tilesObject()

This class I meant to be used with MapMaker program to create a 2d environment file in the proper format.

(1) controls the camera,

(2) stores a single 2d text level environment file (with support for layers), and (3) draws the environment.

Void loadMap(**char []**)

* This function requires a string filename passed to it.
* The file format stores the map text size (row count then column count). The file then stores the number of 2d layers. Then the default X and Y tileSize in pixels. This function will automatically calculate the total map size X and Y size in pixels. It’s important to remember that 1 map world space unit is = 1 pixels. **The 2D text must be created with an extra row and column as the viewable camera range will be row-1 and column-1. This is done to account for screen resolution that does not divide evenly into the default tileSize.** The map stores tile values ranging from 0-999. The max number of different tiles per layer is 999.
* A 3-digit layer type number is stored before each block of numbers for each layer. This is a user defined number in MapMaker during file creation. This number is then typically associated with an enumerated data type for unique layer functionality. For example, separating tiles, items, projectiles, player character, non-player characters, etc.
* There is currently no error trapping, no format integrity checking, etc. As such it is recommended to use MapMaker for file format integrity. Without using MapMaker results may be unstable and crash.
* It does not store all the levels for a game just one level.
* Currently the user cannot modify the 2D map tile value – future enhancement.

loadTileObject(**int layer, string filename**, int red, int green, int blue)

* Takes a filename of an image containing all the possible tile pieces in that layer.
* Takes optional red, green, blue input transparency color for background of image. The default is white, 255,255,255.

Void setTileSizeAndClips(**int layer, int rows, int cols**, int tilesize\_x, int tilesize\_y)

* This function sets the tile image clip coordinates for the image object loaded in loadTileObject for a given layer. The layer of the loadMap, loadTileObject, and this function need to be coordinated.
  + This function requires that the image file is setup so that each clip is the same X pixel and Y pixel space for the entire image (X and Y can be different, but each cannot vary throughout the image file).
* This function does allow for the image file to be setup completely horizontally or vertically, or both. These are determined by the rows and cols input variables. These row and col variables have nothing to do with the 2d text map they are for the tile image only.
* The tileSize for an image or layer CANNOT vary throughout.
* If the tileSize is omitted, the default tilesizes are used from the input file for that layer.

Void draw(**int layer**, int rowCount, int colCount, int xPixelStartPos, int yPixelStartPos, imgOffset)

* This function uses some input along with the internal camera coordinates to render the appropriate tiles.
* Only the layer to render is required, however overrides are provided for the number of rows and cols drawn at the default tileSize.
* The key to this function is that everything is rendered based on the master tileSize specified in the input file. This functionality cannot be changed. IMPORTANT: this function uses the internal camera X and Y offsets. The default starting X, Y pixel drawing position is 0,0; however, the camera will automatically change the starting pixel drawing position based on the camera location. This camera offset is based on the master tileSize only though. This is done to handle partially displayed clipped tiles on the edges of the screen. For example, if the master X tileSize of 64 and the camera X location is at 41, the pixel draw will start X at -41 so that only pixels 42-64 (the 22 rightmost pixels of the tile) are drawn.
* The number of rows and cols drawn are calculated based on the screen X and Y resolution and the X and Y master tileSize. The calculation is roundup(screen\_resolution\_x / tileSizeY)+1. **This calculation is why the 2d text map requires an extra column (see loadMap() function)**. This calculation will automatically handle various resolution and tileSizes. For example, Y resolution=720, Y tileSize=64 leaves 13 possible row tiles viewable at one time- the top and bottom tiles both partially visible. Refer to environmentVars for the value of screen\_resolution\_x and screen\_resolution\_y.
* There is a default tile value to specify a blank space or empty space “air” in the map. The standard is 0, however can be changed in the environmentVars file. All non-empty space tile values in the map are rendered.

Bool moveCameraY(**int yAmount**)

* The camera has no implied speed built-in, a movement amount must be provided by the user.
  + **A positive amount moves the camera up and a negative amount moves the camera down. This is done to align with standard player jump and gravity forces (where y Speed for jumping is a positive value and gravity/falling is a negative value). This is the opposite of Y pixel movement on a monitor.**
* This function has built-in collision detection for movements outside the world boundaries. This is 0 at the minimum (fixed amount) and a max of tileSizeY-1 times screen\_resolution\_y.
* When the collision has been detected, the function will only move the camera the amount to the top or bottom border and stores the amount unable to be moved in an internal variable BoundaryExcessY. For example, if 10 is provided, but that is 4 past the top or bottom world boundaries, the camera will only move 6 and store 4 in the BoundaryExcessY (you can then retrieve this value with getBoundaryExcessY).
  + A value of FALSE is returned if the camera was able to move the entire amount provided.
  + A value of TRUE is returned if the camera was unable to move the amount provided due to collision with the world boundary.

Bool moveCameraX(**int xAmount**)

* The camera has no implied speed built-in, a movement amount must be provided by the user.
  + **A positive amount moves the camera right and a negative amount moves the camera left. This is the same as monitor pixel movement. Depending on implementation of main player horizaontal movement this may align or might be inverted.**
* This function has built-in collision detection for movements outside the world boundaries. This is 0 at the minimum (fixed amount) and a max of tileSizeX-1 times screen\_resolution\_x.
* When the collision has been detected, the function will only move the camera the amount to the left or rightmost border and stores the amount unable to be moved in an internal variable BoundaryExcessX. For example, if 10 is provided, but that is 4 past the top or bottom world boundaries, the camera will only move 6 and store 4 in the BoundaryExcessX (you can then retrieve this value with getBoundaryExcessX).
  + A value of FALSE is returned if the camera was able to move the entire amount provided.
  + A value of TRUE is returned if the camera was unable to move the amount provided due to collision with the world boundary.

Int getTileValue(**int layer, int x, int y**)

* A simple function that returns the 2d file tile value from a provided map layer and x and y location.
* The function contains no boundary checking if an improper layer, x, or y value is provided.