Chapter 6

Implementation

6.1 Introduction

The previous chapter described the design of the Multi Agent System to assist 3D Game Environments Design. The proposed system has been decomposed in to 8 modules and the functionality of each module is described. This chapter discusses about the implementation of each and every module identified in design chapter.

6.2 3D Environment Definition Agent

This agent acts as the main request agent of the system which makes the request to start 3D environment generation. This module is developed with a graphical user interface. The Java Swing technology is used for the creation of graphical user interface. The main purpose of this module is to capture some user inputs about the preferred 3D game environment and these input capturing methods are implemented as specified in Table 6.1.

Input to Capture	**Implemented as**
Height Map	A file browser to load the image file containing the height map
Size of the terrain	Length and width can be selected using drop down lists.
List of 3D models selected from ontology with number of instances required	A list of 3D models of a category (E.g.: Tree category) can be filtered by selecting the required category from a drop down list. From the filtered list of 3D models the required 3D model can be selected using a drop down list. Number of instances required for given 3D model can be specified by selecting “Few”, “Many”, or

	“Average” from a drop down list. However the values associated with above terms are depending on the size of terrain. In addition, a text box is provided to specify the exact amount of instances, if required.
Size and number of towns/ villages/ industries	Size of the towns / villages/ industries can be specified by selecting length and width using drop down lists. Number of towns/ villages/ industries required can be specified by selecting “Few”, “Many”, or “Average” from a drop down list. However the values associated with above terms are depending on the size of terrain. In addition, a text box is provided to specify the exact amount of towns/ villages/ industries, if required.
Water level	A text field with default value 0 (lowest level of terrain). The water level can be adjusted based on the required water level on terrain.

Table 6.1 : Inputs of 3D Environment Definition Agent

After the environment is defined this agent will publish a message with given parameters in *Common Message Space*. Other agent will observe this message and start creating the 3D game environment.

6.3 3D Model Definition Module

This module also has a graphical user interface in order to introduce 3D models and store them in ontology. And this graphical user interface is implemented using Java Swing technology and it provides facilities to define attributes and behaviours of agent. Through this module users can associate 3D modules with agent rule sets which are defined in the ontology.

To define a 3D model, several inputs need to be captured and these input capturing methods are implemented as specified in Table 6.2.

Input to Capture	**Implemented as**
3D model name	Text field
Location of 3D model	A file browser to load the 3D model binary file (E.g.: .obj .3ds)
Category of 3D model (tree, rock etc)	Drop down list
Associated agent rule set (selected from ontology)	Text field
Default scale	Text field (by default this is 1.0. However it is possible to modify this in case a larger or smaller 3D model required).

Table 6.2 : Inputs of 3D Model Definition Module

In addition, this module contains interface elements to Add/ Edit/ Delete 3D models and 3D model categories.

Agent rule sets associated with 3D models are defined as Java classes. To add or modify a rule set, it is required to add/modify the relevant Java class. For example pseudo code for generic plant rule set is similar to Code Listing 6.1. In this example either soil or water are suitable to place a generic plant, because some plants are grown on water.

```
If

given location is free AND
given location is soil OR water AND
given location is not on a road

Then


Location is good to place a Generic Plant
```

Table 6.2 : Inputs of 3D Model Definition Module

The Pseudo code written in Code Listing 6.1 can be implemented as a Java class as listed in Code Listing 6.2.

```
package agents.rulesets.plants;

import agents.rulesets.GenericEntity;

public class GenericPlant extends GenericEntity{

      public boolean goodToPlaceAt(int x, int z) {
            if (goodForGenericPlant(x,z)) {
                  return true;
            } else {
                  return false;
            }
      }

      public boolean goodForGenericPlant(int x, int z) {
            if ((isFree(x, z)) // x, z on a free location

                &&

               (isOnSoil(x, z)|| isOnWater(x, z)) // x, z on soil or
                                                   // water
                &&

               (!isOnRoad(x, z)) { // x, z is not on road

                  return true;

            } else {
                  return false;
            }
      }

}
```

Code Listing 6.2: Sample Java Code of Generic Plant Agent Rule Set Definition

This inheritance of rule sets is shown in design chapter in Figure 5.3. Following methods used in Code Listing 6.2 are inherited from “agents.rulesets.GenericEntity” agent rule set Java class.

- isFree(x, z)   - Returns true if x,z location is free (no other 3D models in x,z)
- isOnSoil(x, z   - Returns true if x,z location is on soil
- isOnWater(x, z)   - Returns true if x,z location is on water
- isOnRoad(x, z)   - Returns true if x,z location is on a road

These methods provides the status of a given location by analyzing the status of Common Game Map Space. In addition to above methods, the “agents.rulesets.GenericEntity” agent rule set Java class contains following methods by default.

- areNeighboursFlat(x, z, N) - Returns true if at least N number of neighbouring cells of x,z location are flat
- areNeighboursSoil(x, z, N) - Returns true if at least N number of neighbouring cells of x,z location are on soil
- areNeighboursWater(x, z, N) - Returns true if at least N number of neighbouring cells of x,z location are on water
- areNeighboursBuildings(x, z, N) - Returns true if at least N number of neighbouring cells of x,z location are buildings
- areNeighboursRoads(x, z) - Returns true if at least N number of neighbouring cells of x,z location are roads
- areNeighboursPlants(x, z) - Returns true if at least N number of neighbouring cells of x,z location are plants
- isNearRoadButNotVeryClose(x, z) - Returns true if x,z location is near a road, but not very close to a road (neighbouring cells should not on roads)
- isOnACity(x, z) - Returns true if x,z location is on a city

Even though above methods are included in “agents.rulesets.GenericEntity” agent rule set Java class by default, it is possible to extend this class by adding new methods based on future requirements.

The example shown in Code Listing 6.1 and Code Listing 6.2 contains only few generic rules, because that rule set is for a generic plant. However if the rule set to be defined for a Coconut Tree for example, the method goodToPlaceAt(x, z) should be overridden at Coconut Tree Agent Rule Set to avoid placing Coconut trees on water as shown in the sample Java code listed in Code Listing 6.3

```java
package agents.rulesets.plants;

import agents.rulesets.plants.GenericPlant;

public class CoconutTree extends GenericPlant {

      public boolean goodToPlaceAt(int x, int z) {

            // Coconut tree should be on soil
            if (isOnSoil(x, z) && goodForGenericPlant(x,z)) {
                  return true;
            } else {
                  return false;
            }
      }
}
```

Code Listing 6.3: Sample Java Code of Coconut Tree Agent Rule Set Definition

In contrast, if the rule set to be defined for a Lotus Plant, the method goodToPlaceAt(x, z) should be overridden at Lotus Plant Agent Rule Set to force placing Lotus palnts on water as shown in the sample Java code listed in Code Listing 6.4.

```java
package agents.rulesets.plants;

import agents.rulesets.plants.GenericPlant;

public class LotusPlant extends GenericPlant {

      public boolean goodToPlaceAt(int x, int z) {

            // Lotus plant should be on water
            if (isOnWater(x, z) && goodForGenericPlant(x,z)) {
                  return true;
            } else {
                  return false;
            }
      }
}
```

Code Listing 6.4: Sample Java Code of Lotus Plant Agent Rule Set Definition

Ability to define agent rule sets as described above enables users of the system to define their own agent rule sets and associate them with 3D models using *3D Model Definition Module.* Also this feature solves the problem of lack of customizable frameworks for 3D game environment generation, which is identified as a major problem of existing 3D environment generation techniques, by providing an extendable framework.

6.4 Ontology

Since the ontology of this system contains agent rule sets and 3D model binaries, the traditional XML based ontology definitions are not enough. Therefore it was decided to use a combination of Java classes and an embedded lightweight database for ontology. The H2 database is used as the embedded database, because it is a very light weight database written in Java and the memory footprint of this database is less than 1 MB. Also it is very easy to deliver the database as a file with the system because of the smaller file size.

The database contains one main table called MODEL table to store the 3D model definitions and the structure of that table is as shown in Table 6.3.

MODEL	
Name	VARCHAR(100)
Model_Category	VARCHAR(100)
Location	VARCHAR(255)
Agent_Rule_Set_Class	VARCHAR(500)
Default_Size	FLOAT

Table 6.3 : The Structure of Model Table

The agent rule set definitions are kept as Java class files in the ontology. 3D model binaries are stored as it is in the file system with a definition of file path in database table MODEL. In addition .properties files are used to store scene graphs of 3D environment and those files are also considered as a part of ontology.

6.5 Common Message Space and Common Game Map Space

The *Common Message Space* is implemented as a Java ArrayList by at which is publicly visible to all agents. Agents can publish messages by adding an element to ArrayList. When a relevant agent observes a request message, the agent will respond by adding another message to the *Common Message Space* to inform that there is an agent working on the request. After catering the request the message can be removed from the Common *Message Space* by calling the remove method of the ArrayList.

The *Common Game Map Space* also acts as a common message space. This map space is used to represent the current state of the environment. The locations in environment are represented using several 2D arrays as described in Table 6.4.

2D Array Name	**Purpose**
terrainNatural	Keeps information about type of natural terrain (E.g.: Soil, Water, Rock)
terrainCultural	Keeps information about type of cultural terrain (E.g.: City, Industry, Estate)
terrainHeight	Keeps information about height of the terrain
roads	Keeps information about availability of road segments

Table 6.4 : 2D Arrays used in Common Game Map Space

To take decisions to place 3D models, agents sense the status of these 2D arrays and determine the current status of game environment. Based on that, those agents act on the environment by marking places for 3D models on 2D arrays. The arrays were defined as integer arrays as much as possible to increase the processing speed. It is sufficient to keep only the X and Z coordinates in 2D arrays, because the Y values are interpolated by 3D display module based on the height of the terrain.

6.6 Agent Implementation

MASON: Multi Agent Simulation Toolkit is used to develop agents in this system. Sean Luke and colleagues have presented this toolkit [13] for a wide range of multi-agent simulation tasks from swarm robotics to machine learning to social complexity environments. This toolkit has been designed with a special emphasis on swarm simulations. However the flexibility is provided to use MASON in a wide rage of other multi-agent applications also. It provides facilities to visualize and manipulate models in both 2D and 3D. Network intrusion and countermeasures, cooperative target observation in unmanned aerial vehicles, ant foraging, anthrax propagation in the human body and wetlands: a model of memory and primitive social behaviour are some of the applications of MASON. This toolkit is licensed as BSD style free and open source software and it is written in Java programming language. This is a very light weight and scalable framework. It was decided use MASON for this project considering above points.

This system contains mainly 3 agent types which are developed using MASON and Java namely Terrain Explorer Agents, 3D Model Placing Agents and Road Network Development Agent. All of these agents are developed by implementing the "sim.engine.Steppable" interface of MASON framework. Following sections describes the implementation of these agents.

6.6.1 Terrain Explorer Agents

Terrain Explorer Agents are created when there is a request *3D Environment Definition Agent* to explore for a terrain suitable for specific requirement such as placing a city. These agents can have many specializations as described in analysis and design chapter under *Section 5.7 Terrain Explorer Agents*. These agents read the values of 2D arrays in *Common Game Map Space* and seek for suitable terrain based on the specialization of agent. When a suitable terrain is located, it is marked as reserved by updating the values of 2D arrays in *Common Game Map Space*. After completing the exploring of suitable terrains, this agent publishes a message to *Common Message Space* and get disposed.

6.6.2 3D Model Placing Agents

3D Model Placing Agents are created when there is a request *3D Environment Definition Agent* to mark locations to place 3D models. These agents can have many specializations as described in analysis and design chapter under section *5.8 3D Model Placing Agents*. These agents read the values of 2D arrays *in Common Game Map Space* and decide the suitability of terrain to place a 3D model, based on the agent rule sets associated with 3D models. When a suitable place is located to place a

3D model, it is marked by updating the values of 2D arrays in *Common Game Map Space*. After completing the exploring of suitable terrains, this agent publishes a message to Common Message Space and get disposed.

6.6.3 Road Network Development Agent

Road Network Development Agents are created when there is a request from *3D Environment Definition Agent* to generate roads. This agent read the values of 2D arrays in *Common Game Map Space* and places to lay roads as described in design section *5.9 Road Network Development Agent*. This agent uses A* algorithm for path finding and obstacle avoidance. Created roads are marked by updating the values of “2D array for roads” in *Common Game Map Space*. Also the generated roads are saved in a .PNG transparent image file to be used as a texture map of roads in game engine. However due to the size of the cells in game world, the roads marked on .PNG image file contains roads with rough edges. To solve that issue those roads on the .PNG image file are smoothed by applying a blur filter before using in 3D environment. Also the primary roads are marked thicker than secondary roads. After completing the road generation, this agent publishes a message to *Common Message Space* and get disposed.

6.7 3D Rendering Module

Since the Java programming language is used as the primary programming technology of the project, the jMonkey Engine (jME) which is available at http://www.jmonkeyengine.com/ has been selected as the game engine to demonstrate the out put of the system. The jME is a high performance scene graph based graphics API. It is completely open source under the BSD license. This engine was built to fulfil the lack of full featured graphics engines written in Java. Using an abstraction layer, it allows any rendering system to be plugged in. Currently, both LWJGL and JOGL are supported.

A sample game with the ability to flythrough the game environment is created using jME in order to display the output of the system. The terrain of the 3D game environment is generated based on the height map provided to *3D Environment Definition Module*. Also a sky box is created based on the size of the terrain and projected water is generated at the level requested by environment definition. In addition, road texture is placed on terrain as a “Splat Texture” (a texture with transparency) based on the road network generated on transparent .PNG image file by *Road Network Development Agent.*

To place 3D models, this module reads the ontology and retrieves the data representation of a generated 3D environment written in a properties file. The content of properties file is converted to a scene graph using the source code of the sample game which is created using jME. At the same time the relevant 3D models are loaded by reading the 3D model definitions from ontology and those 3D models are placed in 3D game environment by attaching to the scene graph. The final output is rendered on a computer screen with the ability to flythrough the created 3D environment.

The prototype system was implemented by integrating the modules described in this chapter. The *Appendix A - Approach in Practice* of this thesis shows the usage of prototype implementation by going through a sample scenario to generate a simple 3D game environment.

6.8 Summary

This chapter discussed about the implementation of the modules which are identified in design chapter. A prototype was implemented using Java, jMonkeyEngine, H2 embedded database and MASON agent development tool. The next chapter reports how the prototype and the proposed approach are evaluated.

Chapter 7

Evaluation

7.1 Introduction

The system has been evaluated by creating a prototype to test the approach proposed by this thesis. Following aspects of the system were considered during the evaluation process.

1. Time taken to generate 3D environments
2. Customizability and Extendibility of the system
3. Adherence to industry standards
4. Portability
5. Cost Effectiveness

Following sections reports the evaluation of above specific aspects.

7.2 Time Taken to Generate 3D environments

Reducing the time taken to create a 3D game environment is one of the main advantages of the approach proposed in this thesis.

To evaluate this it was decided to come up with 2 control experiments.

7.2.1 Control Experiments

It was decided to compare the prototype based on the approach proposed in this thesis with following approaches.

1. Using a coding approach to design the 3D game environment
2. Using a world editor to manually design the 3D game environment

7.2.2 Selection of Participants

It was decided to select 3 users who have experiences with 3D game environments and with programming background in Java programming language as shown in Table 7.1.