Deliverable data

WP6 - D6.2 Software architecture and specifications

LEARN2LEAD (L2Lead)

Action LEONARDO DA VINCI
Multilateral Projects for the Development of Innovation
AGREEMENT NUMBER -2009-2175/001 - 001
PROJECT NUMBER 502903-LLP-1-2009-1-IT-LEONARDO-LMP

Start date of project:	01 January 2010
Duration:	24 months
Project Leader:	ISTC- CNR
Partners:	Entropy Knowledge Network Srl (Entropy), University of Naples (UNINA), MF & Partner Consulting, Universitat Jaume I (UJI), University of Lincoln (ULINC)

This project has been funded with support from the European Commission.

Deliverable data

D6.1 - Functional and non-functional requirements

Workpackage	6
Task	1
Date of submission:	
Leading partner for this deliverable:	ISTC-CNR
Authors:	Massimiliano Schembri
Version:	1
Dissemination level:	Public
Abstract:	
Specification document	

Document Sign-off

Nature	Name	Role	Partner	Date
DRAFT	Massimiliano Schembri	Researcher	ISTC-CNR	10/09/2010
REVIEWED	O. Miglino	Manager	UNINA	30/09/2010
APPROVED	C. Castelfranchi	Manager	ISTC	31/10/2010
SUBMITTED	C. Castelfranchi	Manager	ISTC	31/10/2010

Index

ABSTRACT

This document describes the first prototype of the L2L game from a technical point of view. The software architecture have been designed starting from the Game Design document trying to make it as much as possible independent from the visualization and user interaction. The core of the software is a set of classes written in c# language that implements the basic game mechanics. The name of the classes (i.e. GameLevel, Job, Follower etc.) have been chosen to recall parts of the game design in order to better understand the links between them. The visualization and user interaction has been realized taking inspiration from the Model Vista Controller pattern which was not suitable for our project. A Visualized-Visualizer pattern has been invented on purpose to permit a flexible and modular way of implementing the graphical aspects of the game. This was of vital importance since the graphics of the game had to be outsourced.

The game and software development tool used is Unity 3D V.2.61 (http://unity3d.com/). The main advantage of this development environment is that parts of same project can be easily deployed as a standalone software or a web application without major changes in the code.

INTRODUCTION

The first ideas about the software architecture of the L2L game have been discussed alongside the first draft of the game design document. It was clear that the software had to be some how specular with respect to the game mechanics.. An object-oriented approach has been adopted from the early stages of the design. Every meaning part of the game had to be converted into proper data structure with set of functions to manipulate it. Relations among objects had to be identified and analyzed. The software design has been done in a purely abstract fashion without worrying about visualization and user interaction. Afterwards a specific design pattern for visualization and user interaction have been devised.

The software has been developed using a Game Development Tool called Unity3d which is an integrated authoring tool for interactive 3d content creation which uses Javascript and c# for scripting.

Following we'll describe the architecture of the software showing the main classes and their functioning.

SOFTWARE ARCHITECTURE

Before describing the software architecture let's recap the general structure of the game mechanics. The game is organized in levels of increasing difficulty. In each level the player leads a group of followers who have to accomplish a number of jobs each to be completed by a specific deadline. He can assign jobs to followers based on their personality, motivation and abilities (depending on the game level). During the week he can perform a number of “Leadership Tasks” aimed at influencing follower’s motivations and abilities. The software architecture reflects this structure by implementing the following set of classes: Game, GameLevel, Job, JobAssignment, Follower, Time, WeeklySchedule, LeadershipTask.

Game is the top level class. A Game object contains and controls instances of GameLevel class which in turn contains a set of Follower objects, a set of Job objects, some LeadershipTask and an array of WeeklySchedule objects. Fig. 1 depicts this hierarchical structure.

In the next sections we will describe these classes in more detail.

THE GAME CLASS

As said before the Game class is the top level class of the hierarchy. It can contains a set of GameLevel objects that represent the level of the game. The constructor of this class accepts a string parameter which is the filename of an xml configuration file that contains information about followers, jobs, leadership tasks ecc. of each level. By means of the configuration file it's possible to create different game scenarios for different purposes.

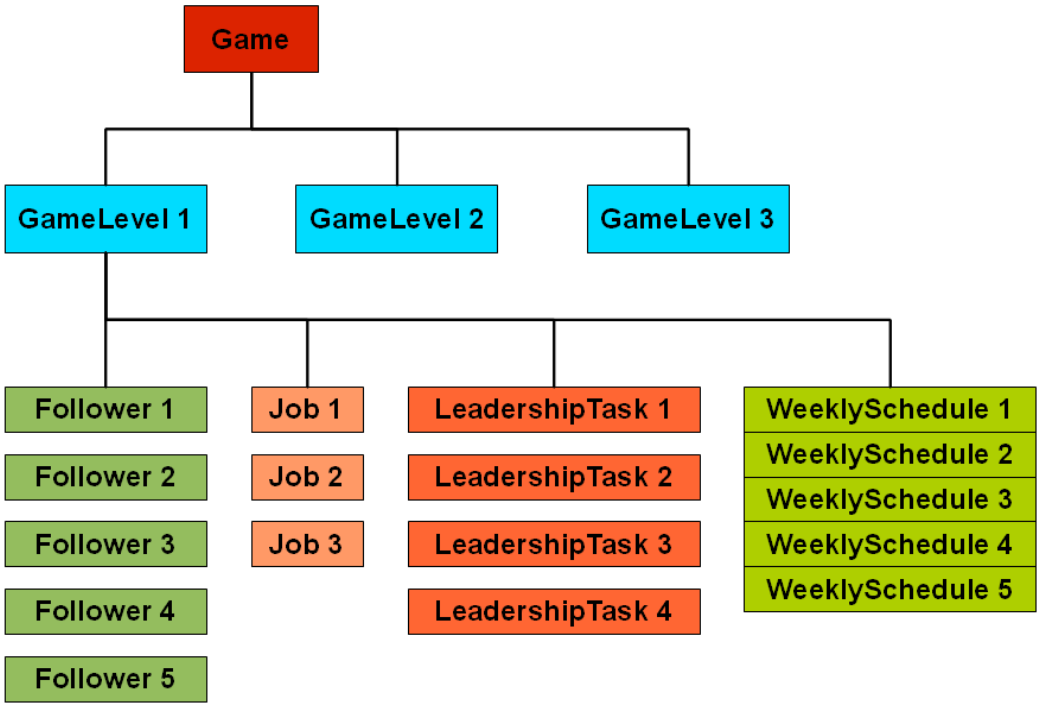

Fig. 1 General class hierarchy of the Learn To Lead software

THE GAMELEVEL CLASS

The GameLevel class controls most of the elements that constitutes the L2L game. A GameLevel object contains a number of Followers, Jobs, LeadershipTask and WeeklySchedule. A set of specific methods permits to add these elements to a game level (addFollower(), addJob(), addLeaderTask()).

An array of WeeklySchedule objects represents the time during which all the jobs must be completed (with different deadlines) and contains information about the activities planned by the leader. An important method of the GameLevel class is the stepGameHour() method. Time is represented with a resolution of 1 hour and this method makes the game advance of 1 hour after which all the consequences of the current game state are calculated and take effect.

THE FOLLOWER CLASS

The Follower class groups all the information of a single follower. The main parts of the follower class are a FollowerBrain object and a list of JobAssignment objects. FollowerBrain is a class that is part of the team dynamics connectionist model. Essentially this class implements a connectionist model of a follower behaviour that models behavioral aspects like motivations, abilities and personalities and calculate his contribution in reducing the jobs workload (see D4.1 for details).

Each follower object contains also a list of JobAssignments. As the name suggests this is a list of objects representing all the jobs a certain follower is assigned to. By means of this list is possible to calculate the total workload requested to a single follower. Actually this value is stored in a specific attribute of the Follower class called *requestedWorkload.*

THE JOB CLASS

The Job class represents information about jobs. Every Job object in the game has a name, a deadline expressed with the Time class and a workload. At the beginning of the game the workload is initialized with a certain value and it decrease as long as the game proceed and followers contribute to it.

Every Job also contains a list of JobAssignment references so that is possible to know which followers are assigned to it.

THE JOBASSIGNMENT CLASS

The JobAssignment class is essentially a link that connect a follower with a job. This connection expresses the fact that a follower is in charge of completing that job giving a certain amount of contribution. Every object contains references to a single Job and a single Follower. Other information stored in this data structure are the total workload of the job and the remaining workload which keep track of the work done by the follower.

THE LEADERSHIPTASK CLASS

The LeadershipTask class represents activities that the Leader can perform during the week to influence his followers like training sessions, meetings, sending email, briefings ecc. The most important attributes of this calss are the LeadershipType, the startingHour and the list of followers attending this activity.

The LeadershipType is a class itself which stores information about the duration and effects of a task. The effect of the LeadershipTask is coded by means of a LeadershipAction class which codes leadership actions based on the motivation (extrinsic, intrinsic), ability and personality (power, achievement, affiliation) of a follower.

LeadershipTasks are planned at specific time during the week. The startingHour attribute indicates this information. Its value can vary from 1 to 40 (considering that a week is formed by 5 days of 8 working hours each).

Every leadership task has a list of followers attending it. The leader can decide that only some of the followers or even a single one need a specific training session, personal communication ecc. The leadership tasks are very important in the game because the player leadership style is deducted by his way of influencing the followers and this is done by planning the right activities at the right moment.

THE WEEKLYSCHEDULE CLASS

As already said the main goal of the player, other than assign jobs to followers, is to plan a series of activities on a weekly basis in order to manage his work group towards a successful accomplishment of the jobs. The WeeklySchedule class represents a 8 hours x 5 days week period used as scheduler. A set of methods allow for the insertion of new activities at any available hours during the working time. It's also possible to query the WeeklySchedule to know if any activity is planned for a particular hour. When time passes the events on the WeeklySchedules change state from "planned" to "done". Each game level contains a collection of WeeklySchedules.

VISUALIZATION AND USER INTERACTION

The software architecture described before is designed to be completely independent from the visualization. A specific pattern has been created to manage the visualization problem. It's based on the use of interfaces since c# doesn't support multiple inheritance. The IVisualized and IVisualizer interfaces are provided to implement this pattern. Every class that represents objects that need a kind of visualization has to implement the IVisualzed interface. Moreover a visualized object contains a reference to its visualizer. This is an object that implements the IVisualizer interface and that is in charge of managing the visualization and user control on the visualized class. A visualized object doesn't need to know what kind of object is its visualizer but through the interface it can invoke an update method that tells the visualizer to update the view of the object. On the contrary the visualizer need to know every detail of the visualized class and has full control on it.

This is a very flexible pattern that permitted us to easily change the visualization type that initially was very simple and based on GUI components like buttons and labels, and then switched to a more sophisticated and user friendly 3D environment.

Fig.2 shows an example of visualization based on GUI elements that was initially adopted to test the game mechanics in a very simple way. Fig.3 shows how the same game can be wrapped with a nicer and more user friendly graphic environment.

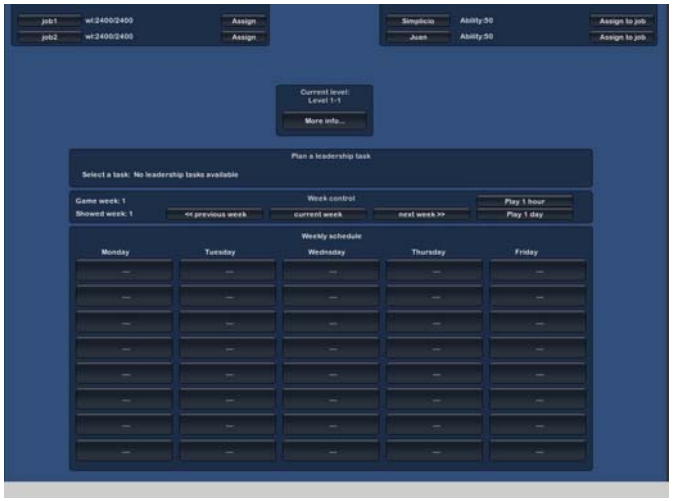

Fig.2 GUI visualization system based on labels and buttons

Fig.3 The 3D visualization system based on drag and drop.