the zoidberg engine

Search this site

136
days since
Project Due Date

Recent site activity

removed stuff
edited by Joe Balough

The Team
edited by rmb5299

removed stuff
edited by Joe Balough

cliCreator
edited by Joe Balough

Testing Framework
edited by Joe Balough

View All

cliCreator

Contents

The cliCreator is a tool written in C++ to be used to build a zbe datafile based off of an XML description. It is designed to be platform independent by being written in C++ using the open-source TinyXML library for all XML parsing routines.

Building the cliCreator

Within the cliCreator directory there is a Makefile that will build the cliCreator for you. All you need to do is run

```
make
```

and it will create an executable called cliCreator.
The build is confirmed to work on Linux, Windows, and Mac OS X with no warnings. It should compile and run without any issues. However, you must install mingw to compile on Windows and add the mingw bin folder to your path environment variable.

Running the cliCreator

cliCreator uses the `getopt()` functionality to parse command-line arguments and supports the following arguments:

`-i filename.xml`	Required, tells the cliCreator to parse the `filename.xml` file to build the zbe file.
`-o filename.zbe`	Optional, defaults to assets.zbe. Tells the cliCreator to output the created zbe file to `filename.zbe`. WARNING: cliCreator will overwrite this file without asking.
`-v`	Optional, enables verbose output. cliCreator outputs no text unless something goes wrong without this option.
`-t`	Optional, enables testing zbe file output. This flag will cause the cliCreator to make a zbe file to be used with the testing built of the zoidberg engine. See Testing Framework for more details.

If you run cliCreator without the `-i` argument, it will print its usage. If you pass the `-v` flag without the `-i` argument, it will give a detailed description of the XML file format.

XML Description

The XML used to generate a zbe file is designed to mirror the structure of the zbe file itself in a way that is human-readable.

Header and root node

The file begins with an XML version tag

```
<?xml version="1.0" ?>
```

The root node of the file should be a `zbe` tag

```
<zbe>
```

```
...
</zbe>
```

Binary data definitions

The first section of the XML file that is parsed by the cliCreator is the `bin` section that describes all of the binary data that should simply be copied into the zbe output file.

```
<bin>
  ...
</bin>
```

In Here there are (currently) three subsections. All three tell the cliCreator where to find binary data output by GRIT that should have been run with the `-gB4`, `-gt`, and `-ftb` flags to generate a 4 bit-per-pixel tiled graphics output into a straight binary file.

Graphics definitions

The graphics section defines binary files that contain the tile data for graphics that are going to be used for sprite graphics. They are all contained within the `graphics` tag and have the following syntax:

```
<graphics>
 <gfx bin="path/to/gfx.img.bin" w="13" h="24" top="3" left="3"/>
 ...
</graphics>
```

The `bin` attribute should point to the binary file for the graphic's tile data as output by GRIT, usually with a `.img.bin` extension. The `w` and `h` attributes refer to the width and height of the graphic contained within the file *not* the dimensions of the file itself. Finally, the `top` and `left` attributes define the coordinates of the top-left most pixel in the graphic. In the example tag above, the graphic contained in the file `./path/to/gfx.img.bin` is 13 x 24 pixels starting from the coordinates (3, 3). You can add as many graphic definitions to this section as you want, the order in which they appear here refer to their gfx id, so this example defines the graphic with id 0.

Background tileset definitions

Backgrounds (see below) are composed of a grid of tile indices. These tiles come from a standard graphic output by GRIT with the same flags set as those for the gfx above. To tell the zoidberg engine about a gfx from which tiles can be used for backgrounds, use the following syntax:

```
<tilesets>
  <tileset bin="path/to/tiles.img.bin" />
  ...
</tilesets>
```

Like with graphics, the bin attribute should point to the binary file for the tiles data as output by GRIT. The image in that file is split into 8 x 8 pixel tiles that can be referenced by index in row-major order. Like with the graphics, the order the tilesets appear here set their tileset id, so this example defines background tileset 0.

Palette definitions

The zoidberg engine is designed to work with 16-color palettes. When adding a sprite to a level, you give it both a graphic and a palette id to use. You can use any 16-color palette with any graphic. And you can use up to 16 unique palettes on screen at a time. Palettes are defined with the following syntax:

```
<palettes>
  <palette bin="path/to/palette.pal.bin"/>
  ...
</palettes>
```

Again, the bin attribute should point to the binary output by GRIT but should instead point to a palette. This output is enabled in GRIT by setting the `-p` flag.

Expectations

The cliCreator will not run GRIT for you, you must create your own respective GRIT rule files and run grit on all graphic files you plan to use in your game. If cliCreator encounters a binary definition with a bin attribute that does not point to an existing file, it will fail.

Background Definitions

DS backgrounds are hardware accelerated and are based off a two-byte map value that points to a tile in the background tileset, horizontal- and vertical-flip values, and a palette id. Backgrounds are defined in XML using the following XML syntax:

```
<backgrounds>
  <background palette="2">
    <row><tile id="3"/>...</row>
    ...
  </background>
  ...
</backgrounds>
```

First, the <backgrounds> tag contains all of the <background> definitions. Within the <background> tag you can use the following attributes:

`palette`	Highly recommended, this defines a palette id to use for all tiles in the background unless otherwise specified in the `<tile>` tag (see below)

`xml`	Points to an external XML file name from which background tile data should be taken. If this attribute is defined, everything else within the `<background> ... </background>` tag will be ignored.
	The file this attribute points at should have the following structure:
	`<?xml version="1.0" ?>`
	`<zbe>`
	`  <backgroundmap>`
	`    <row><tile /> ... </row>`
	`    ...`
	`  </backgroundmap>`
	`</zbe>`

Within the `<background>` tag, there are a series of `<row>` tags. These tags have no attributes are simply used to wrap `<tile>` tags into rows of the background.

Within in each `<row>` tag, there are a lot of `<tile>` tags. These can have the following attributes:

`id`	Required, the id of the background tile to use. This refers to the `id`'th 8x8 pixel square in row-major order in the tileset being used.
`hflip` `vflip`	Optional, indicates whether the tile should be flipped horizontally or vertically. Defaults to not flipping in either dimension. Set equal to `"1"` to turn on flipping in that respective direction.
`palette`	Optional (unless not defined in `<background>`), overrides the default palette to use that was set in the `<background>` tag above. cliCreator will fail if you set a palette attribute in neither the `<background>` tag nor the `<tile>` tag

Object Definitions

Within the <objects> tag, you define the objects that you will use in your game. Use the following syntax:

```
<objects>
   <object weight="45">
     <animations>
       <animation>
         <frame id="2" pal="2" time="10" />
       </animation>
     </animations>
   </object>
</objects>
```

You can define as many objects and objects can contain as many animations with as many frames as you want.
The valid attributes for the `<object>` tag are the following:

`weight`	The weight of the object to be used in collision detection. A heavier object will move a lighter one. This is stored as an 8-bit integer, so the valid range is `0 - 255`.

The valid attributes for each `<frame>` tag are the following:

`id`	The gfx id to use to represent this frame. Corresponds to the `id`'th graphic defined in the `<bin>` section.
`pal`	The palette id to use with this frame. Corresponds to the `id`'th palette defined in the `<bin>` section.
`time`	The number of screen blanks to display this frame of the animation. Can be `0 - 255`.

As of this writing, the animations are not used for anything as the event subsystem has not yet been implemented. When the object is represented on screen, it will only use the gfx defined in the first `<frame>` of the first `<animation>`.

Level definitions

Finally, it comes to define the levels. All levels are contained within the `<levels>...</levels>` tags and are contained within a `<level>...</level>` group.
The level tag can have the following attributes:

`w` `h`	The width and height of the level in pixels. These values are used in the definition of the collisionMatrix and clamping the scrolling bounds,

Levels can be defined to have a name using the `<name>Level name here</name>` group. These are not required but are recommended for use with a future level select screen or in cinematic script.

Backgrounds definitions

Each level can define up to four different backgrounds to use. The only limitation here is that they all must use the *same* tileset. The id of the tileset to use with the backgrounds is defined in the `<backgrounds>` tag using the `tileset="0"` attribute, this attribute is required.

Within the `<backgrounds>...</backgrounds>` group, there can be defined up to four `<background>` tags. These tags have the following attributes:

`layer`	Required the background layer to use. Can be 0, 1, 2, or 3. The background layers 0 - 2 are rendered *behind* the level objects. Layer 3 is rendered *above* the level objects.
`id`	The id of the background defined in the `<backgrounds>` section above to use for this layer.
`distance`	Optional, defines how far from the level objects plane that background is. For layers 0 - 2 the viewport offset is *divided* by this value to determine to where in the background should be scrolled. For layer 3, the viewport offset is *multiplied* by this value to determine its scroll value. Distance defaults to 1 if undefined. The distance values are designed to allow you to create parallax scrolling effects with the backgrounds to make layer 0 appear to be farther away from the objects plane than background layer 1.

Object definitions

Within the `<objects>...</objects>` group you can define all of the basic objects to place in the level. Each `<object>` within can have the following required attributes:

`id`	The id of the object defined in the root <objects> group above.
`x` `y`	The coordinates to place this object
`hgrav`	The horizontal gravity to apply to this object. In XML this is a float value which is converted to a 20.12 fixed point integer.
`vgrav`	The horizontal gravity to apply to this object. In XML this is a float value which is converted to a 20.12 fixed point integer.

Hero definitions

Heroes are defined in the same way as objects and use the same objects defined in the root `<objects>` group.
The difference between a hero and object is that a hero can be controlled by the player.
NOTE: the zoidberg engine is currently designed to only have one hero per level. If you define more than one

here, undefined behavior will occur.

Example

Here is an example of a very basic, four-background level defined with XML:

```
<level>
  <backgrounds tileset="0">
    <background layer="0" id="0" distance="5" />
    <background layer="1" id="1" distance="3" />
    <background layer="2" id="2" />
    <background layer="3" id="3" distance="2" />
  </backgrounds>
  <objects>
    <object id="0" x="155" y="34" hgrav="0.1" vgrav="0" />
    <object id="0" x="200" y="10" hgrav="-0.1" vgrav="0" />
  </objects>
  <heroes>
    <hero id="1" x="32" y="32" hgrav="0" vgrav="0.3" />
  </heroes>
</level>
```

Testing tags

See the Testing Framework page for more information

The level tag contains several tags that are only used if cliCreator is run with `-t` option to generate a testing zbe file. The zoidberg engine's testing build uses these additional level tags to make the level into a *graphical test*. The level is run for a predetermined amount of time during which, something graphical is supposed to happen to test a part of the engine. Before running the level, a message is displayed and after the test is complete, the user will be asked if the test ran as it was supposed to. If not, then it displays a debug message about what may have gone wrong.

The first is a timer attribute in the `<level>` tag that defines how long the level should be run in screen blanks. There also the following groups:

`<name>...</name>`	Defines the graphical test's name. This is not required, but if you don't set it to anything, the test will be represented by a blank line when selecting a graphical test.
`<exp>...</exp>`	Contains a description of what is supposed to happen in this graphical test
`<debug>...</debug>`	Contains an explanation of what may have gone wrong that caused the test to fail. This message is only displayed if the user responds that the test did not run correctly.

Within these messages, there are a series of <line>...</line> tags that allow the test creator to add whitespace to the message. The <lines> are simply appended on to each other with a newline character inserted in between them.

The following is an example of a very basic graphical test:

```
<level timer="180">
  <name>Graphical Test Test</name>
  <exp>
    <line>This is simply to test the</line>
    <line>graphical testing framework,</line>
    <line>you should see a blank screen</line>
    <line>for three seconds.</line>
  </exp>
  <debug>
    <line>You must be tripping</line>
  </debug>
  <backgrounds tileset="0">
  </backgrounds>
  <objects>
  </objects>
  <heroes>
  </heroes>
</level>
```