# Overview

This document contains information on the mechanics of the game such as information on how classes are setup and the initialization of the game.

# How the Game Loads

1. The main class creates a new GameFramework and Message Dispatcher
2. When GameFramework is initialized it creates the Window and OGRE is initialized.
3. The main class then creates the App State Manager.
4. The main class creates the states the App State Manager will use and are automatically added to the manager.
5. The App State Manager then loads the menu state and then goes into a loop of updating whatever the Manager’s current state is.

# App State Manager

This is the main driving object in the game and contains the game loop. The App State Manager has a current state, such as the level, and will continuously update it.

Different states make up different portions of the game. There are only two states that really do anything right now, the MenuState and PhysicsState. The MenuState is the first state loaded and really only allows you to load the PhysicsState. The PhysicsState creates an instance of the Box2D world and loads OGRE objects as well as GameObjects.

# State

An App State has a method to enter, exit, pause, resume and update itself. These are regularly called from the App State Manager.

# Game Framework Object

The Game Framework starts up OGRE initially and creates the Windows process.

The GameFramework object allows for global access to certain parts of the game, like the current scene manager and the current Box2D world. It also takes care of message and other global functions.

# Game Messaging System

The game messaging system allows objects to send messages to each other using the dispatcher.

## Message Dispatcher

Message Dispatcher is a class that has access to all Game Objects that exist. Objects can call upon Message Dispatcher to send Messages to other objects using their unique Id’s.

## Message

A Message contains a sender object, the intended receiver object, a message type, and a pointer to any sort information. The pointer to the information would be context sensitive to the message type so the receiving class would know what to do with it.

# Game Hierarchy

This is the hierarchy for games objects.

# Game Object Overview

Each game object has a creator object that can be created from a game object factory. The factory has a list of the creator objects and chooses which object to create based upon the name passed to it. Each of the creator objects loads the configuration of a game object from XML.

Through a fairly abstract object, Game Object Ogre Box2D, many of the world objects that lack any real logic, like cars and boxes, can be created through XML only. This prevents having to create many specific classes for each object.

## Game Object

This is the lowest form of object in the game. It has several methods that all objects need to have to interact with other objects.

### Properties

#### Object ID

This is a unique number to identify the object.

#### Object Name

This is a unique string to identify the object.

#### Object Type

An integer that identifies what type of object the object is.

#### Is Initialized

This is a Boolean to tell if the object has been initialized.

### Functions

#### Handle Message Function

Handle Message handles a message sent from the Message Dispatcher. This method is defined for the base object so that objects extended from the base object don’t have to create an implementation for this method if they intend to not respond to any messages.

#### Update

This method is defined for the base object in case the object does not need to be updated.

#### Initialize

This method should be called so that we know the object has been finished being loaded and is set up.

### XML Load Scheme

Game Object has only the one property, name.

#### Name

Name is the unique name of the object.

## Game Object Ogre

This extends Game Object with some Ogre components.

### Properties

#### Ogre SceneManager

#### Ogre SceneNode

#### Ogre Entity

### Functions

#### Update Graphics

This method will update the graphics associated with this object. This should be called by update.

### XML Load Scheme

This scheme will be expanded to include several movable types in Ogre. This will include Particle Systems, Planes and Lights.

#### Mesh

This is the name of the mesh that will be loaded.

#### Position

3 Dimensional vector setting the scene node’s position.

#### Scale

3 Dimensional vector setting the scene node’s scale.

## Game Object Ogre Box2D

This object extends Game Object Ogre and adds a Box2D component to it.

### Properties

#### Body

This is the Box2D body.

#### World

This is a reference to the Box2D world which creates all objects in Box2D.

#### Debug SceneNode

This is the scenenode that per object debugging is attached to.

#### Debug Draw On

This is a Boolean that says whether or not debug draw is enabled.

### Functions

#### Begin Contact

This method is called when something touches a shape on this objects body. It reports what touched it and some more information

#### End Contact

This method is called when something touching a shape on this object’s body stops touching it. It reports what touched it and some more information

#### Redraw Debug

This updates the object that represents the debug information for the body of the object.

### XML Load Scheme

Several fixtures can be added to a body by creating several Fixture elements. BodyDef is not necessary but will allow easy setting of positions and other properties of the body.

#### BodyDef

This an element defining the b2BodyDef for the body. This is read using the Box2DXMLLoader.

#### Fixtures

These are elements that define the shapes for the body. These are read using the Box2DXMLLoader. There can be several Fixture elements.

## Game Object Sensor

This object extends GameObjectOgreBox2D. It is meant for detecting when a physical object is contacting something and sending out messages based upon that contact.

### Properties

#### SensorMessage

This is an object that stores an on and off message. The on message is sent when something contacts the sensor and the off message is sent when contact has stopped.

#### MessageList

This is a list of all the game objects that want to be notified when something contacts this sensor. This is stored as a list of sensor messages and game object ids. Each object added to the sensor can either set specific message it wants to be sent out when the sensor is contacted or get back the generic Sensor On and Sensor Off messages.

## Generic XML Loading Scheme

The scheme starts with an <Object> tag with an attribute describing what type of Object this is and then includes the specific Game Object tags. The example here shows a Game Object Ogre Box2D being created as a cube with a mesh.

<Object type="OgreBox2D">

<GameObject name="OgreBox2D\_Demo"/>

<OgreObject mesh="cube.1m.mesh" position="0 0 10" />

<Box2DObject>

<Body>

<BodyDef bodyType="dynamicBody" position="0 10"/>

<Fixture shape="box" boxWidth="1" boxHeight="1" density="10"/>

</Body>

</Box2DObject>

</Object>

# Generic State Machines

A Generic State Machine is a template class that can be added to a class so it can have a kind of modular behavior. A State Machine has an owner object. The owner object calls the state machine to update the owner object.

The state machine has can two states: a global state and a current state. The global state is always updated and is not likely to change between updates. The current state is updated every update but is likely to change between updates.

## Generic State

A state has an **Enter** function which is called when it is entered, and **Update** function which is called when the owners objects is updated, and an **Exit** function wen the state is exited. A state has an owner object that the state is a friend class of so that it can update the owner appropriately.

This is the diagram for a generic state.

Enter State ()

Update State ()

Exit State ()

# Parker’s State Machine States

Parker has three states currently which define all of his motion. The states are defined after the diagram.

Parker State Diagram

Feet Sensor Has Contacts

In Air State

On Ground State

Feet Sensor Has No Contacts

Ledge Grab State

Start

Contact Ledge Sensor Action Pressed

Jumped

Action Pressed –  
Climbed

## **In Air State**

The In Air State is when parker is completely air born and not touching any objects.

### Jump Button

In this state pressing the jump button will do one of three different things.

* Holding in jump after you’ve just jumped from the On Ground state will have you jump higher than if you did not hold in the jump button.
* Press the jump button while the wall jump sensors are hitting something you will do a wall jump off of whatever object you are touching.
* If you have the grappling hook attached to something and you press jump, you will jump with some force up.

### Move Left Button

The move left button will add a small force to the left.

### Move Right Button

The move right button will add a small force to the right.

### Move Down Button

The move down button will have you drop from the grappling hook if you are attached to it.

### Action Button

The action button makes Parker grab a ledge if he’s touching a ledge sensor.

## **On Ground State**

The On Ground State is when Parker’s feet sensors are touching something.

### Jump Button

Parker will jump.

### Move Left Button

The move left button will add a force to the left.

### Move Right Button

The move right button will add a force to the right.

### Move Down Button

The move down button will make Parker will crouch.

### Action Button

The action button will do several things.

* Activate switches
* Go through doors

## **Ledge Grab State**

The Ledge Grab State is s only entered from the In Air State.

### Jump Button

Parker will jump backwards off the ledge

### Move Left Button

The move left does nothing.

### Move Right Button

The move right button does nothing.

### Move Down Button

The move down button will make Parker drop down from the ledge.

### Action Button

The action button will make Parker climb over the ledge.

# Box2D XML Loader

This class returns Fixtures, Bodys, and Joints from XML files describing their properties.

## Fixture

### Properties

#### Shape - String

This can be set to circle, polygon, edge or box for easy creation of boxes.

#### boxWidth – float

This is the width of the box if the shape type is box.

#### boxHeight - float

This is the height of the box if the shape type is box.

#### Angle – float

This is the angle the shape is put at relative to the body.

#### Friction – float

This is the friction on the fixture

#### Restitution – float

This is how bouncy the fixture is

#### Density - float

Density of fixture

#### Category - String

This sets the shape to a certain collision category.

#### Group - String

This sets the shape to a certain collision group.

#### isSensor – bool

The shape is a sensor.

#### Radius – float

Radius of the circle if it a circle shape.

#### Center – 2D Vector

This is the center of the fixture

#### PointCount

This is how many points are in a polygon

#### Point[x]

These would be the points of the polygon. Used like point0=”1 3” point1=”3 5”.