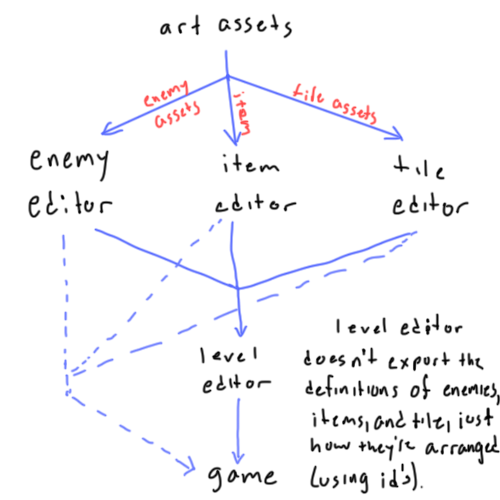
Level Editor Specifications for Project Zechorabra

The purpose of this level editor is to create a finite list of enemies, items, and tiles (by accepting xml's exported by the enemy, item, and tile editors), let a user arrange them into a level, and export an xml that can be read by the game. 

How is the enemy xml formatted?

<enemydef>

<enemy>

<id>0</id>

<path>wraith/</path>

<name>Wraith</name>

…

</enemy>

</enemydef>

The enemy xml only saves the directory (path) where the enemy's assets are located. This is because each enemy MUST have the following assets in that directory:

* idle (4-directional)
* walking (4-directional)
* attacking (4-directional)
* damaged (4-directional)

\*\*Each action (e.g. walking\_right) may have multiple frames to create an animation (i.e. walk\_left0.png, walk\_left1.png).

The level editor is only concerned with the first frame of each idle action (idle\_right0.png, idle\_down0.png, idle\_left0.png, & idle\_up0.png). This way, it can denote which direction the enemy starts off facing.

Enemies should be organized by name.

How is the item xml formatted?

<itemdef>

<item>

<id>0</id>

<path>potion.png</path>

<name>Potion</name>

…

</item>

</itemdef>

In the item editor, the user may organize the items arbitrarily. The order is saved by writing the first item earliest in the xml. Because of this, the item with id 25 can occur earlier than the item with id 0 in the xml. Do not organize the items by id; keep them in the order that they occur in the xml.

How is the tile xml formatted?

The tile xml is formatted 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>

In the tile editor, the user may organize the tiles arbitrarily. The order is saved by writing the first tile earliest in the xml. Because of this, the tile with id 25 can occur earlier than the tile with id 0 in the xml. Do not organize the tiles by id; keep them in the order that they occur in the xml.

Tiles may have multiple paths if the tile is animated. Only use the first path in the level editor (the editor doesn't need to show animation).

How to use the paths provided by the xml's

The xml's save a truncated version of the full relative path. For better understanding, this is how the programs' directories will be structured:

* game.jar
* enemyeditor.jar
* itemeditor.jar
* leveleditor.jar
* tileeditor.jar
* src
  + zecharabra (project files)
  + resources
    - levels
    - enemies
      * wraith
        + idle\_right0.png
        + idle\_right1.png
        + walk\_left0.png
        + walk\_left1.png
    - items
      * potion.png
    - sfx
    - tiles
      * water
        + sea0.png
        + sea1.png

In the example xml above, the path to the tile image is water/sea0.png. The complete relative path should include the prefix src/resources/tiles/. In this case, the complete relative path would be

src/resources/tiles/water/sea0.png

The following prefixes should be added to the corresponding paths:

tiles src/resources/tiles/  
enemy src/resources/enemies/  
item src/resources/items/

How the level editor should be structured

**Tile-based.** The level editor should assume the user would like to use a snap-to-grid system, and let the user define cell size. Cells are squares, so a cell width and a cell height are redundant; there only needs to be 1 field called "cell size". The level width and height should be measured in cells, not pixels.

**Layering**. The level editor should allow you to add an unlimited number of layers. The 0th layer is on the bottom; any new layer will stack on top of that. These layers are movable, hideable, and deletable, much like Photoshop. There is a special layer in which the user can place a generic event tile. Each event has a unique id. When the user places an event tile, the level editor should display the id in that cell for the game programmer's convenience. In game, when the player walks onto a tile, it will check if that tile has an event associated with it. If it does, it will run the event logic as defined by the game programmer.

What the user should be able to do

The user should be able to choose from a list of tiles, enemies, and items. If an enemy is placed, the user should be able to choose which direction it faces. Tiles and items cannot be rotated.

The user can place anything from these lists onto any layer, as well as delete them from the level.

The user should be able to create, move, and delete whole layers.

The user should be able to add and delete events, but only to the event layer.

The user should be able to load an xml level and save a level as xml.

How the exported xml should be formatted

<level>

<cellsize>32</cellsize>

<width>100</width>

<height>100</height>

<eventlayer>

<event>

<id>0</id>

<x>21</x>

<y>25</y>

</event >

</eventlayer>

<layer>

<tile>

<tileid>45</tileid>

<x>21</x>

<y>25</y>

</tile>

</layer>

<layer>

<enemy>

<enemyid>12</enemyid>

<x>15</x>

<y>17</y>

<facing>left</facing>

</enemy>

<item>

<itemid>15</itemid>

<x>10</x>

<y>13</y>

</item >

</layer>

</level>

If there are no events, don't include the eventlayer at all.

There can be multiple layers. The layer earliest in the xml is the bottommost layer in the game.

In the eventlayer, id is a unique event id. This id will be referenced in the game code.

In the layer, the tileid refers to a particular tile's id as defined by the xml exported by the tile editor. The same is true for enemyid and itemid but with the enemy xml and item xml, respectively.

It would be nice to have history functionality (undo, redo).