|  |
| --- |
|  |
| Sisyfos |
| A turn-based strategy game engine |
|  |
| **Frank** |
| **6/27/2015** |

Contents

[Prolog 2](#_Toc423171452)

[Overview 2](#_Toc423171453)

[Prerequisites 2](#_Toc423171454)

[Engine 3](#_Toc423171455)

[Runtime 3](#_Toc423171456)

[Game Logic 3](#_Toc423171457)

[Lua Interface 4](#_Toc423171458)

[Testing 4](#_Toc423171459)

[Unit Tests / Automatic Tests 4](#_Toc423171460)

[Tests in network 5](#_Toc423171461)

# Prolog

This document is currently giving the briefest information about this project. I someone shows any interest I will try to add in more information.

Sisyfos is in *development*.

All these things have been written and created on my spare time as an overgrown hobby project ☺

# Overview

Sisyfos is an experimental project, where I try to use Ada and PolyORB to create a turn-based-strategy game engine that can be played by 2-3 players on a LAN – network.

I was hoping to create an internet game in this way, but I have had problems to make PolyORB work in a useful way via routers and so on, so I have not developed this functionality. Future versions of PolyORB might be more helpful ☺

In the folder Tubastga there is a project file TubastaPolyORB.gpr that will let you compile the experiment game “Tubastga” using PolyORB for distributed communication.

In the same folder there is also two other projects files TubastgaSocketClient.gpr and TubastgaSocketServer.gpr. These project files will build the experiment game without PolyORB, only using sockets directly. Compile these two separately and you will get one executable to be used on the server and one to be used on the client.

# Prerequisites

In order to get this project to compile there are some things you need to install and get to know.

The Sisyfos engine and Tubastga is developed in the programming language Ada. This is a language similar to Pascal in appearance but with some more built-functionality. This project has not seriously taken into use the functionality from the latest standard Ada2012, but relies on the standard Ada2005.

The GUI part is using Gtk3 (with Cairo functionality).

The simplest way, per 1st March 2015 is to download an Ada compilator, Ada IDE, Gtk and PolyOrb from <http://libre.adacore.com/> (2014 version)

I have developed using Windows8.1. Installing all this on Windows is fairly simple. It is now very long since I installed it on Linux so I am not able to tell what it is like there, but it requires some use of Linux command and shells.

If you are new to Ada and especially want to install it on Linux, it might be an idea to dedicate a PC for this purpose so that you can install/uninstall and configure it to this purpose. This doesn’t require a powerful computer.

You will need connection to a LAN.

# Engine

## Runtime

The engine contains a task that starts in the server. This task is a loop that is listening for input from the players. When the task receives any command from any player, it runs logic to try to do these commands. The task uses Ada built-in mechanism to control concurrent activities. Any of the procedures / functions listed above runs alone in the server (no other player is doing anything at the same time).

The engine controls users creating a game, users logging in, users joining a game, users saving and loading a game.

The engine controls the players turns.

The engine deals with communication and interface to the client code.

## Game Logic

Sisyfos contains the framework so that you can create your particular game without tinkering in the engine (you have to adapt your game to the engine). You can have your own pieces or terrain. You can customize this in a different body of logic. The engine let 2-3 players manipulate their pieces, move around on the map fight with each other etc.

It requires you to program functions and procedures that (not complete list):

* Initializes a new piece with you particular needs
* Controls where a created piece can be places on the map. You do this by returning an array of all the positions where a piece can be placed. If the player tries to place the piece somewhere else the engine refuses to place it there.
* Controls if the player is trying to move to patches where he is allowed to move
* Controls if the player is trying to attack a piece on patches where he is allowed to attack
* Calculate the area that each piece is able to observe on the map – sums this up to a report of observations and return it to the client. In this way each player can see what his pieces are able to see. However, since this is a function that you can change you can make it possible to see everything.
* Calculates the outcome of an attack – deals with the win / loss.
* Calculate use of Action Points per piece.
* It can return a narrative text related to the action of the player.
* Grant or revoke effects (according to your implementation) to pieces or patches.
* Do certain tasks at the start of a turn
* Do certain tasks at the end of a turn
* Do certain tasks periodically during a turn.
* Load information particular to your game
* Save information particular to your game

These functions/procedures are a mix of abstract (in C++ called virtual) procedures/functions and callback functions/procedures.

Do not create tasks or protected objects that will be used in the logic that is used by the engine.

Do not create logic that is using IO in any of the abstract procedures/functions or callback functions/procedures except from the events Creating, Saving, Loading, Game Start, Game Upkeep, Start Turn, End Turn or Game End.

These restrictions are valid no matter if you use any other language or script in the game logic.

## Lua Interface

If you have created a game based on the engine, it is possible to adapt you game by adding Lua scripts that will be executed on predetermined events.

The package server.lua\_interface contains the procedures and functions that are made available to the Lua language.

Do not create multi-threaded scripts in Lua. The outcome of doing this, is not known.

The same limitations regarding IO is valid in Lua scripts, as it is in game logic implemented in Ada.

# Testing

## Unit Tests / Automatic Tests

There is a number of tests of the engine (in the Test folder). The tests are coded in a unit test library of Ada. The structure of the tests are in very many cases not ideal.

I have attempted to test the engine, and its behavior. In this work I have got in the situation that there is a lot of initializations to do prior to a test e.g. start engine, place some test pieces etc. In some test scripts I have been lazy, and I have made tests that relies on the situation left from a prior test. In this way it is not always trivial to insert a new test in the middle of an existing test. I am not happy about this – I have still something to learn in order to be able to create automatic tests of an engine like this.

## Lua Tests

There are tests for the Lua interface. The tests validates that the parameters given to the procedure/functions from Lua are translated into the correct parameters in the Sisyfos engine.

## Tests in network

I have only tested it briefly on 2 computers via LAN. I have not tested it at all on 3 computers (two clients and one server).