TWILIGHT

CONFIGURATION MANAGEMENT PLAN

Anıl Yiğit Filiz **1395045**

Berkehan Altınkaya **1394642**

Derya Akpınar **1394600**

Güneş Efe **1394980**

CONTENTS

1. INTRODUCTION

i. Purpose

This software configuration management plan is written for the participants of the Twilight Project to establish and maintain the integrity of the project throughout the development process. This SCM plan is intended to provide guidelines when managing and controlling the changes in the configuration of the Twilight project.

ii. Scope

The scope of this plan extends to source code management, documentation, development tools, executables and the operating system and the hardware to be used during the project development process. It identifies the planning of the implementation process and describes the responsibilities and duties of the project members.

iii. Definitions and Abbreviations

CCB: Configuration Control Board
CM: Configuration Management
CVS: Version Control System
SCM: Software Configuration Management

2. SOFTWARE CONFIGURATION MANAGEMENT

i. Organization

The organizational units that participate in the SCM activities of the Twilight Project are:

• Configuration Control Board (CCB)

All of the participants of the project are active members of the CCB. They will be responsible in approving or rejecting an SCR and update the CM schedule accordingly.

• Testing Team

The testing team is responsible for originating SCRs throughout the testing process. They will also

ensure the implementation of the SCRs. Güneş Efe is responsible for planning test cases and applying them.

• Developer Team

Developer team is responsible for the development of the Twilight project and they implement all the change requests that come from the testing team. Members of the developer team are Berkehan Altınkaya, Anıl Yiğit Filiz and Güneş Efe.

ii. SCM Responsibilities

Responsibilities of the CCB members are :

Commenting the developed functions and files properly : Each individual member is also a part of the developer team. In order to co-operate throughout the development cycle each member is required to write comments and to-do lists for every function and file they develop.

Uploading the finished files to the CVS system : CCB member Berkehan Altınkaya is responsible for uploading the developed files to the CVS system at every prototype step. The names of the developers of the files will be appended to the files they create.

Coordinating the development team for properly implementing the SCRs : The member Anıl Yiğit Filiz is responsible for coordinating the development team for properly implementing the SCRs.

Coordinating the development process according to schedule : Every member is responsible for keeping up with the schedule. Prototype days(in every two weeks) will be the milestones for the tasks of each member.

iii. Tools & Infrastructure

Throughout the development project of Twilight the tools to be used can be listed as follows.

- **DevC++**

DevC++ is an open source IDE for C and C++. It has versions for windows platform and specific packeges for different APIs are present on the web. It supports projects and also it h

as CVS support. But we will make use of another CVS tool TortoiseCVS since it is easier to use.

- **Tortoise CVS**

CVS stands for concurrent versioning system. Tortoise CVS is a user friendly tool for CVS. Since the project will be in windows platform Tortoise CVS is very handy to monitor the different versions of the project.

- **WinMerge**

WinMerge is an editor for window that supports diff functions. It can be integrated into TortoiseCVS and just after checking out the CVS repository the differences of files can be seen from this program. It is up to the user to select which blocks to use in the files.

3. CONFIGURATION MANAGEMENT TASKS

i. Configuration Identification

Twilight implementation consists of modules and supporting functions. The module names are starting with uppercase letters. If the name of a module or function has more than two names, it is written in an appended form and all the words starts with uppercase letters. Each module is in a separate directory according to its usage. Each module has two files, one for the header and one "cpp" file for the actual code. The directory structure is as follows:

Data: This directory is also included with the executable. It contains texture bitmaps, models, sounds and the options file.

Encode-Decode: This directory contains the encode and decode modules.

GameData: This directory contains game data modules for server and client.

Graphics: This directory includes the graphical modules and their auxiliary functions and classes.

Input: This directory includes the input module.

Library: This directory contains the support functions to be used by all of the modules.

Models: This directory contains the classes to be used for importing model files(OBJ).

Network: This directory contains network modules for server and client.

Objects: This directory contains the object classes that are used in the game.

Physics: This directory contains the physics modules and their supporting functions.

Sound: This directory includes the sound module.

World: This directory contains the world class that is used to initialize the map in server.

ii. Configuration Items

Directory	Filename	Description
Encode-Decode	EncodeModule	Module used for encoding network packets.
	DecodeModule	Module used for decoding network packets.
GameData	ClientGameData	Storage for client objects.
	ServerGameData	Storage for server objects.
	UserData	Storage for user inputs.
Graphics	GraphicsModule	Rendering class for the client.
	Particles	Wrapper class for particles.
	Particle	Instant of a particle.
	Stars	Creation and displaying of stars.
Input	InputModule	Input handling.
Library	Vertex	Simple vertex class.
	Options	Storage for options of the game.
	BmpLoader	Bmp loader class.
	Texture	Texture loader class.
	LinearRandom	Random number generation.
Models	Model	OBJ file loader wrapper.
	Group	Support class for Model.
	Face	Support class for Group.
	Material	Support class for Model.
Network	NetworkServer	Packet transfer handler for server.
	NetworkClient	Packet transfer handler for client.
	NetworkAux	Support functions for network classes.
Objects	Object	Wrapper class for objects.
	Displayable	Objects in the client side.
	Spaceship	Spaceship object class.
	Laser	Laser object class.
	Rocket	Rocket object class.
	Mine	Mine object class.

	Headquarter	HQ object class.
	NPC	NPC object class.
	StillObject	Still object class.
Physics	PhysicsModule	ODE wrapper for physics interaction handling.
	Physics	Physics instance of every object.
Sound	SoundModule	Class handling the sounds.
World	World	Class for initializing the universe.
	MissionModule	Mission initializer, handler class.

iii. Management And Control

Change Requests: Change requests are taken on the daily meetings. Each member has the latest working copy of the project and changes are requested informally. Whenever a change is made by a member, the latest working copy of the source code is backed up and named as "Twilight vX.X.". The changes made by the member are then merged together with all the members. If the changes are successful, another working copy is formed.

Build: Whenever a change that causes an advance in the schedule is made a backup of the project with the "Data" folder is taken. These backups are named as "Twilight Working With Data vX.X". These builds are presented in every two weeks to the customer(Oral Dalay).

iv. Accounting

As the project advances, it gets harder to keep track of the version details. To overcome this problem a change log will be included in every backup copy of the project. Naming convention will be as "ChangeLog.txt". The new changes will be entered as the version number on the top and every change listed as a separate line.

v. Auditing

Auditing of the project will be done after a new source code version is created. Each member will test the new version and if an error occurs each member will have equal share of voting to go back to the previous version or to request for a change to correct the error.

4. PROJECT SCHEDULES – CM MILESTONES

Since the project is improving this version of the CM will be outdated after the first release. There will be another version with the first release including these changes. Then after the final release the overall CM plan will be explained.

Date	Milestone	Description
12.03.2007	CM Delivery	Delivery of this document.
01.04.2007	CM Update	An update of changes in the CM with the first release of the project.
01.06.2007	CM Final	Final CM of the project's finished state.

5. PROJECT RESOURCES

The following items will be used for CM activities:

− TortoiseCVS

− WinMerge

− DevC++

6. PLAN OPTIMIZATION

The optimization of the plan will be made during the updates in the SCM plan. These updates are scheduled to the first release and the final release of the project. According to the difficulties faced throughout the project the CM plan will be optimized and changed. These changes will be made by the whole team in order to prevent miscommunication.

7. APPENDIX

Living Schedule

		March	March	April	April	May	May
Task	Member	1 - 15	15-31	1-15	15-30	1-15	15-31
Modelling	Derya	X	X	X	X	X	X
Interface Objects	Berke	X	X				
Collision Detection & Physics	Yiğit	X					
Integration of Weapons	Güneş	X	X				
Universe Objects AI	Berke	X	X				
Particles	Yiğit		X	X			
User Database	Güneş		X	X	X		
NPC AI	Berke			X	X	X	
Interface Integration	Yiğit			X	X	X	
Spaceship Integration	Güneş		X	X	X	X	
Texturing	Yiğit		X	X	X	X	X
Missions	ALL				X	X	X
Sound Integration	Berke				X	X	
Testing	Güneş					X	X

This is a rough one.
The schedule may change a bit.

An Enjoyable Version

First Release