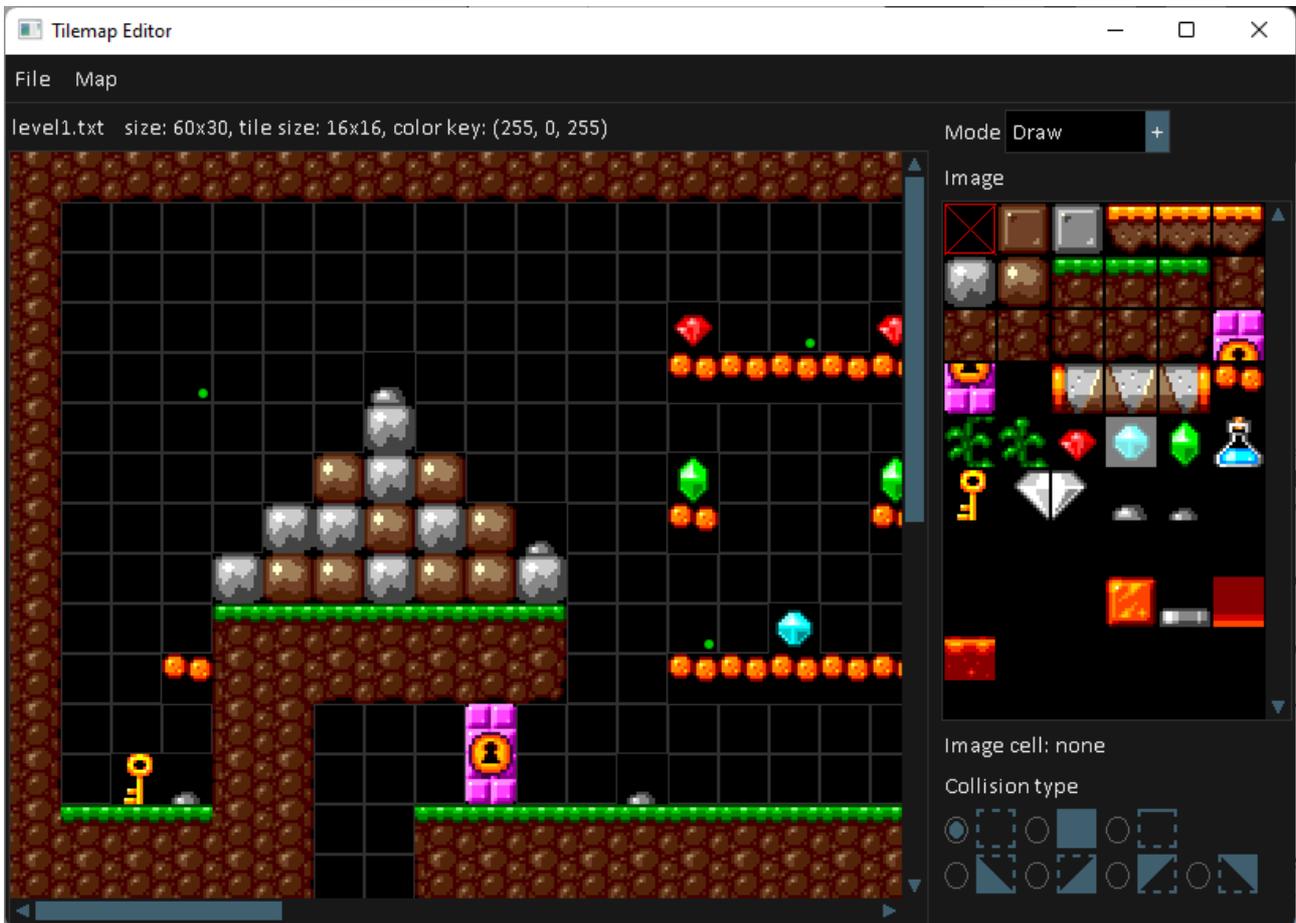
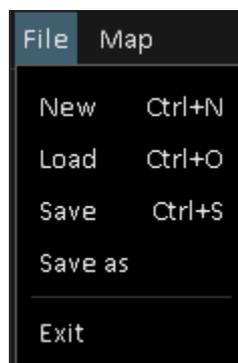


Tilemap editor

Use the Tilemap Editor to create maps that the tilemap library can load.



File menu



New

Create a new map from scratch, this clears everything.

Load

Load a map that you have previously saved. You can also load maps created with the n6 version of the Tilemap Editor.

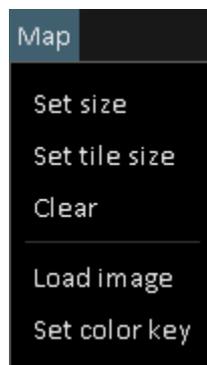
Save/Save as

Save the current map.

Exit

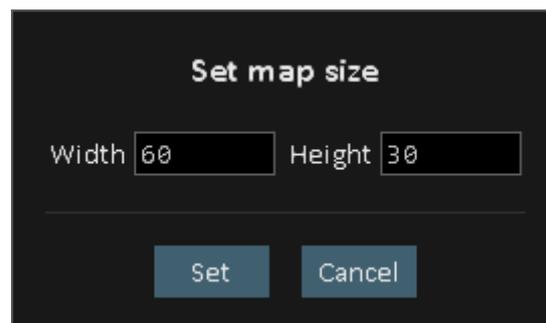
Close the program.

Map menu



Set size

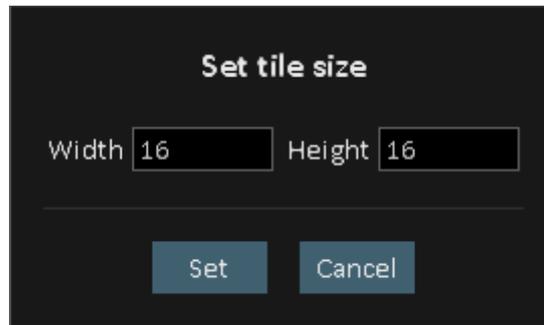
Open a dialog where you can change the width and height of the map.



The width and height is specified in *number of tiles*.

Set tile size

Open a dialog where you can change the width and height of a tile.



The width and height is specified in *pixels*.

Clear

Clear the map but keep its size, tile size and image.

Load image

A map can only use *one* image. This image is assumed to contain cels in a uniform grid, where each cel has the size set in the *Set tile size* dialog. Here is an example of such an image with the tile size 16x16:



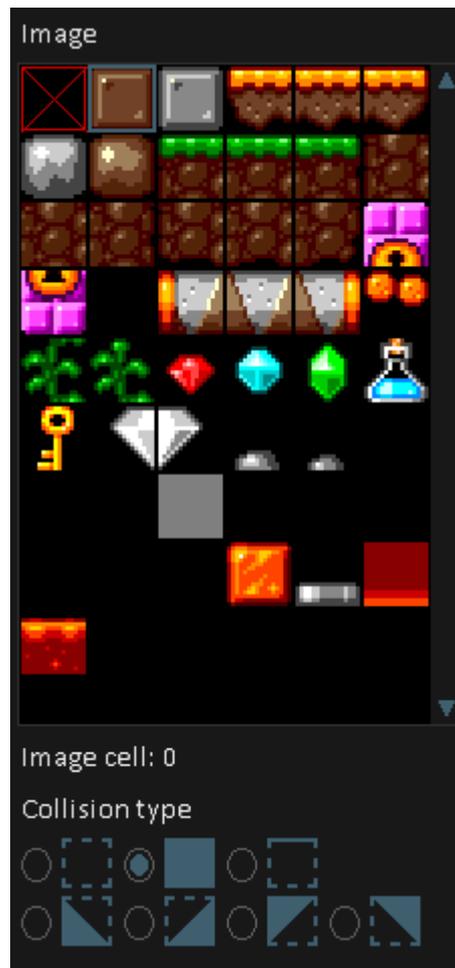
Set color key

If the image loaded contains a color that is to be treated as transparent, you can enter the RGB values of that color in the *Set color key* dialog:



Image cel selection

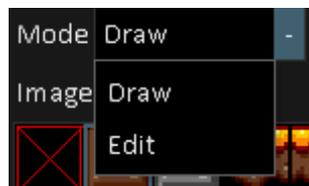
You select an image cel by clicking on it under the *Image* label.



After selecting an image cel, you can set its *collision type* using the radio buttons under *Collision type*.

Modes

There are two modes for editing the map. You can change the mode with the drop-down list next to the *Mode* label:



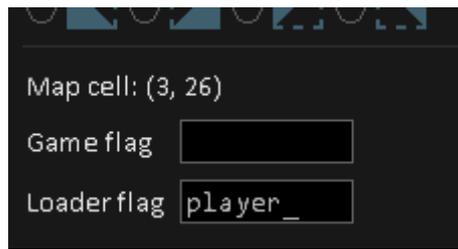
Draw mode

If you select the *Draw* mode, you can draw on the map with your left mouse button using the currently selected tile. The first image tile, the one with a red cross, is used for erasing.

Edit mode

When you select the *Edit* mode, you can select and edit a single tile of the map by clicking on it.

When you select a tile two input boxes appear in the bottom right corner of the window:

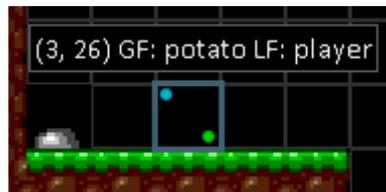


Here you can set a *game flag* and a *loader flag* for the selected map tile. You can use these flags for anything you want.

After loading a map through the tilemap library using *TM_LoadMap*, you can get a list of all set loader flags with *TM_GetLoaderFlags*. As an example, you can use loader flags to position the player and enemies.

The game flag of a map tile can be accessed with the function *TM_GetFlag*. As an example, if your map contains breakable blocks that may contain powerups, you can use game flags to decide what's in a specific block ("coin", "mega gun" ...).

If a game flag has been set for a tile, a blue dot will be displayed in its top left corner. And if a loader flag has been set for a tile, a green dot will be displayed in its bottom right corner. If you hold the mouse cursor over a tile, a tooltip appears, showing its coordinates and flags:



Tilemap library

Units and coordinate systems

A tilemap has a *tile* size, a *map* size and, what I call, a *world* size. The tile size is the size of a single tile in pixels. The map size is the size of the map in tiles. And the world size is the size of the map in pixels. In other words, the *world width* is the map width * the tile width, and the *world height* is the map height * the tile height.

When I speak of *map coordinates*, the unit is tiles, and when I speak of *world coordinates*, the unit is pixels.

Since a tilemap can be larger than the view (usually the window) a *camera* position, in world coordinates, can be set to change what part of the map we're currently viewing. Therefor we also have to make a difference between *screen* (view) *coordinates* and *world coordinates*.

Sprites

Some functions, such as *TM_MoveSprite*, take a "sprite" as an argument. A sprite is defined as any table where *x*, *y*, *w* and *h* represents its position in world coordinates and its width and height in pixels. But you don't have to use these functions, since they have twins using images and world coordinates as parameters.

Functions

(number) **TM_LoadMap**((string)*filename*)

Load map from the filename *filename* created with the Tilemap Editor and return *true* on success.

(number) **TM_MapWidth**()

Return the map width in tiles.

(number) **TM_MapHeight**()

Return the map height in tiles.

(number) **TM_TileWidth**()

Return the tile width.

(number) **TM_TileHeight**()

Return the tile height.

(number) **TM_WorldWidth()**

Return the map width in pixels.

(number) **TM_WorldHeight()**

Return the map height in pixels.

(array) **TM_GetLoaderFlags()**

Return all loader flags set for the map as an array of objects with the keys:

(string)flag

(number)x

(number)y

The position (x, y) is in map coordinates.

TM_SetBorder((number)*leftBorder*, (number)*rightBorder*, (number)*topBorder*, (number)*bottomBorder*)

Decide how the map borders are to be treated. For example, if the left border is to be treated as a wall, set *leftBorder* to *true*, otherwise *false*. By default, all borders are treated as walls.

(number) **TM_GetImage()**

Return the map image.

(number) **TM_GetCel**((number)*mapX*, (number)*mapY*)

Return the image cel of the map image at the map coordinates (*mapX*, *mapY*). -1 is returned if no cel has been set for the position.

(number) **TM_GetCelAt**((number)*worldX*, (number)*worldY*)

Return the image cel of the map image at the world coordinates (*worldX*, *worldY*). -1 is returned if no cel has been set for the position.

TM_SetCel((number)*mapX*, (number)*mapY*, (number)*cel*)

Set the image cel of the map image at the map coordinates (*mapX*, *mapY*) to *cel* (-1 to clear).

(string) **TM_GetFlag**((number)*mapX*, (number)*mapY*)

Return the game flag at the map coordinates (*mapX*, *mapY*). An unset variable is returned if no flag has been set for the position.

(string) **TM_GetFlagAt**((number)*worldX*, (number)*worldY*)

Return the game flag at the world coordinates (*worldX*, *worldY*). An unset variable is returned if no

flag has been set for the position.

TM_SetFlag((number)*mapX*, (number)*mapY*, (string)*flag*)

Set the game flag at map coordinates (*mapX*, *mapY*) to flag (unset to clear).

(number) **TM_Obstacle**((number)*cel*)

Return *true* if the map image cel *cel* is an obstacle.

(number) **TM_ObstacleAt**((number)*worldX*, (number)*worldY*)

Return *true* if there is an obstacle at the world coordinates (*worldX*, *worldY*).

TM_SetView((number)*screenX*, (number)*screenY*, (number)*w*, (number)*h*)

Set the rendering area used by the *TM_Render* function to screen position (*screenX*, *screenY*) and the width and height *w* and *h*. Most likely you will want to render the map over the entire window, and call *TM_SetView(0, 0, width(primary), height(primary))* once you've created a window.

TM_SetCamera((number)*worldX*, (number)*worldY*)

Set the camera's top left position to the world coordinates (*worldX*, *worldY*).

TM_CenterCamera((number)*worldX*, (number)*worldY*)

Center the camera at the world coordinates (*worldX*, *worldY*).

(number) **TM_CameraX**()

Return the x world coordinate of the camera.

(number) **TM_CameraY**()

Return the y world coordinate of the camera.

(number) **TM_ToScreenX**((number)*worldX*)

Return the world x coordinate *worldX* converted to a screen x coordinate.

(number) **TM_ToScreenY**((number)*worldY*)

Return the world y coordinate *worldY* converted to a screen y coordinate.

(number) **TM_ToWorldX**((number)*screenX*)

Return the screen x coordinate *screenX* converted to a world x coordinate.

(number) **TM_ToWorldY**((number)*screenY*)

Return the screen y coordinate *screenY* converted to a world y coordinate.

(number) **TM_ToMapX**((number)*worldX*)

Return the world x coordinate *worldX* converted to a map x coordinate.

(number) **TM_ToMapY**((number)*worldY*)

Return the world y coordinate *worldY* converted to a map y coordinate.

(number) **TM_SpriteVisible**((table)*sprite*)

Return *true* if the sprite *sprite* would be visible if rendered.

(number) **TM_Visible**((number)*img*, (number)*worldX*, (number)*worldY*)

Return *true* if the image *img* at world coordinates (*worldX*, *worldY*) would be visible if rendered.

TM_Render()

Render the map.

TM_MoveSprite((table)*sprite*, (number)*dx*, (number)*dy*)

Move the sprite *sprite* with the x- and y-speed *dx* and *dy* with collision handling.

(table) **TM_Move**((number)*img*, (number)*worldX*, (number)*worldY*, (number)*dx*, (number)*dy*)

Move the image *img*, currently at position (*worldX*, *worldY*) with the x- and y-speed *dx* and *dy* with collision handling. The new coordinates are returned as *x* and *y* in a table.

(number) **TM_CollisionLeft**()

Return *true* if the last call to *TM_MoveSprite* or *TM_Move* resulted in a leftward collision.

(number) **TM_CollisionRight**()

Return *true* if the last call to *TM_MoveSprite* or *TM_Move* resulted in a rightward collision.

(number) **TM_CollisionUp**()

Return *true* if the last call to *TM_MoveSprite* or *TM_Move* resulted in an upward collision.

(number) **TM_CollisionDown**()

Return *true* if the last call to *TM_MoveSprite* or *TM_Move* resulted in a downward collision.

(number) **TM_SpritesCollide**((table)*sprite1*, (table)*sprite2*)

Return *true* if the sprites *sprite1* and *sprite2* overlap.

(number) **TM_ImagesCollide**((number)*img1*, (number)*x1*, (number)*y1*, (number)*img2*, (number)*x2*, (number)*y2*)

Return *true* if the images *img1* and *img2* at the positions (*x1*, *y1*) and (*x2*, *y2*) overlap.

Additional functions when not using the Tilemap Editor

TM_InitMap((number)*w*, (number)*h*)

Set the map width and height in tiles to *w* and *h*.

TM_SetImage((number)*img*)

Set the map image to *img*. The tile width and height of the map will be set to the tile width and height of the image (set with *set image grid*).

TM_SetObstacle((number)*cel*, (number)*value*)

Make the cel *cel* of the map image an obstacle if *value* is *true*. By default, all cels are non-obstacles.

TM_SetOnlyDown((number)*cel*, (number)*value*)

Make the cel *cel* of the map image an obstacle that only allows collision *downwards* if *value* is *true*. This may be useful in platform games where the user can jump through certain platforms from below but still land and walk on them.