repeat = "yes", when there is a fountain that constantly regenerates health)

Actions:

- level - set players level

- text - window containing inputted information is displayed.

Those values are added to statistics. Using dice expressions, it is also possible to get negative values!

- health
- mana
- speed
- strength
- dexterity
- intelligence
- bonusStats - points that can be used to manually increase statisctics
- bonusSkills - points that can be used to manually increase skills

5.4 Skills.xml

This section is under development. It is hard to predict what might be possible for a skill and design a clear system to introduce it in the game. Some examples can be found in skills.xml file. As for now, skills contain only draft parameters:

Skill parameters:

- name - skill name
- desc - skill description
- valuePerPoint - how many ‘skill points’ are needed to increase skill by one. This would be useful to balance skills
- strength, dexterity, health, armor, etc. - modifications to stats the skill might bring (per one point)

5.5 Maps.xml

Maps.xml file contains information about available maps. Every map is being loaded from a text file and arranged according to XML properties. Parent node: “maps” contains different “map” elements. Then, every map child contains many field subchildren that describe possible field types.

Map parameters:

- name - map name
- filename - path and name if a file containing map data. The simple text file contains symbols of fields arranged in a map.
- description - map description
- creatureNumber - number of creatures appearing on a map
- respawn - number of turns all monsters will be reseted (not supported yet)
- itemNumber - number of items appearing on a map
- visibility - default field visibility
- mobility - default field mobility
- harmful - default field damage (not supported yet)
- field - describe specific field properties

Possible field elements:

- name - field name
- symbol - one character used as a symbol of a field
- filename - image displayed
- mobility - describes how easy it is to travel
- visibility - describes how easy it is to see
- fieldItemsNr - amount of appearing items

Following parameters apply to field that has a name: teleport which means that it is some kind of a passage (like stairs, cave entry, city gate).

- targetMap -specifies map name player goes to
- targetX -specifies X coordinates player goes to
- targetY -specifies Y coordinates player goes to

6. Graphics

One of the key aims was to make different graphics options possible for GM. Every picture is stored in GM chosen directory (in example configuration – “images”). Image format is a GIF file, 50 pixels width and 50 pixels height. It is possible to make and display bigger images (ex. for bigger monsters), but the head part of it (the one used to action calculations) will remain in the left-upper 50x50 square. The GIF (Graphics Interchange Format) has been chosen for some reasons:

- simple to create, modify and display (especially for the editor which could be added to the application)
- supports transparency so that many objects can lie on each other
- supports animation (which could also be introduces in the game)

Images are load into an ImageIcon object (javax.swing.ImageIcon library) and stored in every visual Game object (creatures, fields, items).
Note: graphics is being displayed in a certain order. First, the map is displayed, starting from up-left corner and moving through lines and columns. Then, all items are being displayed. Lastly, all creatures are being displayed (including Player). It is useful to know that monsters images will overlap items and items images will overlap ground images.

7. Technical info

The whole game has been written in Java jdk1.5.0_03 using Eclipse Platform. The graphic is created in Swing libraries. Some graphical interface parts have been designed using CloudGarden's Jigloo SWT/Swing GUI Builder (free for non-commercial use). There are several classed used to design whole application. All classes are stored in the archive: *rpg.jar* which enables execution by using:
java –jar rpg.jar

As the application is not a final version and is going to be further developed, the Programmer allowed himself to skip some elements of “good programming style” (leaving some data containing object attributes public)

Inside the application, almost all data is being preserved as objects in Vectors (dynamic, easy to maintain) which allow to freely add data.

Used classes:

7.1 Creature class

It represents a single creature. It can be a pattern creature (a model of a creature) or a unique (existing in the game) creature. The difference between both is that every unique creature is different from another (even created from the same pattern). They can have different items and money making it less boring to kill many of them. First, a Vector of “pattern creatures” is being created from CREATURES.XML file and then, every unique creature is being created and added to maps. Pattern creatures are preserved in “Logic” object and all unique creatures are preserved by specific “Map” object they are located in.

7.2 Item class

It is used to preserve info about every item. It uses the same pattern/unique technique creation (look above).

7.3 Event class

It is used to preserve all available events in the game. Events are stored in “Logic” object and are usually erased after a use (by default) according to “repeat” parameter.

7.4 Map class

It contains information for every particular map. Every map contains Vectors of unique items, creatures and fields it contains. There are a 2-dimensional array of field symbols which shows the map and an array of integers to point to the right field type for every map position. This technique prevents storing duplicated data. There is also an array showing if the location has been seen before or should it be hidden. All creatures are being stored in “creatures” vector starting from the player and following from the fastest to the slowest creature (that order is needed for creature action algorithm).

7.5 Field class

It represents a single field which can appear many times on the map. It is being stored on specific map in the “fields” Vector.

7.6 Dice class

There is only one dice object in the whole game although all objects use it. Making only one dice is necessary for better randomization. As the pseudo random generator (java.util.Random library) bases on system time, two different random generators create the same value when executed one after another (for example). To make them different, there is need for only one such generator, which would be executed twice.

7.7 World class

This is a main application containing Main() method. It initializes and creates a main object from “Logic” class and loads all XML data into it. XML files are processed using libraries:

- org.w3c.dom.Document;
- org.w3c.dom.*;
- javax.xml.parsers.DocumentBuilderFactory;
- javax.xml.parsers.DocumentBuilder;
- org.xml.sax.SAXException;
- org.xml.sax.SAXParseException;

They allow XML parsing and maintain its searching.

It also creates a GUI (Graphic User Interface).

7.8 Logic class

This is the main class that controls all events in the game. It stores all items and creatures' patterns and all available maps. It supervises maps adding (including creatures and items) and plays a main engine that other objects refer to.

It is triggered by keyboard events launched by Show object. According to the pressed key it can execute creature moving, attacking or item picking methods. Every turn begins with a Player action and ends with the action of the last (slowest) creature on the map followed by event handling. Every creature (including player) can move and attack other creatures. Other creatures do not attack each other and only the hostile ones will try to follow and attack Player if he gets in their sight (distance equal their perception). Otherwise, creatures move randomly in all directions or wait turn.

Player can also enter "teleport" location (teleport him to another map), chat and pick/drop/equip/use items. Those actions do not generate new turn.

Every new Player's move executes new 'lookAround' to check which fields are visible. The method recursively executes itself to checks surroundings, and surroundings of surrounding, etc, until all viewing range is used. Obstacles with visibility equal 0 forbid to see further in that direction.

Every new turn ends with event checking. All stored events are being checked if they match executing criteria. Events can be executed by following conditions:

- Player enters particular location (or even whole map)
- Player achieves particular amount of experience

Every event can display a window message which informs Player about a new circumstance. It can also modify Player's statistics, or add skill points which Player can freely distribute among attributes.

7.9 Show class

This class inherits from a JFrame class and contains main JTabbedPane. It allows Player to easily choose between different views. The default window size is 800 x 600 pixels but it can be easily adjusted by dragging window corner. This, user manual adjusting, method was used because of ambiguous execution of Toolkit.getScreenSize() method. Depending on operating platform, it was returning whole screen dimension, or just available screen, hiding part of JFrame with a Task Bar.

7.10 Intro class

Simple welcoming screen contains a button describing how the controls look like and some formal information.

7.11 Display class

This class contains main map canvas, message panel and stat panel. It handles Player mouse clicks. Clicking on map displays information about position coordinates type of field, creature and item info (if any). When a creature is right next to Player, dialog function is activated allowing Player to ask questions to a creature (from possible questions).
Map canvas contains useful scroll which center at the Player whenever it moves. Centering method sets a visible rectangle according to Player coordinates multiplied by image side size (50 pixels).
Apart from map canvas, display class initializes also message and statistics panels. It also controls displaying updated Player stats, map name and turn number.

7.12 displayMap class

This object is responsible only for displaying map. It gathers information straight from Logic object, and displays them if necessary at the end of each turn. Every error in loading image which is to be displayed effects in skipping only the display (object is still active). Map is being displayed field after field by going through all lines and columns. Next, items and creatures (including) player are displayed. Creatures are marked with a blood stain if they were injured in the turn.

7.13 StatsPanel class

Show Player statistics just down below the map. This is just a brief info as all the maintenance lies at the Stats view. The panel displays also current map name and current turn number.

7.14 Stats class

This is a statistics view allowing Player to change his character by modifying stats, equipping or using items. All those actions influence greatly Player preferences as well as playing tactics. Player can experiment with different sets of items, seeing how they modify stats.
Items are displayed in a JTable, contained by a JScrollPane which. Every click on particular character body part will try to equip chosen item there, or dis-equip already equipped one.

7.15 Inventory class

This view is usable mostly at item maintaining (as in shops). It allows choosing what item we pick or drop. Both item groups (lying and picked) are stored in JTable supported by JScrolls which allows flexible expanding.

7.16 InventoryDetails class

This simple JPanel displays detailed information about an item.

7.17 QuitPanel class

Contains buttons to execute FileChooser object, initialize FileStreams (in or out) and maintain of data flow. All saved data and contained object are serialized.

8. Possible improvements (things to add):

Of course there is no limit of game improvements, as the game could be large as life. But, to give only an impression of details, I could mention introducing:

- Some monsters are more or less immune to certain kinds of weapons, ex: golems (stone, clay) can be harmed only by blunt weapons (like hammers), or shoots do not have any effect on oozes (jelly crawling thing).
- Aging that could have an effect on player's condition (such as dropping of speed, strength).
- System of spells. The game needs some way of killing monsters and I decided that normal physical attacks are more important and simply did not have time to work on magic.
- Item durability, materials that items are made from and connected with specific consequences. Metal objects could have a possibility of rusting when exposed to water. Wooden items are sensitive for fire and some materials could attract monsters.
- System of goods and religions. Every creature (including character) could have an alignment. Worshipped god could (or not) help in need. Different gods would require different proofs of faith (prayers, sacrifices, etc).
- Character differences. The existence of good and evil could be enriched with neutral and other transitions. Different deeds would push a character to specific alignment. Character would have connection with the deity, used items, accomplished quests.

But the most urgent corrections should consider:

8.1 standardizing

There is no way that I could know what the relations of the game will be. It might occur that a normal player will have strength at 10, dexterity at 50 and speed at 200 to match higher level creatures. Those statistics need to be normalized (not to differ so radically). This is not a difficult programming task (only introducing some multiplications factors) but requires a lot of more time spend on probing the application from another point of view (other humans).

8.2 editors

Not only for maps but also: creatures, items, classes, professions.

8.3 shooting

This is a broad description as many events require this to appear:

- material shooting (arrows, spears, axes, knives)
- range spells

Shooting possibility has not been introduced for the lack of time. It would require an algorithm that checks if a target is available. One cannot shoot someone who is too far, behind the wall, or another creature. This makes some character statistics less important (like dexterity and intelligence) but they can be still used in setting the minimal requirements for weapons. This simplified system has been used successfully in series of Diablo games.

8.4 professions, classes, races, specialization

It was one of the first ideas to introduce them from the beginning (in the hero creation) and resulted just in few example implementations. Fighter, as experienced in pure physical fighting, would have different bonuses (ex: swords, armors, dodge). Every class/profession/specialization should be unique, have a set of features that would change the whole game strategy. Also, by continuing this universal tool, classes should be created dynamically. Those to reasons are the cause for giving up on that. It would be not hard to introduce few classes. But the decision has been made that instead creating something that will have to be changed better not to do it at all. In this way the idea could be considered deeper and designed in a proper way. Having only one type of hero, player can modify statistics, items.

8.5 traps, doors, chests, locks

Of course a profession like thief, rogue or assassin should have a considerable bonus in this field. A blacksmith could also have a possibility to make keys. As for races that have a gift for that, gnomes should stand on the first place. The major stat influencing the skill should be intelligence.

8.6 improved food consumption

Present way of eating could be greatly improved. Instead of just increasing stats, other features could be introduced.

Getting tired depends strictly on action a player takes. Moving in the mountain field or river tires Player more than on the road. A player might first get hungry, then, starving (decreasing some statistics like: strength, dexterity, speed) and eventually may die of hunger. There could be also several kinds of food varying of nourishes, weight and price of course. Also different classes might find different kind of food better, for example gnomes prefer fruits.

Food could be found while searching surroundings (using an 'orientation' skill). The chance could be bigger while in wilderness. Also, the food could be obtained in a shop or found at killed monster's corpse (or even the corpse could be eatable). It would be nice to add rotting feature to the food and 'cooking' skill to preserve it, also increasing its nutriousity.

8.7 armor factor improvement

For simplicity, armor factor is being counted as a sum of all equipped armor elements. It would be logical to differentiate armor elements among body part. Going further, every attack should have a probability on hitting different parts and different damage on them. For example, hitting head would cause critical damage or hitting a hand would disarm it.

9. Summary

I would like to point out that the application was never supposed to be a good RPG game as it does not include this everyday, but important factor, which is human intelligence. Actually, the world is created by creative players which participate in making it live by itself. This is a reason why internet connection, allowing many people to interact, is a necessity for a successful RPG game. Nevertheless, it also creates too many difficulties that can only be solved by a professional company. First, time system should be continuous, which would provide Players synchronization, which implies a full 3D display, to occupy Player's attention during time flow. Making a 3D world exceeds even my rich imagination and creativity skills.

As I said in the introduction, this application does not satisfy all of my first imaginary ideas but I also realize that it could be improved forever. The real aim was not to make everything possible, but to make a good structure to make it extendable. The structure bases mostly on objects and flexible data organization. Logics and graphics have been separated to make it possible to change them individually. As for details, it could be said that following technologies have been mastered:

- a vision of world with it's rules, activities, possibilities

- graphic display – mainly the map
- swing elements which preserve it
- XML configuration files
- mouse and keyboard user events
- some simple but fancy algorithms such as:
    - throwing of the dice
    - recurrence distance checking
    - great deal of object oriented programming (i.e. source objects creating end objects)

It is hard to say which tasks have been achieved and which not because ideas were being born while creating. At this stage of development, basing on a well designed backbone, it is easy to implement any detail from the mentioned above. The only exception requiring a new way of looking is a nicely designed spell system, with lots of configurable features, new spells.xml file, and some animation.

10. Acknowledgments

The knowledge about how an imaginary, fantasy world should work has been taken from years of constant touch with fantasy world. Starting from dozens books, several movies and games, I guess I posses a good overview in this subject. Though, while books and movies showed me only the beauty of fantasy, games forced me to actually feel like a part of that world. So, I would like to thank especially:

- **ADOM** (Ancient Domain of Mystery) for making a text based application that is so enormously detailed and synchronized and which I was trying to imitate
- **Warcraft 2** and **Heroes of Might and Magic 4** for beautiful 2D graphics and handy editors (from which I have taken my example graphics)
- **World of Warcraft** for making RPG system (although greatly simplified) popular among such a huge part of society
- **Diablo** for being the first RPG game I played.