The Rainfall River Generator

ic111

November 30, 2014

Contents

1 Introduction

The river generator so far present in OpenTTD generates rivers by some path-finding algorithm, without considering landscape at a large scale. Its results are often far from desired.

The Rainfall River Generator aims at doing a better job, and generating real-world-looking rivers and lakes in OpenTTD. The basic idea is calculating

water flow for the whole landscape, and adding rivers and lakes wherever flow is sufficient. As flow is always directed downwards, this automatically adds rivers to valleys. In contrast, the old generator often fails to put rivers into valleys, as they don´t contain that many tiles suitable for rivers, and its search doesn´t consider landscape at a large scale.

Note that the Rainfall River Generator aims at generating real world looking like rivers and lakes to OpenTTD. This comes at a cost in terms of calculation speed: Don´t expect it to calculate the rivers for a big map within a second. This wasn´t my aim, nor are 4096x4096 maps my primary usecase for such a generator, as they are ways to big to play them sensefully anyway. But as map-generation is a one-time-step that is done once before playing a map for days or weeks of real-world-time, IMHO the performance is ok.

1.1 Features

Features of the Rainfall River Generator:

- Each tile receives one unit of water, that water flows downwards, sums up in valleys and lakes. Rivers and lakes are produced wherever more flow than some configurable bounds is available.
- Thus, the amount of rivers and lakes is completely free to the player. Needed flow 151 for a river gives a bit more rivers than needed flow 150.
- Nevertheless, some predefined settings *Few*, *Moderate*, and *Lot* exist, that translate into the configuration world of the Rainfall River Generator.
- Rivers are connected, i.e. if a river starts somewhere, it proceeds without any gap until it disappears in a lake or the ocean or the map edge.
- Lakes are generated wherever the landscape generator (or heightmap) leaves a lake basin. Adding such lake basins in the generator itself would be possible (but is not yet implemented), but both tgp and heightmaps of mountainious areas aren´t exactly known for generating too few basins...

- Depending on how much flow is consumed per lake tile volume, lakes can overflow into a river to the ocean or the next lake, or just consume all rivers flowing into.
- Lakes generated by the Rainfall River Generator conceptionally have a lake depth (the difference between their surface height, and the original tile height). Unfortunately, the game doesn´t support the concept of a lake depth, but maybe this changes in future?
- Wider rivers are supported: If a river has more flow than some configurable bounds, then it will be generated with width 2, 3 or whatever the player causes by configuring the generator.
- Optionally, islands, fan deltas or additional shore tiles can be added to lakes with some configurable probability.
- Optionally, lakes can dig an outflow canyon with some configurable probability, which essentially transforms them into a river.
- By using the latter feature, there is a way of transforming the often inverted-pyramid-lake landscape generated by tgp into a landscape containing much more real-world-like valleys.
- As valleys generated from heightmaps are often quite small at there basis, and don´t offer much usable, flat, space, the generator optionally can make valleys with a river in them wider.

1.2 Generation Steps

The generator performs the following steps to generate the water. Detailed descriptions are subject to the following chapters.

1. Generate an index of the heightlevels on the map. This aims at being able to do some work *for all tiles of a given heightlevel*, without needing to inspect all tiles separately.

2. Bottom-up, for each tile, calculate the number of lower tiles needed on a path to the ocean or a lake basin. Bottom-up means, first for all flat tiles of height zero, then for all inclined tiles of height zero, then for all steep tiles of height zero, then for all flat tiles of height one, and so on.

3. Top-down, calculate the flow amount and flow direction for each tile. Flow is basically the sum of all water that rained on higher tiles, and flowed to the tile at hand.

4. Modify flow in a random manner, to make rivers less straight and more realistic looking, with curves in them. Of course maintain the properties that flow never leads upwards, and is never cyclic.

5. So far, lake basins were just marked as lake centers, that receive flow which disappers there. Now, define real lakes, by iteratively adding tiles to the lake, until an outflow is found, or the needed flow per lake tile volume is exhausted. In the outflow case, all flow is progagated downwards, i.e. a big river flowing into a lake doesn´t disappear there.

6. Find out which tiles actually become river and lake tiles

7. Terraform river beds, and lakes to their surface height, and modify lakes according to the configuration (possible options: reduction to the absolutely necessary tiles to keep the lake connected, addition of islands, fan deltas, expanding the shore).

8. Optionally, generate wider rivers and wider valleys.

9. Bottom-up, perform final terraforming tasks, to make all tiles planned to be water tiles have a suitable slope.

2 Generator details

Here, you can find a detailed look on both how the generator works.

2.1 Height Index

The aim of the height index is being able to quickly find all tiles of a given height. For this purpose, it partitions map into 2x2, 4x4, 8x8, and so on, rectangle. See figure **??**. The height index stores, that the minimum height present in the bottom 4x4 rectangle is 1, and the maximum height is 3. All other marked 4x4 have at most height 1. Thus, if we search for all tiles of height 3, we only have to inspect the bottom one of the blue grids.

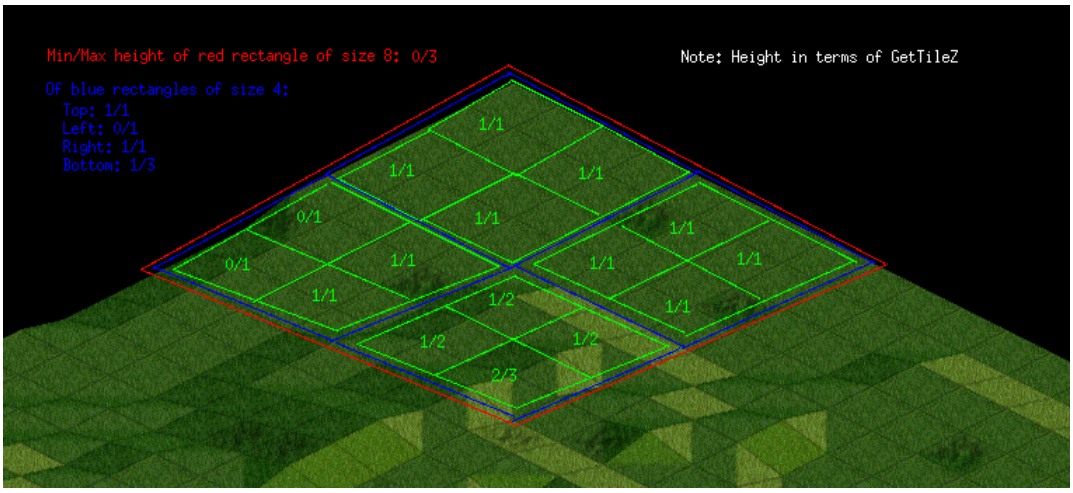

Figure 1: Structure of the height index.

2.2 Number of Lower tiles

For each tile, the generator determine the number of lower tiles. The aim of this number is giving the flow calculation algorithm a clue, where in a huge, flat, valley it must look for the exit into the ocean. Without that number, that decision would have to be a random one, and would often be wrong.

Have a look at screenshot 2.2. We are somewhere in a huge, flat valley. The red numbers of lower tiles that tell us that the ocean (or some lake basin) is somewhere in the south-west, but more than 160 tiles away. Thus, the next algorithm can direct flow to the south-west.

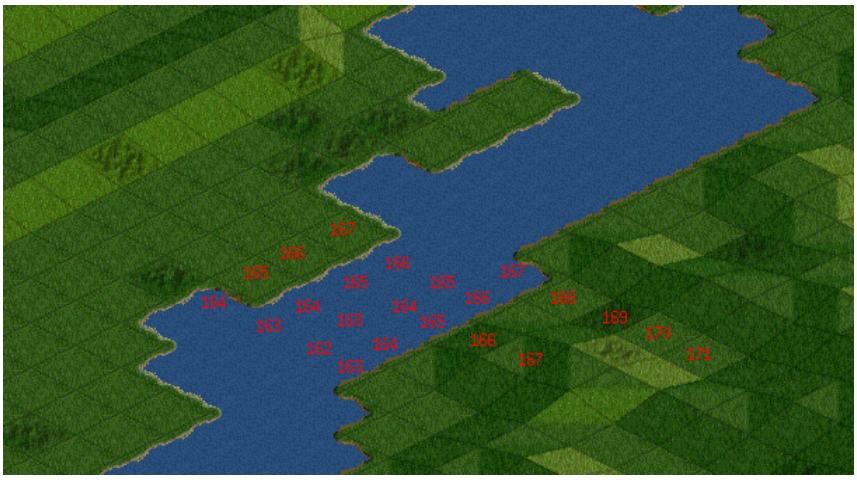

Figure 2: Number of lower tiles

2.3 Flow

Based on the number of lower tiles, flow calculation then calculates both flow amount and flow direction. It assumes that each tile receives one unit of water. Screenshot 2.3 shows a small example: The main flow path in the valley goes from 198 to 199 to 207 (where it gains the small side flow 6) to 209 to 210. The river starts at flow 207, because this is the first time the flow exceeds the minimum flow for a river, which in this example was configured to the value 200.

This essentially means, that for each river starting somewhere, 199 tiles without river exist somewhere above it. 199 because the tile itself also adds one unit of flow. Thus, in the example, 199 plus 1 for the tile between 199 and 6, plus 6 gives 206, plus one for the tile itself gives 207.

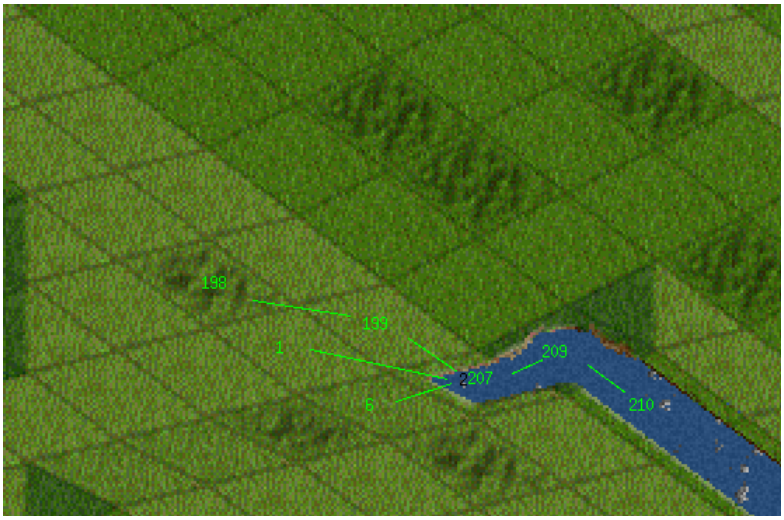

Figure 3: Flow amount and flow directions (from higher to lower numbers)

2.4 Debugging

Those values (number of lower tiles, flow) are calculated for the whole map. To debug what the algorithm does, I temporarily added these values to the landinfo window, where they are available if and only if the map was just generated. This, however, is not meant to be a permanent part of the generator, its just there for debugging purposes.

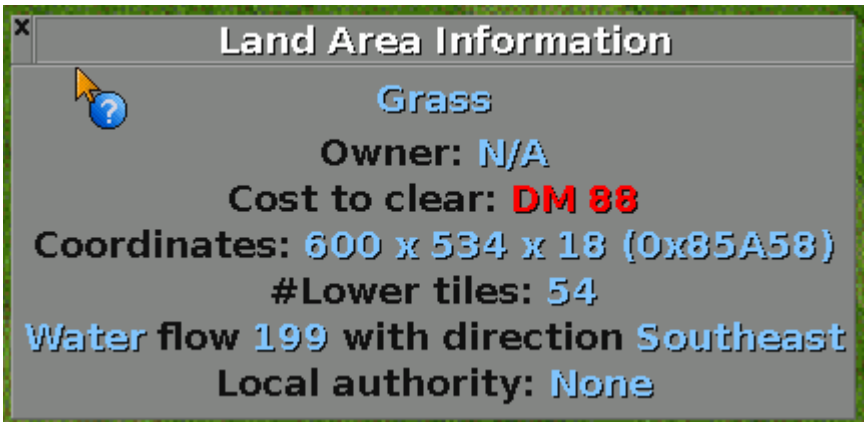

Figure 4: Debugging information in land info window

2.5 Flow Modifications

The flow generated so far leads to quite straight rivers, as you can see in the following screenshot.

Figure 5: Generated Rivers without the flow modification step

The reason for this is, that the number of lower tiles metrics basically consists the distance to water, and that distance tends to increase along a straight line, at least in a flat area.

To improve those rivers, the generator can apply flow modifications on the flow generated above. The number of flow modifications (per 1000 tiles) can be configured. One flow modification works as follow:

1. Find a random tile
2. Follow its flow until it exceeds have the flow necessary for a river
3. Determine an angle based on current flow direction.
4. In phases *Straight*, *Left*, and *Right*, step forward tile by tile, and based on the phase, modify the angle.

5. From time to time, switch phase, with a bias towards Straight, and towards heading to the opposite direction after one of the direction phases was chosen. This is to decrease the probability of ending up completely elsewhere than the original flow.

Obviously, some of the flow modifications will not work, as the above algorithm of course can produce cycles, or end up somewhere were it could only head upwards.

(Nearly) the same area as above looks as follows, if one applies 30 flow modifications per 1000 tiles:

Figure 6: Generated Rivers with 30 flow modifications per 1000 tiles

The straight lines have disappeared, instead the upper river now has a 180 degree turn before proceeding further to the east, just as rivers sometimes have in reality.

2.6 Wider rivers

As described above, a river starts once flow exceeds the configured necessary flow for a river. One can optionally enable wider rivers, and specify a multiplier. Consider for example necessary flow 200, and multiplier 10. Then,

- Rivers with flow between 200 and 1999 have width 1
- Rivers with flow between 2000 and 19999 have width 2
- Rivers with flow between 20000 and 199999 have width 3
- and so on

The result looks as follows:

Figure 7: Wider Rivers

The river at the right, flowing from south to north comes with flow 172000, and is thus 3 tiles wide. The river passing Solfoten in the South flows from

West to East, and has about flow 3000. The river passing Solfoten in the Center has about flow 8000. Thus, those two rivers have width 2. Finally, the river in the North of Solfoten is the smallest one, it just has flow 1500.

Note that here one can see a special artifact of the generator: Crossing rivers. The 8000-river doesn´t flow into the 172000-river right away, but crosses it and flows further to the east. It finally joins the big river about 25 tiles to the east:

Figure 8: Wider Rivers

I don´t regard this a bug, but a feature, since it can give the impression of river deltas and river islands.

2.7 Wider valleys

Valleys in heightmaps tend to be quite small, as the mountain slopes in real world are too steep for OpenTTDs world. In essence, many flat areas where in real world cities are situated are consumed by this.

Screenshot 2.7 shows a valley as it is generated with the raw river generator. The river consumes the whole basis of the valley (in fact, it already lowered some tiles to make space as it is wider than one tile). Space for positioning a city, or a railroad? Hardly any.

Figure 9: Valley generated without wider valleys

Now generate the same, with wider valleys multiplier 5. The mountains around have become somewhat lower, but now, we actually have space in our valley.

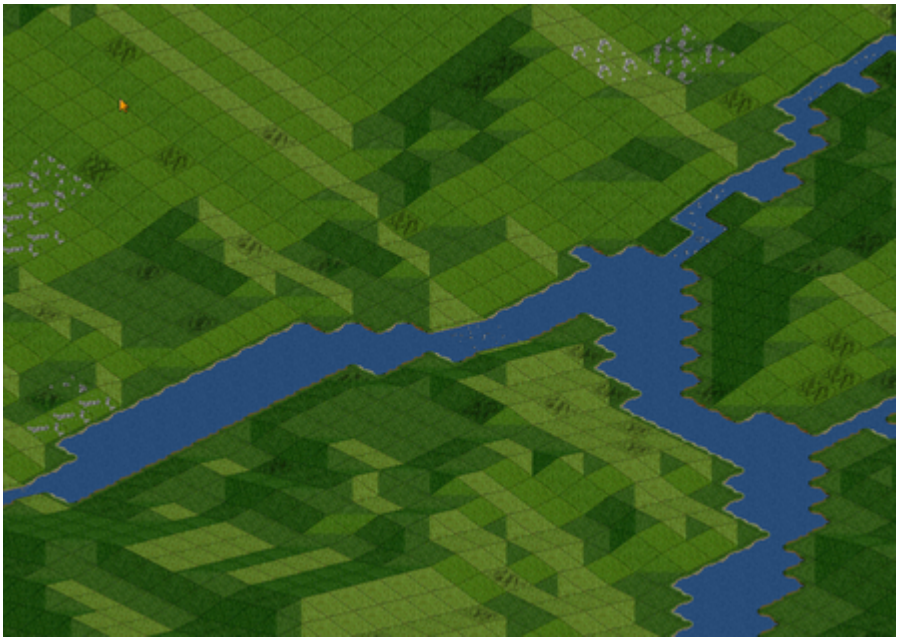

Figure 10: Valley generated with wider valleys enabled

The meaning of the multiplier is: At each side of the river, have a look at a number of tiles equal to the river width multiplied with the multiplier. For example here: 10 tiles as the river is 2 tiles wide and the multiplier is 5. Start at the river, follow a line away from the river. In each step, increase height by one with probability *number of tiles away from river divided by max distance from river*. Thus, tiles near to the river stay low with quite high probability, tiles farer away get somewhat higher, just as in a real world valley.

Note that the rivers paths in the second screenshot look a bit different, as number of lower and flow generation contains some probabilistic elements.

2.8 Lakes

The flow generation algorithm described above just recorded a lake center if it found a lake basin, where it coulnd´t proceed downwards.

Lake calculation then works as follows:

- Start with the inflow of the lake, set the surface height to the height of the lake center.
- Add tiles of equal height as the surface height to the lake.
- For each tile added to the lake, decrement the configured flow needed per lake volume from the available flow.
- If we run out of flow, we stop adding further tiles, the lake stays at it is without outflow.
- If before that happens, a tile is found, whose flow doesn´t end up in the lake, it is an outflow tile. It will receive all the flow of the lake.
- If no further tile of surface height can be added to the lake, and flow isn´t exhausted yet, then the surface height will be raised by one, and the algorithm proceeds with the new surface height.
- Raising the surface height costs number of lake tiles times the flow needed per lake tile volume, i.e. needed flow for a lake is really a three-dimensional concept.
- If during that algorithm, another lake center is found, the respective lake will be consumed, i.e. the two lakes will be merged, and the algorithm will proceed after adding the flow of the consumed lake. As both heightmap loading and tgp produces many lake basins at the small scale (i.e. one tile down, one tile up), this is a quite frequent case.

Propagating the flow of the lake downwards the outflow causes the need to have a look at all lakes downwards (rivers weren´t calculated yet at that step, just flow, i.e. this has no relation to wider rivers). Of course, in a recalculation an already found outflow will be reused.

If, in contrast, no outflow was found yet, the algorithm above will be continued.

In a later step, optionally islands, fan deltas, and additional shore tiles can be added to lakes. In essence, the result may look as follows:

Figure 11: A lake

Note that if you set the flow needed per lake tile volume to zero, then every lake, regardless how deep its basin is, will get an outflow. This can be useful to make the very best of the landscape generated by tgp, i.e. the player now sees a lake, and no longer the ugly inverted pyramid tgp maybe calculated.

Also, one can optionally dig outflow canyons for lakes with some configured probability (i.e. lower the surface height accordingly). Together with the reduction of lakes to their necessary tiles to keep them connected (also configurable) this can help adding some more or less realistic valleys to a landscape generated by tgp.

3 Configuration

The patch changes the map generation and generate from heightmap window as follows:

Figure 12: The adjusted Generate from Heightmap window

Basically, now one can choose the river generator, and optionally configure detail settings for it. More interesting, the options of the Rainfall River Generator look as follows:

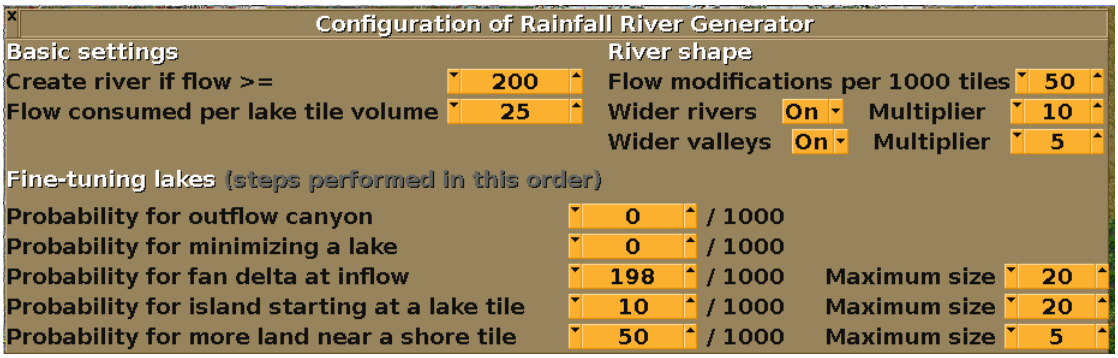

Figure 13: Configuration window of the Rainfall River Generator

Most of the settings where already described before (look into the respective chapters of the manual), thus just some general remarks:

- Probabilities are always given in the range 0 to 1000, to have fine control.
- The maximum sizes in the bottom section refer to the maximum size of e.g. an island that can be generated in a step. A step is one run of the respective algorithm, that will be executed on a tile with the given probability.
- Example: The screenshot tells: If you have an arbitrary lake tile, start an island there with at one percent probability, with maximum size 20.
- The actual size is a random number smaller than 20.
- Note that if the start tile of an island is near the edge of the lake, then it will merge with the shore, maybe forming a peninsula.

Note that before actually calculating islands etc., the calculator will have a look which tiles it needs to keep the lake (i.e. all inflow tiles and the outflow) connected. It will never declare such tiles land tiles.

4 The patch queue

000_Debug Just a small patch for easier debugging. Replaces GetTileZ by GetTileHeight in the land info window, and removes the automatic print-out of land-info everytime the user clicks somewhere (it is really tedious if you inspect a log output, and on each click the log scrolls away...). Not meant to be permanent.

002_RiverGeneratorOO The river generator so far lived in landscape.cpp, together with various other things. This patch refactors river generation to be object oriented, i.e. moves all code of the original generator into a new class named PathRiverGenerator. Can be applied independent of the remaining patch queue.

003_SaveGameBump Introduces a new savegame version. The only savegame relevant change of the patch queue are some new config settings.

005_AdjustMapGeneratorGUI Changes to the genworld_gui, to support choosing from more than one river generator on GUI level. Also adds

a button meant to open a window for river generator related expert option.

007_RiverExpertGUISkeleton A new window (without any content at this point) meant to contain expert options for the rainfall river generator. Content and functionality will be added in later patches.

010_RiverExpertGUIWidgets Widgets (not yet functionality) for the rainfall expert options gui.

011_SkeletonRainfallRiverGenerator A skeleton class for the new Rainfall River Generator, without any functionality yet.

012_ConfigSettings New config settings for configuring the work of the Rainfall River Generator in detail, plus constants defining default values and bounds for those settings.

015_RainfallOptionsRead Populating the expert options window with the respective config values.

016_RainfallOptionsChangeSettings Adding the functionality to actually change the settings to the expert settings window.

017_RainfallOptionsWidgetState Widgets changing state because the current configured state changes.

018_UseDefaultValuesForRiverAmount Translate the general Few / Moderate / Lot of Rivers setting into the config settings world of the Rainfall River Generator.

019_ActivateRainfallRiverGenerator After this patch, the Generate function of the Rainfall River Generator (though at this point still empty) will be actually called if the generator is chosen and landscape will be generated.

020_HeightIndex Add the concept of a HeightIndex for fast access of all tiles of a given height. Without this, nearly ten times the maximum heightlevel, all tiles of the map would have to be scanned. As it is general, not specifially river-related code, it is added to a new file landscape_util.

030_HeightIterator Adds a HeightIterator for iterating over all tiles of a given height to landscape_util.

035_DebugNumberOfLowerTiles Debug code for being able to inspect the result of the next patches in the land info window. Not meant to be added permanently.

037_ConnectedComponentCalculator Adds an API for calculating a connected component of tiles based on some properties.

040_NumberOfLowerTiles For each tile, calculate the length of a reasonable path downwards to a depression (forming a lake lateron) or the sea. Do this bottom-up.

045_DebugWaterFlowInLandInfo Show water flow (which will be calculated by the next patch) in the land info window. Patch is just for debugging and not meant to be permanently.

050_CalculateFlow Based on the measure calculated in patch 040, this patch calculates water flow in a top-down manner. Water flows to lower tiles, and if height is equal, to tiles with a lower number of lower tiles as calculated by patch 040. This way, water will head towards the sea, even if we are in a huge plain area. If flow finds a depression which is not the sea, then this patch just records a lake center and leaves that lake center for the lake generation code in patch 070.

060_TerraformFromMapGen Refactor the well-known terraformer code in terraform_cmd, in order to be able to terraform from landscape generation. This is needed, as in contrast to the landscape generator, the river generator always assumes to operate on a valid landscape (i.e. we cannot just set heightlevels in an arbitrary way). However, sometimes the river generator needs to terraform landscape on the small scale, in order to make room for river beds, to terraform lake surfaces, to dig outflow canyons for lakes.

062_MoveTerraformerStateFunctions Make some terraformer state related functions member functions of struct TerraformerState, for the sake of software design and easier access to those functions from the river generator code.

063_FixShores This patch is a workaround patch: According to my tests, the TerraformTileToSlope function introduced in patch 060 leaves tiles sea even if they are raised to height greater than zero. This then leads to an assertion in drawing code. This patch adds an extra step after the river generator has run, where those tiles are fixed. Hints how this can be done the right way are welcome.

065_LakePathSearch This patch adds the ability to search a path inside a Lake, using Aystar. This is used lateron e.g. when declaring the tiles that are needed to keep the lake connected guaranteed.

070_DefineLakes The flow calculation in patch 050 simply declared a depression a lake center, if it couldn´t find any path further downwards. Here, we define which of these lake centers grow to lakes, and where their water flows to after the lake has been filled up. The logic implemented in this patch is probably the most sophisticated in the whole patch queue, as one has to take care about lakes consuming other lakes, lakes that have a higher surface height than the lake where their water comes from, and so on.

090_TerraformRivers This patch terraforms as many of the river tiles as possible to have a valid slope for a river. A 100 percent success percentage cannot be guaranteed in this step. This is no harm, as a later fine-tuning step does the rest.

100_TerraformLakes Here, lakes are terraformed to their surface height.

105_FineTuneTiles The above terraforming algorithms typically leave some tiles planned as water, but with invalid slope. This is no harm, and perfectly within the concept of those algorithms, but still, it needs to be fixed by terraforming tiles on the small scale. This is the task of this patch.

130_ModifyFlow The flow (and thus the river paths) as calculated by path 050) are quite straight, as they depend on a simple number of lower tiles to water measure. This patch randomly adds some curves to flow while of course keeping the property that water flows downwards.

140_GuaranteedLakeTiles Some of the subsequent patches modify lakes in some way, e.g. by adding islands or peninsulas. This patch declares

lake tiles guaranteed in the sense that they may not be touched by those algorithms. Otherwise, those algorithms may accidentally split up a lake where e.g. a huge river flows through into pieces. I.e. guaranteed lake tiles are a measure to keep a lake connected.

150_LakeModificator Lakes generated so far e.g. have seldom islands, or tend to fill up valleys completely, leaving few space for towns etc. This patch sets up an API for modifying already defined lakes.

160_OnlyGuaranteedLakeModificator This LakeModificator reduces a lake to its guaranteed lake tiles.

170_FanDeltaLakeModificator This LakeModificator adds some sort of a fan delta at some inflow tiles (i.e. where a river flows into a lake)

180_IslandLakeModificator This LakeModificator adds some islands to a lake.

190_ExpandLakeShore This LakeModificator expands the shore of a lake, to make room for other stuff.

200_WiderLakes So far, only lakes of width 1 tile were calculated. This patch adds the ability to generate rivers and guaranteed lake paths wider than one tile, if there is really much flow available. This way, a distinction between small side-rivers and large main-rivers is visible to the player.

210_WiderValleys When generating landscape from heightmaps for mountainious real-world-heightmaps, valleys tend to the very deep and small. The background for this is that the slopes the valleys have in reality are too steep for the dimensions available in OpenTTDs world. If rivers are added to such valleys, there quickly isn´t any space usable for cities, railroads, etc. left. This patch adds the option to automatically make all valleys with a river wider, by lowering tiles in an appropriate manner.

250_GenerationProgress Add steps for the Rainfall River Generator to the world generation progress.