# How to add buttons in Gdx

rst I create my button image, add it to the assets folder and load the texture region. Now I make a sprite out of it.

Sprite button1=new Sprite(myTextureRegion);

To check if the button is touched I can use the rectangle from the sprite to check if you touched the image. In your touchUp method you will do something like

if(button1.getBoundingRectangle.contains(screenX,screenY))

# [java - merge adiacent rectangles into a polygon](http://stackoverflow.com/questions/20997254/java-merge-adiacent-rectangles-into-a-polygon)

[**Rectangle**](https://libgdx.badlogicgames.com/nightlies/docs/api/com/badlogic/gdx/math/Rectangle.html#Rectangle-float-float-float-float-)(x start, y start, float width, float height)

Constructs a new rectangle with the given corner point in the bottom left and dimensions.

[**merge**](https://libgdx.badlogicgames.com/nightlies/docs/api/com/badlogic/gdx/math/Rectangle.html#merge-com.badlogic.gdx.math.Rectangle-)([**Rectangle**](https://libgdx.badlogicgames.com/nightlies/docs/api/com/badlogic/gdx/math/Rectangle.html) rect)

Merges this rectangle with the other rectangle.

Drawing the I shape

1. Create an l shape
2. Create the Tshape

Drawing the L shape

1. Create an l shape
2. Create an - shape
3. Merge I with – but do we merge precicely. Check your positioning!
4. You now have L
5. Rotate it around
6. Rotate up() to get 7
7. Rotate down to get l-
8. Rotate again to get –l

Drawing the square

1. Make height and width the same
2. No rotations needed

Drawing the

ines **#12-#15** define some constants we will use to calculate the speed and positions in the world. These will be tweaked later on.

We also need some blocks to make up the world.  
The Block.java class looks like this:

|  |  |  |
| --- | --- | --- |
| 01 | package net.obviam.starassault.model; | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | import com.badlogic.gdx.math.Rectangle; | |
| 04 | import com.badlogic.gdx.math.Vector2; |

|  |  |
| --- | --- |
| 05 |  |
| 06 | public class Block { | |

|  |  |
| --- | --- |
| 07 |  |
| 08 | static final float SIZE = 1f; | |

|  |  |
| --- | --- |
| 09 |  |
| 10 | Vector2  position = new Vector2(); | |

|  |  |  |
| --- | --- | --- |
| 11 | Rectangle  bounds = new Rectangle(); | |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | public Block(Vector2 pos) { | |
| 14 | this.position = pos; |

|  |  |
| --- | --- |
| 15 | this.bounds.width = SIZE; |
| 16 | this.bounds.height = SIZE; | |

|  |  |  |
| --- | --- | --- |
| 17 | } | |
| 18 | } |

Blocks are nothing more than rectangles placed in the world. We will use these blocks to make up the terrain. We have one simple rule. Nothing can penetrate them.

**libgdx note**

You might have noticed that we are using the Vector2 type from libgdx. This makes our life considerably easier as it provides everything we need to work with Euclidean vectors. We will use vectors to position entities, to calculate speeds, and to move thing around.

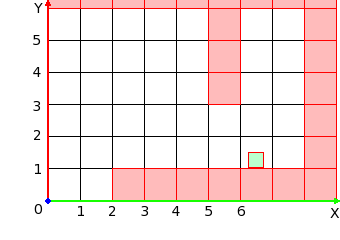
**About the coordinate system and units**

As the real world, our world has dimensions. Think of a room in a flat. It has a width, height and depth. We will make it 2 dimensional and will get rid of the depth. If the room is 5 meters wide and 3 meters tall we can say that we described the room in the metric system. It is easy to imagine placing a table 1 meter wide and 1 meter tall in the middle. We can’t go through the table, to cross it, we will need to jump on top of it, walk 1 meter and jump off. We can use multiple tables to create a pyramid and create some weird designs in the room.

In our star assault world, the world represents the room, the blocks the table and the unit, the meter in the real world.

If I run with **10km/h** that translates to **2.77777778 metres / second ( 10 \* 1000 / 3600)**. To translate this to Star Assault world coordinates, we will say that to resemble a 10km/h speed, we will use 2.7 units/second.

Examine the following diagram of representing the bounding boxes and Bob in the world coordinate system.

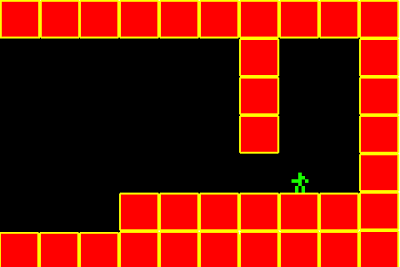
[](http://1.bp.blogspot.com/-sKL9IRmoKbo/T6JJRP8cmkI/AAAAAAAAAUM/cc5Nh3Gccok/s1600/coordinate-system.png)

The red squares are the bounding boxes of the blocks. The green square is Bob’s bounding box. The empty squares are just empty air. The grid is just for reference. This is the world we will be creating our simulations in. The coordinate system’s origin is at the bottom left, so walking left at 10.000 units/hour means that Bob’s position’s X coordinate will **decrease** with **2.7 units every second**.

Also note that the access to the members is package default and the models are in a separate package. We will have to create accessor methods (getters and setters) to get access to them from the engine.

**Creating the World**

As a first step we will just create the world as a hard-coded tiny room. It will be 10 units wide and 7 units tall. We will place Bob and the blocks following the image shown below.

[](http://2.bp.blogspot.com/--jbdeqKWJ80/T6JJdvMOmzI/AAAAAAAAAUU/gGFySbB5IBY/s1600/level011.png)

The World.java looks like this:

|  |  |  |
| --- | --- | --- |
| 01 | package net.obviam.starassault.model; | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | import com.badlogic.gdx.math.Vector2; | |
| 04 | import com.badlogic.gdx.utils.Array; |

|  |  |
| --- | --- |
| 05 |  |
| 06 | public class World { | |

|  |  |
| --- | --- |
| 07 |  |
| 08 | /\*\* The blocks making up the world \*\*/ | |

|  |  |  |
| --- | --- | --- |
| 09 | Array<Block> blocks = new Array<Block>(); | |
| 10 | /\*\* Our player controlled hero \*\*/ |

|  |  |  |
| --- | --- | --- |
| 11 | Bob bob; | |
| 12 |  |

|  |  |
| --- | --- |
| 13 | // Getters ----------- |
| 14 | public Array<Block> getBlocks() { | |

|  |  |  |
| --- | --- | --- |
| 15 | return blocks; | |
| 16 | } |

|  |  |  |
| --- | --- | --- |
| 17 | public Bob getBob() { | |
| 18 | return bob; |

|  |  |
| --- | --- |
| 19 | } |
| 20 | // -------------------- | |

|  |  |
| --- | --- |
| 21 |  |
| 22 | public World() { | |

|  |  |  |
| --- | --- | --- |
| 23 | createDemoWorld(); | |
| 24 | } |

|  |  |
| --- | --- |
| 25 |  |
| 26 | private void createDemoWorld() { | |

|  |  |  |
| --- | --- | --- |
| 27 | bob = new Bob(new Vector2(7, 2)); | |
| 28 |  |

|  |  |
| --- | --- |
| 29 | for (int i = 0; i < 10; i++) { |
| 30 | blocks.add(new Block(new Vector2(i, 0))); | |

|  |  |  |
| --- | --- | --- |
| 31 | blocks.add(new Block(new Vector2(i, 7))); | |
| 32 | if (i > 2) |

|  |  |  |
| --- | --- | --- |
| 33 | blocks.add(new Block(new Vector2(i, 1))); | |
| 34 | } |

|  |  |
| --- | --- |
| 35 | blocks.add(new Block(new Vector2(9, 2))); |
| 36 | blocks.add(new Block(new Vector2(9, 3))); |

|  |  |
| --- | --- |
| 37 | blocks.add(new Block(new Vector2(9, 4))); |
| 38 | blocks.add(new Block(new Vector2(9, 5))); |

|  |  |
| --- | --- |
| 39 |  |
| 40 | blocks.add(new Block(new Vector2(6, 3))); | |

|  |  |
| --- | --- |
| 41 | blocks.add(new Block(new Vector2(6, 4))); |
| 42 | blocks.add(new Block(new Vector2(6, 5))); |

|  |  |  |
| --- | --- | --- |
| 43 | } | |
| 44 | } |

It is a simple container class for the entities in the world. Currently the entities are the blocks and Bob. In the constructor the blocks are added to the blocks array and Bob is created. It’s all hard-coded for the time being.

Remember that the origin is in the bottom left corner.