Creating a Dragonwind like Game

This document describes how to create a game using the *Dragonwind Like Project* template, and the rest of the QGAMES libraries.

Content

[Introduction 3](#_Toc26807356)

[Installing the template in Visual Studio 3](#_Toc26807357)

[Tools needed to extend the template 3](#_Toc26807358)

[Installing QGAMES first 3](#_Toc26807359)

[Creating basic Dragonwind like games 3](#_Toc26807360)

[Reading Conf.xml 4](#_Toc26807361)

[Scene definition 5](#_Toc26807362)

[Attributes 7](#_Toc26807363)

[Action blocks 7](#_Toc26807364)

[Connections 8](#_Toc26807365)

[Map definition 10](#_Toc26807366)

[Background maps 10](#_Toc26807367)

[Play maps 12](#_Toc26807368)

[Base 13](#_Toc26807369)

[Solid 14](#_Toc26807370)

[Liquid 14](#_Toc26807371)

[Ladder 15](#_Toc26807372)

[Decoration 15](#_Toc26807373)

[Notification 15](#_Toc26807374)

[Location 15](#_Toc26807375)

[ObjectZone 16](#_Toc26807376)

[Presentation 18](#_Toc26807377)

[Exit. Type 7000 20](#_Toc26807378)

[Null. Type 10025 20](#_Toc26807379)

[Waiting time. Type 10000 20](#_Toc26807380)

[Waiting until enter key or mouse left button is pressed. Type 10001 21](#_Toc26807381)

[Waiting until a key or any mouse button is pressed. Type 1002 21](#_Toc26807382)

[Waiting a couple of seconds or until a key or mouse button is pressed. Type 10006 21](#_Toc26807383)

[To Show a Text. Type 10003 21](#_Toc26807384)

[To show the number of the player. Type 10004. 22](#_Toc26807385)

[To show the FPS: Type 10005. 22](#_Toc26807386)

[To play a sound. Type 10010 22](#_Toc26807387)

[To show a fixed form in the screen. Type 10020 22](#_Toc26807388)

[To show a form that appears or disappears. Type 10030 23](#_Toc26807389)

[To show an entity. Type 10040 23](#_Toc26807390)

[To load all resources of a game. Type 10050 23](#_Toc26807391)

[To create a count down effect. Type 10060 24](#_Toc26807392)

[To show a hall of fame. Type 10070 25](#_Toc26807393)

[To roll figures on the display. Type 10080 26](#_Toc26807394)

[To select an option from a menu. Type 10100 26](#_Toc26807395)

[Main menu. Number 10000 26](#_Toc26807396)

[Keys menu. Number 10100 27](#_Toc26807397)

[To show a list of credit titles. Type 10090 27](#_Toc26807398)

[To show a set of instructions. Type 10091 28](#_Toc26807399)

[To control how the game states flow. Type 10101 29](#_Toc26807400)

# Introduction

## Installing the template **in** Visual Studio 2017

Copy the .zip file into the directory *./Visual Studio 2017/My Exported Templates* usually located in *./Users/XXX/Documents* in the same disk where *Windows OS* is installed.

To a better understanding of how to extend the library or the template and basic games themselves, it is needed a deeper understanding of how QGAMES library has been built and works. Please read the book “How to make a 2D / 3D engine for complex games”.

### Tools needed to extend the template

To extend the template there are a number of tools that can (should maybe) be used to facilitate the job. We suggest downloading and installing them before moving forward:

* ***Tiled*** (<https://www.mapeditor.org/>): Probably the most important one. It is needed to build the maps being part of the different scenes and worlds of the game.
* ***TexturePackcageGUI*** (<https://www.codeandweb.com/texturepacker>): Used to create sprite sheets starting from the individual elements.
* ***SpriteFontBuilder (SFB)*** (<https://www.johnwordsworth.com/projects/sprite-font-builder/>): Used to create fonts. It is extremely useful indeed.
* ***GIMP*** (<http://www.gimp.org.es/>): To build the o modify the individual element of both sprites and backgrounds. The sprites built usually are integrated in a sprite sheet, using the **TexturePackageGUI**.

### Installing QGAMES libraries first

Before creating any new game, it is needed to install QGAMES library.

Download it as a .zip file from <https://sourceforge.net/projects/qgames-library/>, and expand the different zips and elements into your preferred folder (usually ./Workpaces/QGAMES). That library has the files needed to compile any new game you want to create based on it. Navigate a little bit across the different folder unfolded, to know in more detail what each of them contains.

QGAMES libraries do not include other required dependencies. Install those ones before moving forward: SDL, OGRE, … See how to install fully QGAME library in its *Read.rtf* file carefully.

Let us take some time to analyse a little bit the structure of directories. It is important to understand it, because the templates that are going to be used, rely on it. You must see a folder called *games*, where the different simple examples are located, and where we also suggest you put your own ones. More complex examples can be found under the folder *samples* as well. Please, go to them after you have totally understood how QGAMES works using the examples.

### Creating basic Dragonwind like games

Add a new project. Select the option *Add a new project*. Then, under the folder *Visual C++* should appear the type *Dragonwind Like Project* (represented with a red icon and a dragon image at the centre). Introduce the name of the game and select the folder under which you want to locate it (we suggest *games* or *samples*). Click *Ok*.

Visual Studio will expand all elements included in the template.

By default, the template will locate *.obj* elements in the folder *../../obj* (taking the game folder as the origin) and the executables in the folder *../../exe*. The template will look for the libraries it needs under the folder ../../lib. If you have located the base of your game in a folder different that “games” or “examples”, please check that the compilation process will find what it is needed correctly.

The template comes with a totally executable example (quite simple), so you just can simply compile the game created and execute it (press F5 e.g.). It should work properly! If not, review the previous step, probably either you will have forgotten something important, or the resolution of the configuration file has not been defined according to your computer standards (see next point).

The template does not include the need runtime libraries (SDL, TinyXML, ZIP. The new executable will try to find them under the new game directory. By default, the template will expect an SDL executable. So, copy the libraries (*.dll* files) from any another example in the folder of the new game.

### Reading Conf.xml

Let us take a look to the *conf.xml* file (in the game’s directory). It will look like such as:

<?xml version="1.0"?>

<Properties>

<!-- General Properties -->

<!-- The name of the game -->

<Property id="GAMENAME" value="Dragonwind"/>

<!-- The icon of the game -->

<Property id="GAMEICON" value="50000"/> <!-- Represents the logo of dragonwind -->

<!-- Where the temporal files are stored in release version, and also the configuration saved -->

<Property id="DATADIR" value="C:/Temp"/>

<!-- When windows is in Full 4K and the scale letters are in other proportion than 1 (take care of this prm) -->

<Property id="XSCALE" value="2.5"/>

<Property id="YSCALE" value="2.5"/>

<!-- Related with the visible screen -->

<Property id="SCREENWIDTH" value="900"/>

<Property id="SCREENHEIGHT" value="600"/>

<Property id="SCREENXPOSITION" value="10"/>

<Property id="SCREENYPOSITION" value="10"/>

<!-- The number of lives -->

<Property id="LIVES" value="3"/>

<!-- Whether to show the FPS -->

<Property id="FPS" value="YES"/>

</Properties>

Many of the attributes defined can be changed to accomplish later your own needs.

All of them are quite self-explained except maybe *XSCALE* and *YSCALE*. SDL libraries (version 2.0) have a bug. They don’t manage windows’ scale properly. So, it is needed to set it externally (using software). Those variables define that scale. Their values should be equivalent to the values set at *Configuration -> System -> Display -> Scale & Distribution:*

Una captura de pantalla de una computadora

Descripción generada automáticamente

Compare what is define here with what appears in the *Conf.xml*. If there is any difference, align the two values, and finally restart the game. Now, it should work if it did not do before properly.

Let us start to understand the different elements of the game and how to modify them.

# Scene and maps definition

The most basic thing to do in a Dragonwind like game is to add a new map.

Consider that any element in a QGAMES project is always identified by a number (id). This number should be also unique within its similar. It means that the number (e.g.) 50000 could be used for either a baddy or for a scene, because the belong to different “similar”. What is not allowed e.g. is to use that number (50000) for two different scenes in two different worlds.

Open *dwworlds.xml*. You will see something like:

<?xml version="1.0"?>

<Worlds>

<!-- Includes -->

<Include file="ptworlds.xml"/>

<!-- For the game to run properly,

all worlds have to have a different id from the scenes, and these from the actions blocks.

The standard actionblocks are from 7000 onwards and from 71000 onwards.

This give us a maximum of 1000 different action blocks per game. -->

<!-- The very basic world.

Used just like a template to create the rest in a real game

It is made up of a single scene -->

<World id="50000">

<Scenes>

<Scene id="60000">

<Actions>

<!-- Type of action related with flying monsters -->

<Action id="70000">

<Attributes>

<Attribute id="TypeOfMonster" value="0"/> <!-- Bees -->

<Attribute id="TimeToAppear" value="2"/> <!-- Seconds -->

</Attributes>

</Action>

<!-- Type of action related with flying monsters -->

<Action id="70001">

<Attributes>

<Attribute id="TypeOfMonster" value="1"/> <!-- Birds -->

<Attribute id="TimeToAppear" value="4"/> <!-- Seconds -->

</Attributes>

</Action>

<!-- Type of action related with the monsters on platforms -->

<!-- Monster in block 1 -->

<Action id="71000"> <!-- @defined at themap.tmx (whatever the map is), parameter 8 block 1 -->

<Attributes>

<Attribuye id="MonitorId" value="50001"/> <!-- @see dwmonitor.xml -->

<Attribute id="Fire" value="YES"/>

<Attribute id="TypeOfFire" value="0,6"/> <!-- Arrow ot star flying -->

<Attribute id="ImpulseOfFire" value="-3"/>

<Attribute id="SecondsToFire" value="5"/>

<Attribute id="SecondsBetweenFires" value="0.5"/>

<Attribute id="SecondsFiring" value="5"/>

</Attributes>

</Action>

<!-- Type of action, consisting in an environment condition -->

<Action id="72000" active="NO">

<Attributes>

<Attribute id="TypeCondition" value="4"/>

<Attribute id="SecondsToStart" value="1"/>

<Attribute id="SecondsCondition" value="-1"/> <!-- No end -->

<Attribute id="RainDensity" value="0.35"/>

<Attribute id="RainBackgroundId" value="50160"/>

<Attribute id="CloudsDensity" value="2"/>

<Attribute id="LightningsProbability" value="95"/>

<Attribute id="LightningsFrecuency" value="1"/>

</Attributes>

</Action>

</Actions>

<Attributes>

<Attribute id="FUNDAMENTAL" value="YES"/>

<Attribute id="NAME" value="Scn Example"/>

<Attribute id="MOVEMENT" value="1,0,0"/> <!-- To indicate the axis the scene moves towards -->

<Attribute id="BACKGROUNDMUSIC" value="50002"/> <!-- The music in the background if it is on -->

<Attribute id="LIQUIDSFLUIDITY" value="6.50"/> <!-- Nothing is 1!! -->

</Attributes>

<Conexions>

<Conexion id="0" with="60001" at="0"/>

</Conexions>

<Maps>

<!-- The backgrounds -->

<Map id="50100" file="maps/BKDessert.xml"/>

<Map id="50110" file="maps/BKDarkForest.xml"/>

<Map id="50120" file="maps/BKCity.xml"/>

<Map id="50130" file="maps/BKLandscape.xml"/>

<Map id="50140" file="maps/BKMountains.xml"/>

<Map id="50150" file="maps/BKTemple.xml"/>

<!-- And a tile map just like an example again -->

<Map id="60000" file="maps/Example.tmx"/>

</Maps>

<Entities>

<!-- Over which layer have the entities to be located? -->

<Entity id="50100" layer="Solid\_1"/> <!-- The male main ninja -->

<Entity id="50101" layer="Solid\_1"/> <!-- The female main ninja -->

<SetOfEntities fromId="51000" number="110" layer="Solid\_1"/> <!-- The shootings -->

<SetOfEntities fromId="30000" number="200" layer="Solid\_1"/> <!-- The food -->

<SetOfEntities fromId="31000" number="200" layer="Solid\_1"/> <!-- Things to be caught -->

<SetOfEntities fromId="50110" number="10" layer="DecorationFront\_1"/> <!-- The bees -->

<SetOfEntities fromId="50120" number="10" layer="DecorationFront\_1"/> <!-- The birds -->

<SetOfEntities fromId="32000" number="10" layer="Solid\_1"/> <!-- The bad guys -->

</Entities>

</Scene>

<!-- Just to show how two scenes connect -->

<Scene id="60001">

<Actions>

<Action id="71001">

<Attributes>

<Attribuye id="MonitorId" value="50002"/>

<Attribute id="Fire" value="YES"/>

<Attribute id="TypeOfFire" value="0"/>

<Attribute id="ImpulseOfFire" value="1"/>

<Attribute id="SecondsToFire" value="5"/>

<Attribute id="SecondsBetweenFires" value="0.5"/>

<Attribute id="SecondsFiring" value="5"/>

</Attributes>

</Action>

<Action id="71002">

<Attributes>

<Attribuye id="MonitorId" value="50003"/>

<Attribute id="Fire" value="YES"/>

<Attribute id="TypeOfFire" value="1"/>

<Attribute id="ImpulseOfFire" value="10"/>

<Attribute id="SecondsToFire" value="5"/>

<Attribute id="SecondsBetweenFires" value="2"/>

<Attribute id="SecondsFiring" value="4"/>

</Attributes>

</Action>

</Actions>

<Attributes>

<Attribute id="FUNDAMENTAL" value="YES"/>

<Attribute id="NAME" value="Scn Example Next"/>

<Attribute id="MOVEMENT" value="1,0,0"/>

<Attribute id="BACKGROUNDMUSIC" value="50002"/>

<Attribute id="LIQUIDSFLUIDITY" value="5.50"/>

</Attributes>

<Conexions>

<Conexion id="1" with="60000" at="2"/>

</Conexions>

<Maps>

<Map id="50100" file="maps/BKDessert.xml"/>

<Map id="50110" file="maps/BKDarkForest.xml"/>

<Map id="50120" file="maps/BKCity.xml"/>

<Map id="50130" file="maps/BKLandscape.xml"/>

<Map id="50140" file="maps/BKMountains.xml"/>

<Map id="50150" file="maps/BKTemple.xml"/>

<Map id="60001" file="maps/ExampleNext.tmx"/>

</Maps>

<Entities>

<Entity id="50100" layer="Solid\_1"/>

<Entity id="50101" layer="Solid\_1"/>

<SetOfEntities fromId="51000" number="110" layer="Solid\_1"/>

<SetOfEntities fromId="30000" number="200" layer="Solid\_1"/>

<SetOfEntities fromId="31000" number="200" layer="Solid\_1"/>

<SetOfEntities fromId="50110" number="10" layer="DecorationFront\_1"/>

<SetOfEntities fromId="50120" number="10" layer="DecorationFront\_1"/>

<SetOfEntities fromId="32000" number="10" layer="Solid\_1"/>

</Entities>

</Scene>

</Scenes>

<Attributes>

<Attribute id="NAME" value="Wrld Example"/>

</Attributes>

</World>

</Worlds>

These are the very basic world and scene example included in the template.

The number of the world example is 50.000, and the first scene example is the number 60.000. They are configured to be the first world and scene executed (see *conf.xml* file). Many others can be later defined, rewritten / extending the file *dwworlds.xml* file.

What is defined in this file example has to aligned with the information of the file *Example.tmx*. Open the first one using *Notepad++* and the second one using *Tiled*, to have a comprehensive understanding of them.

As it has been introduced at the beginning, every scene in the game must have a unique id. There is no reserved numbers for them, except the number 60.000 that is for a *DRAGONWIND::SceneExample* instance.

Let us review what are the parameters that define the scene:

## Attributes

Any scene is defined by a set of attributes (under the XML paragraph *<Attributes>*). Those are:

* **FUNDAMENTAL**: YES or NO. NO by default. To define whether the scene is fundamental in the game. It means that worlds’ and scenes’ name are announced first time the player enters them.
* **NAME**: A string. “---” by default. To define the name of the scene.
* **MOVEMENT**: A vector. “1,0,0” by default. To define the direction the scene progresses, and it is also used to move the background map if defined (The background is defined by code).
* **BACKGROUNDMUSIC**: A reference to an id defined in the *music.xml* file structure. Nothing by default, what means no background sound will be played even if music is on.
* **LIQUIDFLUIDITY**: A number above 1. It defines the density of the liquids in the scene. 4.50 would mean “normal” water. The bigger the value is, the thicker the liquid is, and the lesser the value is the more fluid the liquid will be.

It is also possible to define your own attributes, but it will be necessary to create a new class and maybe specific methods to read and interpret them as well.

However all values are stored in the attribute *\_properties* (std::map <std::string, std::string>) of the QGAMES::Scene class (parent of any). That class also adds methods to manage that parameters:

*bool existsProperty (const std::string& n)*, to verify whether an attribute exists or not, and

*std::string property (const std::string& n)*, to get the value of the parameter (error when the parameter does not exist). Notice that the value returned is always an string, so it should move into other type if needed.

In the default template all Worlds created (regardless its id) are DRAGONWIND::World instances, and all Scenes created are DRAGONWIND::PlayingScene instances except the one for the id 60.000 that is a *DRAGONWIND::SceneExample* (inheriting from *DRAGONWIND::PlayingScene*). This example adds simple things (mainly as an example for programmers), and usually the parent is enough for many uses.

It will be analysed later how to manage that situations.

## Action blocks

Any scene in Dragonwind is made up of a set of blocks of actions (under   
XML paragraph *<Actions>*).

There can be several different types of them, and you can even create your own ones. This specifically will be covered later. The type of an action block is self – defined by its id. The world builder must understand them (in the method *createSceneActionBlockObject*).

The template comes with 5 types of action blocks already implemented:

* Blocks from 70.000 to 70.999 are reserved to control **flying monsters** (such as bees or birds).
* Blocks from 71.000 to 71.999 are to control **how enemies move and shoot in the scene** (dogs, cats, robots, or jacks).
* Blocks from 72.000 to 72.999 are to **control environmental conditions** (thunders, lightings, rain, wind, or storms,).
* Blocks from 73.000 to 73.999 are to control **how a layer or a set of those appear or disappear** in the scene and, finally
* Blocks from 74.000 to 74.999 are to control **how a layer or a set of layers moves**.

Notice that the scene example has one block of flying monsters and one more to manage an enemy. A scene could have more enemies (more dogs…), but they wouldn’t be controlled through any action block if it is not defined here.

An action block is always a set of attributes. The meaning of each attribute depends on the type of action block. So, in the case of an action block defining a flying monster those attributes are the following ones:

* **TypeOfMonster**: A positive integer number (or cero). 0 by default. 0 means a flock of bees. 1 means a flock of birds.
* **TimeToAppear**: A positive real number, representing seconds. 2 by default. Time to appear in scene.

In the case of one defining an enemy, it is a little bit more complex:

* **Fire**: YES or NO. NO by default. To point whether the enemy fires or not.
* **TypeOfFire**: A number. 0 by default. -1 would mean random among the ones allowed for a character. Makes sense only when the previous value is YES. To define what the enemy will fire. The possible values for this parameter are the following ones. New types can be defined.
  + **10**: An arrow flying and disappearing when collisions against something.
  + **16**: A star flying and disappearing when collisions against something.
  + **11**: A bomb that falls, bounces and finally explode.
  + **20**: A pool ball that falls, doesn’t bounce and rolls along a surface (and fall again if there is no platforms under).
* **ImpulseOfFire**: An integer, bigger than 0. The force the fire is moved. -1 by default, meaning a random force. Again, it makes only sense when the parameter Fire is YES.
* **SecondsToFire**: A number. Bigger than 0. How long takes to start to fire. In seconds. 2 seconds by default. Makes sense if Fire is YES.
* **SecondsBetweenFires**: 2 seconds by default. Defines how long to wait between two consecutives shootings.
* **SecondsFiring**: 2 seconds again by default. How long shooting, before stopping. After that time, the SecondsToFire variable takes back the control.
* **MonitorId**: A reference to an id define in the *monitor.xml* structure. The monitor is a set of steps to control how one character moves automatically. A set of monitor types have been defined for a Dragonwind like games, but more can be added easily.

You can define your own blocks but you will have then to extend the builder *DRAGONWIND::WorldBuilder*, and extend also *DRAGONWIND::Game* to redefine the method *createWorldBuilder* to create that new builder instead the default one. Let’s see later how to do so. Yourr game should then inherit from the new game class defined.

## Connections

The scenes can be connected to others through out the connections. When a player reaches a specific type of point (we will see how to define those points in the map later), the connection process is then executed, ending the current scene and starting a new one at the position pointed by the connection definition.

To connections are defined in the XML paragraph *<Conexions>*. To define a connection *<Conexion>* the following attributes are needed:

* **id**: To represent the connection
* **with**: The id of the scene to connect to.
* **at**: the connection point at the destination scene, where the character will appear.

The scene example has no connections defined. If when reaching a connection there is no other one defined, the game finishes and the player wins. So, in this scene example, after reaching the goal the game finishes.

# Map definition

The scene must include a reference to one (or many) map. In a Dragonwind like game two types of maps can be referred: background maps and play maps. They are referred using its id (a reference to the identification located in *maps.xml* structure) and the name of the file where its full definition is. Open also the file *dwmaps.xml* to understand a little bit more how to define maps.

## Background maps

A background map is all what is drawn behind the platforms where the game is taking place and it usually moves when the player advances through the maze. It can be made up of several layers, as it is the case in the scene example. To define a background map no special tool is needed, just the *notepad++* or *Visual Studio IDE*.

There are a couple of background maps defined for the scene example and just one play map. This is because the scene example (DRAGONWIND::SceneExample) choose random one when the game starts (it is one of the simple things added over the default behaviour defined in DRAGONWIND::PlayingScene).

Searching them at the file *dwmaps.xml*:

<?xml version="1.0"?>

<Maps>

<!-- Includes -->

<Include file="ptmaps.xml"/>

<!-- The standard background maps -->

<Map id="50100" type="10000" file="maps/BKDessert.xml"/>

<Map id="50110" type="10000" file="maps/BKDarkForest.xml"/>

<Map id="50120" type="10000" file="maps/BKCity.xml"/>

<Map id="50130" type="10000" file="maps/BKLandscape.xml"/>

<Map id="50140" type="10000" file="maps/BKMountains.xml"/>

<Map id="50150" type="10000" file="maps/BKTemple.xml"/>

<!-- -->

<!-- Just an example map to show the examples -->

<Map id="60000" type="1" file="maps/Example.tmx"/>

<!-- -->

</Maps>

The background maps are type 10.000. Type 10.000 means an Object Map. They are made up of different simple elements (circles, squares, lines, polygons and backgrounds).

Go to *./maps/* at the game folder and open one of the background map definition file (e.g. *BKCity.xml*):

<?xml version="1.0"?>

<!-- Background map city -->

<Map name="50120" width="3200" height="600">

<Layers>

<Layer id="0" name="background">

<Element id="50120" X="0" Y="0" Z="0"/>

<Attributes>

</Attributes>

</Layer>

</Layers>

<Attributes>

<Attribute id="BASE" value="470"/>

</Attributes>

</Map>

There is a simple layer with just one element being part of it. The one with the id 50120. Open the file *dwobjects.xml* and go to the block title named *The city*.

…<!-- The City -->

<Background id="50120" form="50120" times="2">

<Attributes>

<Attribute id="FADEGRADE" value="255"/>

<Attribute id="AUTOMATICMOVE" value="YES"/>

<Attribute id="MOVDIRECTION" value="1,0,0"/>

<Attribute id="SPEED" value="0"/>

<Attribute id="PIXELS" value="1"/>

<Attribute id="LINKEDBK" value="50121"/>

<Attribute id="SPEEDLINKEDBK" value="2"/>

<Attribute id="ROLL" value="YES"/>

</Attributes>

</Background>

<Background id="50121" form="50121" times="2">

<Attributes>

<Attribute id="FADEGRADE" value="255"/>

<Attribute id="AUTOMATICMOVE" value="YES"/>

<Attribute id="MOVDIRECTION" value="1,0,0"/>

<Attribute id="SPEED" value="0"/>

<Attribute id="PIXELS" value="1"/>

<Attribute id="LINKEDBK" value="50122"/>

<Attribute id="SPEEDLINKEDBK" value="2"/>

<Attribute id="ROLL" value="YES"/>

</Attributes>

</Background>

<Background id="50122" form="50122" times="2">

<Attributes>

<Attribute id="FADEGRADE" value="255"/>

<Attribute id="AUTOMATICMOVE" value="YES"/>

<Attribute id="MOVDIRECTION" value="1,0,0"/>

<Attribute id="SPEED" value="0"/>

<Attribute id="PIXELS" value="1"/>

<Attribute id="LINKEDBK" value="50123"/>

<Attribute id="SPEEDLINKEDBK" value="2"/>

<Attribute id="ROLL" value="YES"/>

</Attributes>

</Background>

<Background id="50123" form="50123" times="2">

<Attributes>

<Attribute id="FADEGRADE" value="255"/>

<Attribute id="AUTOMATICMOVE" value="YES"/>

<Attribute id="MOVDIRECTION" value="1,0,0"/>

<Attribute id="SPEED" value="0"/>

<Attribute id="PIXELS" value="1"/>

<Attribute id="LINKEDBK" value="50124"/>

<Attribute id="SPEEDLINKEDBK" value="2"/>

<Attribute id="ROLL" value="YES"/>

</Attributes>

</Background>

<Background id="50124" form="50124" times="2">

<Attributes>

<Attribute id="FADEGRADE" value="255"/>

<Attribute id="AUTOMATICMOVE" value="YES"/>

<Attribute id="MOVDIRECTION" value="1,0,0"/>

<Attribute id="SPEED" value="0"/>

<Attribute id="PIXELS" value="1"/>

<Attribute id="LINKEDBK" value="50125"/>

<Attribute id="SPEEDLINKEDBK" value="2"/>

<Attribute id="ROLL" value="YES"/>

</Attributes>

</Background>

<Background id="50125" form="50125" times="2">

<Attributes>

<Attribute id="FADEGRADE" value="255"/>

<Attribute id="AUTOMATICMOVE" value="YES"/>

<Attribute id="MOVDIRECTION" value="1,0,0"/>

<Attribute id="SPEED" value="0"/>

<Attribute id="PIXELS" value="1"/>

<Attribute id="LINKEDBK" value="50126"/>

<Attribute id="SPEEDLINKEDBK" value="0"/>

<Attribute id="ROLL" value="YES"/>

</Attributes>

</Background>

<Background id="50126" form="50126" times="1">

<Attributes>

<Attribute id="FADEGRADE" value="255"/>

<Attribute id="AUTOMATICMOVE" value="YES"/>

<Attribute id="MOVDIRECTION" value="0,0,0"/> <!-- Sun doesn't move -->

<Attribute id="SPEED" value="0"/>

<Attribute id="PIXELS" value="1"/>

<Attribute id="LINKEDBK" value="50127"/>

<Attribute id="SPEEDLINKEDBK" value="0"/>

<Attribute id="ROLL" value="YES"/>

<Attribute id="FIX" value="YES"/>

</Attributes>

</Background>

<Background id="50127" form="50127" times="1">

<Attributes>

<Attribute id="FADEGRADE" value="255"/>

<Attribute id="AUTOMATICMOVE" value="YES"/>

<Attribute id="MOVDIRECTION" value="0,0,0"/> <!-- Last Background (orange) doesn't move -->

<Attribute id="SPEED" value="0"/>

<Attribute id="PIXELS" value="1"/>

<Attribute id="ROLL" value="NO"/>

<Attribute id="FIX" value="YES"/>

</Attributes>

</Background>…

You see a list of background elements linked each other with different speeds, what produces a background effect with different layers.

Notice that each background refers to a form defined in the *forms.xml* structure.

## Play maps

A play map is usually where the platforms, blocks, water, etc.. are defined. It is more complex to define it, so more time will be dedicated later to understand it better.

They are type 1. Type 1 means maps defined following the standard TMX (<https://doc.mapeditor.org/en/stable/reference/tmx-map-format/>).

To manage a play map a tool such as *Tiled* is always recommended.

The map is made up of different layers. Every layer has always a name. The map is always drawn from the layer at the bottom to the top one defined in the list. Depending on how the name has been written, it associated attributes will be ones or others. The words than can be included in the different names are the following ones:

### Base

A layer with tiles that the characters of the game can walk over. Valid names could be *Base\_1*, *MainBase*, etc. Probably this is the most important type of layer.

This type of layers can include some attributes to move it. They are different depending on the type of movement.

For linear movements:

* **DIRECTION**: A vector to indicate the direction of the movement and the size of every step. Makes sense only when the type of movement (see TYPEMOV for more details) is a linear one. So, e.g a value of 0,-2,0 will indicate that the layer will move up two pixels per step.
* **LENGTH**: The length of the movement (in steps). Following the example, 96 steps of 2 pixels each, will imply that the layer will move 192 pixels, or 8 tiles if each one is 24x24 pixels (as it is the case of the examples included).
* **SPEED**: The time, in loops, that the game must wait before moving one step.
* **TYPEMOV**: The type of linear movement. The standard types of linear movements defined are:
  + **0**: Straight movement. Goes to DIRECTION and after LENGTH steps, stops.
  + **2**: Straight cyclical movement. Goes to DIRECTION and then back, and so on and so forth.

For Circular movements, SPEED and TYPEMOVE are also needed, and additionally:

* **RADIX**: The radix, in pixels, of the circle containing the movement.
* **INITANGLE**: The angle (in radians, from 0 to PI) initial of the movement.
* **ANGLE**: The angle to move.
* **CLOCKWISE**: TRUE or FALSE. To point whether the movement takes place following the clock hands or the other way around.
* **TYPEMOV**: The type or circular movement. The standard types of circular movements defined are:
  + **1**: Simple circular movement. The layer moves ANGLE starting from INITANGLE, and when it reaches it, finishes.
  + **3**: Same than the previous, back going back.
  + **4**: Cyclic. It has been though for pure total circular movement.

Other types of movements can be added, even those made up of the composition of others. It will be also analysed later.

Really any layer can have associated a movement. It can be added by code always. But in the case of Dragonwind layers named Base the movement is added automatically after reading the attributes.

### Solid

A layer with solid tiles that no character can pass through. Valid names could be *Solid\_1*, *BackSolid*, etc.

### Liquid

A layer with tiles that can move like water and where the main character can swim into. Valid names could be *Liquid\_1*, *PossionLiquid*, etc.

The liquid moves. Two types of movements can be added. The type of movement must be defined in a variable called **TYPEUPDATE**.

When 1, a simulation of flowing water will be implemented, and the next other attributes will be also needed:

* **TILES**: The list of the forms used to simulate the movement. In the loop that the movement must take place every tile frame of the list will be changed by the next in it, and the last one will be changed by the first in the list. By default the list is empty.
* **LOOPS**: Number of loops for the movement to take place. By default the value is 60. It means that the actualization will take place every 1 second (if the frame rate has been set to 60 frames per second).

When 2, a more complex simulation takes place. The movement will simulate a wave. Other attributes will be need, apart of also the previous ones (TILES and LOOPS). Each tile implicated in the movement has associated a speed value. Speed value is incremented every Xs loops (LOOP) by the ACCELERATION. When the speed reaches a maximum value (SPEED), the speed will be reduced instead by the ACCELERATION up to 0 and so on and so forth.

* **SPEED**: Maximum speed. As any tile frame cannot be ever greater than the last one in the list, the greater this value is, the more time the wave will stay in the bottom or in the top, and lesser realistic the effect will be maybe. By default, the value is 1.
* **ACCELARATION**: The value the speed is incremented per loop. The greater the value is, the quicker the wave moves. The default value is 0.5, but a value around 0.35 produces a nice effect.

Tile Updater can be manually created and added to any layer, but Dragonwind games adds automatically one out of those two if the parameters are well defined.

In Dragonwind games, one form is mainly are used in water layers: Water.png. Open it. You will see that the one tile is a little bit smaller than the previous one. Except the two first. The are equal. Use always 1 to draw the top part of the moving water, and 0 to draw the rest.

### Ladder

A layer with tiles representing things that can be used to climb by the main character. Valid names could be *Ladder\_1*, *UpLadder*, etc.

### Decoration

Just tiles to decorate. Valid names could be *DecorationFront*, *TreesDecoration*, etc.

Additionally, two other layers must be defined. They are critical in any Dragonwind like game because they define where important points of the game are located. None of both are visible when the game takes place. These two layers are (and with exactly that name):

### Notification

The tiles pointed here generate important events when the main character touches them: e.g. A goal has been reached, an intermediate point has been reached, a switch to open a hidden door has been pressed, etc.

In the default implementation of the library, the use of the form *Notification.png* is recommended for this layer. You will see that the form is grouped in blocks of 10 tiles named from the A to the J each and with different background colours.

Only the two first groups has a defined use by default:

* **First block** (background colour black) is to point connections points among scenes. The tile A will be equivalent to the connection point number 0, and so on and so forth. So, in a single scene there couldn’t be more than 10 connection points defined.

If any of these tiles are included in the scene, a connection attribute should be defined in the worlds.xml file to indicate where the connection goes. As follows:

<Connections>

<Connection id=”0” // This is the number of the connection (the A)

with=”XX” // Here the id of the scene where to go

at=”YY” // The id of the location (see next layer) where the main character should appear/>

</Connections>

* **Second block** (background colour brown) is for issuing events when the main character pass over them. The default behaviour of this brown tiles is to change the starting point of the main character in the scene if he / she dies there. So, if a brown notification tile named B is put into the scene, there must a green location tile named also B (maybe near) to indicate where the main character will appear back if dies.

### Location

The tiles here point places to locate characters: Where to initialize locate an enemy, or the main character when enter first a scene, or tools, etc.

In a similar way than in the case of the layer notification, the use of the form Location.png is strongly recommended. This form Is also divided in blocks of then tiles each. Their uses are:

* **First block**: With a blue background and named from the A to the J. They are used to locate enemies, with the following meaning:
  + **A**: Cat.
  + **B**: Dog.
  + **C**: Robot.
  + **D**: Jack.

The rest are not used. We will see later how to add new enemies in the game.

* **Second block**: With a green background and named from the A inverse to the J inverse. They are used to locate tools, with the following meaning:
  + **A inverse**: Arrow.
  + **B inverse**: Bomb.
  + **C inverse**: Coin.
  + **D inverse**: Heart.
  + **E inverse**: Paper.
  + **F inverse**: Question mark.
  + **G inverse**: Star.
  + **H inverse**: Thunder.
* **Third block**: With different symbols and backgrounds. It is used to locate pieces of food. The power of the piece will be different depending on the background colour, and similar thing regarding the behaviour depending on the symbol of the tile. The specific piece of food (its aspect) shown in the game is chosen random. The meaning of the background colours is:
  + **Blue**: simple power.
  + **Orange**: Double power.
  + **Red**: Triple power.

And the meaning of the symbol:

* + **Star**: A piece of food that falls until a platform.
  + **Item**: A piece of food that stays in the air.

These all are the type of tile layers that a Dragonwind like game admits, but another type of layer can be added to describe properties of a zone. The layer has always to contain the name ObjectZone.

### ObjectZone

The ObjectZone (there must be just only 1), must define squared zones containing a location tile for an enemy. and assign an attribute to each of them. Those attributes are:

* **0**: Can the character be injured by its enemies? Whether someone is or not an enemy is defined in the method: DRAGONWIND::DragonArtist::isEnemy. YES or NOT. By default, YES.
* **1**: Can the character be injured by its congeners? If someone is not an enemy it is a congener. YES or NOT. By default, NO.
* **2**: Can the character be injured by flying monsters? YES or NOT. By default, YES.
* **3**: Energy lost per cycle. A real number, between 0 and 100. Percentage of energy that the monster loses per cycle. By default, 1/ frames-per-second. It means that in 1 minute it will lose all energy.
* **4**: Points given to the enemy who hit him/her the most when dies. An integer, By default, 10.
* **5**: Shooting energy. An integer with the energy of him/her shootings. This energy will be decremented (if possible) from an enemy when he/she is hit. By default, 5.
* **6**: Action block id: A reference to the id of the action block that manages the enemy (see action block definition). By default, -1, what means that no action block is controlling the enemy.

So, to fully define an enemy in a scene, follow the next steps:

1. In the Location layer, insert a tile to define the type of monster and its initial position.
2. Round the tile with a square located in a layer including the text ObjectZone.
3. Define the attributes above for that square.
4. Define an action block to control enemy’s movement.
5. Link action block and properties through out the attribute 6 of the square of the ObjectZone.

# Presentation

Any Dragonwind game can define its presentation in a file belonging to the structure *gamestates.xml*. In that file a set of instances of the class *QGAMES::GameState* (or others inheriting from it) must be defined. How they flow is also defined in that file. Open the file *dwgamestates.xml* to see how it looks like:

<?xml version="1.0"?>

<GameStates>

<!-- Includes -->

<Include file="ptgamestates.xml"/>

<!-- The usual controlling form -->

<GameState id="50000" type="50101" name="Controller" main="YES">

<FlowConditions>

<FlowCondition on="50002" when="0" to="50300"/> <!-- After the game is loaded, the game is initialized -->

<FlowCondition on="50300" when="0" to="50003"/> <!-- After the game is initialized the logo of the owner appears -->

<FlowCondition on="50003" when="0" to="50004"/> <!-- After the loogo appears, then the introduction starts -->

<FlowCondition on="50004" when="0" to="50009"/> <!-- After introduction, the menu is shown -->

<FlowCondition on="50009" when="0" to="50014"/> <!-- When the play option is selected, then playing starts asking for the name of the player -->

<FlowCondition on="50009" when="1" to="50028"/> <!-- When the option selected is exit -->

<FlowCondition on="50009" when="2" to="50025"/> <!-- When the option selected is hall of fame -->

<FlowCondition on="50009" when="3" to="50029"/> <!-- When the option selected is instructions -->

<FlowCondition on="50014" when="0" to="50100"/> <!-- After asking for the name of the player, then the game starts showing the name of the wotld and scene -->

<FlowCondition on="50100" when="0" to="50019"/> <!-- After dieing when playing, the official label is shown -->

<FlowCondition on="50100" when="1" to="50100"/> <!-- After reaching a goal when playing, start back if there is more scenes to deal with -->

<FlowCondition on="50100" when="2" to="50009"/> <!-- After selecting (while playing) the game finishes and the menu is back -->

<FlowCondition on="50100" when="3" to="50400" /> <!-- After reaching a goal, there is no more scene to deal with so the player wins!! -->

<FlowCondition on="50019" when="0" to="50014"/> <!-- After die, the game starts back -->

<FlowCondition on="50019" when="1" to="50022"/> <!-- ..or the game is over -->

<FlowCondition on="50022" when="0" to="50025"/> <!-- After game is over, the hall of fame appears -->

<FlowCondition on="50025" when="0" to="50009"/> <!-- After hall of fame to menu -->

<FlowCondition on="50029" when="0" to="50009"/> <!-- After the instructions the menu is back -->

<FlowCondition on="50400" when="0" to="50022"/> <!-- After showing that the player has won, then game over -->

</FlowConditions>

<Attributes>

<Attribute id="PAUSEGAMESTATE" value="50001"/> <!-- A reference to the nested state when nested -->

</Attributes>

</GameState>

<!-- The pause game.

It is just a state showing an entity in the middle -->

<GameState id="50001" type="10040" name="Pause" main="YES"> <!--Type ENTITY Shown -->

<Attributes>

<Attribute id="ENTITYID" value="10000"/> <!-- The entity used when pause -->

<Attribute id="FADE" value="255"/>

<Attribute id="CONTROLPOSITION" value="YES"/>

<Attribute id="LOOPSPERFADECHANGE" value="3"/>

<Attribute id="POSITION" value="0.5,0.5,0"/>

<Attribute id="POSREFERENCE" value="0"/>

</Attributes>

</GameState>

<!-- To load the game.

One of the houses used as background in many scenes full filled little by little -->

<GameState id="50002" type="10050" name="0Loading" main="YES"> <!-- Type LOADING -->

<Attributes>

<Attribute id="FORMID" value="50002"/> <!-- The big sun -->

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.5,0"/>

<Attribute id="POSREFERENCE" value="0"/>

<Attribute id="REVERSE" value="NO"/>

<Attribute id="HORIZONTAL" value="YES"/>

</Attributes>

</GameState>

<!-- To initialize the game -->

<GameState id="50300" type="50102" name="1Initializing" main="YES"/> <!-- Type INITIALIZING -->

<!-- To present the game.

Just the commty logo comming to live little by little -->

<GameState id="50003" type="10030" name="2CommtyLogo" main="YES"> <!-- Type FORM Shown -->

<Attributes>

<Attribute id="FORMID" value="10002"/> <!-- The Big CommtyLogo -->

<Attribute id="INITIALFADE" value="0"/>

<Attribute id="FINALFADE" value="255"/>

<Attribute id="FADEINCREMENT" value="5"/>

<Attribute id="LOOPSPERFADECHANGE" value="2"/>

<Attribute id="POSITION" value="0.5,0.5,0"/> <!-- In the middle -->

<Attribute id="POSREFERENCE" value="0"/> <!-- Reference to the center -->

</Attributes>

</GameState>

…

Any game state can be defined by a set of Attributes (XML paragraph *<Attributes>*).

The definition can also include an XML *<Nested>* paragraph, to define one (and only one) nested game state. If defined, the nested game state will be included into its parent and initialized, updated, drawn and finalized when the parent is, and always before it. So, it is possible to build complex game states using this feature and other fundamental game states (or even complex).

Other possibilities (more than attributes and one nested game state) can be added to a game state depending on its type.

To build a game state a builder must be used. That builder must inherit from *QGAMES::GameStateBuilder* and include all add – on needed to understand what is defined in the *gamestate.xml* file. The default game states builder for a Dragonwind like game is set in the method: *DRAGONWIND::Game::createGameStateBuilder*.

Any game state must define the following parameters:

* **id**: Something unique in the system.
* **type**: The type. It will determine the add – on used to build the game state actually.
* **name**: The name of the game state.
* **main**: Whether the game state is a main one or not. It is supposed than the No main will be nested in others.

The game states can be created using code as well, but a definition file (such as *dwgamestates.xml*) can be a simpler way to do so and, if needed, modify their behaviour without recompiling the code.

The fundamental game states are:

## Exit. Type 7000

Macro for type: *\_\_QGAMES\_EXITGAMEGAMESTATE\_\_*

class QGAMES::NULLGameState

Just exit the game

## Null. Type 10025

Macro for type: *\_\_QGAMES\_GAMESTATENULLTYPE\_\_*

class QGAMES::NULLGameState

Just issue an event with the code number 7016 (*\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*). The data of the event is a pointer to the GameState.

## Waiting time. Type 10000

Macro for type: *\_\_QGAMES\_GAMESTATEWAITINGTIMETYPE\_\_*

class QGAMES::SpendingTimeGameState

Spend time. The attributes needed are:

* **TIME**: The time in seconds to wait before issuing the event 7016 (*\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*).

## Waiting until enter key or mouse left button is pressed. Type 10001

Macro for type: *\_\_QGAMES\_GAMESTATEUNTILENTERTYPE\_\_*

class QGAMES::ForEverUntilIntroKeyOrLeftButtonMousePressedGameState

Wait until the referred keys or mouse buttons are pressed, issuing the event 7016 (*\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*). No attributes are needed.

## Waiting until a key or any mouse button is pressed. Type 1002

Macro for type: *\_\_QGAMES\_GAMESTATEWAITORUNTILKEYTYPE\_\_*

class QGAMES::SpendingTimeOrWaitingUntilAnyKeyPressedGameState

Waiting until the referred keys or buttons are pressed, issuing then the event number 7016 (*\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*). No attributes are needed.

## Waiting a couple of seconds or until a key or mouse button is pressed. Type 10006

Macro for type: *\_\_QGAMES\_GAMESTATESPENDUNLESSKEYTYPE\_\_*

class QGAMES::SpedingTimeUnlessKeyIsPressedGameState

Spend time unless a key o mouse button is pressed, issuing then the event number 7016 (*\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*). The only attribute needed is TIME.

## To Show a Text. Type 10003

Macro for type: *\_\_QGAMES\_GAMESTATESHOWSTDTEXTTYPE\_\_*

class QGAMES::ShowingStandardTextGameState

A fixed text is shown. The state never finishes, so it must be nested in one of the previous ones (e.g.) to control how long the text is the screen.

The attributes needed to define it are:

* **TEXT**: The text to show. No default values.
* **FONT**: The id of the font. The fonts are built using the Font Builder. The font builder is created in the method: *QGAMES::AdvancedGame::createTextBuilder*. See the appendix for a reference with all standard fonts considered in a Dragonwind like game. It is easy, by the way, to add others. No default values.
* **FADE**: The fade value for the text. From 0 to 255.
* **POSITION**: A set of three variables separated by comma and from 0 to 1. They indicate the reference point to locate the text. 0.5, 0.5, 0 means the centre of the screen. 0,0,0 the left – up corner, and 1,1,0 the right – bottom of the screen. By default, the centre.
* **POSREFERENCE**: How to interpret that initial position. 0 means that the centre of the text will be locate at the previous position, 1 that the left – up corner of the text will be there, and 2 the right – bottom corner of the text. By default, left – up.
* **ORIENTATION**: A vector to indicate the orientation of the text. 1,0,0 means to the right. By default, to the right.

## To show the number of the player. Type 10004.

Macro for type: *\_\_QGAMES\_GAMESTATESHOWPLAYERNUMTYPE\_\_*

class QGAMES::ShowingCurrentPlayerNumberGameState

Like the previous one but adding the number of the player to the text defined.

## To show the FPS: Type 10005.

Macro for type: *\_\_QGAMES\_GAMESTATEFPSTYPE\_\_*

class QGAMES::ShowingFrameRateGameState

Like the previous ones but adding the current FPS before the text defined. This game state can be useful to see how the game is performing. The example included in the template uses this game state (press F1 in any moment to switch on / off the FPS).

## To play a sound. Type 10010

Macro for type: *\_\_QGAMES\_GAMESTATEPLAYSOUNDTYPE\_\_*

class QGAMES::PlayingASoundGameState

To play a sound. The game state never finishes. So, it should be part of another game state with controlling features.

The attributes needed are:

* **SOUNDID**: A reference (id) to a sound defined in the *sounds.xml* structure. By default, 0. Probably it will produce a bug.
* **CHANNELID**: Number of the channel used to emit the sound. By default, 0.

## To show a fixed form in the screen. Type 10020

Macro for type: *\_\_QGAMES\_GAMESTATESHOWFIXFORMTYPE\_\_*

class QGAMES::ShowingFixFormGameState

To show a form in the screen. The game state never finishes. So, it should be included as part of other having controlling features.

The parameters are:

* **FORMID**: A reference to the form to show. Defined in the *forms.xml* structure. By default, 10002 that is defined to be a big logo of Community Games.
* **FADE, POSITION and POSREFERECE** with the same meaning than in the case of showing a text.

## To show a form that appears or disappears. Type 10030

Macro for type: *\_\_QGAMES\_GAMESTATESHOWFORMTYPE\_\_*

class QGAMES::ShowingAppearingFormGameState

Is like the previous one, but the fade is not constant. The parameters are:

* **INITAILFADE**: The initial fade of the form. From o to 255. By default, 255.
* **FINALFADE**: The final fade. From 0 to 255. By default, 255.
* **FADEINCREMENT**: How many the fade must be incremented (or decremented if it is negative). By default, 10.
* **LOOPSPERFADECHANGE**: How many loops takes place before the fade is incremented (or decremented).
* **FORMID, POSITION and POSREFERENCE**, with the same meaning than above.

## To show an entity. Type 10040

Macro for type: *\_\_QGAMES\_GAMESTATESHOWENTITYTYPE\_\_*

class QGAMES::ShowingEntityGameState

To show an entity. The entities move and have much more possibilities than a simple form. The game state never finishes. So, it should be included as part of other with time / action controlling features.

The attributes needed are:

* **ENTITYID**: A reference to an entity defined in the *entities.xml* structure. By default, 10000, which is a logo to represent pause and it is defined in the adentities.xml file.
* **FADE**: fade level when drawing the entity. By default, 255 (no fade at all).
* **CONTROLPOSITION**: Regarding the state of the entity, it can even move. This parameter is to point whether the position has to be controlled or not. YES or NOT. By default, YES.
* **POSITION**: Like previous ones. It makes only sense when CONTROLPOSITION is YES. By default, 0.5,0.5,0. It is at the centre of the screen.
* **POSREFERENCE**: Idem. 0 by default, meaning the centre of the entity.

## To load all resources of a game. Type 10050

Macro for type: *\_\_QGAMES\_GAMESTATELOADINGTYPE\_\_*

class QGAMES::LoadingGameState

This game state shows a shape getting fulfilled as the different game resources are loaded. When all resources are loaded the event number 7016 (*\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*) is issued.

The parameters needed are:

* **FORMID, FADE, POSITION and POSREFERENCE** used in a similar way than others.
* **REVERSE**: To define whether the form is fulfilled normally (from right to left or from bottom to top) or the other way around. YES or NOT. NOT by default.
* **HORIZONTAL**: To define whether the form is fulfilled from left to right (or from right to left if REVERSE is YES) or from top to bottom (or from bottom to top if REVERSE is YES). YES or NOT. NOT by default.

## To create a count down effect. Type 10060

Macro for type: *\_\_QGAMES\_GAMESTATECOUNTINGDOWNTYPE\_\_*

class QGAMES::CountingDownGameState

With this game state a series of frames of a series of forms are shown. The game state starts showing the first frame of the first form, after a period it presents the same frame of the next form and so on and so forth. As the form is changing its fade level is going down little by little. When almost the frame of the last form is about to be shown, the game state starts to display also the next frame of the first form and so on and so forth until there is more frames to show.

It is possible to emit a sound when the first form of any frame is shown.

Let’s figure out that the forms represent e.g. different sizes of numbers, and the frames represent the numbers itself. If the first frame had represented the bigger number, and the last frame the lowest and if the first form had represented the lowest size and the last from the biggest size, a countdown effect could have been created.

Now it is easier to understand the attributes needed.

* **FORMID**: The initial form. The other ones required must be sequential to this one. By default, it is 30000 that represent a set of small numbers (3), and it is defined in the file *adforms.xml*.
* **BEEPSOUNDID**: A reference to the sound emitted when the first form per frame is shown. The sound must be defined in the *sounds.xml* structure.
* **FRAMESPERNUMBER**: Integer. Indicates the number of forms after the one defined in FORMID. 9 by default.
* **FRAMESOVERLAPPED**: An integer. Define how many forms before the end of a frame, the next one starts. 2 by default. It cannot be greater than FRAMESPERNUMER.
* **NUMBERS**: An integer. Define the number of frames per form. 3 by default.
* **POSITION**: In a similar way than in others. In this case always the centre of any frame is considered the reference.

So, as you can appreciate the default behaviour is a counter from 3 to 1.

When the last beep is emitted the event number 70016, *\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*, is issued.

## To show a hall of fame. Type 10070

Macro for type: *\_\_QGAMES\_GAMESTATEHALLOFFAMETYPE\_\_*

class QGAMES::ShowingHallOfFameGameState

This game state is to show the typical hall of game of almost any game.

The hall of fame is a map (*std::map*) with two entries: a string, and a SetOfOpenValues, which is made up of OpenValues. This map is stored in the configuration object of the game and accessible using the method:

((QGAMES::AdvancedArcadeGame::Conf\*) game () -> configuration ()) -> hallOfFame ();

What the map represents depends on the game itself. Usually the first entry is the order number in the hall of fame, and the SetOfOpenValues represents the name of player, his/her score, time, number of enemies killed, etc.

The class defines two important methods that can be (and usually are) overloaded to manipulate the information displayed in the hall of fame:

* **orderHallOfFame (hall of fame)**: To sort the entries of the hall of fame. Those entries had been added to the hall of fame as they have been generated, using the number of the player as the first entry usually. Maybe a presentation based on the score is more natural. This method is invoked when QGAMES::ShowingHallOfFameGameState::onEnter method is executed. By default, it does do anything, but it can be overloaded as needed.
* **createTextFor (entry index, hall of fame entry)**: To create the text to be displayed per entry. The method receives the two parts of the any entry in the map hall of fame. What is returned, depends on the need. By default, a text with the index and the first openValue in SetOfValues chained is returned. It can be overloaded.

The Dragonwind like games overload this class to create a new one:

class DRAGONWIND::ShowingHallOfFameGameState

Type 50070. Macro for type: *\_\_DRAGONWIND\_GAMESTATEHALLOFFAMETYPE\_\_*

The SetOfOpenValues manage in a Dragonwind like game gas 2 openValues: 0, with the score of the player, and 1 with his/her initials (they are introduced before the first scene per player appears).

The entries in the hall of fame are order using the score as criteria (from least to greatest), and they are show aligning the score to 5 digits (filling with ceros if lesser).

This game state never finishes too. So, it should be included into another controlling the time or whether the player presses something.

## To roll figures on the display. Type 10080

Macro for type: *\_\_QGAMES\_GAMESTATEROLLINGFRMSTYPE\_\_*

class QGAMES::RollingFormsGameState

It is typical to see a set of backgrounds that fade out little by little while moving to the left of the screen and while other’s new fade in also little by little over the previous ones. This game state is to produce effects such as that one.

The parameters needed are:

* **FORMS**: A list of ids referring to forms identified in the forms.xml structure. The all have to exist. No default values are allowed.
* **DELAY**: Loops to move any form. 6 by default.
* **SECONDSONSCREEN**: Number of seconds (or fraction) that every background is on the screen. 1 by default.
* **DIRECTION**: A vector to point the direction the display is moved to. By default, 1,0,0 meaning that the forms move to the opposite direction (-1,0,0).
* **INSTANCES**: Number of times a background is copied in the moving direction. Just only 1 by default.
* **SEQUENTIAL**: No commonly used. YES or NOT. NO by default. If YES, the forms wouldn’t be chained.

The game state never finishes. It rolls the backgrounds cyclically.

## To select an option from a menu. Type 10100

Macro for type: *\_\_QGAMES\_GAMESTATESELECTOPTIONSTYPE\_\_*

class QGAMES::StandardInitialSelectionOptionsGameState

Maybe one of the most interesting game states, and the one almost any game should stop on.

It is to manage player’s interaction over two menus of options. These menus must be defined in the *guisystem.xml* structure. The menus managed are the number 10.000 and the number 10.100, both defined in the file *adguisystem.xml*.

The options for each are:

### Main menu. Number 10000

* **1 Start the game**: To start the game.
* **3 Load last game**: To load the last saved state of the game. Saved in the data directory with the name *QGAMES::StandardInitialSelectionOptionsGameState::\_CONFFILENAME*. By default, it is: *Game.cfg*, but it can be changed as needed.
* **5 Number of players**: To introduce the number of players of the game.
* **6 Joystick / Keyboard**: Iterates between both. Which device do you want to play with?
* **7 Configure keys**: Open the next menu. Number 10100.
* **8 Easy / Medium / Difficult**: Iterates among the different difficulty levels.
* **9 Music On / Off**: To switch on or off the background music in any scene.
* **10 Hall of fame**: To show the hall of fame.
* **11 Exit**: To exit the game.

### Keys menu. Number 10100

* **UP, DOWN, LEFT, RIGHT, FIGHT, PAUSE, SAVE**: Options to define the keys used to play (if the keyboard had been selected as the device to play with). They all have a default value and clicking on it that value iterates among all different possibilities.

Dragonwind like games modifies a little bit this game state, with the class:

class DRAGONWIND::SelectionGameState

Type 50100. Macro for type: *\_\_DRAGONWIND\_GAMESTATEMENUTYPE\_\_*

And introducing the options:

* **2 Instructions**: To show the instructions of the game.
* **4 Default player Daiki**: Iterates between Daiki (male character) and Akira (female character).

In the main menu, and the options:

* **ITERATE, CATCH & LEAVE and JUMP** in the key configuration menu.

But it has been modifying a little bit the file adguisystem.xml, adding new widgets in the middle of the standard Advanced Arcade definition (numbers 10000 and 10100), instead defining new menus.

To be accurate, when in the main menu any out of the options 1,2,10 or 11 the game state finishes (event number 7016, *\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*, is issued) storing the keeping the option selected in an internal variable, accessible using the method:

lastOptionSelected ()

Usually there will be a flow controller game state containing this one and making what needed to move to the next game state. We will see a in the following pages which on is this external controller.

Any time the game state finishes saves the options selected into the configuration object of the game. In the same way, any time the game state starts, the configuration object of the game is read, and the options are set regarding its values.

## To show a list of credit titles. Type 10090

Macro for type: *\_\_QGAMES\_GAMESTATECREDITTITLESTYPE\_\_*

class QGAMES::CreditTitlesGameState

This game state is maybe more complex than many of the previous ones. It is aimed to show a set of titles that appears out of the blue and fade out little by little before the next one is shown.

The attributes are:

* **MAXLETTERSPERTITLE**: Each title can be made up of many different lines of text. This parameter defines the number of maximum letters per line. No limit by default.
* **POSITION & POSREFERENCE**: in the same way they have already been explained in other game states. By default, 0.5, 0.5, 0 (meaning the centre of the screen) and 0 (meaning the centre of the text).
* **SECONDSONSCREEN**: Number of seconds that every text stays totally visible, before starting to fade out. By default, 1 second.
* **FONT**: Id of the font used to represent the texts. 20022 (*\_\_QGAMES\_RAVIE24ORANGELETTERS\_\_*) by default. Defined in *adfonts.xml*.

Additionally, it is needed to define the texts in the XML paragraph *<Titles>*. Each entry must be like:

*<Title value=”…”/>*

Dragonwind like games modifies a little bit this class introducing the possibility that one of the titles shown represents the main title of the game and with the Dragonwind logo behind.

class DRAGONWIND:: CreditTitlesGameState

Type 50090. Macro for type: *\_\_DRAGONWIND\_GAMESTATECREDITTITLESTYPE\_\_*

The game state never ends. When the last text is shown noting else happens, but no event is issued either.

This class adds two new attributes:

* **NUMBERTITLE**: The number of the title where the logo has to be presented. By default, 2. Starting from 0.
* **CONTENTTITLE**: The text that must being part of the title that is shown bigger. Dragonwind, by default. The letters for that title will be green but belonging to the same font (if they exist) than the rest.

## To show a set of instructions. Type 10091

Macro for type: *\_\_QGAMES\_GAMESTATESHOWINSTTYPE\_\_*

This game state is just to show a set of texts that behave as a paragraph. The paragraph moves from the bottom to the top always.

The attributes needed are:

* **SEPARATION**: In pixels. The separation between the bottom part of a line and the top part of the next. By default, 0 pixels.
* **VISIBLEZONE**: A list of 6 values (all of them between 0 and 1) describing the top – left corner (the first three) and the bottom – right corner (the last three) of a rectangle defining in which zone of the screen the instructions will be visible. So, 0,0,0,1,1,1 (by default) would mean that the full screen will be used for the paragraph. 0,0.15,0,1,0.85,0 implies to use a reduced zone of it.
* **FADE**: fade level (maximum) of the text. 255 by default.
* **MOVEMENT**: integer. To define the speed of the paragraph movement. 1 by default.
* **TOMOVE**: Integer. Number of loops to wait before moving (MOVEMENT). 0 by default. The movement is continuous.

Additionally, it is needed to define each line. To do so, the paragraph *<Instructions>* should be added. Each member of it should have the following aspect:

*<Instruction line="Welcome" type="0" font="20031" position="0.5" reference="0"/>*

The meaning of the different values are:

* **line**: The text itself. Nothing, by default.
* **type**: It is not used still.
* **font**: ide of the font used for the text. 20022 (*\_\_QGAMES\_RAVIE24ORANGELETTERS\_\_*) by default.
* **position**: Between 0 and 1. The relative position of the line where to put the first point of the text. 0.5 by default.
* **reference**: 0, 1, 2 which is the reference point of the text? 0 the middle, 1 the left – top corner y 2 the right – bottom corner of the text. 0 (CENTER) by default.

Every text enters always the visible rectangle fading in and exit it fading out.

When the last text has exited the visible zone the event 7016 (*\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*) is issued.

## To control how the game states flow. Type 10101

Macro for type: *\_\_QGAMES\_GAMESTATEADVANCEDCTRL\_\_*

class QGAMES::AdvancedGameStateControl

This is with no doubts the most important game state. It is mainly aimed to control how the flow of a game happens. It could have nested another game state control if needed.

The attributes needed are:

* **PAUSEGAMESTATE**: A reference to the game state shown when the game is in pause mode. No default values. It must exist.

The main body of the game state definition is the flow. It is described in the paragraph *<FlowConditions>*. Each element of the paragraph has the following aspect:

*<FlowCondition on="50002" when="0" to="50300"/>*

The controlling game state has always nested (only one is possible at the same time) the game state under execution in every specific moment of the time. When it finishes it is replaced by the one determined by a flow condition.

To let the controlling know that the nested game state has finished, the event number 7016 (*\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*) must be issued (notice that many of the previous game states issue this event when finish).

The meaning of the different parameters is:

* **on**: Id of the game state finalizing. No default values.
* **when**: The finalization condition. The same game state could finish in different circumstances and the next game state could be different in each case. No default values. The finalization condition must be defined using a number (integer). It is necessary to programme which condition to issue in every case.
* **to**: The next game state. If this value es -1 means that the controlling game state has to finish too, and the event number 7016 (*\_\_QGAMES\_GAMENESTEDSTATEFINISHED\_\_*) will be issued for, maybe, other control game state to take decisions o simply (if there is none) exit the game. No default values.

The control game state must be usually adapted to every specific case. Some specific game states can have many different ending conditions, and what to do could be different in each of them. Imagine e.g. that there is a game state in which the active part of the game happens. The main character can die or reach the main goal, or try to open a set of options, or… Anytime one of those happens the game state will finish, but its ending condition will be totally different and what the controlling game state (parent of this one) should do too.

The only way to do it is modifying the behaviour of the class *QGAMES::AdvancedGameStateControl*. That class defines a couple of methods to manage the behaviour:

* **firstGameState**: Returns the name of the state being the first to set up when the controlling game state starts. The controlling game state maintain a list of all the game states managed, sorted using its id. By default, the first game state in the list is returned. It means the one with the lower id.
* **nextGameState (current game state, event)**: Returns the name of next the game state after an event received from the one being executed right one. The default implementation invokes the method *circuntanceWhen*, and uses the value returned to determine the right game state to move to. it is not necessary to modify it in most of the circumstances.
* **circunstanceWhen (current game state, event)**: Returns an integer. O by default. This integer is the one used to determine the flow condition to apply (attribute when). This is the method to usually overload when redefining a game control state.
* **finishControl**: This method is executed in every loop. When returns true the game state control finishes and then issue the event number 7016. It returns false by default. Usually it is not needed to change it behaviour.

Dragonwind like games define two controlling game states:

class DRAGONWIND::PlayingControlGameState

Type 50080. Macro for type: *\_\_DRAGONWIND\_GAMESTATECONTROLPLAYINGTYPE\_\_*

This controlling game state is aimed to manage the game when the player is playing or about to play. The main nested game state controlled here is:

class DRAGONWIND::PlayingGameState

Type 50081. Macro for type: *\_\_DRAGONWIND\_GAMESTATEPLAYINGTYPE\_\_*

This game state just observes the input handler (keyboard or joystick) and send instructions to the main character to move him / her. The game state also observes how the main character is behaving.

When the main character dies or he/she reaches a goal (final or an intermediate one), this game state will receive the event, and will notify to the owner class (the *PlayingControlGameState*) an event number 7016.

The controlling class will receive that event and will transform it in the following values in the method *circuntanceWhen*:

* **0** if the main artist has died,
* **1** if the main artist has reached the goal and the scene were not visited ever.
* **3** same, but when the scene has been visited previously.

The differences between the case 1 and 3 are than in the first situation no celebration is done and the game will chose immediately and automatically the next world/scene, whilst in the second situation a celebration intermediate game state will be shown.

If during the execution of *DRAGONWIND::PlayingGameState* the key ESC were pressed an event number 70016 would be also notified and translated (in this case) in a 2 in the *circunstanceWhen* method.

The *DRAGONWIND::PlayingGameState* game state is now substituted by:

class DRAGONWIND::OptionsWhenPlayingGameState

Type 50082. Macro for type: *\_\_DRAGONWIND\_GAESTATEOPTIONSWHENPLAYTYPE\_\_*

In this game state, the game keeps going. The enemies keep moving, etc. The player cannot control the main character but only select one of the options shown. The GUI system shown is the number 50200, and the position of the menu will be at the top – left corner of the screen. Both definitions are written in the code. To change them several class should be changed.

The options for this GUI are:

* **Save the game**: The game is saved in the file *QGAMES::StandardInitialSelectionOptionsGameState::\_CONFFILENAME* (*Game.cfg* by default) in the folder *DATADIR* (defined in the *Conf.xml* file). After doing so an event number 70016 is issued and the controlling game state over it (*DRAGONWIND:: PlayingControlGameState*) will translate it returning 0 in its method *circunstanceWhen*.
* **Exit the game**: If exit the game had been selected, that method (*cirunstanceWhen*) would return 1.
* **Continue playing**: 0 is also returned when to continue with the game was selected.

Now is easier to understand what is defined in the file *dwgamestates.xml*

<!-- The group of states related to the game playing -->

<!-- When Playing Game a Control Game State type is used too -->

<GameState id="50100" type="50080" name="11Control Playing" main="YES">

<FlowConditions>

<!-- The first state (50101) is just to determinate what to do next:

It could be just to start to play or whether to show the name of the world and sceme and counting down first -->

<FlowCondition on="50101" when="0" to="50102"/> <!-- After starting the game the name of the world and the scene must be shown -->

<FlowCondition on="50101" when="1" to="50107"/> <!-- After starting the game control is handed over the player -->

<FlowCondition on="50102" when="0" to="50105"/> <!-- After the name of the world and scene...counting down -->

<FlowCondition on="50105" when="0" to="50107"/> <!-- After counting down...the control to the player -->

<FlowCondition on="50107" when="0" to="50109"/> <!-- When playing stops because the artist has died, then stop a while -->

<FlowCondition on="50107" when="1" to="50111"/> <!-- When playing stops because the artist has reached the goal and the scene hasn't been completed before-->

<FlowCondition on="50107" when="3" to="50114"/> <!-- When playing stops because the artist has reached the goal but scene has been completed before-->

<FlowCondition on="50107" when="2" to="50200"/> <!-- Whwn playing stops because the artist is asking for options -->

<FlowCondition on="50109" when="0" to="-1"/> <!-- After waiting when dieing, the control fisishes -->

<FlowCondition on="50111" when="0" to="-1"/> <!-- After waiting when a goal is reached, the control finishes -->

<FlowCondition on="50114" when="0" to="-1"/> <!-- After entering this No game state just the control finishes -->

<FlowCondition on="50200" when="0" to="50107"/> <!-- After the options to continue in the game has been choseen -->

<FlowCondition on="50200" when="1" to="-1"/> <!-- After the options to exit the game has been choseen -->

</FlowConditions>

</GameState>

<!-- Just a NULL Game State to launch the verification of what is next before

giving the control to the player -->

<GameState id="50101" type="10025" name="110Decide Before Playing" main="YES"/> <!-- Type NULL -->

<!-- To show the name of the world and the name of the scene -->

<GameState id="50102" type="10002" name="111World and Scene Name" main="YES"> <!-- Until TIME or KEY PRESSED -->

<Attributes>

<Attribute id="TIME" value="1.0"/> <!-- Seconds to wait if no key is pressed -->

</Attributes>

<Nested>

<GameState id="50103" type="50010" name="World Name" main="NO"> <!-- Type WORLD NAME -->

<Attributes>

<Attribute id="TEXT" value="World"/> <!-- This text is completed with the name of the word after -->

<Attribute id="FONT" value="20031"/>

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.45,0"/>

<Attribute id="POSREFERENCE" value="0"/> <!-- Meaning Center -->

<Attribute id="ORIENTATION" value="1,0,0"/>

</Attributes>

<Nested>

<GameState id="50104" type="50011" name="Scene Name" main="NO"> <!-- Type SCENE NAME -->

<Attributes>

<Attribute id="TEXT" value="Scene"/> <!-- This text is completed with the name of the word after -->

<Attribute id="FONT" value="20031"/>

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.55,0"/>

<Attribute id="POSREFERENCE" value="0"/> <!-- Meaning Center -->

<Attribute id="ORIENTATION" value="1,0,0"/>

</Attributes>

<Nested>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

<!-- To count down -->

<GameState id="50105" type="10001" name="112Counting Down" main="YES"> <!-- Until KEY PRESSED -->

<Nested>

<GameState id="50106" type="10060" name="Counting Down" main="NO"> <!-- Type COUNTING DOWN -->

<Attributes>

<Attribute id="FORMID" value="30010"/> <!-- The initial form. Next 9 (FRAMESPERNUMBER) will also be used -->

<Attribute id="BEEPSOUNDID" value="10011"/>

<Attribute id="FRAMESPERNUMBER" value="9"/>

<Attribute id="FRAMESOVERLAPPED" value="2"/>

<Attribute id="NUMBERS" value="3"/>

<Attribute id="POSITION" value="0.5,0.5,0"/> <!-- In the middle of the screen -->

</Attributes>

</GameState>

</Nested>

</GameState>

<!-- This game state is used to play.

The controls are then managed by the user -->

<GameState id="50107" type="50081" name="113Just Playing" main="YES"/> <!-- Type PLAY -->

<!-- This game state is used with the options when playing -->

<GameState id="50200" type="50082" name="116OOptions When Playing" main="YES"/> <!-- Type OPTIONS WHEN PLAYING -->

<!-- This game state is used when the artist die just to wait a couple of seconds -->

<GameState id="50109" type="10000" name="114Wait After Dieing" main="YES"> <!-- Type WAIT -->

<Attributes>

<Attribute id="TIME" value="2.0"/> <!-- Seconds to wait -->

</Attributes>

<Nested>

<GameState id="50110" type="10010" name="When Dieing Sound" main="NO"> <!-- Type SOUND -->

<Attributes>

<Attribute id="SOUNDID" value="50014"/>

<Attribute id="CHANNELID" value="3"/> <!-- In the Background of the Scene -->

</Attributes>

</GameState>

</Nested>

</GameState>

<!-- This game state is used when the artist reaches the limit of the current scene (the goal)

It is needed to wait just for a couple of seconds while celebrating -->

<GameState id="50111" type="10000" name="115Wait After Reaching Goal" main="YES">

<Attributes>

<Attribute id="TIME" value="4.0"/> <!-- Seconds to wait -->

</Attributes>

<Nested>

<!-- A message to point that the goal has been reached -->

<GameState id="50112" type="10003" name="Reach Goal Message" main="NO"> <!-- Type TEXT Shown -->

<Attributes>

<Attribute id="TEXT" value="You did it!"/>

<Attribute id="FONT" value="20032"/>

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.5,0"/>

<Attribute id="POSREFERENCE" value="0"/> <!-- Meaning Center -->

<Attribute id="ORIENTATION" value="1,0,0"/>

</Attributes>

<Nested>

<GameState id="50113" type="10010" name="When Reaching Sound" main="NO"> <!-- Type SOUND -->

<Attributes>

<Attribute id="SOUNDID" value="50013"/>

<Attribute id="CHANNELID" value="3"/> <!-- In the Background of the Scene -->

</Attributes>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

<!-- This game state is a no game state. That is: simply finishes when starts

The state doesn't need any attribute -->

<GameState id="50114" type="10025" name="116JustFinishes" main="YES"/>

<!-- -->

And also, how the main controlling game state is defined:

<?xml version="1.0"?>

<GameStates>

<!-- Includes -->

<Include file="ptgamestates.xml"/>

<!-- The usual controlling form -->

<GameState id="50000" type="50101" name="Controller" main="YES">

<FlowConditions>

<FlowCondition on="50002" when="0" to="50300"/> <!-- After the game is loaded, the game is initialized -->

<FlowCondition on="50300" when="0" to="50003"/> <!-- After the game is initialized the logo of the owner appears -->

<FlowCondition on="50003" when="0" to="50004"/> <!-- After the loogo appears, then the introduction starts -->

<FlowCondition on="50004" when="0" to="50009"/> <!-- After introduction, the menu is shown -->

<FlowCondition on="50009" when="0" to="50014"/> <!-- When the play option is selected, then playing starts asking for the name of the player -->

<FlowCondition on="50009" when="1" to="50028"/> <!-- When the option selected is exit -->

<FlowCondition on="50009" when="2" to="50025"/> <!-- When the option selected is hall of fame -->

<FlowCondition on="50009" when="3" to="50029"/> <!-- When the option selected is instructions -->

<FlowCondition on="50014" when="0" to="50100"/> <!-- After asking for the name of the player, then the game starts showing the name of the wotld and scene -->

<FlowCondition on="50100" when="0" to="50019"/> <!-- After dieing when playing, the official label is shown -->

<FlowCondition on="50100" when="1" to="50100"/> <!-- After reaching a goal when playing, start back if there is more scenes to deal with -->

<FlowCondition on="50100" when="2" to="50009"/> <!-- After selecting (while playing) the game finishes and the menu is back -->

<FlowCondition on="50100" when="3" to="50400" /> <!-- After reaching a goal, there is no more scene to deal with so the player wins!! -->

<FlowCondition on="50019" when="0" to="50014"/> <!-- After die, the game starts back -->

<FlowCondition on="50019" when="1" to="50022"/> <!-- ..or the game is over -->

<FlowCondition on="50022" when="0" to="50025"/> <!-- After game is over, the hall of fame appears -->

<FlowCondition on="50025" when="0" to="50009"/> <!-- After hall of fame to menu -->

<FlowCondition on="50029" when="0" to="50009"/> <!-- After the instructions the menu is back -->

<FlowCondition on="50400" when="0" to="50022"/> <!-- After showing that the player has won, then game over -->

</FlowConditions>

<Attributes>

<Attribute id="PAUSEGAMESTATE" value="50001"/> <!-- A reference to the nested state when nested -->

</Attributes>

</GameState>

<!-- The pause game.

It is just a state showing an entity in the middle -->

<GameState id="50001" type="10040" name="Pause" main="YES"> <!--Type ENTITY Shown -->

<Attributes>

<Attribute id="ENTITYID" value="10000"/> <!-- The entity used when pause -->

<Attribute id="FADE" value="255"/>

<Attribute id="CONTROLPOSITION" value="YES"/>

<Attribute id="LOOPSPERFADECHANGE" value="3"/>

<Attribute id="POSITION" value="0.5,0.5,0"/>

<Attribute id="POSREFERENCE" value="0"/>

</Attributes>

</GameState>

<!-- To load the game.

One of the houses used as background in many scenes full filled little by little -->

<GameState id="50002" type="10050" name="0Loading" main="YES"> <!-- Type LOADING -->

<Attributes>

<Attribute id="FORMID" value="50002"/> <!-- The big sun -->

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.5,0"/>

<Attribute id="POSREFERENCE" value="0"/>

<Attribute id="REVERSE" value="NO"/>

<Attribute id="HORIZONTAL" value="YES"/>

</Attributes>

</GameState>

<!-- To initialize the game -->

<GameState id="50300" type="50102" name="1Initializing" main="YES"/> <!-- Type INITIALIZING -->

<!-- To present the game.

Just the commty logo comming to live little by little -->

<GameState id="50003" type="10030" name="2CommtyLogo" main="YES"> <!-- Type FORM Shown -->

<Attributes>

<Attribute id="FORMID" value="10002"/> <!-- The Big CommtyLogo -->

<Attribute id="INITIALFADE" value="0"/>

<Attribute id="FINALFADE" value="255"/>

<Attribute id="FADEINCREMENT" value="5"/>

<Attribute id="LOOPSPERFADECHANGE" value="2"/>

<Attribute id="POSITION" value="0.5,0.5,0"/> <!-- In the middle -->

<Attribute id="POSREFERENCE" value="0"/> <!-- Reference to the center -->

</Attributes>

</GameState>

<!-- To introduce the game.

A set of titles appearing in the middle of the screen,

with a background made of rolling entities

The state can be stopped clicking any key -->

<GameState id="50004" type="10001" name="3Introducing The Game" main="YES"> <!-- Type UNTIL KEY PRESSED -->

<Nested>

<!-- A little commtynets logo is shown is the left down corner -->

<GameState id="500041" type="10020" name="Little Commty Logo" main="NO"> <!-- Type FIX FORM Shown -->

<Attributes>

<Attribute id="FORMID" value="10001"/> <!-- The Little CommtyLogo -->

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.01,0.925,0"/> <!-- In the left side -->

<Attribute id="POSREFERENCE" value="1"/> <!-- Reference to the left up corner -->

</Attributes>

<Nested>

<!-- The titles appears -->

<GameState id="50005" type="50090" name="Introducing Titles" main="NO"> <!-- Type CREDIT TITLES -->

<Titles>

<Title value=""/>

<Title value="Community Networks Presents"/>

<Title value="A new game to have fun"/>

<Title value="Dragonwind"/>

<Title value="by Ignacio Cea"/>

<Title value="From his laboratories"/>

<Title value="and using many resources downloaded from Internet"/>

<Title value="all combined for you in this new adventure"/>

<Title value="Press any key to menu"/>

</Titles>

<Attributes>

<Attribute id="NUMBERTITLE" value="3"/>

<Attribute id="CONTENTTITLE" value="Dragonwind"/>

<Attribute id="MAXLETTERSPERTITLE" value="25"/>

<Attribute id="POSITION" value="0.5,0.5,0"/> <!-- In the middle -->

<Attribute id="POSREFERENCE" value="0"/>

<Attribute id="SECONDSONSCREEN" value="3"/>

<Attribute id="FONT" value="20032"/> <!-- Ravie 36 Orange -->

</Attributes>

<Nested>

<!-- A fixed form with the form of the dragon

The form will be visible only when the title on the screen is the name of the game

This is something that the previous game state controls -->

<GameState id="50006" type="10020" name="Game Logo" main="NO"> <!-- Type FIXED FORM -->

<Attributes>

<Attribute id = "FORMID" value = "50001"/>

<Attribute id = "FADE" value = "255"/>

<Attribute id = "POSITION" value = "0.5,0.5,0"/>

<Attribute id = "POSREFERENCE" value = "0"/>

</Attributes>

<Nested>

<!-- A roolling background appears, under the titles -->

<GameState id="50007" type="10080" name="Rolling Background" main="NO"> <!-- Type ROLLING FORMS -->

<Attributes>

<Attribute id="FORMS" value="50010,50011,50012,50013,50014,50015,50016,50017"/>

<Attribute id="DELAY" value="1"/>

<Attribute id="SECONDSONSCREEN" value="2"/>

<Attribute id="DIRECTION" value="1,0,0"/>

<Attribute id="INSTANCES" value="3"/>

<Attribute id="SEQUENTIAL" value="FALSE"/>

</Attributes>

<Nested>

<!-- An a sound plays in the background -->

<GameState id="50008" type="10010" name="Rooling Sound" main="NO"> <!-- Type SOUND -->

<Attributes>

<Attribute id="SOUNDID" value="50000"/>

<Attribute id="CHANNELID" value="0"/> <!-- Game Background -->

</Attributes>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

<!-- The main menu of the game -->

<GameState id="50009" type="50100" name="4Menu" main="YES"> <!-- Type DRAGONWIND MENU -->

<Nested>

<!-- The ninja male running -->

<GameState id="50010" type="10040" name="Male Running" main="NO"> <!-- Type ENTITY Shown -->

<Attributes>

<Attribute id="ENTITYID" value="50000"/>

<Attribute id="FADE" value="255"/>

<Attribute id="CONTROLPOSITION" value="YES"/>

<Attribute id="LOOPSPERFADECHANGE" value="6"/>

<Attribute id="POSITION" value="0.75,0.75,0"/>

<Attribute id="POSREFERENCE" value ="0"/>

</Attributes>

<Nested>

<!-- The ninja female running -->

<GameState id="50011" type="10040" name="Female Running" main="NO"> <!-- Type ENTITY Shown -->

<Attributes>

<Attribute id="ENTITYID" value="50001"/>

<Attribute id="FADE" value="255"/>

<Attribute id="CONTROLPOSITION" value="YES"/>

<Attribute id="LOOPSPERFADECHANGE" value="6"/>

<Attribute id="POSITION" value="0.80,0.75,0"/>

<Attribute id="POSREFERENCE" value ="0"/>

</Attributes>

<Nested>

<!-- A little commtynets logo is shown is the right up corner -->

<GameState id="500091" type="10020" name="Little Commty Logo" main="NO"> <!-- Type FIX FORM Shown -->

<Attributes>

<Attribute id="FORMID" value="10001"/> <!-- The Little CommtyLogo -->

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.99,0.07,0"/> <!-- In the left side -->

<Attribute id="POSREFERENCE" value="2"/> <!-- Reference to the right bottom -->

</Attributes>

<Nested>

<!-- A roolling background under the menu -->

<GameState id="50012" type="10080" name="Rolling Menu Background" main="NO"> <!-- Type ROLLING FORMS -->

<Attributes>

<Attribute id="FORMS" value="50012,50013,50014"/>

<Attribute id="DELAY" value="1"/>

<Attribute id="SECONDSONSCREEN" value="5"/>

<Attribute id="DIRECTION" value="1,0,0"/>

<Attribute id="INSTANCES" value="3"/>

<Attribute id="SEQUENTIAL" value="FALSE"/>

</Attributes>

<Nested>

<!-- A sound in the background -->

<GameState id="50013" type="10010" name="Menu Sound" main="NO"> <!-- Type SOUND -->

<Attributes>

<Attribute id="SOUNDID" value="50001"/>

<Attribute id="CHANNELID" value="0"/> <!-- Game Background -->

</Attributes>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

<!-- Asking for the name of the player, if it is needed -->

<GameState id="50014" type="10006" name="5Player Info" main="YES"> <!-- Type WAIT UNLESS KEY -->

<Attributes>

<Attribute id="TIME" value="4"/> <!-- Two seconds -->

</Attributes>

<Nested>

<!-- A message to point the number of the player -->

<GameState id="50015" type="10004" name="Player Number" main="NO"> <!-- Type PLAYER NUMBER Shown -->

<Attributes>

<Attribute id="TEXT" value="Player"/> <!-- This text is completed with a number after -->

<Attribute id="FONT" value="20030"/>

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.45,0"/>

<Attribute id="POSREFERENCE" value="0"/> <!-- Meaning Center -->

<Attribute id="ORIENTATION" value="1,0,0"/>

</Attributes>

<Nested>

<!-- This state will be shown only when the name has been already introduced -->

<GameState id="50016" type="50000" name="Player Name" main="NO"> <!-- Type PLAYER NAME (Shown or ask for) Shown -->

<Attributes>

<Attribute id="TEXT" value=""/> <!-- This text is completed with the name after -->

<Attribute id="FONT" value="20030"/>

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.55,0"/> <!-- A little bit under the previous one -->

<Attribute id="POSREFERENCE" value="0"/> <!-- Meaning Center -->

<Attribute id="ORIENTATION" value="1,0,0"/>

</Attributes>

<Nested>

<GameState id="50017" type="10020" name="Background Sun" main="NO"> <!-- Type FIXED FORM -->

<Attributes>

<Attribute id="FORMID" value="50003"/>

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.5,0"/>

<Attribute id="POSREFERENCE" value="0"/>

</Attributes>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

<!-- When the player has dies but the game has to restart back

because there are still lives enough to continue

The main state waits until a key is pressed -->

<GameState id="50019" type="10002" name="6Die" main="YES"> <!-- Type WAIT OR UNTIL KEY Pressed -->

<Attributes>

<Attribute id="TIME" value="3"/> <!-- Seconds with no key iteraction -->

</Attributes>

<Nested>

<!-- A message to point the player has died -->

<GameState id="50020" type="10003" name="Died Message" main="NO"> <!-- Type TEXT Shown -->

<Attributes>

<Attribute id="TEXT" value="Player has died"/>

<Attribute id="FONT" value="20030"/>

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.5,0"/>

<Attribute id="POSREFERENCE" value="0"/> <!-- Meaning Center -->

<Attribute id="ORIENTATION" value="1,0,0"/>

</Attributes>

<Nested>

<!-- A sound in the background -->

<GameState id="50021" type="10010" name="Died Sound" main="NO"> <!-- Type SOUND -->

<Attributes>

<Attribute id="SOUNDID" value="50010"/>

<Attribute id="CHANNELID" value="0"/> <!-- Game Background -->

</Attributes>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

<!-- The game has finished

The main state waits until a key is pressed -->

<GameState id="50022" type="10002" name="7Gameover" main="YES"> <!-- Type WAIT OR UNTIL KEY Pressed -->

<Attributes>

<Attribute id="TIME" value="6"/> <!-- 6 seconds waiting -->

</Attributes>

<Nested>

<!-- A message to point the game is over -->

<GameState id="50023" type="10003" name="Gameover Message" main="NO"> <!-- Type TEXT Shown -->

<Attributes>

<Attribute id="TEXT" value="Gameover"/>

<Attribute id="FONT" value="20041"/>

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.5,0"/>

<Attribute id="POSREFERENCE" value="0"/> <!-- Meaning Center -->

<Attribute id="ORIENTATION" value="1,0,0"/>

</Attributes>

<Nested>

<!-- The logo sun in the middle -->

<GameState id="500220" type="10020" name="Gameover background logo" main="NO"> <!-- Type FIX FORM Shown -->

<Attributes>

<Attribute id="FORMID" value="500022"/> <!-- The temple -->

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.5,0"/> <!-- In the middle -->

<Attribute id="POSREFERENCE" value="0"/> <!-- Reference to the center -->

</Attributes>

<Nested>

<!-- A sound in the background -->

<GameState id="50024" type="10010" name="Gameover Sound" main="NO"> <!-- Type SOUND -->

<Attributes>

<Attribute id="SOUNDID" value="50011"/>

<Attribute id="CHANNELID" value="0"/> <!-- Game Background -->

</Attributes>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

<!-- The player wins!! -->

<GameState id="50400" type="10002" name="12Player wins" main="YES">

<Attributes>

<Attribute id="TIME" value="6"/> <!-- 6 seconds waiting -->

</Attributes>

<Nested>

<!-- A message to point the game is over -->

<GameState id="50401" type="10003" name="Player wins message" main="NO"> <!-- Type TEXT Shown -->

<Attributes>

<Attribute id="TEXT" value="You won!!"/>

<Attribute id="FONT" value="20042"/>

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.5,0"/>

<Attribute id="POSREFERENCE" value="0"/> <!-- Meaning Center -->

<Attribute id="ORIENTATION" value="1,0,0"/>

</Attributes>

<Nested>

<!-- The logo sun in the middle -->

<GameState id="50402" type="10020" name="Player wins background logo" main="NO"> <!-- Type FIX FORM Shown -->

<Attributes>

<Attribute id="FORMID" value="50002"/> <!-- The Sun with the dragon -->

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.5,0.5,0"/> <!-- In the middle -->

<Attribute id="POSREFERENCE" value="0"/> <!-- Reference to the center -->

</Attributes>

<Nested>

<!-- A sound in the background -->

<GameState id="50403" type="10010" name="Player wins sound" main="NO"> <!-- Type SOUND -->

<Attributes>

<Attribute id="SOUNDID" value="50018"/>

<Attribute id="CHANNELID" value="0"/> <!-- Game Background -->

</Attributes>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

<!-- The hall of fame

The main state waits until a key is pressed -->

<GameState id="50025" type="10002" name="8Hall Of Fame" main="YES"> <!-- Type WAIT OR UNTIL KEY Pressed -->

<Attributes>

<Attribute id="TIME" value="6"/> <!-- 6 seconds waiting -->

</Attributes>

<Nested>

<!-- The list of players awarded -->

<GameState id="50026" type="50070" name="Awarded Players" main="NO"> <!--Type DRAGONWIND HALL OF FAME -->

<Attributes>

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0,0,0"/> <!-- Left top corner -->

<Attribute id="SEPARATION" value="10"/>

<Attribute id="FONT" value="20022"/> <!-- Ravie 24 Orange -->

</Attributes>

<Nested>

<!-- A little commtynets logo is shown is the left down corner -->

<GameState id="500251" type="10020" name="Little Commty Logo" main="NO"> <!-- Type FIX FORM Shown -->

<Attributes>

<Attribute id="FORMID" value="10001"/> <!-- The Little CommtyLogo -->

<Attribute id="FADE" value="255"/>

<Attribute id="POSITION" value="0.01,0.925,0"/> <!-- In the left side -->

<Attribute id="POSREFERENCE" value="1"/> <!-- Reference to the left up corner -->

</Attributes>

<Nested>

<!-- A sound in the background -->

<GameState id="50027" type="10010" name="Hall Of Fame Sound" main="NO"> <!-- Type SOUND -->

<Attributes>

<Attribute id="SOUNDID" value="50012"/>

<Attribute id="CHANNELID" value="0"/> <!-- Game Background -->

</Attributes>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

<!-- To exit the game -->

<GameState id="50028" type="7000" name="9Exit" main="YES"/> <!-- Type EXIT -->

<!-- To show the instructions -->

<GameState id="50029" type="10001" name="10Instructions" main="YES"> <!-- Type UNTIL KEY PRESSED -->

<Nested>

<!-- The instructions shown to the player

This set of instructio ns is just an example. It should be adapted in each implementation -->

<GameState id="50030" type="10091" name="The Titles" main="NO"> <!-- Type INSTRUCTIONS LINES -->

<Instructions>

<Instruction line="Welcome" type="0" font="20031" position="0.5" reference="0"/>

<Instruction line="to Dragonwind's land" type="0" font="20031" position="0.5" reference="0"/>

<Instruction line=" " type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="A cruel, sad, and hopefulless land." type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="A land commanded by Jsan II," type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="who calls himself Dragonwind." type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="Probably because," type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="as dragons spitting fire," type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="he leaves nothing alive behind him..." type="0" font="20022" position="0.5" reference="0"/>

<Instruction line=" " type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="It seems to have nothing to do" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="with you, until one day" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="he steals your most precious treasure:" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="Your father's holy ashes," type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="source of you knowledge and power," type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="and something that Dragonwind" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="did wish from long time ago" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="to become even more powerful." type="0" font="20022" position="0.5" reference="0"/>

<Instruction line=" " type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="You now are convinced it is your war" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="and you decide to take part of it." type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="Your mission is quite simple:" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="Freeing the world from this yoke." type="0" font="20022" position="0.5" reference="0"/>

<Instruction line=" " type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="You will depart from home," type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="after the attack." type="0" font="20022" position="0.5" reference="0"/>

<Instruction line=" " type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="Pass levels until you get" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="Jsan II's mansion," type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="where you will have to beat him" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="to free the world." type="0" font="20022" position="0.5" reference="0"/>

<Instruction line=" " type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="During the travel you could take" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="tools or weapons to either" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="open doors or access to hiden" type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="places or to beat other enemies." type="0" font="20022" position="0.5" reference="0"/>

<Instruction line=" " type="0" font="20022" position="0.5" reference="0"/>

<Instruction line="Enjoy and good luck" type="0" font="20021" position="0.5" reference="0"/>

</Instructions>

<Attributes>

<Attribute id="SEPARATION" value="5"/>

<Attribute id="VISIBLEZONE" value="0,0.15,0,1,0.85,0"/>

<Attribute id="FADE" value="255"/>

<Attribute id="MOVEMENT" value="1"/>

<Attribute id="TOMOVE" value="1"/>

</Attributes>

<Nested>

<!-- A background for the instructions -->

<GameState id="50031" type="10020" name="Instructions Background" main="NO"> <!-- Type FIXED FORM -->

<Attributes>

<Attribute id = "FORMID" value = "50009"/>

<Attribute id = "FADE" value = "100"/>

<Attribute id = "POSITION" value = "0,0,0"/>

<Attribute id = "POSREFERENCE" value = "1"/>

</Attributes>

<Nested>

<!-- A sound in the background -->

<GameState id="50032" type="10010" name="Instructions Sound" main="NO"> <!-- Type SOUND -->

<Attributes>

<Attribute id="SOUNDID" value="50003"/>

<Attribute id="CHANNELID" value="0"/> <!-- Game Background -->

</Attributes>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

</Nested>

</GameState>

To change the flow of the game is complex but possible.

# Understanding Entities