Tile Editor Specifications for Project Zecharabra

The purpose of this tile editor is to take individual tiles images and give them definition.

What definition do we give to the tile images?

* a unique permanent id
* the path to the tile image
* the path to another tile image (if you want the tile to be animated/have multiple frames)
* the relative path to a passive sound effect (plays within earshot)
* the relative path to an active sound effect (plays when stepped on)
* passability - from which directions can the player enter this tile (4-directional)

\*\*When a tile is created, it receives a unique id. This id will NEVER be changed. Even if the tile is deleted, the id will never be used again. Take not that the user should be able to manually organize the tiles so (e.g.) all the water tiles are grouped together. In your code, you should not use the id's to organize the tiles. You should use some other variable only available to your code. When the tile editor exports its xml, it will record how the tiles are organized by writing the tiles to the xml file in the appropriate order. It's possible that the tile with id 25 will occur earlier in the xml file than the tile with id 0.

What is a relative path in this case?

This is how the program's directories will be structured:

* game.jar
* enemyeditor.jar
* leveleditor.jar
* tileeditor.jar
* src
  + zecharabra (project files)
  + resources
    - levels
    - tiles
      * water
        + sea0.png
        + sea1.png
    - sfx
      * water
        + sea\_passive.wav
        + sea\_active.wav

Note that the tile editor is going to be placed in the same directory as the game and the rest of the editors; this way, each program has the same perspective.

The relative path of each tile image will have the form

src/resources/tiles/…/<image\_name>.<image\_extension>

Since the first part of the path is redundant and implied, you only have to save the path to the right of "tiles/". The same is true for sound effects (sfx), except the relative path will have the form

src/resources/sfx/…/<image\_name>.<image\_extension>

How is tile definition being saved?

The tile editor will export an xml that can be used in other editors and the game. The xml will be structured as follows:

<tiledef>

<tile>

<id>0</id>

<path>water/sea0.png</path>

<path>water/sea1.png</path> //if multiple frames

<psfx>water/sea\_passive.wav</psfx>

<asfx>water/sea\_active.wav</asfx>

<passability>F</passability> //hex notation, flags

</tile>

</tiledef>

\*\*Subtags are used rather than attributes because subtags are faster.

There must be at least one path tag. If there are multiple path tags, the level editor and game will group them together into one animated tile (looped).

If the user hasn't specified a particular field, do not include it in the xml. For example, if the user didn't specify a passive sound effect (psfx), simply don't record that tag in the xml. You may do this for the following tags:

* path (note that at least 1 path must exist)
* psfx
* asfx
* passability

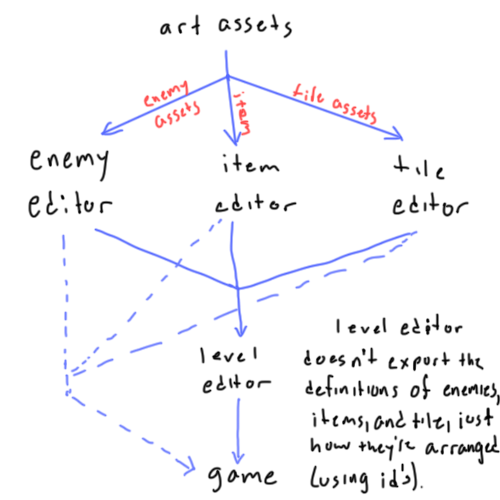
The user can't modify the id.

The passability will have a value from 0~F, depending on which flags are set. Consider the bits 03020100. The bits represent the following:

* 00 - passable from the right
* 01 - passable from the bottom
* 02 - passable from the left
* 03 - passable from the top

Why is this tile editor important?

The tile editor serves as a bridge between tile art files and the level editor and game.



Note that the enemy, item, and tile editors each export an xml that gives definition to a different type of thing, and that the level editor reads all three xml's. The level editor now has a finite list of things that can arranged into a level. The level editor will then export an xml of its own; however, this xml only contains *references* to enemies, items, and tiles (i.e. tile id). It doesn't export the definitions themselves; if the level editor's xml alone were read by the game, it wouldn't have enough information to be able to run it. This is why the xml's exported by the first three editors are also loaded into the game.

What's interesting is that a tile can be updated in the tile editor and automatically reflect changes in the game without having to go through the level editor again.