Hello, although I think that everybody knows who I am, (possibly because all we are student of the same class of Cross Platform Development or maybe because Yes!, I’m a little egomaniacal), let me introduce myself: My name is Luis Pastor Abia and I’m going to talk about Box2D: Joints.

I know what can you think now: what the hell is the meaning of Box2D: Joints? (At least it was my first thought when I chose the topic). So, before coming to the point of my presentation I’ll try to answer that question simply.

Box2D is a 2D physics library. It is one of the most popular physics libraries for 2D games.

Yes. I know that all of you are programmers (or future programmers) and gamers so everybody knows what the word Physics means in games but I will not be happy if I don’t try to explain the subject for dummies so…

When we talk about physics we are not talking directly about Newton laws, complicated differentials equations, integrals, maths and… Well, really we are talking about that things but in a more… applied point of view: we are introducing the laws of physics into a simulation or game engine for the purpose of making the effects appear more realistic to the gamer.

It’s easy to guess the value of this topic and for this reasons although the Box2D library was originally a C++ library has been ported to many languages and many different engines, including libgdx.

There are many things that we can be made with Box2D and I have only 20 minutes so I recommend that if you want more exhaustive information you have to read the fucking manual (although it’s for C++) or the wiki for the libgdx wrapper.

I know, I know. You have little time so I have done a little summary for the libgdx users.

The first thing that we need to do is to initialize the Box2D calling Box2D.init. Easy, not? Then we have to create a world and it’s as simple as indicate the gravity of the world in a vector. If we want the real thing: 0 gravity in x axis and -9.8 in the y axis.

Then we need to see the debug renderer (the real thing… even when you sell the game you don’t show this and show the pretty textures that you are going to do).

Do you remember the delta? We need to step the simulation of our objects and world in the render loop. In an ideal world the frames and delta are all the same and we don’t have to do many things but I leave you a link to how time step. The main thing is to do the world step with the time step needed, and give the iterations needed to calculate the velocity and position. Remember this is a resume so feel free to investigate more…

Then we render the world using the camera and vualá…. Nothing happens because we only define the world and the world is empty… So we need to introduce other things (objects) in the Box2D:

We have static bodies (a wall for example), dynamic bodies (can move and you can apply forces and impulses to them) and kinematic bodies (for example you can do platforms).

Also we can move or modified the moves of the bodies applying forces or impulses.

We can make sensors or contact listeners to interact between objects in a more complex way that with only the collisions and a direct interaction.

Moreover We can use box2d-editor (free open source tool) to define complex shapes and load them into the game and finally we can use Box2D’s User Data to manage the link between our objects of Box2D and the pretty Sprites

And last but not least we have the JOINTS. Because without them all we can do are objects alone. With joints we can make complex joints between objects making more realistic actors.

But before talk about the joints.. Are you following me? I hope that but I think that this is better seen in one little example.

(Example 1)

As you see this is a pretty empty project. I initialize the Box2D, create the word, and the debug Renderer, create a body definition for a dynamic body and give it a little velocity and a circular shape, some other properties (I will talk after about some of that but let me to go to another point and only to run the project).

Pretty simple, not? And not too much different with things that we do in the flappee bee game… But you can sense the power of the physics… I don’t do anything to make the circle fall, I don’t do nothing to get the parabollic move of the circle. If I set the initial conditions of the physic I can obtain reallistic results (if my physics its reallistic). And if you modify the parameters you’ll obtain the results of the physics. Let me for example to make the restriction to cero and you will see that when the “circle” collisionate with the “ground” (I put the ground in movement for everybody loves platforms games…). I don’t have to tell you all you can do for pool games when the ellastic collision of the balls. Moreover, to apply forces or changes in speed is as simple as you can see if I press some keys… All the world under the Newton’s Law can be simulated and you can do other physics only changing the world!

Perhaps you think that it’s not as amazing as it seems to me (at the end seem that we have done those things easily without physics in the flappee bee game). But the thing is more amazing when you will see that we can do interaction between bodys physically accurates. That’s the work of the joints.

Joints are simply the junctions between bodies. We have several types of joints in Box2D. I’ll try to sumarice in the next slides. However I can say one important thing about all joints:

We have the union of two bodies with different anchors and constraints to give several degrees of freedom. Next we have to create the joint and the Box2D library will make the rest.

Weld joint: basically is that. You glue two bodys rigidly. You can imagine that it’s very useful to make more complex bodies.

Distance Joint: you define the anchors to the 2 body and the length of the union. The 2 bodies cand rotate around them but if you fixed and modified the length in the code you could do things as the example of the piston. Do you have in mind pinballs games? Angry bird games? Open and close doors making the length=0?

Friction Joint: Angular and linear friction between bodies…

Prismatic joint: one body can move in a axis around the other body. If the axis is paralel to one body It’s similar to change the length in the distance joint. Platforms ;) ?

Revolute joint: one body spin around the other over a point . Machines… pistons more realistic, Wheels in vehicles. Althought for that it’s better the wheel joint.

Gear joint: joint of joints. Do you have a machine in mind as a bycicle string?

Pulley joint: The word said it all: pulley. One length grown another get small. Nobody play games with the character activating things?

Rope joint: ropes ropes ropes.. The distance maximum it’s one but the joint can simulate a rope with flexibility. Somebody has in mind how simulate spiderman movement? or a pendulus with axes to try to kill your character?

wheel joint: one body is spinning and another moving linearly… Suspensions.

In summary: how can be do this thing?

Two body, making the joint definition with the anchor, and constraints and the type of joint that we want.

Creating the joint in the world.

Next we can see some simple examples.

Example 2: Distance Joints (Repeat the things)

Example 3 : Mouse Joints (It’s not very physical but for a gamer point of view…) The things a little different its with the input adapter.