Scratch game “Kersje”

Extension Guide

August 29th 2021

Jan & Ruben De Beer

Overview

This is a guide on how to extend the Scratch game “Kersje” with custom elements such as backgrounds, object types and characters.

After deployment, the extensions can be included in new maps using the Game Map Maker software (GMM) and played in an updated (or remixed) Scratch project.

Introduction

Map anatomy

The world in which the character moves around is called a *map*. As the character moves from start to finish, the map scrolls during the gameplay. The visible part of the map is called a *screen*. A screen is made up of an invisible grid of tiles. One *tile* measures 30 pixels wide by 30 pixels high. A screen measures 15 tiles horizontally (450px) by 11 tiles vertically (330px). The bottom row is typically reserved for a solid *ground* on which the character runs, slides and jumps.

GMM extension files

There are two important extension files in the GMM source repository (see Deploy GMM). They are referred to in the remainder of the document using these names.

- **Inventory file**
  Located at 'src/main/webapp/data/map-inventory.json'. Contains descriptions of all the available backgrounds and object types.
- **Shop file**
  Located at 'src/main/webapp/data/map-shop.json'. It defines the layout (grouping and order) of the items in the GMM "shop".

Extensions

Custom backgrounds, object types (including atmospheres) and characters can be added to the game. The sections below describe how.

Backgrounds

A background fills the entire map visually in the distance and scrolls (more slowly) as the character moves around. A background is a repeated sequence of 1 or more background screens. In the example below, the background is made up of 2 screens.


...

This is the procedure to add a new custom background.

1. Create background graphics

   Create images for the background screens. Every image is 450px wide by 330px high and should be fully opaque. Note that the bottom 30px is typically where the ground will be laid out. Also note that the sequence is repeated so for a seamless design, make sure the far end matches the beginning of the sequence.

2. Store background graphics in GMM

   The background images should be stored in a custom subdirectory of the GMM source repository under ‘src/main/webapp/media/backgrounds’.

3. Describe background in GMM

   Add one element to the ‘backgrounds’ array in the GMM inventory file. Here is an example element.

   ```
   {
       "id": "MountainTree",
       "code": 0,
       "label": "Mountain and tree",
       "images": [
   ```

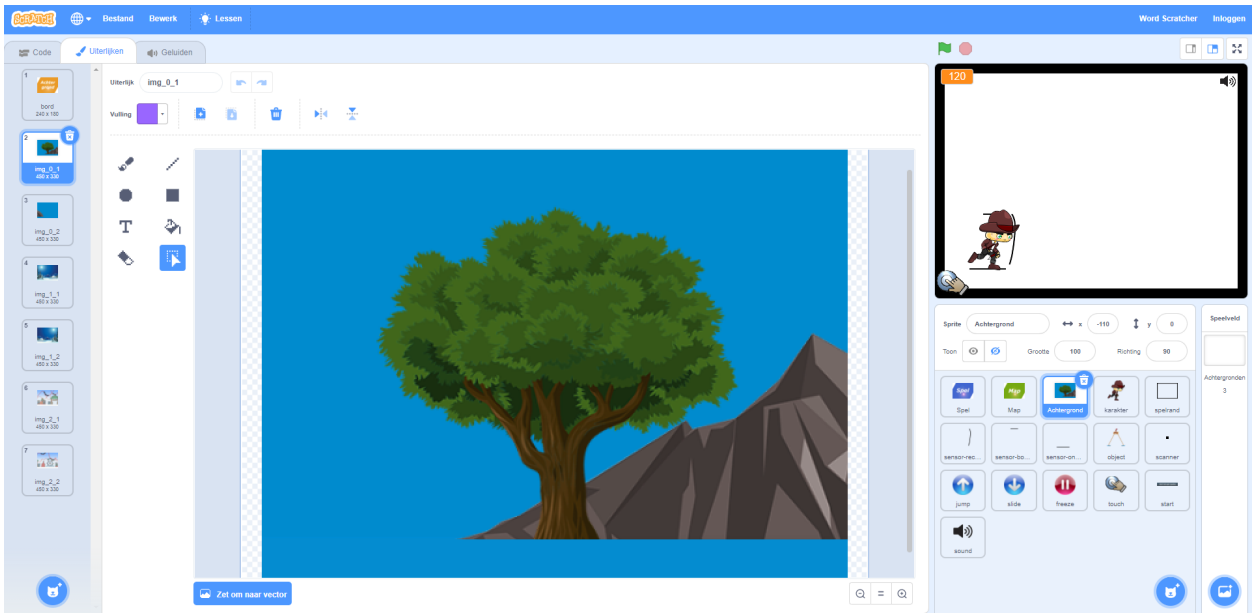
```
            "MountainTree/MountainTree1.png",
            "MountainTree/MountainTree2.png"
        ]
}
```

The following table describes the properties of a background.

Property	Description
id	Unique background symbolic id It's used in map json definitions
code	Unique background numeric id It's a sequential number within the backgrounds It's used in map bitmaps and in Scratch
label	A human readable label for the background It's used in the 'Set background' dialog of GMM
images	Array of relative file paths for the background images, in order of appearance

4. Deploy background in Scratch

Inside the Scratch project, go to the costumes of the sprite called "*Achtergrond*". Upload the individual background images. You should rename every costume as follows : "img_${code}_${i}". The ${code} represents the background's numeric id. The ${i} is a one-based sequence number for the background images.



5. Deploy background in GMM

   Deploy the GMM software containing the updated resources as described in Deploy GMM.

6. Validate

   In GMM open or create a new map. Click "*Set background*" from the menu. In the dialog window, your new background should be found. Select it. Save the map and deploy it for validation in Scratch (see Deploy new maps).

Object types

An object type is a type of element that can be placed inside maps. One map can contain many objects of the same type. Object types *interact* with the character in different ways. Some are blocking the character from moving, others are intangible (used for decoration), there are value items (like coins), fatal items (like bombs) and so on.

Objects have an initial (anchor) position and rotation on the map but they can move and rotate around that anchor in few supported ways. Also, objects can have a single constant appearance or an alternating appearance via a repeated sequence of images (as in “stop motion”). A combination of both is possible as well, as illustrated in the table.

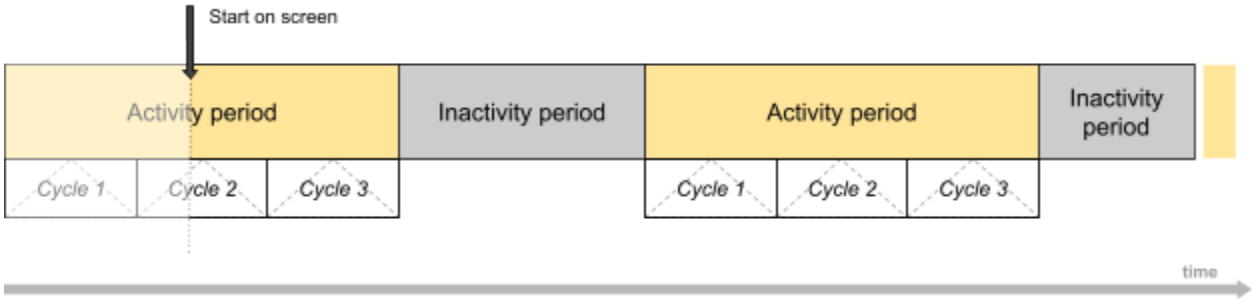
	Fixed	**Moving**
Single appearance	"appearances": 1 "movement": "none" **= static object** 	"appearances": 1 "movement": "smoothRadialSweep" 
Alternating appearance	"appearances": 6 "movement": "none" 	"appearances": 2 "movement": "smoothVerticalSweep" 

We refer to an object as *static* when it does not move and has a single appearance. When an object is not static, it is *alive* and will endlessly alternate between an *active state* and an *inactive state*.

- In the *active state*, the object goes through a predefined number (1 or more) of repeated movement or appearance *cycles*. Every cycle represents either one full movement if there is a movement, or one iteration through all appearances if there is no movement.
- In the *inactive state*, the object sticks to its starting position, rotation and appearance for the duration of the inactivity.

The duration of both states can be controlled. The duration of the active state is defined through the *speed* of motion or appearance change. The duration of the inactive state, called the *inactivity time*, can be either constant over periods of inactivity or each time random within predefined bounds.

When an *alive* object first appears on screen, it will always start within an activity period, either completely random or a fixed offset within a randomly chosen cycle. The randomness is intentional to have a lesser predictability between gameplays.



This is the procedure to add a new custom object type.

1. Create object type graphics

   For every appearance of the object type, create a separate image. There can be up to 15 alternating appearances. All images should have the same size which is a multiple of tiles. The maximum width is 15 tiles. The maximum height is 11 tiles. The images can be (partially) transparent. As a convention, the following filename pattern is suggested : "${baseName}_${tileWidth}x${tileHeight}[_${i}]", where ${i} is a one-based sequence number in case of multiple appearances. Optionally, you can have one separate image for the inactive state. As a convention, use the suffix "sleep" instead of ${i} in its filename.

2. Store an object type graphic in GMM

   One representative image is needed for GMM. This is the image that is shown in the GMM shop and placed on maps created in GMM. Store that image in the GMM source repository under 'src/main/webapp/media/objectTypes'.

3. Describe object type in GMM

Add one element to the 'objectTypes' array in the GMM inventory file. Here is an example element.


```
{
        "id": "barrel_2x2",
        "code": 6,
        "label": "Barrel",
        "image": "barrel-dark_2x2.png",
        "appearances": 1,
        "movement": "none",
        "transparency": 0,
        "widthInTiles": 2,
        "heightInTiles": 2,
        "interaction": "tangible"
}
```


The following table describes the properties of an object type.

Property	Description
id	Unique object type symbolic id It's used in map json definitions
code	Unique object type numeric id It's a sequential number within the object types It's used in map bitmaps and in Scratch
label	A human readable label for the object type It's used as a tooltip in the GMM shop
image	Relative file path for the image shown in GMM When an object type has multiple appearances, choose one representative image (it could even be a completely different image)
appearances	The number of appearances (between 1 and 15)
movement	Specifies the type of movement, if any. These are the supported values : • When "*none*", the object's position and rotation

	is fixed • When "*horizontalStroke*", the object travels in one direction along a horizontal path in a constant velocity • When "*horizontalSweep*", the object travels back and forth along a horizontal path in a constant velocity • When "*smoothHorizontalSweep*", the object travels back and forth along a horizontal path, slowing down as it approaches both endpoints • When "*verticalStroke*", the object travels in one direction along a vertical path in a constant velocity • When "*verticalSweep*", the object travels back and forth along a vertical path in a constant velocity • When "*smoothVerticalSweep*", the object travels back and forth along a vertical path, slowing down as it approaches both endpoints • When "*radialStroke*", the object travels in one direction along a radial path in a constant velocity • When "*radialSweep*", the object travels back and forth along a radial path in a constant velocity • When "*smoothRadialSweep*", the object travels back and forth along a radial path, slowing down as it approaches both endpoints • When "*vector*", the object travels outwards from its anchor position along a vector in a constant velocity • When "*vectorWithGravity*", the object travels outwards from its anchor position along a vector however pulled down as if by force of gravity. At the end of the movement, the object touches the ground ⚠ One problem with movement is that objects cannot entirely move off-screen in Scratch (they will be snapped to the edges). Beware especially of horizontal translation. As objects enter the screen on the right side and scroll leftwards, one suggestion is to make use of the "*movementStartAt*" property to enter with a leftward translation.
transparency	Transparency to apply to the object images. The range is from 0 (opaque) to 1 (fully transparent).
widthInTiles	The width of the object type in tiles (maximum 15)
heightInTiles	The height of the object type in tiles (maximum 11)

interaction	Specifies the way in which objects of this type interact with the character. These are the supported values : • When "*tangible*", the object blocks movement of the character • When "*intangible*", there is no interaction (used mainly for decorative objects, think clouds) • When "*intangible-back*", there is no interaction and the object is on the "back" layer (see Drawing order) • When "*intangible-front*", there is no interaction and the object is on the "front" layer (see Drawing order) • When "*score*", the character earns 1 point • When "*scoreValue*", the character earns a number of points specified by *"value"* • When "*fatal*", the character dies • When "*finish*", the character has reached the end of the map • When "*atmosphere*", an atmospheric condition applies (see Atmospheres)
Optional properties	
movementParameter1	For path-like movements, a parameter value that defines the precise path. The range is between -508 and +512, in steps of 4 • For horizontal movements, the first endpoint's x-coordinate relative to the object anchor position, measured in pixels • For vertical movements, the first endpoint's y-coordinate relative to the object anchor position, measured in pixels • For radial movements, the first endpoint's angle rotation, measured in degrees • For vector movements, the horizontal component of the vector
movementParameter2	For path-like movements, a parameter value that defines the precise path. The range is between -508 and +512, in steps of 4 • For horizontal movements, the second endpoint's x-coordinate relative to the object anchor position, measured in pixels • For vertical movements, the second endpoint's y-coordinate relative to the object anchor position, measured in pixels • For radial movements, the second endpoint's

	angle rotation, measured in degrees • For vector movements, the vertical component of the vector
movementStartAt	For moving object types, the "on screen" offset within the activity cycle. These are the supported values : • When "*0%*", the offset is at the movement's starting point • When "*25%*", the offset is at one quarter of the movement • When "*50%*", the offset is midway the movement • When "*75%*", the offset is at three quarters of the movement • When "*random*", the offset is random within the movement
activityCycles	For object types that are alive, the number of activity cycles. The range is from 1 to 15.
activitySpeed	For object types that are alive, the speed of moving (when there is movement) or changing appearances (when there is no movement). The range is from 0 (slowest) to 255 (fastest).
appearancesSpeedWhile Moving	For moving object types that also have an alternating appearance, the speed of changing appearances. The range is from 0 (slowest) to 255 (fastest).
inactivityTimeMinimum	For object types that are alive, the minimum duration of the inactivity periods. The range is from 0 (no inactivity) to 15, a relative unit of time.
inactivityTimeMaximum	For object types that are alive, the maximum duration of the inactivity periods. The range is from 0 (no inactivity) to 15, a relative unit of time.
inactivityAppearance	For object types that are alive, the one-based index number of the appearance while in the inactive state. Defaults to 1. The special value 0 allows you to have a special costume in Scratch used exclusively for the inactive state, see Deploy object type in Scratch.
offsetHorizontal	Optional horizontal position offset for the object type, measured in pixels. May be needed to re-align the object to its intended anchor position when panning the costume in the Scratch editor (for example, to rotate around a particular point). The range is

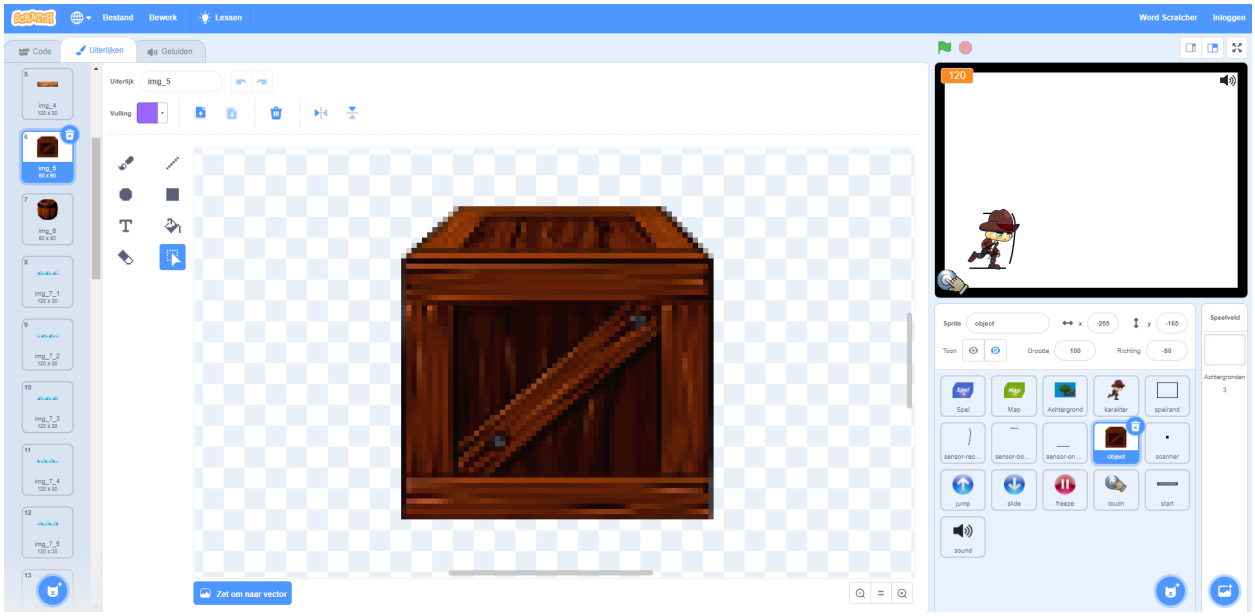
	between 0 and -255 (only negative horizontal offsets are supported).
offsetVertical	Optional vertical position offset for the object type, measured in pixels. May be needed to re-align the object to its intended anchor position when panning the costume in the Scratch editor (for example, to rotate around a particular point). The range is between -255 and +255.
value	A custom value depending on the *"interaction"*. The range is between 0 and 255. • When "*scoreValue*", the number of points that is earned
parts	See Composite object types

4. Include object type in GMM shop

   Reference the object type (by symbolic id) as an *item* to a *rack* within a *department* in the GMM shop file (a way to group object types). Doing so will make the object type available for selection in GMM. Hint: if you ctrl-click a shop item in GMM, it will show you the item's inventory definition.

5. Deploy object type in Scratch

   Inside the Scratch project, go to the costumes of the sprite called "*object*". Upload the graphic image(s) for all the different object appearances. You should rename every costume as follows : "img_${code}[_${i}]". The ${code} represents the object type's numeric id. The ${i} is a one-based sequence number in case of multiple appearances. If you have a separate image for the inactive state, it should be uploaded and renamed to "img_${code}_sleep".



6. Deploy object type in GMM

   Deploy the GMM software containing the updated resources as described in Deploy GMM.

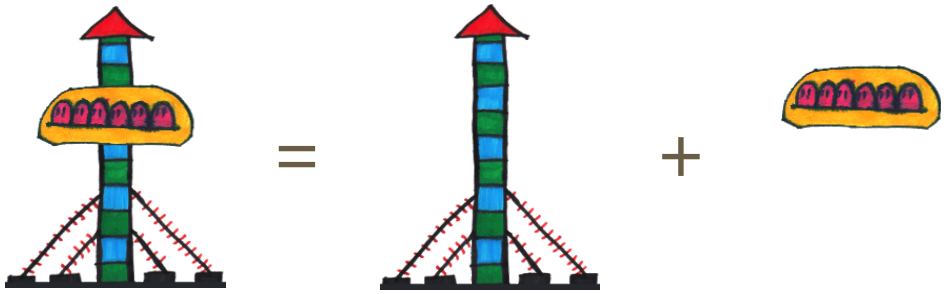
7. Validate

   In GMM open or create a new map. In the shop, your new object type should be found. Select it and add new objects to the map. Save the map and deploy it for validation in Scratch (see Deploy new maps).

Composite object types

The object types in the previous section are called *singular* object types. They define a single interaction, a single movement etc. A *composite* object type is an object type that consists of multiple parts, each of which is a singular object type that can have its own interaction, movement and appearance.

As an example, consider a drop tower. It is one composite object consisting of a static column (part 1) and seats (part 2) that are moving vertically along the column. Touching the seats is fatal, however the column is not and can be passed freely when the seats are high up in the air.



Composite objects have several advantages.

- They are treated in unity in the GMM software, meaning they can be added or removed with a single click in a map
- Their parts are assembled in predefined ways, meaning which parts and their relative positions to one another (like a “template”)
- Their parts are objects that can be overlapping in space, which is more constrained with singular objects in GMM (see Drawing order). You have full control over the relative drawing order of the parts within a composite object
- By assembling singular objects each constrained in maximum size, you can create much larger objects that still look as one object
- You can more efficiently add (partial) variations in objects by having different composite object types assembled from a shared pool of singular object type parts (like “lego”)

Composite objects also have some restrictions concerning their parts.

- A part cannot itself be a composite object type. It must be singular
- A part should be on the “default” layer (see Drawing order), meaning it cannot have one of these interactions : “*intangible-back*”, “*intangible-front*” or “*atmosphere*”

To create a new composite object, follow this procedure.

1. Create composite parts

   For every part of the composite object type, create or reference an existing object type as described in Object types. The same object type can be referenced (reused) in multiple composite object types. Note the above mentioned restrictions on parts.

2. Create a composite graphic for GMM

   GMM requires one graphic to represent the entire composite object type. This is the image that is shown in the GMM shop and placed on maps created in GMM. Follow the same guidelines for creating graphics as with singular object types.

3. Describe composite in GMM

   Add one element to the 'objectTypes' array in the GMM inventory file to represent the composite object type. Let it reference the parts by symbolic id. Here is a full example.

```
{
        "id": "droptower_7x9",
        "code": 19,
        "label": "Drop tower",
        "image": "droptower_7x9.png",
        "transparency": 0,
        "widthInTiles": 7,
        "heightInTiles": 9,
        "parts": [
                {
                        "idRef": "droptower-column_7x9",
                        "dx": 0,
                        "dy": 0
                },
                {
                        "idRef": "droptower-seats_5x2",
                        "dx": 1,
                        "dy": 1
```

```
                }
            ]
    },
    {
            "id": "droptower-column_7x9",
            "code": 20,
            "label": "Drop tower column",
            "image": "droptower-column_7x9.png",
            "appearances": 1,
            "movement": "none",
            "transparency": 0,
            "widthInTiles": 7,
            "heightInTiles": 9,
            "interaction": "intangible"
    },
    {
            "id": "droptower-seats_5x2",
            "code": 21,
            "label": "Drop tower seats",
            "image": "droptower-seats_5x2.png",
            "appearances": 1,
            "movement": "smoothVerticalSweep",
            "movementParameter1": 148,
            "movementParameter2": 0,
            "movementStartAt": "0%",
            "activityCycles": 2,
            "activitySpeed": 220,
            "inactivityTimeMinimum": 3,
            "inactivityTimeMaximum": 5,
```

```
    "transparency": 0,
    "widthInTiles": 5,
    "heightInTiles": 2,
    "interaction": "fatal"
}
```

The following table describes the properties of the composite object type.

Property	Description
id code label image transparency widthInTiles heightInTiles	Same as for singular object types. See Describe object type in GMM The image is meant to represent the entire composite object type
parts	Array describing the constituent parts. The order defines the relative drawing order of the parts, from back to front-facing
Properties of a "part"	
idRef	Reference to a singular object type's symbolic id within the GMM inventory file
dx	The part's relative horizontal offset from the composite object's anchor position, measured in tiles
dy	The part's relative vertical offset from the composite object's anchor position, measured in tiles

4. Include composite in GMM shop

   Similar to singular object types, reference the composite object type (by symbolic id) in the GMM shop file.

5. Deploy composite in Scratch

   There is no special deployment needed in Scratch other than deploying the constituent parts.

6. Deploy composite in GMM

   Deploy the GMM software containing the updated resources as described in Deploy GMM.

7. Validate

   In GMM open or create a new map. In the shop, your new composite object type should be found. Select it and add new objects to the map. Save the map and deploy it for validation in Scratch (see Deploy new maps).

Atmospheres

An atmosphere is meant to represent an atmospheric condition in a region of the map, for example rain, snow or shades of darkness. An atmosphere is a singular object type that has these special characteristics.

- An atmosphere object always spans the full height on a map
- An atmosphere object resides on the foremost “atmos” layer (see Drawing order), so atmospheric images are typically (partially) transparent.

To create a custom atmosphere, follow the same procedure as in Object types but with these special instructions.

- Create 1 image as the visual indicator in GMM
    - Its width is the width of the atmospheric condition, a multiple of tiles
    - Its height is 1 tile (30px) so it fits the *atmosphere strip* in GMM
    - This image is typically non-transparent
    - The image file is conventionally named “${baseName}_${tileWidth}”
    - Store the image file under ‘src/main/webapp/media/objectTypes/ATMOS’
    - Reference this image from the GMM inventory file
    - In the GMM inventory file, set the property “interaction” to ”atmosphere”. Set the “movement” to “none”. Set the “heightInTiles” to 1
- Create 1 image file for a static atmosphere (for example, darkness) or a sequence of image files for an animated atmosphere (for example, rain) as the actual appearances in Scratch (the costumes)
    - The width is the same as the indicator image
    - The height is 11 tiles (330px), so it spans the full height of the map
    - This costume image is typically (partially) transparent, however you could also use the inventory’s “transparency” attribute
    - Upload in Scratch the same way as with other object types

Here is an example atmosphere in the GMM inventory file.


```
{
        "id": "ATMOS_dusk_8",
        "code": 14,
        "label": "Dusk",
```

```
        "image": "ATMOS/dusk_8.png",
        "appearances": 1,
        "movement": "none",
        "transparency": 0,
        "widthInTiles": 8,
        "heightInTiles": 1,
        "interaction": "atmosphere"
}
```

Characters

A character is the persona in the game that the player controls. A character can run, hold still, jump and slide. The objective of the game is to safely guide the character from the beginning of the map until the finish while earning as many points as possible by collecting value objects.

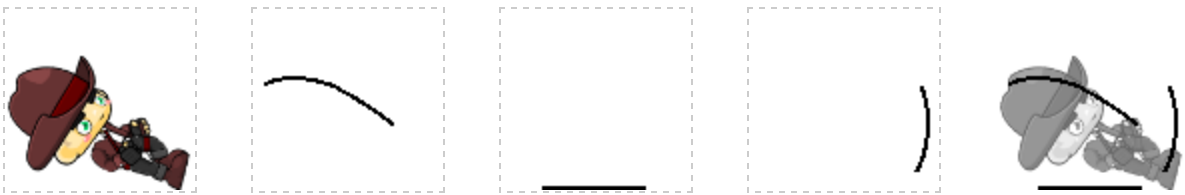
The GMM software allows you to set the character that goes with a particular map. See Deploy new maps. The default character is the cowboy figure.

In order to create a new character, you need to create the appearances for running and one appearance for sliding. In addition, there is one set of *sensor images* for running and one similar set for sliding. The sensors are invisible contour lines that are used to detect whether a character is blocked upwards, downwards or at the front. You can draw the contours in any color, Scratch will make them invisible using full transparency. The sensor images for the cowboy are illustrated below.

Running

character appearance | sensor up | sensor down | sensor front | composite image (for illustration purposes)

Sliding



This is the procedure to add a new character.

1. Create character graphics

For every running appearance of the character, create a separate image. There can be up to 15 alternating appearances. All images should be 3 tiles in height (90 pixels) and the width can be variable (the appearances will be centered around the character's position in the map). Create one image for sliding. Also create the sensor images (up, down, front) for both the running and sliding mode.

2. Store a character graphic in GMM

   GMM requires one graphic to represent the character. This is the image that is shown in the GMM shop and that can be used for simulations when creating a map in GMM. You can use one of the running appearances as the reference graphic. Store that image in the GMM source repository under 'src/main/webapp/media/characters.

3. Describe character in GMM

   Add one element to the 'characters' array in the GMM inventory file. Here is an example element.

```
{
        "id": "Cowboy",
        "code": 0,
        "label": "Cowboy",
        "image": "cowboy.png",
        "appearancesRunning": 10
}
```

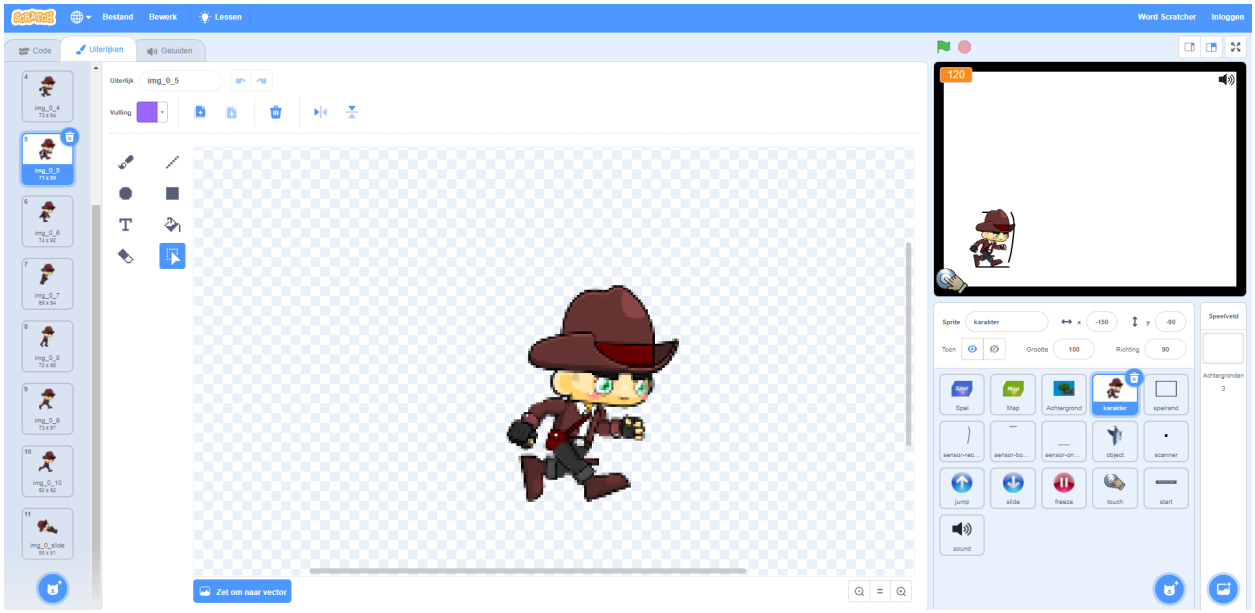
   The following table describes the properties of a character.

Property	Description
id	Unique character symbolic id It's used in map json definitions
code	Unique character numeric id It's a sequential number within the characters It's used in map bitmaps and in Scratch
label	A human readable label for the character It's used in the 'Set character' dialog of GMM
image	Relative file path for the image shown in GMM
appearancesRunning	The number of alternating appearances when in the

	running mode (between 1 and 15)

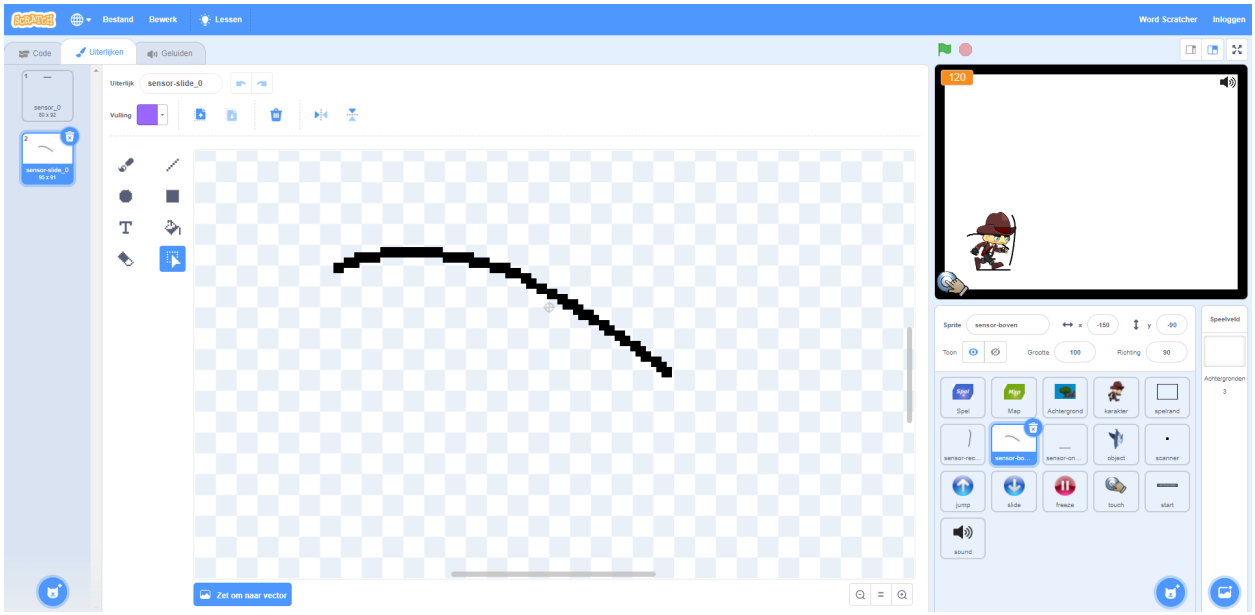
4. Deploy character in Scratch

Inside the Scratch project, go to the costumes of the sprite called "*karakter*". First, upload the graphic images for the running appearances. You should rename every costume as follows : "img_${code}_${i}". The ${code} represents the character's numeric id. The ${i} represents a one-based sequence number for multiple appearances. Second, upload the single image for the sliding appearance and rename it "img_${code}_slide".



Then upload the sensor image for running and sliding to the sprites called "*sensor-boven*" (up), "*sensor-onder*" (down) and "*sensor-rechts*" (front). You should rename every costume as follows : "sensor_${code}" and "sensor-slide_${code}",

respectively. The ${code} represents the character's numeric id.



5. Deploy character in GMM

   Deploy the GMM software containing the updated resources as described in Deploy GMM.

6. Validate

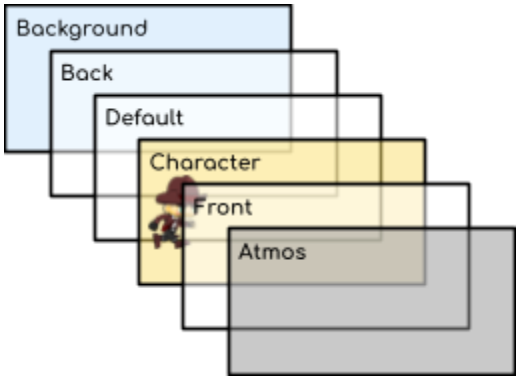
   In GMM open or create a new map. Click "*Set character*" from the menu. In the dialog window, your new character should be found. Select it. Save the map and deploy it for validation in Scratch (see Deploy new maps).

Drawing order

The elements in a map are logically organized in layers that define their relative drawing order. Different layers can overlap. Objects in the same layer cannot overlap when in their anchor positions, with the exception of the object parts of composite objects. Composite object types explicitly define the drawing order of their parts (see Composite object types). The relative drawing order of moving objects in the same layer is unspecified.

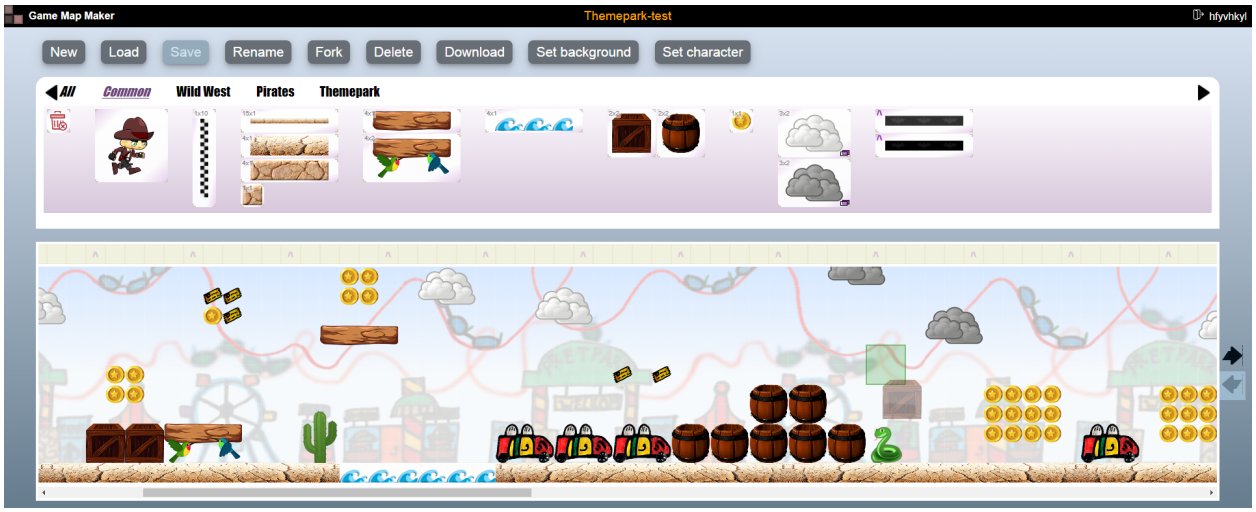
These are the layers, from back to front-facing.

- The **background** layer contains the map's single background. This is the only layer that scrolls at a reduced speed to simulate visual distance
- The **back** layer contains decorative objects with interaction = "*intangible-back*". These objects are behind any higher-layered objects as well as the character. In GMM, the shop identifies these object types with the badge
- The **default** layer contains all regular objects. These objects are behind the character. This is the only layer that accepts tangible objects and composite objects
- The **character** layer contains the map's single character
- The **front** layer contains decorative objects with interaction = "*intangible-front*". These objects are in front of any lower-layered objects as well as the character. In GMM, the shop identifies these object types with the badge
- The **atmos** layer contains atmospheric objects with interaction = "*atmosphere*". These objects are in front of everything else. In GMM, there is a special strip with the emblem ∧ where to place the atmospheric objects



Deploy new maps

You can use the GMM application as a map editor tool (see also Deploy GMM). Map editors can set up their own private collection of maps or join an already existing collection to collaborate. New maps can be created and existing maps can be changed or forked (copied). You can freely add or remove objects including atmospheres, set the background and the character.



When you have finished creating a map, follow this procedure to deploy it to Scratch.

1. Save map

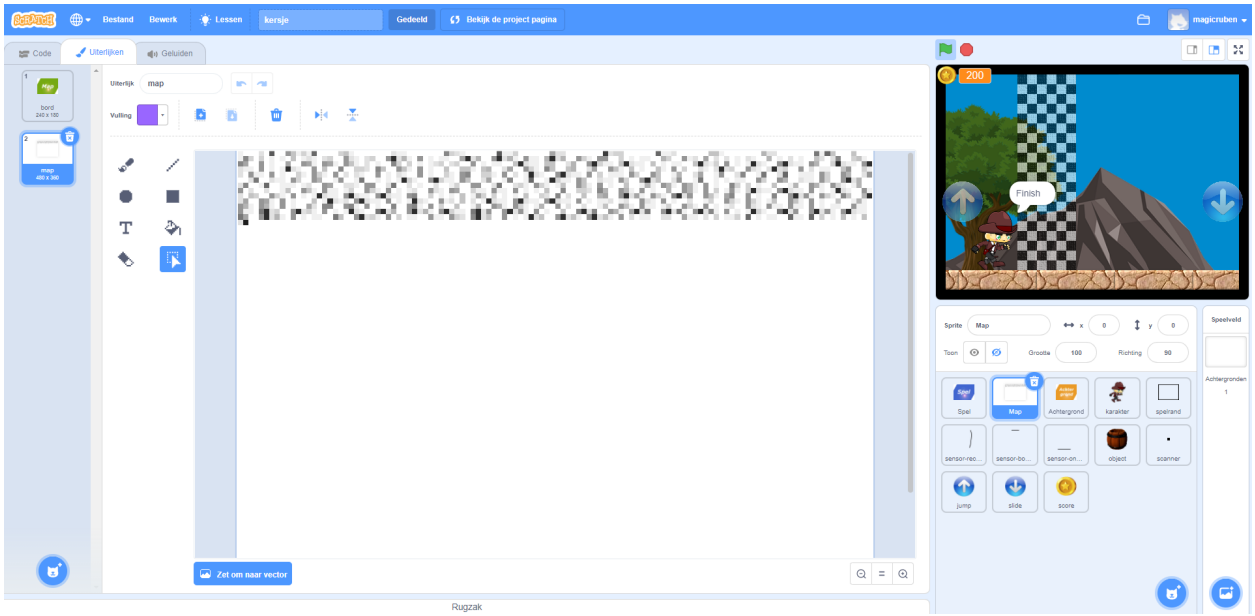
   Make sure to save your map in GMM. Any unsaved changes are not included in the deployed map.

2. Download map

   Click download to store the map's bitmap file (by default called 'map.png') to your local computer. It depends on your browser's settings if it will prompt for the download location and filename.

3. Deploy map in Scratch

   Remix the reference Scratch project "Kersje" to create your own new version. Inside the remixed Scratch project, go to the costumes of the sprite called "*Map*". First delete the existing costume named "map", then upload the new map's bitmap file and make sure that costume is again named "map".



4. Validate

   Simply start playing in Scratch. Have fun!

Deploy GMM

GMM is a single-page Javascript web application that runs in your browser. The application needs to be hosted on a pc or server such that your browser can access the resource files (html, css, js, images, etc.) and make use of the backend functionalities (load, save, download maps etc.). This means you need to deploy the web application to a Java application server (Jetty, Tomcat, JBoss, etc.). Maps are stored on file storage that is accessible from the application server.

GMM source repository

The GMM source code is available on GitHub. You can clone this git repository locally and start making local changes, in particular adding extensions as described in this document.

https://github.com/jandebr/gmm

GMM persisted maps

The GMM application saves maps in a directory structure on the application server's file system. You need to configure the base directory path in one of these ways :

- The Ant property "**MAPS_BASE_DIRECTORY**" that is used by the Ant build script (see Build and deploy WAR). Its value will go inside the *web.xml* deployment descriptor in the WAR file that is generated by the script
- Alternatively and to override the value inside *web.xml*, you can specify the custom Java system property "**mapsBaseDirectory**", for example
  > java -DmapsBaseDirectory="/basepath/to/maps" ...

Build and deploy WAR

To build and package the GMM application as a Java WAR file, simply run the default "*war*" target of the Ant build.xml script that is available from the source repository. Remember to first configure the Ant property "MAPS_BASE_DIRECTORY" as described above. When run, the Ant script will produce an updated "gmm.war" file in the "dist" directory. Deploy the WAR file to your application server (servlet container) of choice.

Validate by pointing your browser to the context path of the deployed GMM application (for example, http://localhost:8080/gmm/)