Scratch game “Kersje”

Extension Guide V7

July 9th 2022

—

Jan & Ruben De Beer

Overview **2**

Introduction **3**
Map anatomy 3
GMM extension files 3

Setup GMM **4**
GMM source repository 4
GMM persisted maps 4
Build and deploy GMM 5

Deploy new maps **6**
Save map 6
Download map 6
Deploy map in Scratch 6

Extensions **7**
Backgrounds 7
Create background graphics 7
Store background graphics in GMM 7
Describe background in GMM 7
Deploy background in Scratch 8
Deploy background in GMM 9
Validate 9
Object types 10
Create object type graphics 12
Store an object type graphic in GMM 12
Describe object type in GMM 12
Include object type in GMM shop 18
Deploy object type in Scratch 18
Deploy object type in GMM 18
Validate 18
Composite object types 19
Create composite parts 20
Create a composite graphic for GMM 20
Describe composite in GMM 20
Include composite in GMM shop 22
Deploy composite in Scratch 22
Deploy composite in GMM 22
Validate 23

Teleports 24
Atmospheres 25
Characters 27
Create character graphics 28
Store a character graphic in GMM 28
Describe character in GMM 28
Deploy character in Scratch 29
Deploy character in GMM 30
Validate 30
Drawing order 31

GMM Limitations 32

Game modifications in Scratch 34
Splash screen 34
Create graphic 34
Add graphic to Scratch 34
Apply in Scratch 34
Background music 36
Add sound to Scratch 36
Apply in Scratch 36
Levels and lives 37
Add a map per level 37
Apply in Scratch 37

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.

This guide also documents a few other game modifications including splash screens and sounds, which are only customizable from the Scratch project editor.

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 Setup 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, characters 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

Setup GMM

GMM (Game Map Maker) is a map editor tool that runs in the browser. 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 (see Extensions).

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 below). 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 GMM

GMM is a web application containing static resource files (html, css, js, images, etc.) and dynamic elements (Java servlets). For this reason, the application should be deployed to a web Servlet Container (Jetty, Tomcat, JBoss, etc.).

After configuring the Ant property "MAPS_BASE_DIRECTORY" (see above), you can 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. This 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/

Deploy new maps

When you have finished creating a map in GMM, follow these steps to deploy the map to Scratch and start playing.

1. Save map

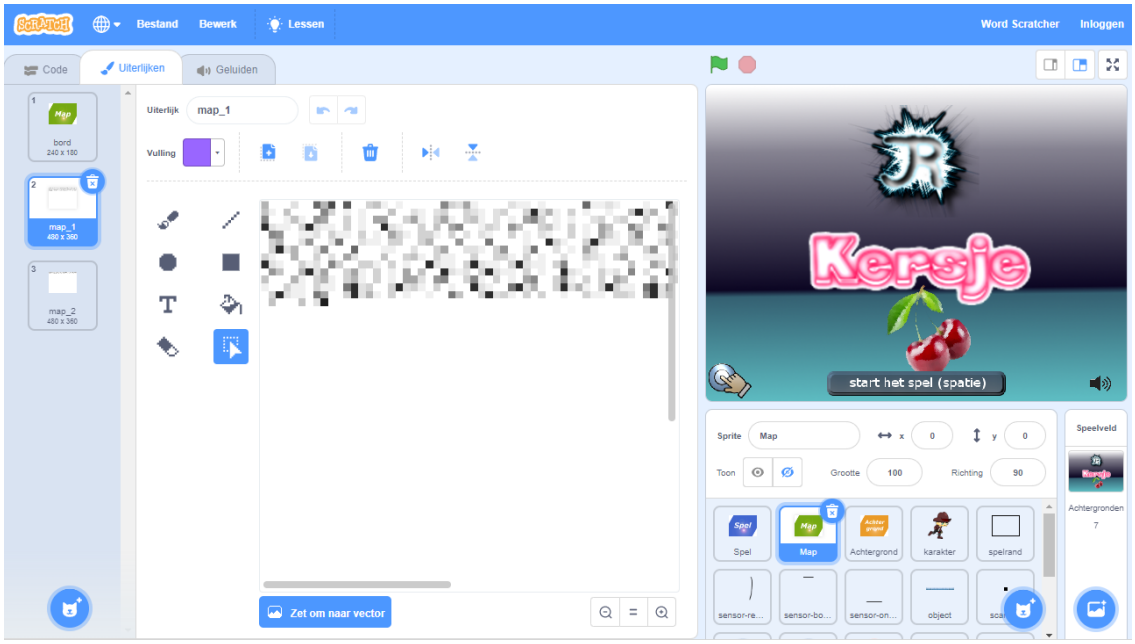
   Make sure to save your map in GMM. Any unsaved changes are not included in the deployed map.

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*". Upload the map's bitmap file and rename as follows : "map_${level}". The ${level} represents the level number (starting at 1), see Levels and lives. You can now start playing.



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": [
   ```

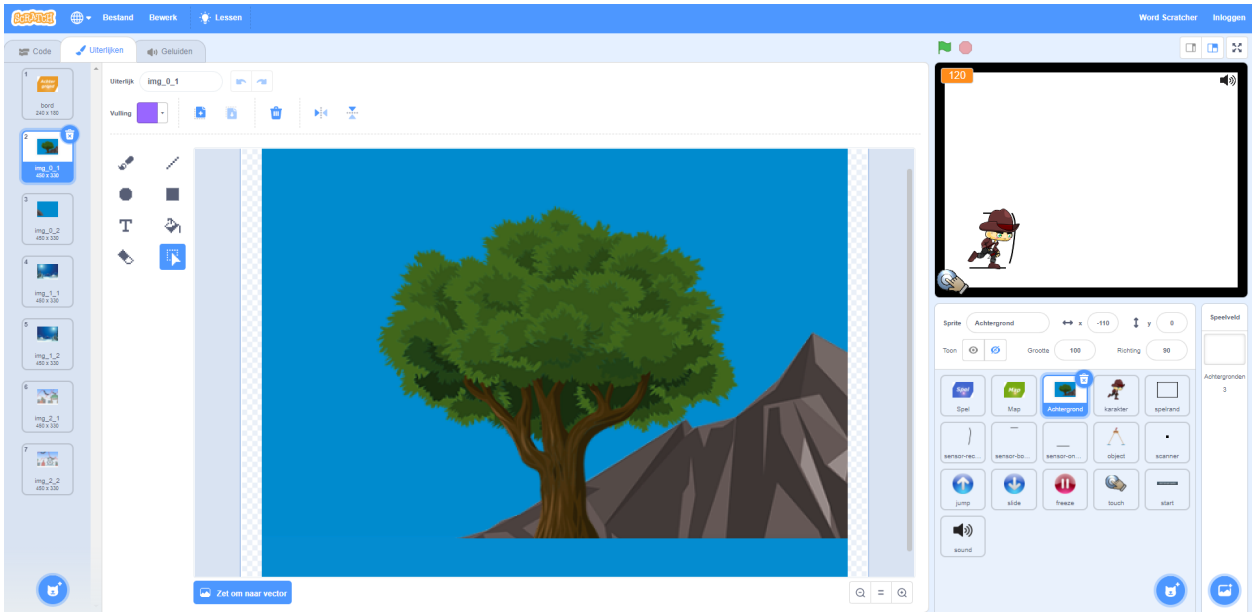
```
            "MountainTree/MountainTree1.png",
            "MountainTree/MountainTree2.png"
        ],
        "gmm-style": {
            "opacity": 0.3,
            "grid-brightness": 1.0
        }
}
```

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
gmm-style	(optional) style information when rendered in the map editor of GMM
Properties of "gmm-style"	
opacity	The opacity of the background in GMM, between 0 (fully transparent) and 1 (fully opaque)
grid-brightness	The brightness of the tile grid overlaying the background in GMM, between 0 (black) and 1 (white)

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" 

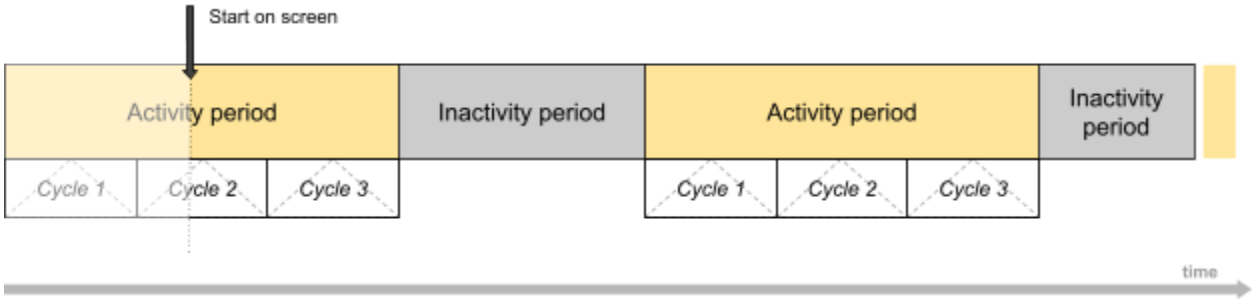
We refer to an object as *static* when it does not move and has a single appearance. When an object is not static, it is *alive* and will endlessly alternate between an *active state* and an *inactive state*.

- 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 can be controlled whether it should start at or within an activity period, at the beginning of an inactivity period, or an inactivity period after a threshold proximity to the character is reached. As for the first activity case, the following holds.

- If the object has movement, you can control if it should start at a fixed or at a random offset within the activity cycle. If there are multiple cycles during one activity period, the cycle is always chosen randomly to have a lesser predictability between gameplays
- Always alive objects without movement (ever alternating appearances), will be synchronous in appearance throughout the map. This is intentional. For example, stretches of multiple sea objects will result in synchronous waves



A vertically moving, tangible object can *lift* the character up or down when the character stands on top of the object. For the best effect, the object should have a flat, non-transparent shape at the top of the object bounding box.

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 "*verticalJump*", let the object jump between two vertical positions. Due to gravity, the object slows down as it approaches its highest position • 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 • 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 ⚠ 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 "*lifeStartAt*" 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 "*score*", the character earns a number of points specified by *"value"* • When "*fatal*", the character loses one life when it touches the object (see Levels and lives) • When "*fatal-active*", the character loses one life when it touches the object (see Levels and lives) but only when the object is in the active state • When "*finish*", the character has reached the end of the map • When "*atmosphere*", an atmospheric condition applies (see Atmospheres) • When "*teleport*", the character is teleported to a distant place in the map. See Teleports

	• When "*size-shrink*", the character shrinks in size • When "*size-restore*", the character restores to its original size • When "*shield*", the character is protected from harm for an amount of time specified by *"value"*
depthLayer	The layer on which objects of this type reside (see Drawing order). These are the supported values : • "*default*" • "*back*" • "*front*" • "*atmos*" When unspecified, "*default*" applies
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
lifeStartAt	For object types that are alive, the "on screen" offset. These are the supported values : • When "*activity*", the offset is at the start of an

	activity cycle • When "*activity 25%*", the offset is at one quarter of an activity cycle • When "*activity 50%*", the offset is midway of an activity cycle • When "*activity 75%*", the offset is at three quarters of an activity cycle • When "*activity random*", the offset is random within the activity period • When "*inactivity*", the offset is at the start of an inactivity period • When "*activity in reach*", the offset is at the start of an activity period, conditioned by the object being close enough to the character (see property *lifeStartAtProximity*) • When "*inactivity in reach*", the offset is at the start of an inactivity period, conditioned by the object being close enough to the character (see property *lifeStartAtProximity*) When unspecified, "*activity random*" applies
lifeStartAtProximity	For object types whose *lifeStartAt* is either "*activity in reach*" or "*inactivity in reach*", the horizontal distance between the object and the character before the (in)activity kicks in. The range is between 0 and 255 pixels, measured center-to-center.
activityCycles	For object types that are alive, the number of activity cycles. The range is from 1 to 15. The special value 0 indicates that the object has 1 activity cycle and then is gone.
activitySpeed	For object types that are alive, the speed of activity, from 0 (slowest) to 255 (fastest) • When there is movement, the speed of moving • When there is no movement, the speed of changing appearances
appearancesSpeedWhileMoving	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

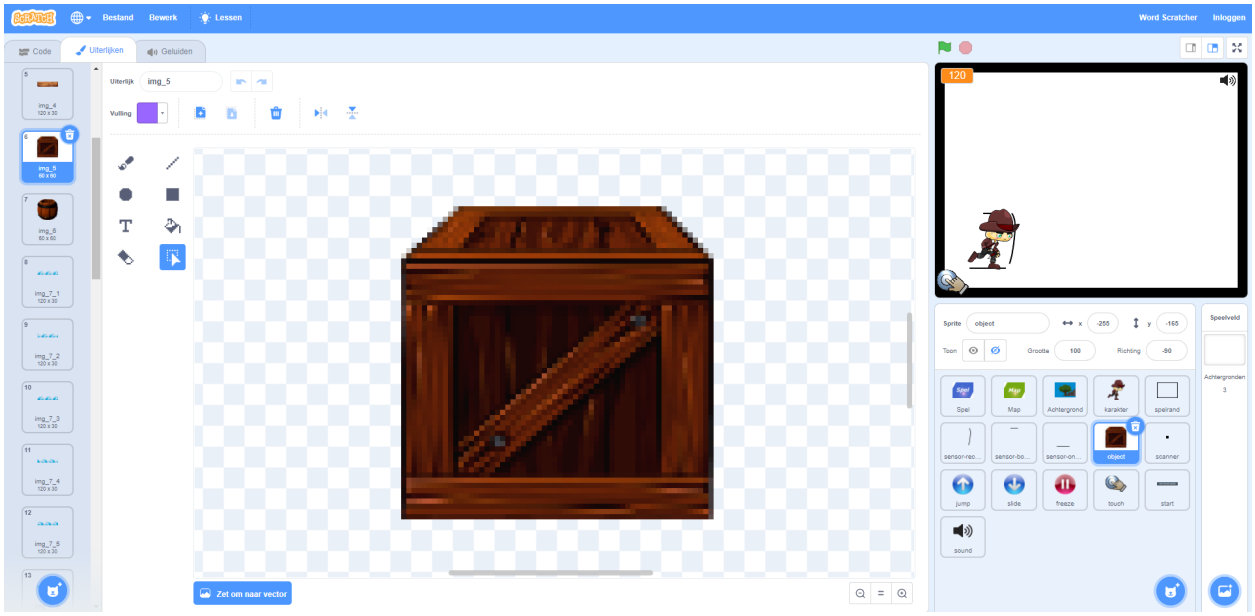
	inactivity) to 15, a relative unit of time.
inactivityTimeMaximum	For object types that are alive, the maximum duration of the inactivity periods. The range is from 0 (no inactivity) to 15, a relative unit of time.
inactivityAppearance	For object types that are alive, the one-based index number of the 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"*. ● When "*score*", the number of points that are earned. The range is between 0 and 255. ● When "*teleport*", the number of screens ahead that the character is teleported to. The range is between 2 (technical minimum) and 255. ● When "*atmosphere*" : ○ 0 = nothing special (default) ○ 1 = wind. The character floats slightly when jumping ○ 2 = blast. The character floats heavily when jumping ○ 3 = lightning effect. ● When "*tangible*" : ○ 0 = nothing special (default) ○ 1 = slippery underground, like ice ● When "*shield*", the duration in seconds that the character is protected. The range is between 0 and 255
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.

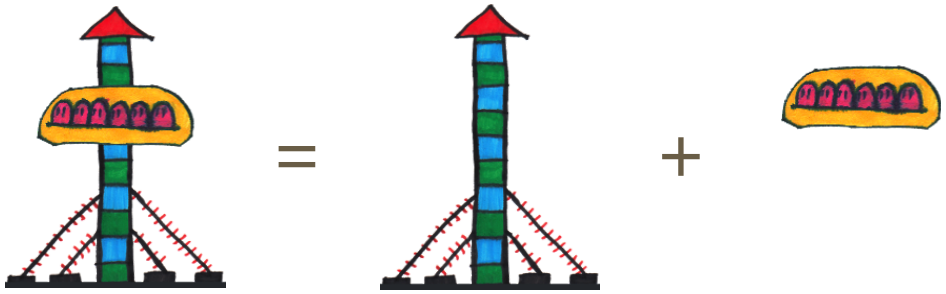
7. Validate

   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 must be on the “default” layer (see Drawing order)

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,
                "lifeStartAt": "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).

Teleports

A teleport is a special object having the interaction type "*teleport*". When the character hits a teleport, it is instantly moved ahead a fixed number of screens. This action is called *teleporting*. Since objects cannot be parameterized, a few teleport object types are provided for a varying number of screens (2 as a technical minimum, 5, 10, 20 and 50). One can place teleports in series to target more locations in a map.

Teleports can be very useful when building and testing maps, in order to skip otherwise repetitive parts of the map. But also during gameplay they can be used to create shorter passages through a map.

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
- An atmosphere object type can have a value to represent certain special interactions with the character, such as floating in whirlwinds (see Describe object type in GMM)

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.

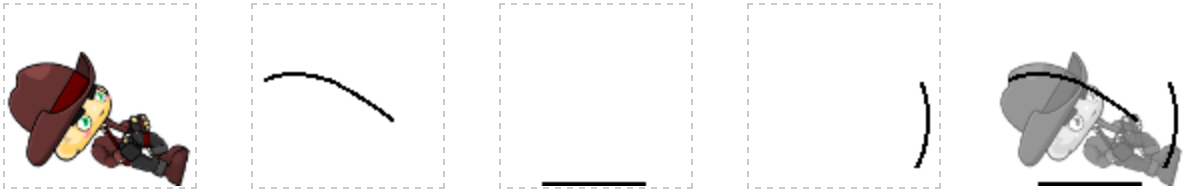
The GMM software allows you to set the character that goes with a particular map. See Deploy new maps. The default character is the cowboy figure.

In order to create a new character, you need to create the appearances for running, one appearance for sliding and one appearance for gliding. In addition, there is one set of *sensor images* for running/gliding and another 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. In Scratch they will be invisible using full transparency. The sensor images for the cowboy are illustrated below.

Running or gliding

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. Images should be around 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* (slanted) and one image for *gliding* (on slippery surfaces, like ice) . Also create the sensor images (up, down, front) for both the *running/gliding* and the *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,
           "height": 92
   }
   ```

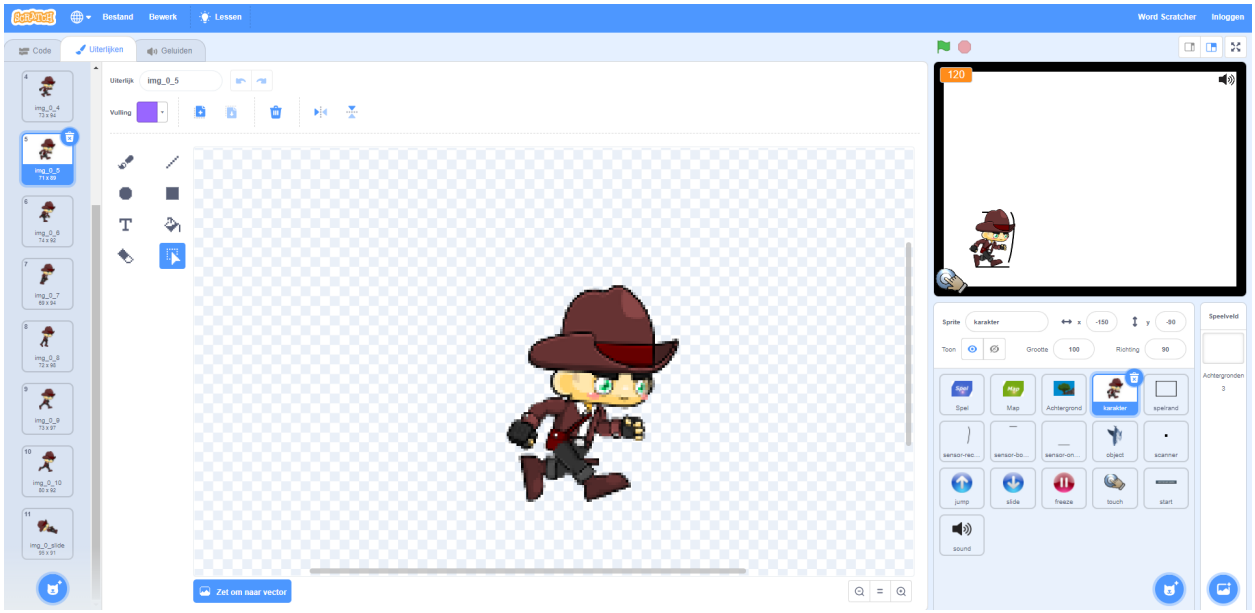
   The following table describes the properties of a character.

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

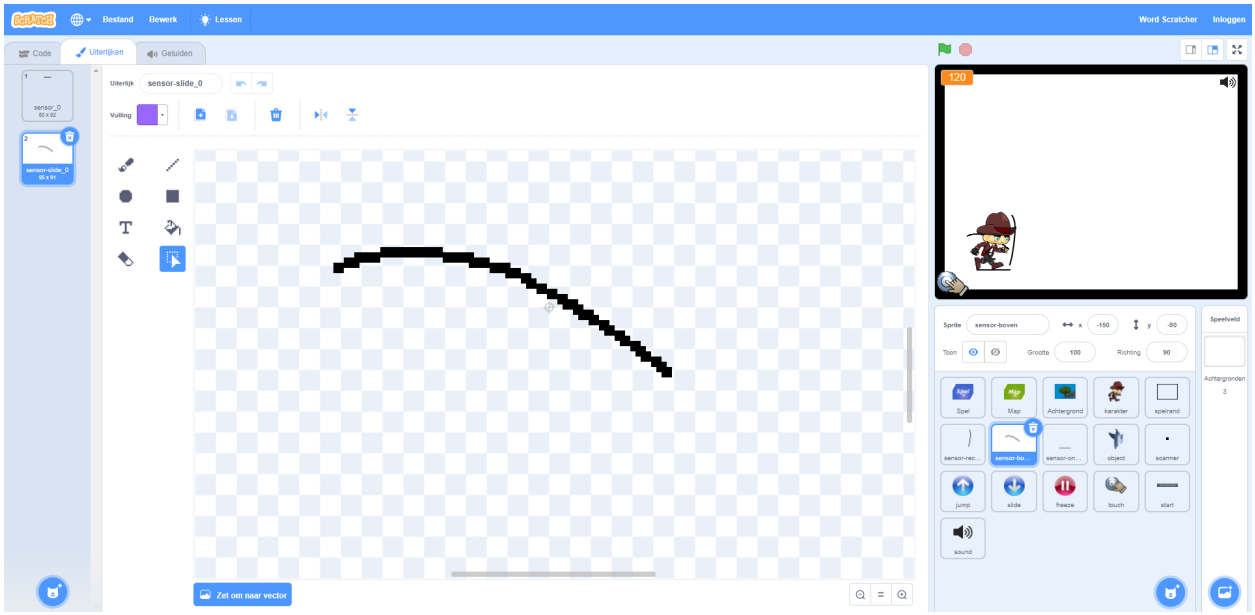
label	A human readable label for the character It's used in the 'Set character' dialog of GMM
image	Relative file path for the image shown in GMM
appearancesRunning	The number of alternating appearances when in the running mode (between 1 and 15)
height	The (average) height of the character, in pixels. This is used, for example, to accurately elevate the character when standing on vertically moving platforms

4. Deploy character in Scratch

Inside the Scratch project, go to the costumes of the sprite called "*karakter*". 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 ${i} represents a one-based sequence number for multiple appearances. Also upload the single image for the *sliding* appearance and rename it "img_${code}_slide". Also upload the single image for the *gliding* appearance and rename it "img_${code}_glide".



Then upload the sensor image for running/gliding 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]).

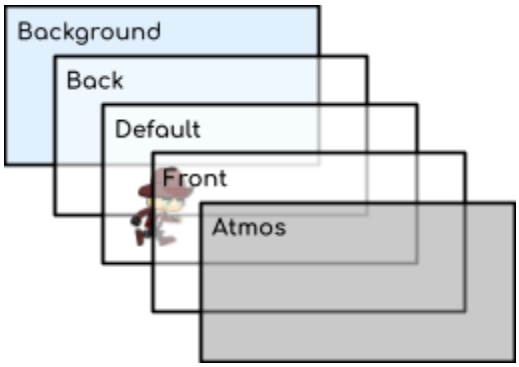
Drawing order

The elements in a map are logically organized in layers that define their relative drawing order. Objects in different layers can overlap. Objects in the same layer cannot overlap when in their anchor positions, with the exception of the 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.

All objects of the same object type share the same layer. That layer is defined with every object type (see Describe object type in GMM).

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 typically contains decorative objects, which go behind the character and objects in the default layer. In GMM, the shop identifies these object types with the badge
- The **default** layer contains the character and typically tangible objects. An important constraint is that all parts of a composite object type must be on this layer
- The **front** layer typically contains decorative objects, which go in front of the character and objects in the default layer. In GMM, the shop identifies these object types with the badge
- The **atmos** layer contains atmospheric objects. These objects are in front of everything else. In GMM, there is a special strip with the emblem ∧ where to place the atmospheric objects



GMM Limitations

This section documents the current GMM limitations. Please note that Scratch has its own set of limitations.

Property	Limitation
Maps	
Maximum screens in a map	100 could be less depending on total map size
Maximum objects in a map	2000 could be less depending on total map size
Maximum different object types used in a map	256
Backgrounds	
Maximum backgrounds in the inventory	256
Maximum screens of a background	255
Objects	
Maximum object types in the inventory including composite parts, teleports and atmospheres	4096
Maximum width in tiles of an object	15
Maximum height in tiles of an object	11
Maximum score value of an object	255
Maximum appearances of an object	15
Maximum activity cycles of an object	15
Teleports	
Minimum screens distance of a teleport	2
Maximum screens distance of a teleport	255

Property	Limitation
Atmospheres	
Maximum appearances of an atmosphere	15
Characters	
Maximum characters in the inventory	256
Maximum running appearances of a character	15

Game modifications in Scratch

Some game modifications are only available from the Scratch project editor and not within GMM. You can apply these in an updated (or remixed) [Scratch project] and start playing.

Splash screen

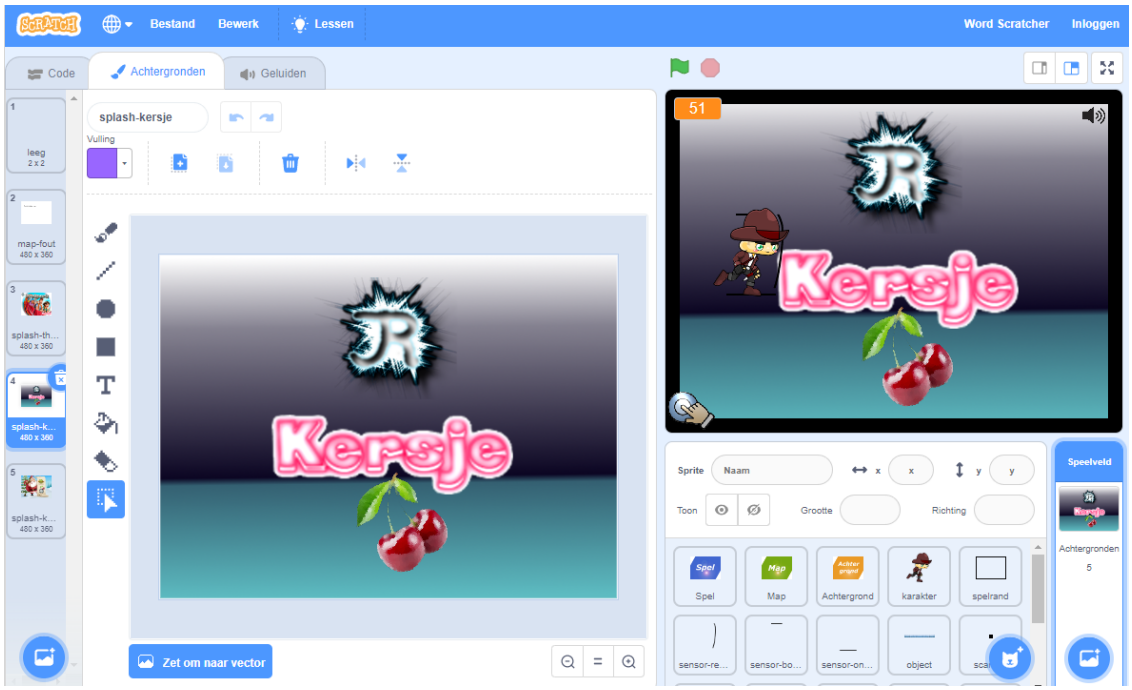
To customize the splash screen, follow these steps.

1. Create graphic

   Create an image that fills the entire screen in Scratch (480px wide by 360px high).

2. Add graphic to Scratch

   Add the splash image as a background to the Scratch project.



3. Apply in Scratch

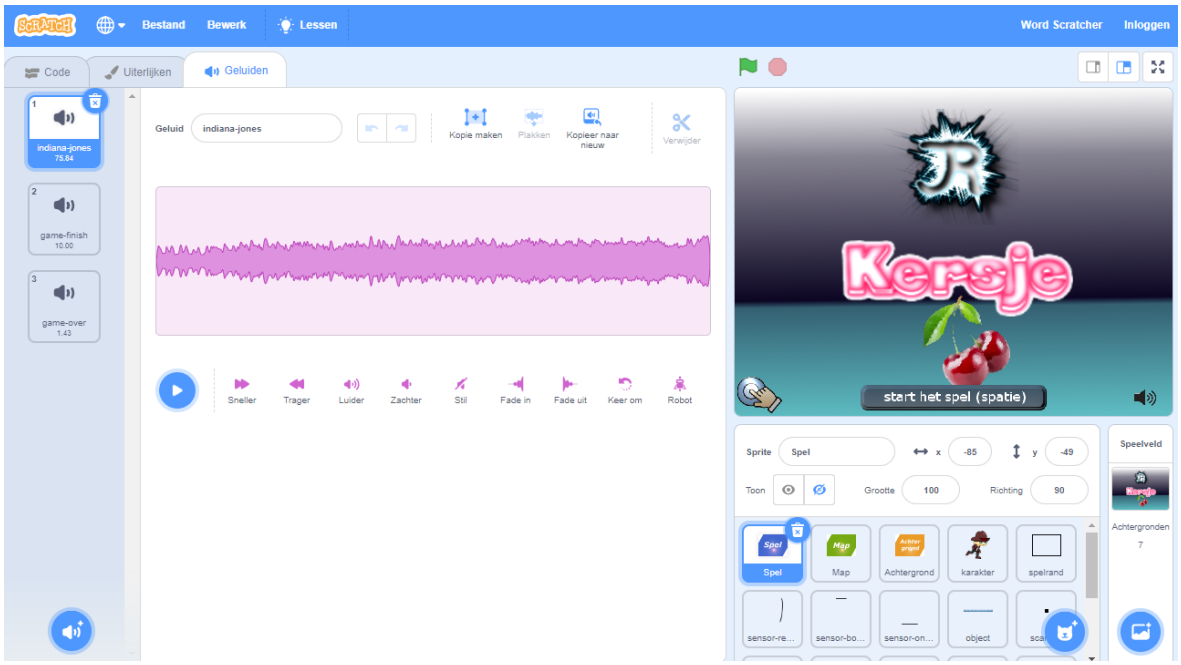
Go to the code editor of the sprite "*Spel*". In the block "*Init spel variabelen*", set the variable "*spel-splash-scherm-naam*" to that background name.

Background music

To customize the background music while playing, follow these steps.

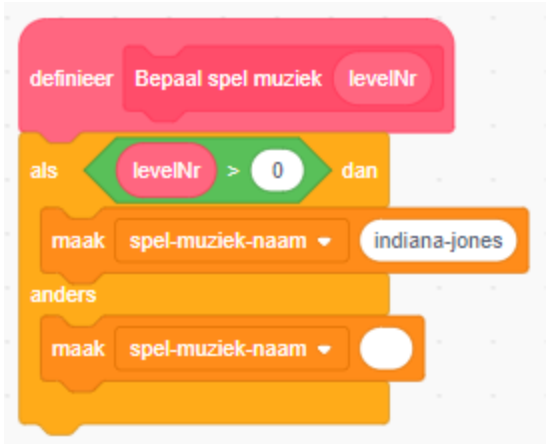
1. Add sound to Scratch

   Go to the sounds panel of the sprite "*Spel*" and upload the background music file.



2. Apply in Scratch

   Go to the code editor of that same sprite. In the block "*Bepaal spel muziek*", set the variable "*spel-muziek-naam*" to that sound name. You can differentiate the background music per level. Note, if you leave the variable empty, no background music will play.



Levels and lives

One map represents one *level* in the game. You can create multiple levels in the game and as the game gets more difficult to complete, grant the character more than one life. With every failure, one life is lost and the character goes back to the start of the current level.

❤️❤️❤️

To have more than one level and alter the number of lives, follow these steps.

1. ### Add a map per level

   For every level in the game, create a map and deploy it as outlined in Deploy new maps

2. ### Apply in Scratch

   Go to the code editor of the sprite "*Spel*". In the block "*Init spel variabelen*", set the variable "*spel-aantal-levels*" to the number of levels. Likewise, set the variable "*spel-aantal-levens*" to the initial number of lives.