Libgdx - Tiled Maps Tutorial

When designing a **board game** like **RoboRally**, one may think of possible already available technologies that can be used to visually represent and interact with such a board. **Game engines** in general were built for exactly these tasks. They provide different visual representations and various graphical interaction methods that support you creating such games. In this tutorial, we will introduce you to **Libgdx**, a free and open-source game-development application framework written in Java. Libgds already provides a generic **maps API** and also a more specific **Tiled Map representation**. We will provide you with the necessary steps required to design levels using a level editor called **Tiled**, export it and then import it in your personal project. First, we will start by designing a grid-based level.

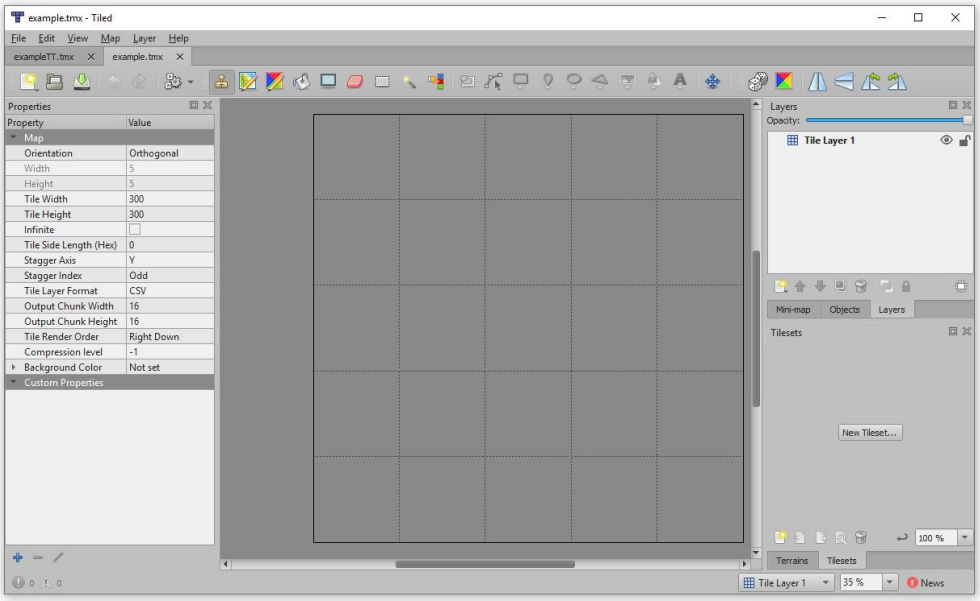
Tiled - Level Editor:

Tiled is a free and easy to use level editor (**https://www.mapeditor.org/index.html**). In this tutorial, we will use a typical top-down level design based on a grid similar to the one you will need to develop in *RoboRally*. The first step is downloading and running the software. After starting the program you can create a new level by pressing: **File/New/New Map…** or **Ctrl+N**

The opening dialog will let you choose important parameters of the level. In this tutorial, we will go with the following settings (for your group project you will need to adapt this configuration accordingly):

Orientation	- Orthogonal
Tile layer format	- CSV
Tile render order	- Right Down
Map size	- fixed
Width	- 5 tiles
Height	- 5 tiles
Tile size:	
Width	- 300 px
Height	- 300 px

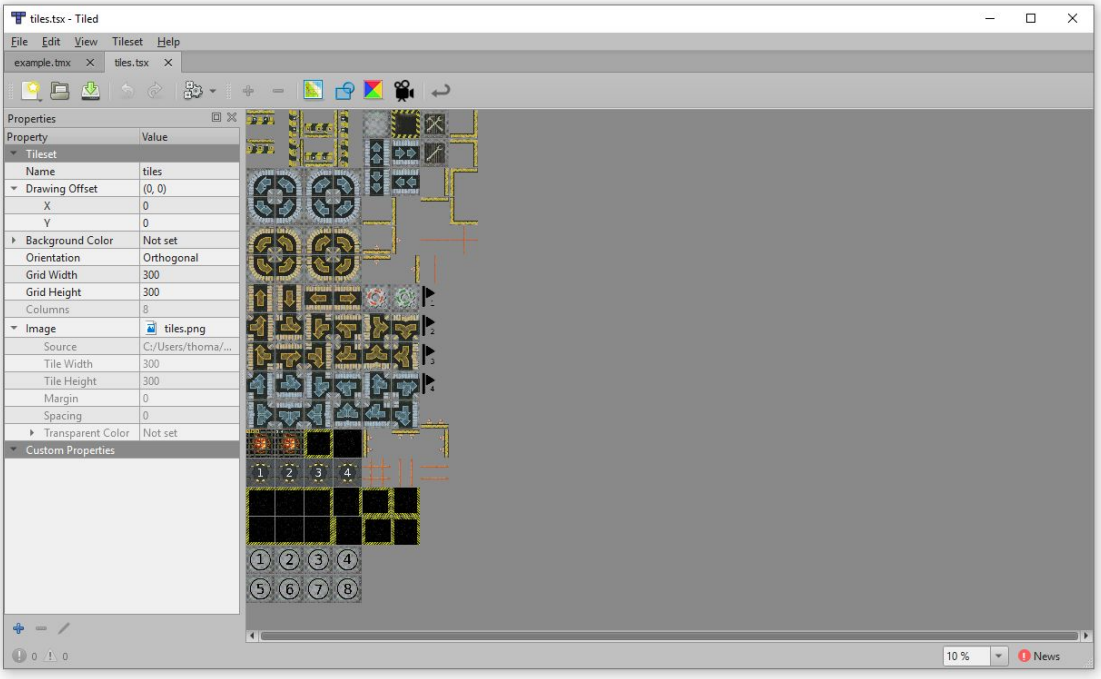
This file should then be saved in the **assets** folder of your Java project (in the following we will assume you use the **mavenTemplate-master** project). If you have not created an **assets** folder, it is time to do this in your projects directory. The **assets** folder is also the right place for you to copy the two images provided by us (**tiles.png** and **player.png**). We are using RoboRally tiles provided by http://www.yeoldewebsiteknight.co.uk/roborally. In the meantime, **Tiled** will show you something similar to the following screenshot:



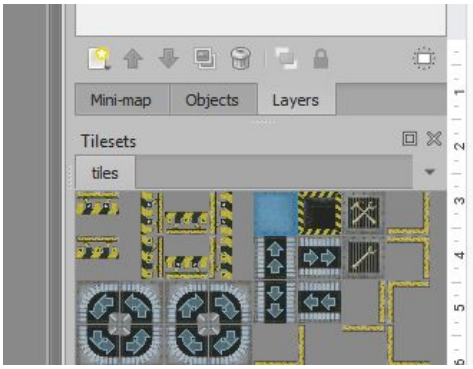
As you can see, the level consists of 5x5 tiles where every tile has the dimensions of 300x300 px.

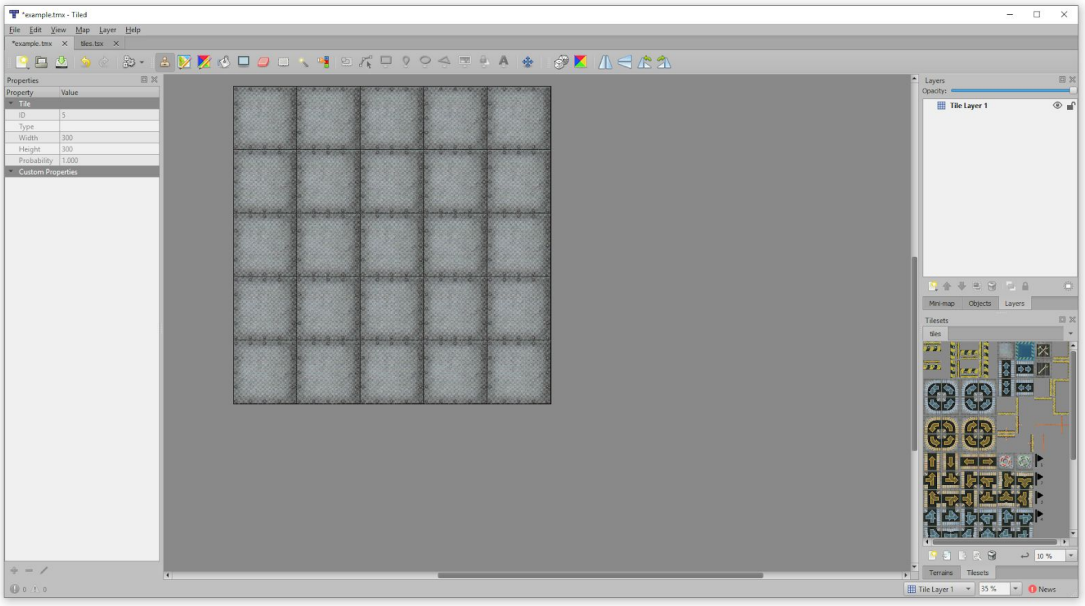
Next, click on the **New Tileset…** button on the lower right side which will open a new dialog. There you need to browse for the **tiles.png** file, the one you just copied into the **assets** folder. After selecting the file, make sure that **Tile width** as well as the **Tile height** are both 300 px and **Margin** and **Spacing** are both 0 px. Next, save this file as **tiles.tsx** in your assets **folder** by pressing the **Save As…** button.

This should lead to something that looks similar to the following screenshot (maybe it is necessary for you to **zoom out** to see the complete image):



Next, go back to the **example.tmx** tab. In the lower right corner you will now see the tiles you have just loaded. Here, we recommend to change the percentage from **100%** to, for example **10%**. You can simply do this by manually typing numbers in the text field. Then, click on the **default floor tile** in the first row (the fifth element starting from the left), highlighted in **light blue** in the figure on the right. You can now assign this tile to every tile within the grid. Our current level will then look similar to the screenshot below:

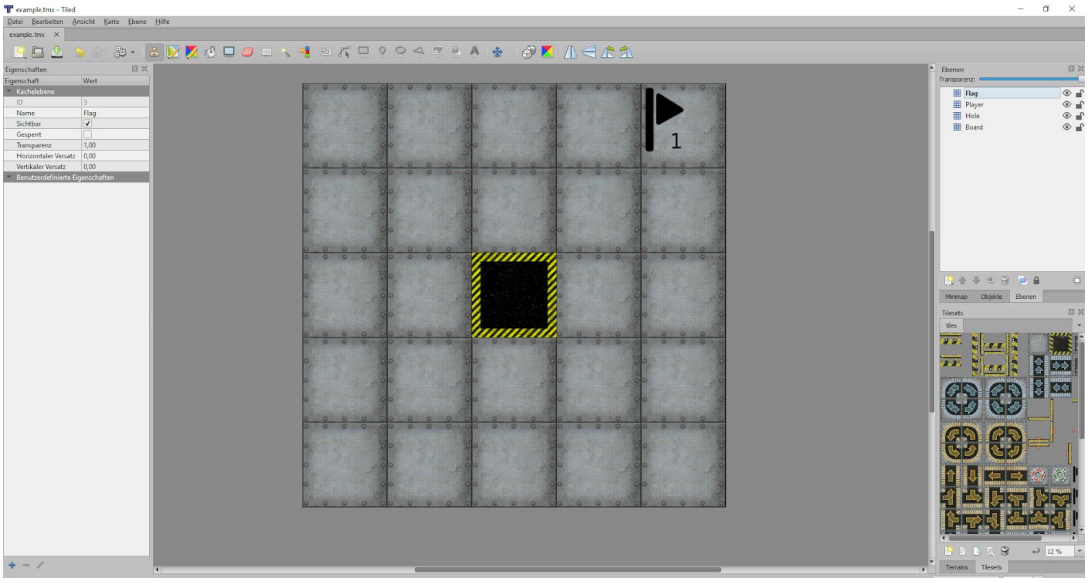




Next, we want to make this level a little bit more exciting. What about possible obstacles or a goal the player should move to? Before we do so, we need to change the name of the current **Layer**. This will help us to keep track of the individual layers. Instead of **Tile Layer 1**, we change it to **Board**. You can do so by double-clicking on the name in the upper right corner.

Let's add an obstacle in the middle of the level. Therefore, we add an additional layer called **Hole**. Do this by clicking on **Layer/New/Tile Layer…** and name it accordingly. Next, select the sixth element in the first row of the **Tileset** (right next to the **default floor tile** we used before). We will also use this hole tile to later trigger a *losing-condition* in our **libgdx** tutorial.

Next we want to add a possible *winning-condition* as well. First, we create a new layer called **Flag**, then select the flag tile from **Tilesets** and put it in the top-right corner of the level. In the end, also add a new layer called **Player**, where the player will be placed and can move around afterwards. The level should then look similar to the following screenshot:



[Here we want to mention again that in the final RoboRally game, you will need to come up with your own solution of how to deal with other barriers, tools, conveyor belts, lasers, and so on. Do you want those in separate layers? What are possible advantages/ disadvantages? The intention of this tutorial is just to show you some possible approaches.]

Let's continue with the tutorial. Since we have now done quite some updates, save all your recent changes in the **example.txm** file. Next, open the file with an editor (**Notepad**, **Notepad++**, **Sublime**, or similar). Look at the files we have just created. What do you notice? How is the visual representation (our tiled board) encoded in this computer readable format (.tmx)t? As a final step, make sure that file **tiles.tsx** has an **image source** address that is simply the png image **source="tiles.png"** (remove a possible path like **../user/folder/folder/.../tiles.png** if there is one). This concludes the first section of how to create a level.

Congratulations!! You now know how to create grid-based levels supporting multiple layers in **Tiled**.

Libgdx:

For this section, you can use the already provided **mavenTemplate-master** project. The result of the prior step has been saved in the **assets** folder of your project. In order to make them available to use them in your code, you have to configure **Maven** to load them as **resources**. To do so you have to edit the **pom.xml** file (located in your projects directory). In the file there is a build section, where you have to add the assets folder as a resource and for making it available while testing as a test resource. Add the following code to support this functionality:

```
<build>
      ...
      <resources>
       <resource>
         <directory>${basedir}/assets</directory>
       </resource>
```

```
        </resources>
        <testResources>
         <testResource>
           <directory>${basedir}/assets</directory>
         </testResource>
        </testResources>
        ...
        ...
        ...
</build>
```

Now, open **Main.java** and change the the **width** and **height** of the **LwjglApplicationConfiguration** to 500 each and choose an appropriate application **title** and change this accordingly.

Showing the prior created board

Let's start by displaying the map. Therefore, open **HelloWorld.java** and add class variables that represent the **TiledMap**, and multiple **TiledMapTileLayer** for all the three individual layers (**Board**, **Player**, **Hole**, and **Flag**) we created using the **Tiled** level-editor.

Then, initialize the **TiledMap** in the **create()** function using a new instance of the **TmxMapLoader** and its **load(...)** function. The load function requires a String containing the path to the **example.tmx** file. Next, you can initialize the **TiledMapTileLayer** using the **getLayers().get(...)** function that the instance of **TiledMap** provides. Here, it is important that you do not forget to cast the result of **getLayers(...)** to a **TiledMapTileLayer**.

```
boardLayer = (TiledMapTileLayer) map.getLayers().get("Board");
```

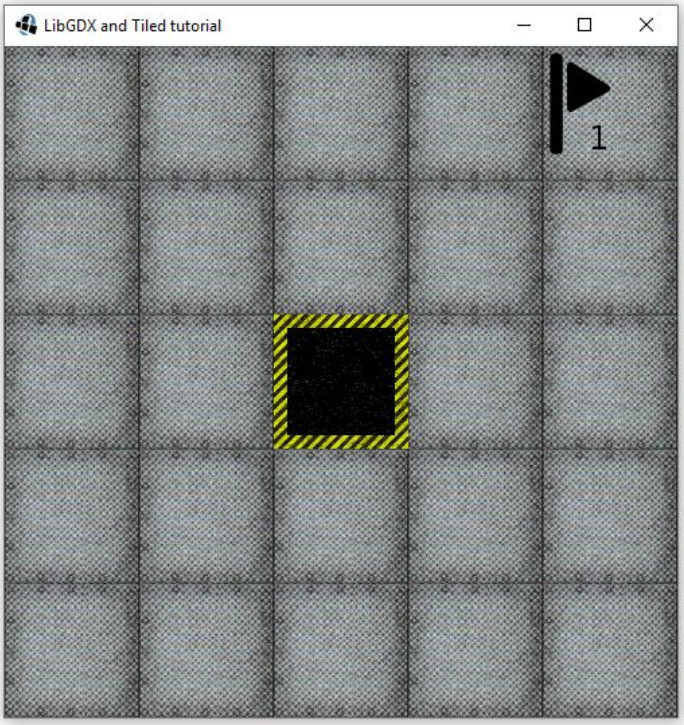
To be able to "see" the level, we need a camera and a renderer. Therefore, you need to create two new member variables of the type **OrthogonalTiledMapRenderer** and **OrthographicCamera**. Next, go back to the **create()** function and initialize both variables. For the camera, simply call the constructor. Then call **.setToOrtho(...)**. This function requires three variabel, the first is the boolean **yDown** which we will set to false and the other two represent the **viewportWidth** and the **vieportHeight**. Set both to 5 since our level consists of 5x5 blocks.

```
camera.setToOrtho(false, MAP_SIZE_X, MAP_SIZE_Y);
```

Then, change the **x-position of the camera** to a float value between 0 and 5 since we want the camera to be centered over the level. *What value do you have to pick for this?* Finally, call the **update()** function of the camera to update it.

Next, we need to initialize the **OrthogonalTiledMapRenderer**. The constructor requires the already created instance of **TiledMap** and a **float** unitScale which is computed as 1 divided by the number of pixels a single tile has (check the **Tiled** part of the tutorial if you are unsure what we picked back then). Now, we need to assign the camera to the renderer by calling the **.setView(...)** function.

Next, go to the **render()** function and remove everything except setting the clear-color and clearing the color buffer bit. Instead, we now simply call the **render()** function of the **OrthogonalTiledMapRenderer**. Run the program and you should see something like this:

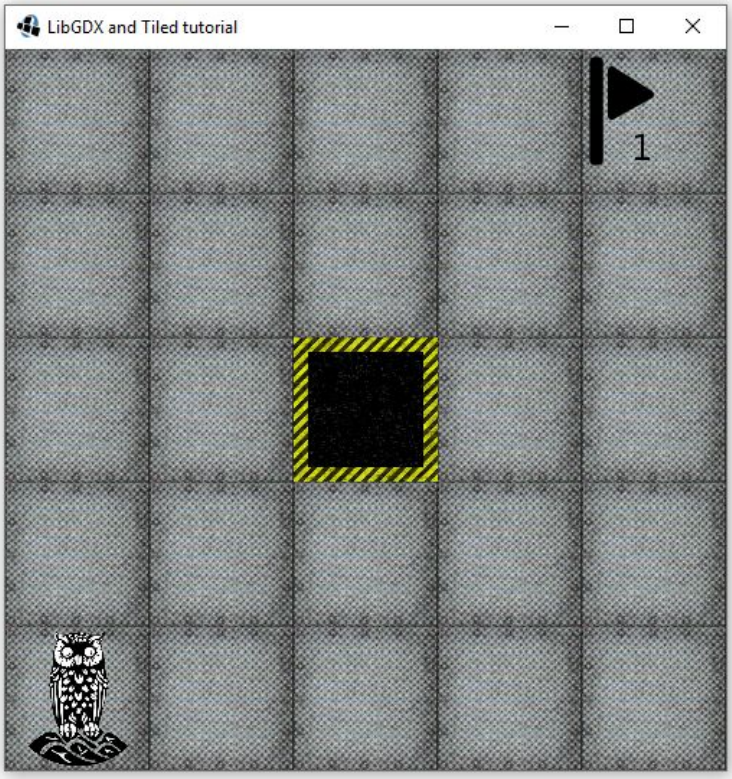


Displaying a player instance on the board

Next, we want to add a player. Therefore we need some more member variables for the different **Cells** representing the normal player **Cell**, the player won and the player died **Cell** as well as the 2D position **Vector2** of the player.

After creating those variables, go back to the **create()** function. In the create function we load the **Texture** py passing a String containing the name **player.png** which should also be in the **assets** folder of your project. The picture provided contains all the player pictures, so we have to split it. Therefore use the **split** functionality of the **TextureRegion** class. The function needs the prior loaded Texture as input as well as the tileWidth and tileHeight of the Player. Check the values, which are the same as the tiles of the board used in the Tiled tutorial. After splitting the player.png you will end up with a two dimensional array containing the different player textures. Create now a new **Cell** for each player option **(playerWonCell, playerDiedCell, playerCell)** by calling the constructor and afterwards the function **setTile(...)**. The tile needs a new **StaticTiledMapTile** as input, which constructor takes a **TextureRegion** as input. This **TextureRegion** has been obtained by calling the split function. Now, you should have the three cells representing the three states of the player.

Initialize the 2D vector containing the players position, by putting the owl representing the player in the lower left corner of the grid (think of what coordinates are required for this and play around with them). Next, go to the **render()** function. To display the player on the board you need to add the player cell to the **playerLayer** by calling the function **setCell(...)**. The function takes the cell position and the playerCell as input. The result will look like this:



Cool right? But wait a minute… we cannot even move the player! Let's change this.

Moving the player over the board

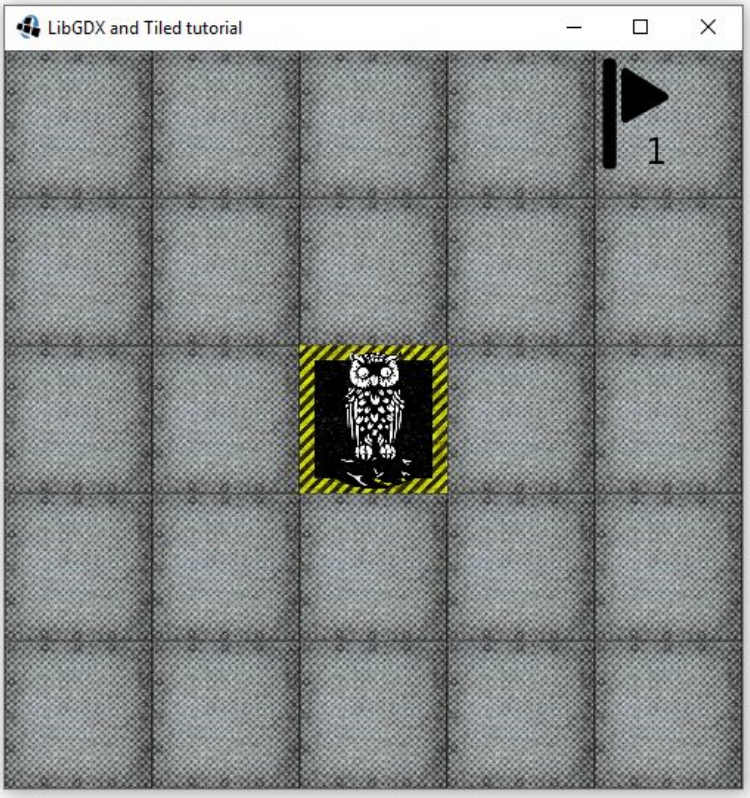
Make the **HelloWorld.java** class extend **InputAdapter**. This class can be used to access pressed buttons on the keyboard. Therefore, we need to additionally register the input processor. This can be done by calling **Gdx.input.setInputProcessor(this);** inside your **create()** function.

Next, we need to override the **keyUp(...)** function. (Why do we overwrite **keyUp** and not **keyDown**? Take a minute to think about it, what would change if we use **keyDown** instead? In which cases is the keyDown function useful?) As parameter the function gets an int code that can be used to find out which button was pressed. Check if the passed code is similar to Input.Keys.LEFT, Input.Keys.RIGHT, Input.Keys.DOWN, Input.Keys.UP. This way we can use the arrow keys of our keyboard to change the position of the player. (What would you need to do to use W,A,S, and D instead? Do not forget to try it out afterwards!). Before moving the player also think about clearing the cell where the player is currently on the player board. If you don't clear the cell the player will be in two cells at a time.

```
@Override
public boolean keyUp(int keycode) {...}
```

If one of these buttons was pressed, change the position of the player accordingly. Additionally, you have to think about situations where the player could "fall" out of the map. How do you want to deal with these situations? After solving this problem, let's run the program again. The result should look

something like the following screenshot (notice the position of the owl is of course dependent on where you have moved the player. Note that preventing the player to fall off the board is maybe not important for *RoboRally*):

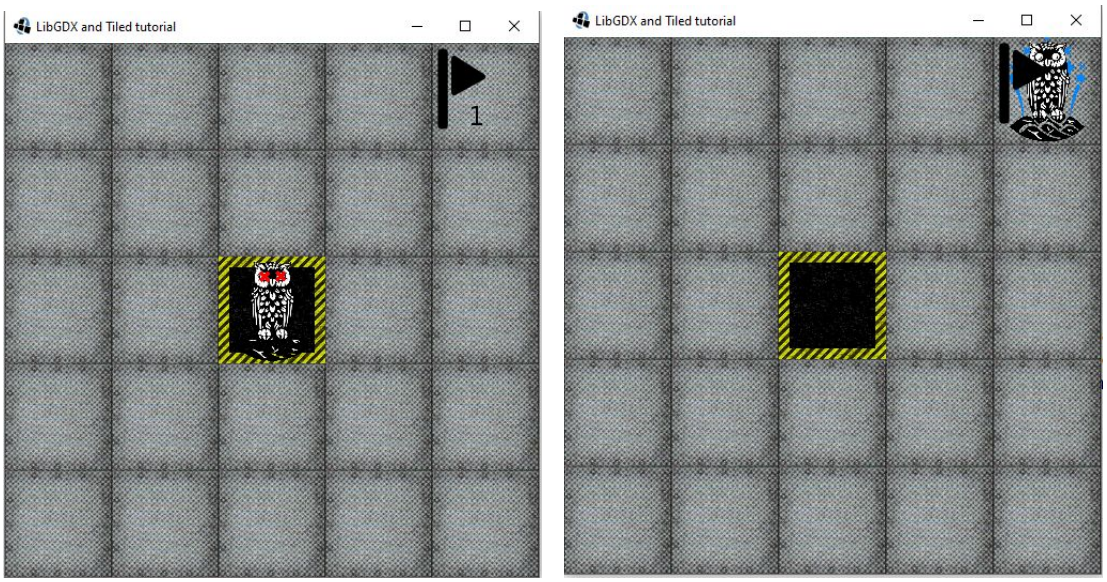


We can now additionally control and move the player. The only problem we still have is: our player is ignoring the win-lose conditions. Currently, we can simply move over a hole or the flag without any consequences. Let's fix that!

Consider special fields of the board

To take the special fields into account we are using the two other cells reflecting the win and lose condition. Next, go to the **render()** function and use the **TiledMapTileLayer** instances you created for the hole and flag layer to call **getCell(...)** with the current player coordinates. This helps you writing game logic that reacts to dynamic player positions. If this function returns **null**, there is no tile on the corresponding layer, otherwise you know that the player just walked over one of those special fields.

If this happens, change the **playerLayer.setCell(...)** call to use either the **playerWonCell** or the **playerDiedCell**. This will change the player representation according to the field the player is standing on. The result should look something like the following screenshots:



Congratulations, you have now finished the second part of the tutorial! Now you know how to load a level that was created using **Tiled**, create an avatar for a player, move the player dependent on keyboard buttons, and wrote a simplified game logic that reacts to the dynamic movements of the player! Well done!