Bridges and Tunnels

Bridges and tunnels for streets and tracks are defined with config files too.

Bridges

A bridge configuration files specifies a set of parameters to define the properties, dimension, features, and visuals of tracks. They are stored in `.lua` files in the folder `res/config/bridge/`. The file has the following format:

```
function data()
return {
  -- property definitions

  materialsToReplace = {
    -- material definitions; only applies to carriers = { "ROAD" }
  },

  updateFn = function
}
end
```

Property definitions

The bridge properties start with the common metadata properties:

```
  name = _("Stone bridge"),
  yearFrom = 0,
  yearTo = 0,
  cost = 200.0,
  carriers = { "RAIL" , "ROAD"},
  speedLimit = 90.0 / 3.6,
  categories = { "misc" },
```

Their meaning is as follows:

- `name` is the name of the bridge type that is used in the tooltip. It can be translated in a strings.lua file.
- `yearFrom` is the year from when the bridge should be available. Unset or values below 1851 mean from start.
- `yearTo` is the year until when the bridge should be available. Unset or value 0 means unlimited availability, values below 1850 result in a never available bridge.
- `cost` is the building cost coefficient. Additional costs depending on the street or track type, height and upgrades apply.
- `carriers` is a list of carrier keys. Possible values are “RAIL” and “ROAD”.
- `speedLimit` is the maximum speed in meter per second. If the track or street has a lower

speed restriction, the bridge speed limit is ignored.
- To provide additional filter possibilities, it is possible to assign a bridge to one or more categories. The string keys are added to the `categories` list.

There are several properties available to control the pillars:

```
pillarLen = 3,

pillarMinDist = 12.0,
pillarMaxDist = 48.0,
pillarTargetDist = 12.0,

pillarGroundTexture = "shared/dirt.gtex.lua",
pillarGroundTextureOffset = 10.0,
```

- `pillarLen` is the length of pillar on ground level that is used for collision calculation and ground texture.
- `pillarMinDist` is the minimum distance between 2 pillars.
- `pillarMaxDist` is the maximum distance between 2 pillars.
- `pillarTargetDist` is the optimum distance between 2 pillars that is used as default.
- `pillarGroundTexture` is a reference to a ground texture relative to `res/config/ground_texture/` that should be used around the pillars.
- `pillarGroundTextureOffset` is a distance which is added to all sides of the pillar in which the pillar ground texture is used too.

Another property is `noParallelStripSubdivision`. If it is set to `true`, the `updateFn` will be called for the whole bridge instead of seperate segements. This might be relevant for bridges with large gaps between pillars. If set to `false` or the property is unset, the `updateFn` will be called several times. Be aware that bridge segments are limited to a length of 100 meters. Larger pillar distances then will result in no pillars.

Material definitions

The list is used to overwrite materials defined in the street configuration. See there for further details on the purposes of the different types.

UpdateFn

The `updateFn` function is used to construct the bridge model from individual parts. It receives a `params` table that contains a list of properties which are provided with the function call:

- `pillarHeights` contains a list with the heights of all the pillars.
- `pillarWidth` is the width of a pillar.
- `pillarLength` is the length of a pillar.
- `railingWidth` is the width of the bridge.
- `state.models` is the list with all loaded bridge models like described above.
- `railingIntervals` is list of bridge parts between the pillars. There are four properties for each section:
  - `curvature` is the inversion of the radius in this railing section (`1/radius`).

    - `hasPillar`: is a pair of two values that are the ids of the pillars at the start and end of the railing section. `-1` tells that it is the start or end of the whole bridge part that is built with the `updateFn` call.
    - `lanes` is a list of lanes that are built with the bridge. For streets and single tracks, there is only one entry in the list but for parallel tracks, there is an entry for each track. Each entry has two properties:
        - `offset` is the orthogonal offset from the bridge center lane. For tracks this is usually a multiple of 5.
        - `type` is the collision type of this particular lane.
    - `length` is the length of the railing section between the pillars.

The collision types are:

- `0` no collision
- 1 collision on the left (in the dragging direction)
- 2 collision on the right (in the dragging direction)
- 3 collision on both sides

It is expected that the result contains two lists:

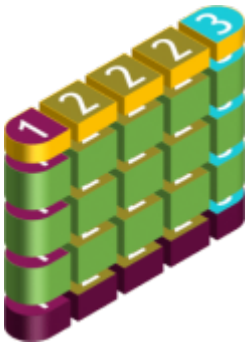
- `result.pillarModels` is a list of pillars. Each element of the list corresponds to one of the `params.pillarHeights` entries, thus both lists should have the same length. The elements are lists themselves containing all the rows of models for each pillar.
- `result.railingModels` is a list of railing sections. Each element of the list corresponds to one of the `params.railingIntervals` entries, thus both lists should have the same length. The elements are lists themselves containing all the rows of models for each railing section.

```
return {
  pillarModels = {
    { -- pillar 1
      { -- row 1
        { id = "bridge/stone/pillar_btm_side.mdl", transf = { ... } },
        { id = "bridge/stone/pillar_btm_rep.mdl", transf = { ... } },
        { id = "bridge/stone/pillar_btm_side2.mdl", transf = { ... } },
      },
      ... -- more rows
    },
    ... -- more pillars
  },
  railingModels = {
    { -- section 1
      { -- row 1
        { id = "bridge/stone/railing_end_side.mdl", transf = { ... } },
        { id = "bridge/stone/railing_end_rep.mdl", transf = { ... } },
        { id = "bridge/stone/railing_end_rep.mdl", transf = { ... } },
      },
      ... -- more rows
    },
    ... -- more sections
  },
}
```

Default UpdateFn

In scripts/bridgeutil.lua a prefabricated function is offered, which is also used by the vanilla bridges. This function gets a config table as parameter. The contents of this table are described below.

Pillars



Bridge pillars consist of three mandatory layers, of which the middle layer can be repeated several times:

- `pillarTop` (yellow) is a list of models used for the upper end of the pillar.
- `pillarRepeat` (green) is a list of models used for the height variable part of the pillar. These models are put together vertically several times and, if necessary, scaled slightly to reach the required height.
- `pillarBase` (purple) is a list of models that are used for the bottom layer of the pillar that touches the ground.

An additional `pillarMain` layer list can be added that is used on top of the `pillarTop` layer and above the track or street bed.

All four lists may vary in length. Possible lengths are:

1 model reference		This model is set in the middle as a pillar.
2 model references		The first model is used for the edges of the pillar (rotated once accordingly), the second model is placed next to each other for the middle of the pillar and scaled slightly until it fills the required width.
3 model references		The first model is used for one side of the pillar, the second model is placed next to each other for the middle of the pillar and scaled slightly until it fills the required width and the third model is used for the other side, but not rotated. Be aware that the boundingbox of the third model needs to be rotated by 180° around the origin.

If you do not want to use the pattern described above to assemble the pillars, but instead assemble another pillar based on the height of the pillar, you can specify a separate function for this:

```
...
config.configurePillar = function(modelData, params, i, height, width)
  ... --function code
end,
...
```

The function receives various parameters:

- `modelData` is a list of available bridge element models and their properities.
- `params` is a list with the parameters for the bridge (section) that currently should be built.
- `i` is the index of the pillar that currently is under construction with the function call.

- `height` is the height of the pillar to be built in meters.
- `width` is the width of the pillar to be built in meters.

The `modelData` is the list of all loaded bridge element models (not only from the current bridge) that is retrieved from the `updateFn` parameter `state`. For each model, the bounding box information is provided:

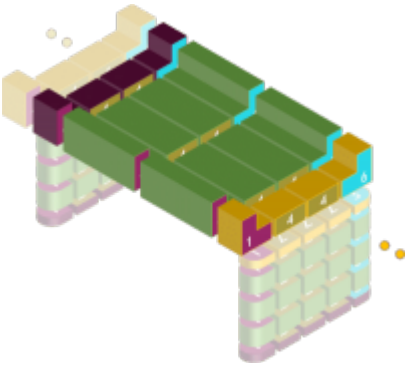
```
...
["bridge/iron/pillar_top_side.mdl"] = {
  ["min"] = { [1] = -1, [2] = -4.6595997810364, [3] = -3.5, },
  ["max"] = { [1] = 1,  [2] = 0, [3] = -2.0499999523163, },
},
...
```

Due to the game loading and extracting all these bounding box data, it is required that all bridge related mdl files are located under `res/models/model/bridge/`!

The `params` parameter receives the forwarded content of the `updateFn` parameters.

Railings

The bridge railing also consists of several rows that are placed next to each other:



- `railingBegin` (purple) is a list of models that are used for the beginning of a bridge railing segment. A railing segment starts at the beginning of the bridge and at each pillar.
- `railingRepeat` (green) is a list of models that are used for the center of a bridge girder segment. These models are lined up horizontally several times and, if necessary, scaled slightly to reach the required length.
- `railingEnd` (yellow) is a list of models that are used for the end of a bridge railing segment.

All three lists may vary in length. There can be either 5 or 8 models in the list. If only 5 elements are included, the elements 1-3 are used as a substitute for 6-8 in a rotated version. The purpose of the elements are:

1. side element
2. side element with no roof because of a collision on the other side
3. side element with no railing and roof because of a colission on this side, e.g. a track switch
4. middle element (that is repeated to fill the bridge width)
5. middle element with no roof because of a collision on at least one of the sides
6. other side element
7. other side element with no roof because of a collision on the other side
8. other side element with no railing and roof because of a colission on this side

Be aware that the boundingbox of the three last models needs to be rotated by 180° around the origin.

UI icons

The ui icons for bridges are displayed in the context menu when some track or street segments are above ground. The icons are located in the `res/textures/ui/bridges/` folder and are named as their corresponding `.lua` file. It is recommended that the bridge icons have a resolution of 60×38 pixels.

Tunnels

A tunnel configuration files specifies a set of parameters to define the properties, dimension, features, and visuals of tracks. They are stored in `.lua` files in the folder `res/config/tunnel/`.

Configuration

The file has the following format:

```
function data()
return {
  name = _("Standard tunnel"),
  carriers = { "ROAD" },   -- or { "RAIL" }
  portals = {
    { "railroad/tunnel_old.mdl" },
    { "railroad/tunnel_double_old.mdl" },
    { "railroad/tunnel_large_start.mdl", "railroad/tunnel_large_rep.mdl",
"railroad/tunnel_large_end.mdl" },
  },
  cost = 1200.0,
  categories = { "misc" },
}
end
```

The `name` property is the text that is used in the tooltip. It can be translated in a strings.lua file. Whether a tunnel is available for rail and/or road is defined in the `carriers` list. Possible values are “`RAIL`” and “`ROAD`”. The cost is influenced by the `cost` coefficient. To provide additional filter possibilities, it is possible to assign a tunnel to one or more categories. The string keys are added to the `categories` list.

The portals block has three different lists for different cases:

1. a single track/lane portal
2. a double track/lane portal
3. a portal for more tracks/lanes

There are two possible combinations for the lists:

- a single model for portals with fixed width
- a set of three models, where the middle one is repeated as often as needed to fill the width.

Street Tunnels usually use the second one for all three lists.

UI icons

The ui icons for tunnels are displayed in the context menu when some track or street segments are below terrain surface. The icons are located in the `res/textures/ui/tunnels/` folder and are named as their corresponding `.lua` file. It is recommended that the tunnel icons have a resolution of 60×38 pixels.

Tracks and Streets

Waypoints and Signals