Gunivers-Lib

Gunivers

Mar 03, 2023

INFO


Gunivers.Libs

CHAPTER

ONE

WHAT IS IT?

The Gunivers-libs is a modular library datapack designed to help mapmakers to implement common or complexe systems.

Watch the presentation video: https://www.youtube.com/watch?v=E2nKYEvjETk

CHAPTER

TWO

FEATURES

- Mathematical functions (sin, cos, exp, log, sqrt…)
- ID system for entities
- Block <-> Score conversion system supporting BlockStates
- Pathfinding and other NPC AI basic tools
- Vectors management to give customized trajectories to your entities, which can be deflected by the wind, bounce on blocks etc.
- Entity Link to synchronize the movement of entities, create moving entity cohérent structures or adding mirrors effects

And much more!

CHAPTER

THREE

INSTALLATION

Do not download directly from Github

This repository contain a lot of things that you probably don't need and the datapacks are not compressed. Instead, go on the Glib Manager to select what you want to download.

This project contain a library datapack and a map that is used as a test zone, wich can help new users to handle this tool. Also, the map show some possible features that are made possible thanks to the library datapack.

Install the datapack only

1. Go on Glib Manager.
2. Select the version you want (recommanded to use the last one)
3. On the right panel, select the modules you want to use. You can check the modules' content in the Documentation
4. Click on the "Download" button
5. Put the zip in the "datapacks" folder that you can find in your map folder.
6. In game, use */reload* to load the datapack.

or

Install the sandbox map (contain the datapack)

- A clean installation process is coming soon. For now, you have to download the content of the Github repository and put it in the "map" folder. This repo contain all the modules of the lib so it's heavy. For this reason, we recommand you to go on `play.gunivers.net`, where the map is already hosted.

Be sure that "cheats" are enabled. Or, if you are on a server, be sure that `enable-command-block` is set on `true` in you `server.properties` file.

Some modules can be heavy. Like the `glib.block` or `glib.item` that contain search trees. If you download them, the reload can take some time. If you edit the datapack, be sure to compress it in a zip file before uploading it somewhere. In the other cases, the search trees will slow down the upload due to thousands of little files.

CHAPTER

FOUR

DOCUMENTATION & HELP

The only link you need to know, which contain all the usefull links and informations about the project: glib.gunivers.net.

But to spare you 2 clicks, here is the most important links:

- Documentation (including FAQ)
- Our Discord

CHAPTER

FIVE

MOTIVATION

As developers, we know the importance of using libraries to avoid losing time by re-inventing the wheel in each project. But in Minecraft, we often see that mapmakers are not familiar with this concept.

That's why we created this library, to propose a lot of re-usable tools and try to convice mapmakers to become real developers by looking for and using the available tools.

Thus, this lib is not made to propose the best optimized functions or the more accurate ones. Instead, it is designed to be easy to install and use, and propose various features. We give a huge importance to the accessibility and we recommend talented creators to fork this project in order to make their own optimized versions of the lib.

CHAPTER

SIX

FOLLOW AND/OR CONTRIBUTE

You can come on our Discord server to talk with us and/or take part of the project!

If you want to contribute, please read at least the "Getting started" section in the "Contributing" page that contain all the development convention used in this project.

6.1 Contribute

Pour les francophones

Ce projet est réalisé principalement par des membres francophones mais à destination d'un public international. C'est pourquoi toute la lib ainsi que la documentation est en anglais.

Si vous souhaitez contribuer mais que votre anglais n'est pas au point, vous pouvez utiliser des outils comme DeepL puis demander une vérification.

The Gunivers-Libs is a community project and can therefore be developed by several people at the same time. To keep the project clean and coherent, it is necessary for the different developers to respect certain conventions when developing functions.

To contribute, you must

- Know the structure of a minecraft datapack (5-10 min to learn on internet)
- Know the basic usage of git projects (20-30 min to learn on internet)
- Be accessible to answer questions about your work

6.1.1 Getting started

The only link you need to know : glib.gunivers.net. On this page, you can find:

- The Github repo
- Our Discord server

Definitions

In order to speak with the same vocabulary, here is some words that have a specific meaning in this project:

- **Module**: a group of features that share a common purpose and that are contained in a namespace (i.e. datapack structures)
- **Feature**: something designed for the final user in order to allow him to perform a task. A feature can be a function, a loot_table, a structure, a predicate or whatever.
- **MVP**: for "Minimum Viable Product" correspond to a feature that can not be decomposed in several smaller features. In this lib, all features should be decomposed in MVP, and use theses MVP instead of redefining them.
- **Brigadier**: This is the command completer and highlighter of Minecraft that you can see when you enter a command in the chat. We define it here because most of people doesn't know that this system have a name.

Nomenclature

The Gunivers-Libs respects a certain tree structure which can be similar to the Java packages, called "modules" in this project. The added features must therefore be positioned in these various folders (corresponding to a namespaces) according to their usefulness. If no namespace folder (i.e. module) seems appropriate for the addition of a feature, it can be considered to add a new namespace. A category must respect a particular structure:

- File and folders use the snake_case convention. Example: `my_function`
- Score objectives use the camelCase convention with the `glib.` prefix. Example: `glib.myScore`
- Fake players use the SCREAMING_SNAKE_CASE convention. Exemple: `MY_FAKE_PLAYER`. Add `#` before the name to hide the fake player in the scoreboard when there is no point to let it visible.
- Tags use the UpperCamelCase convention with the `glib.` prefix. Example: `glib.MyTag`
- A feature is equal to an unique utility, so we should not hesitate to decompose its features in MVP in order to make it more readable and to promote the reusability of the MVP. In addition to these few constraints, the contributor is free to organize his files as he wishes as long as it remains coherent and understandable, and it respect the global structure detailed below.
- In some folders are files nammed "_". The main purpose of these files is to reorganize the display of the Gunivers-Libs folders during the auto-completion proposed by Brigadier. Thus, the first proposals are not all the files of a particular folder but the folder itself, followed only by "_" (wich can be easily removed to allow to press 'tab' again and continue to explore the tree structure). These files can be added in all folders, and if possible, they may describe and/or represent, the category in question or a redirection to the section of the documentation related to the category.

Tree structure

The lib must respect this global structure:

```
data
├  <module1>
│  └  functions/predicates/loot_tables/...
│     ├  <feature1>
│     ├  <feature2>
```

(continues on next page)

(continued from previous page)

```
│     ├─ <feature1>
│     └─ <feature2>
├─ <module2>
│  └─ ...
└─ ...
```

Note: Functions, predicates, structures, loot tables etc. must respect the same structure.

Important: Modules can be separated in 2 types:

- library: a set of tools for datapackers and/or builds for builders/terraformers
- system: ready-to-use system, for demos, mini-games or user interface that allow to configure a complex system

The prefix of the namespace must be respectively:

- `glib.<moduleName>`
- `gsys.<moduleName>`

In this structure, you can find a "feature" file and it's associated folder. This "feature" représent a "minimal viable product" (something that have a specific utility but that can require to be a part of another system in order to work)

Features files must respect this structure:

```
<featureName>
<featureName>
    ├─ accuracy
    │  # Same function, but with another precision, ex:
    │  ├─ 10-3.mcfunction
    │  └─ ...
    ├─ child
    │  # sub-functions, not destined to be executed by the user, ex:
    │  ├─ loop.mcfunction
    │  └─ ...
    ├─ config
    │  # mcfunction editable by the user to allwo him to customize the function␣
↪behavior, ex:
    │  ├─ entity_concerned.mcfunction
    │  └─ ...
    └─ debug
       # tools dedicated to debug a system, ex:
       ├─ print.mcfunction
       └─ ...
```

Note: The main file is the only file that is required. Other files/folders depends on the needs of the feature.

Folder	Description
Accuracy	They allow to manage the precision of the functions. Minecraft allowing to store only integers, to use decimals, you have to be clever. Thus, a function in "accuracy/10-3" will be a function which will see its parameters (at least most of them), multiplied by 1000 to be able to store 3 digits after the decimal point (3.14159 * 10^3 = 3141.59, which gives 3141 once in a score). Not all functions have an equivalent in the above specifications. If you need a function with a precision that is not supported, contact us on our Discord, a dev will do that quickly ;)
Child	Child folder contain every function used by other functions to make them works. These functions are not supposed to be executed or edited by the user.
Config	The lib has several systems that manage different behavior (e.g. pathfinding, a bat will not have the same behavior as a villager). You will then find a "main" file that will list the different files and call the right one according to a certain condition. This allows the user of the lib to create his own behavior by copying an existing behavior file, adapting it, and linking it to the system via the "main" file.
Debug	The "debug" folders contain functions that are intended to display a certain number of parameters specific to the folder in which they are located (e.g. debug in the vector folder will display the different vectors). These functions are usually called by other functions but can also be executed by the user in order to debug one of his systems at a specific location.

File format

All the functions of the Gunivers-Lib implement documentation to describe for other developers as well as for users what the function is for and how to use it. This is what this one looks like:

```
#____________________________________________________
# INFO     Copyright © 2021 Gunivers.

# Authors       :
# Contributors  :
# MC Version    :
# Last check    :

# Original path :
# Documentation :
# Note          :

#____________________________________________________
# PARAMETERS

# Input  : Foo (score) : Lorem Ipsum
# Input  : Bar (tag)   : Lorem Ipsum
# Output : Qux (score) : Lorem Ipsum

#____________________________________________________
# INIT

#____________________________________________________
# CONFIG

#____________________________________________________
# CODE
```

We can find various information about the function itself (the example is not exhaustive), of which the following is a complete list:

Field	Description
Authors	The list of authors of the function.
Contributors	The list of contributors to the function. A contributor is someone who helps to create the function without developing it (the one who gives a track to realize the function or the one who fixes a bug for example).
MC Version	Version of Minecraft for which the system was created.
Last check	Version of Minecraft until which the system is certified functional.
Original path	The path to the function so that it can be copied to a /function command.
Documentation	Link to the documentation of the function
Note	Allows you to provide additional information about the function such as a description of what the function does, how to use it if the use is particular, the behaviors of the function or the side effects of its use.

Special scores

This lib use global scores, that are automatically created:

Scores name	Description
`glib`	Dedicated to temporary storage (ex: fake players)
`glib.config`	Allow to define the behavior of some systems
`glib.const`	Contain a huge numer of fakeplayers with constant score value
`glib.data`	Uses by the `glib.core` module. Do not use it without knowing exactly what you are doing.
`glib.debug`	Determine the behavior of debug features
`glib.lifetime`	If positive (default behavior), count the number of tick the entity exist. If negative, determine the number of ticks the entity will live (killed when the score reach -1). This socre is automatically incremented each tick.
`glib.res[0-9]`	Default score for outputs
`glib.var[0-9]`	Default score for inputs

Conservation principle

> "Nothing is lost, nothing is created, everything is transformed" - Antoine Lavoisier

The lib must have a minimum impact on the scores and other data in order to avoid some overwritings of data. It means that we should avoid as much as possible to :

- Create new scores
- Delete scores (totally forbidden)
- Avoid unnecessary scores rewriting

To do so, each input - as well as other data used by the fonction - must be saved in the begining of the function (in a fake player for example) and restored at the end. Only the outputs of the functions should change.

Note: This is a new directive, so most of the functions doesn't respect it for now. Please do not hesistate to update the existing functions in order to apply this directive.

Also for scores, by default, the input and outputs should use respectively the scores `glib.var[0-9]` and `glib.res[0-9]`. But they can use others scores when it's is more appropriate, for exemple to allow writing chains of calling functions like :

```
# Multiply the X coordinate of the source entity and place the entity at the new location
function glib.location:get
scoreboard players operation @s glib.locX *= 2 glib.const
function glib.location:set
```

In this exemple, the `get` function will return `glib.loc[X,Y,Z]` scores, that are also used as input for the `set` function.

6.1.2 Advanced stuff

If you want to deeply take a part of the project, here is more advanced stuff for you!

Welcome in the dark side!

Initialization

In order to make use as easy as possible, each function must limit its dependencies as much as possible. It must then declare each of the variables it uses in the "INIT" part. It is not necessary to initialize variables used by child functions because child functions are supposed to initialize them. On the other hand, it is forbidden to neglect a declaration for any other reason (example: "Var1 is already used everywhere").

Some scores, used by the lib in a global way, do not need to be declared. You can find the list of global scores by here. Also, in order to simplify arithmetical operations, the lib define plenty of constants stored on the score `glib.const`. You can find them here. All constants used in the lib must be defined in this file.

Configuration

Some functions require parameters that are usually fixed. However, the function can manage other parameters and the user, in a particular case, may need to change this parameter. So we call them configuration parameters, which are parameters with a default value. These values are initialized in the "CONFIG" part.

As you can see in several files, some lines in the configuration part call the "glib.config.override" tag. It allows you not to rewrite the score values (or other) if they have been voluntarily set to another value. So, if you want to use something else than the default value for a function, add the "glib.config.override" tag before executing the function, then remove this tag immediately afterwards.

Comments

The development of the Gunivers-Libs is collaborative, which means that other people can read the code. Moreover, the Gunivers-Lib is also meant to be pedagogical and accessible to people curious about the way the functions of the Gunivers-Lib work. Therefore, it is important to make it understood by other developers or users, and this, in addition to the documentation, also goes through the commentary of the code. Thus, it is important to regularly and cleanly comment on functions in order to explain how the function works.

Debug

It is possible to add debug lines anywhere in the code. However, these must be subject to conditions. For the debug to be displayed to a player, that player must have the tags;

- `Glib_Debug`
- `Glib_Debug_<tag_path>`

Where is the path to the function after the namespace, replacing the / with . (tag format requires)

- Example: `glib:entity/vector/get_from_actual_orientation` becomes `entity.vector.get_from_actual_orientation`

Error Messages

Error tellraws must be displayed to all players with the glib.debug tag and must be in this form:

```
tellraw @a[tag=glib.debug] [{"text":"[ERROR] in <PATH>","color":"red"}]
function glib:core/debug/message/error/entity_info
tellraw @a[tag=glib.debug] [{"text":"   <MESSAGE>","color":"red"}]
```

For readability, all lines except this [ERROR] container must have a 3 space indentation.

Lines of code concerning error messages must be preceded by `## Start Error` and followed by `## End Error` in order to be removed by a program.

Debug messages

In the same logic, debug messages must be conditioned to an additional tag linked to the path of the function concerned and must start with:

```
tellraw @a[tag=glib.debug.<TAG_PATH>] [{"text":"> DEBUG | <PATH>","color":"green",
↪"clickEvent":{"action":"run_command","value":"/tag @e remove glib.debug.<TAG_PATH>"},
↪"hoverEvent":{"action":"show_text","value":["",{"text":"Remove this debug"}]}}]

execute as @e[tag=glib.debug.<TAG_PATH>] run function glib:core/debug/message/info/
↪entity_info
```

In order to distinguish between nested function debugs, this debug must be followed by

```
execute if entity @a[tag=Glib_Debug_<TAG_PATH>] run function glib:core/debug/message/
↪info/end_debug
```

Lines of code concerning debug messages should be preceded by `## Start Debug` and followed by `## End Debug` in order to be removed by a program.

Special functions

The “ata” functions

This is a reduction of “as to at”. Several functions require 2 positions to work (example: retrieve the distance between 2 points) or an entity and a position. To simplify the use, no need to pass 3 scores for each position. You will be able to place an entity on point 1 (if it is useful), then execute the function on point 2 by executing it on the entity on point 1.

The “tti” functions

This is a reduction of “to target id”. Several functions require 2 a source and a target entity (example: get a vector to another entity). To simplify the use, no need to always use the `id/check` function.

6.2 FAQ

- **Can I use the lib in my map/server project?**

  Of course you can! The only thing we ask is that you get proper credit. If you are a very nice person, you can even add links to our site and Discord. If you are even nicer, you can also leave us a feedbak or your suggestion to improve the lib. Then you can also make a donation to help us develop other projects but that only happens in my dreams haha… :sob:

- **How did you do some functions?**

  We have set up a dedicated space on this site and on Discord to share the things we do, and especially how we do them so that everyone can learn to do the same. You can find this space here: https://project.gunivers.net/projects/ressources/wiki/Datapacks

6.3 Biome

`glib.biome:` all function concerning biome properties.

6.3.1 Can it rain?

`can_rain`: Determine if it can rain or not.

- Requires that the `glib.temperature` score is defined on the executing entity
- Returns the tag `glib.canRain` if it can rain.

Example:

Knowing if it can rain where the players are

```
# Once
execute as @a run glib.biome:get
execute as @a run glib.biome:get_temperature
execute as @a run glib.biome:can_rain

# See the result
execute as @a[tag=glib.canRain] run say Where I am, it can rain!
execute as @a[tag=!glib.canRain] run say Where I am, it never rains...
```

6.3.2 Can it snow?

`can_snow`: Determine if it can snow or not.

- Requires the `glib.temperature` score to be set on the running entity
- Returns the tag `glib.canSnow` if it can snow.

Example:

Knowing if it can rain where the players are

```
# Once
execute as @a run glib.biome:get
execute as @a run glib.biome:get_temperature
execute as @a run glib.biome:can_snow

# See the result
execute as @a[tag=glib.canSnow] run say Where I am, it can rain!
execute as @a[tag=!glib.canSnow] run say Where I am, it never snows...
```

6.3.3 Get biome ID

`get`: Detects the biome in which the current entity is located and stores it in the `glib.biome` score of the entity (see list of Biomes).

Example:

Get the biome in which each villager is located.

```
# Once
execute as @e[type=villager] run glib.biome:get

# See the result
tellraw @a ["",{"text":"<"},{"selector":"@s"},{"text":"> "},{"text":"Mon biome: ","color
↪":"dark_gray"},{"score":{"name":"@s","objective":"glib.biome"},"color":"gold"}]
```

6.3.4 Get biome temperature

`get_biome_temperature`: Allows to retrieve the temperature of the biome at the execution position of the function.

- The result will be stored on the score `glib.temperature`

Example:

Get the temperature of the biome in which each octopus is located

```
# Once
execute as @e[type=squid] run glib.biome:get_biome_temperature

# See the result
tellraw @a ["",{"text":"<"},{"selector":"@s"},{"text":">"},{"text": "The temperature of␣
↪my biome: ", "color": "dark_gray"},{"score":{"name":"@s", "objective": "glib.biome"},
↪"color": "gold"}]
```

6.3.5 Get block temperature

`get_temperature`: Allows to retrieve the temperature at the execution position of the function taking into account the temperature of the biome and its altitude.

- The result will be stored on the score `glib.temperature`

Example:

Get the temperature at each polar bear

```
# Once
execute as @e[type=polar_bear] run glib.biome:get_temperature

# See the result
tellraw @a ["",{"text":"<"},{"selector":"@s"},{"text":">"},{"text": "The temperature␣
↪where I am: ", "color": "dark_gray"},{"score":{"name":"@s", "objective": "glib.biome"},
↪ "color": "gold"}]
```

6.4 Block

`glib.block:` Systems for manipulating blocks

6.4.1 Get block ID

`get` : Gives the executing entity a score corresponding to a unique identifier of the block + blockstate located where the function is executed.

- The result is stored on the score `glib.blockId`

Example:

Make the player named Steve retrieve the block id under his feet.

```
# Once
execute as Steve at @s positioned ~ ~-1 ~ run function glib.block:get

# See the result
scoreboard objectives setdisplay sidebar glib.blockId
```

6.4.2 Set block from ID

`set`: Places a block (and associated blockstates) corresponding to the identifier stored on the executing entity.

- The identifier must be given via the score `glib.blockId`.

Example:

Make the player named Steve put the block on top of him, corresponding to the identifier he has given on his score `glib.blockId`

```
# Once
execute as Steve at @s positioned ~ ~2 ~ run function glib.block:set

# See the result
# Look above Steve
```

6.4.3 Convert block ID to item ID

`convert_to_item`: Convert a block id stored on the executing entity to an item id.

- The block identifier must be indicated by the `glib.blockId` score
- The item identifier will be stored on the score `glib.itemId`

Example:

Make the player named Steve get the item identifier corresponding to the block that is indicated by his score `glib.blockId`

```
# Once
execute as Steve run function glib.block:convert_to_item

# See the result
scoreboard objectives setdisplay sidebar glib.itemId
```

6.5 Cache

No documentation here...

Note: This is a communautary project. Writing docs can take a lot of time and creators don't ever have this time. You can help us by contributing to the project on the Github repository and by joining us on Discord

6.6 Color

No documentation here...

Note: This is a communautary project. Writing docs can take a lot of time and creators don't ever have this time. You can help us by contributing to the project on the Github repository and by joining us on Discord

6.7 Health

`glib.health:`: Management by scoreboard of the life of an entity.

6.7.1 Safe kill

`safe_kill`: Allows to cleanly delete an entity. This function erases the scores of the entity and teleports it to layer 0 before killing it.

- Players cannot be killed
- Entities with the tag `glib.permanent` cannot be killed
- It is recommended to put the `glib.permanent` tag to all decorative entities (arrays, item frames etc ...)

Example:

Kill all zombies:

```
# Once
execute as @e[type=zombie] run function glib.health:safe_kill
```

6.7.2 Time to live

`time_to_live`: Allows to define the time of life of the entities.

- By default, this time is 10 seconds (= 200 ticks)
- The entity will live before being applied the end of life action (default: function glib.health:safe_kill).
- Before being applied the end of life action, the entity will receive the tag `glib.ttl.timeOut` during 1 tick
- You can change each of the default values by opening the file and going to the "CONFIG" section.

Example:

Give the Creepers a 10 second life time:

```
# At each tick
execute as @e[type=creeper] run function glib.health:time_to_live
```

Give a time to live of 20 seconds to the Cow:

```
# At least once
scoreboard players set @e[type=cow,tag=glib.ttl.default] glib.ttl 400
# At each tick
execute as @e[type=cow] run function glib:time_to_live
```

Give an explosion effect to Creepers at the end of their life

```
# At each tick
execute as @e[type=creeper] run function glib.health:time_to_live
execute as @e[tag=glib.ttl.timeOut] at @s run playsound minecraft:entity.generic.explode␣
↪master @a
execute as @e[tag=glib.ttl.timeOut] at @s run article minecraft:explosion_emitter ~ ~ ~

Warning: if the TTL function is called twice on the same entity, its
lifetime will decrease twice as fast.
```

6.8 ID

`glib.id:` : The management of entity IDs allows to identify precisely an entity. Useful for example in the case of a shooting game to know who is shooting at whom.

6.8.1 Check ID

`check` : Allows to compare the `glib.id` scores of the entities with the `glib.targetId` score of the entity having executed the function.

- The latter receives the tag `glib.id.checker`.
- **The other entities receive the tag**
  - `glib.id.upper` if their `glib.id` is higher than the `glib.targetId`.
  - `glib.id.lower` if their `glib.id` is lower than the `glib.targetId`.
  - `glib.id.match` if their `glib.id` is equal to `glib.targetId`.

Example:

Find the entity (or entities) with ID 3:

```
# Once
scoreboard players set @s glib.targetId 3
function glib.id:check

# See the result
execute unless entity @e[tag=glib.id.match] run tellraw @a [{"text": "No entity found :'(
↪", "color": "dark_gray"}]
execute as @e[tag=glib.id.match] run tellraw @a ["",{"text":"<"},{"selector":"@s"},{"text
↪":">"},{"text": "Hey! Are you looking for me?", "color": "dark_gray"}]
```

6.8.2 Check parent ID

`check_parent` : Compares the `glib.parentId` scores of the entities with the `glib.id.target` score of the entity that executed the function.

- The latter receives the tag `glib.parentId.checker`.
- **The other entities receive the tag**
  - `glib.parentId.upper` if their score `glib.id.parent` is higher than the `id.targetId`
  - `glib.parentId.lower` if their score `glib.id.parent` is lower than the `id.targetId`
  - `glib.parentId.match` if their score `glib.id.parent` is equal to the `id.targetId`

Example:

Find all child entities of Bob:

```
# Once
execute as @e[name=Bob,limit=1] run scoreboard players operation @s glib.targetId = @s␣
↪glib.id
function glib.id:check_parent

# See the result
execute unless entity @e[tag=glib.parentId.match] run tellraw @a [{"text": "No entity␣
↪found :'(", "color": "dark_gray"}]
execute as @e[tag=glib.parentId.match] run tellraw @a ["",{"text":"<"},{"selector":"@s"},
↪{"text":">"},{"text": "Hey! Are you looking for me?", "color": "dark_gray"}]
```

6.8.3 Get simple unique ID

`get_suid` : (Simple Unique ID) Allows the entity executing the function to get a `glib.id` score different from all other entities that have already executed the function.

- Returns the ID on the glib.id score of the executing entity.
- Gives the tag `glib.id.set` and `glib.id.type.suid` to the entities that have already executed the function

Example:

Give an ID to all players:

```
# In a loop to give an ID to the players who connect
execute as @a[tag=!glib.id.set] run function glib.id:get_suid

# See the result
scoreboard objective setdisplay sidebar glib.id
```

6.8.4 Get chain unique ID

`get_cuid` : (Chain Unique ID) Allows the entity running the function to get a score `glib.id` different from all other entities that have already run the function. The difference with `get_suid` is in the way the scores are assigned.

- ID scores are assigned dynamically based on the available scores, so that they form a string. So, if there are 5 entities, they will be numbered from 1 to 5, without any "hole". In order not to break this string, you must also execute the `update_cuid` function in a loop.
- Returns the ID on the `glib.id` score of the executing entity.
- Give the tag `glib.id.set` and `glib.id.type.cuid` to the entities that have already executed the function

Example:

Give an ID to all players:

```
# In a loop to give an ID to the players who connect
execute as @a[tag=!glib.id.set] run function glib.id:get_cuid

# See the result
scoreboard objective setdisplay sidebar glib.id
```

6.8.5 Update chain unique ID

`update_cuid` : Allows to update all the CUID of the entities. Executes globally (the source entity does not matter, executing it multiple times per tick will have no effect)

Example:

Keep the ID string free of holes and duplicates:

```
# To be executed once at the beginning of each tick
function glib.id:update_cuid

# See the result
scoreboard objective setdisplay sidebar glib.id
```

6.9 Item

`glib.item:` : Systems allowing to manipulate items

6.9.1 Get item ID

`get`: Gives to the item executing the function a score corresponding to a unique identifier corresponding to its nature.

- The result is stored on the score `glib.itemId`
- Must be executed on an item only

Example:

Make sure that all the items in the world have their identifier indicated by their score `glib.itemId`

```
# Once
execute as @e[type=item] run function glib:item/get

# See the result
scoreboard objectives setdisplay sidebar glib.itemId
```

6.9.2 Set item from ID

`set`: Create an item corresponding to the identifier stored on the executing entity.

- The identifier must be given via the score `glib.itemId`.

Example:

Make the player named Steve create an item in front of him, corresponding to the identifier he has given on his score `glib.itemId`

```
# Once
execute as Steve at @s anchored eyes positioned ^ ^ ^1 run function glib:item/set

# See the result
# Look in front of Steve
```

6.9.3 Convert item ID to block ID

`convert_to_block`: Convert an item id stored on the executing entity to a block id

- The item identifier must be indicated by the `glib.itemId` score
- The block identifier will be stored on the score `glib.blockId`

Example:

Make the player named Steve get the block identifier corresponding to the item that is indicated by his score `glib.itemId`

```
# Once
execute as Steve run function glib:item/convert_to_block

# See the result
scoreboard objectives setdisplay sidebar glib.blockId
```

6.10 Link

`glib.link:`: The "Link" functions allow to link an entity to another. This link consists in preserving the position and the relative orentation between the two entities, allowing then to imitate or to reverse the movements and rotations of the parent entity.

6.10.1 Create link to target ID

`create_link_tti`: Allows to create the link between two entities.

- The `glib.targetId` score of the executing entity must match the `glib.id` score of the entity to which it will be linked.
- Multiple entities can be linked to a single entity (generally recommended for armor_stand structures).
- The child entity (having performed the function) will then have 9 distinct scores:
  - `glib.link.r[x,y,z,h,v]` representing the relative coordinates (position + orientation)
  - `glib.link.l[x,y,z]` representing local coordinates (position only)
  - `glib.link.to` identifies the entity to which it is linked
- These scores should generally not be modified because they are used as parameters for other link functions.

Example:

- Link all armor_stand to the entity with ID 3

```
# Once
scorebaord players set @e[type=armor_stand] glib.targetId 3
execute as @e[type=armor_stand] run function glib.link:create_link_tti

# See the result
# In loop
execute as @e[type=armor_stand] run function glib_debug:link/display_link
```

6.10.2 Create link “as to at”

`create_link_ata`: In the same way as `create_link_to_target_id`, this function creates a link between the entity executing the function and the entity closest to the execution position.

- Multiple entities can be linked to a single entity (generally recommended for armor_stand structures).
- The child entity (having executed the function) will then have 9 distinct scores:
  - `glib.link.r[x,y,z,h,v]` representing the relative coordinates (position + orientation)
  - `glib.link.l[x,y,z]` representing local coordinates (position only)
  - `glib.link.to` identifies the entity to which it is linked
- These scores should generally not be modified because they are used as parameters for other link functions.

Example:

- Link all armor_stand to the nearest sheep

```
# Once
execute as @e[type=armor_stand] at @e[type=sheep,limit=1,sort=nearest] run function␣
↪glib.link:create_link_ata

# See the result
# In loop
execute as @e[type=armor_stand] run function glib_debug:link/display_link
```

Imitate location

`imitate_loc`: Allows to replace the entity at its relative position. This operation repeated in a loop is to imitate the movements of the parent entity.

- This function also has declinations on x, y and z to allow you to imitate the movements on one or two chosen axes.

Example:

- Make armor_stands mimic your moves

```
# Once
execute as @e[type=armor_stand] at @s run function glib.link:create_link_ata

# In a loop
execute as @e[type=armor_stand,tag=glib.linked] run function glib.link:imitate_loc
```

6.10.3 Imitate orientation

- `immitate_ori`: Allows to replace the entity to its relative orientation. This operation repeated in a loop is to imitate the rotations of the parent entity.
- This function also has variations on h and v to allow you to mimic only the horizontal or vertical rotation.

Example:

- Make armor_stands mimic your orientation changes

```
# Once
execute as @e[type=armor_stand] at @s run function glib.link:create_link_ata

# In a loop
execute as @e[type=armor_stand,tag=glib.linked] run function glib.link:imitate_ori
```

6.10.4 Keep local location

`keep_local_location`: Allows to keep the local position corresponding to the position of the child entity in the repository of the parent entity.

- This reference frame, unlike the relative coordinates, takes into account the orientation of the entity. Thus, when the parent entity turns on itself, the child entity will turn around it keeping its distance and the angle formed between the direction of the parent entity's look and the parent->child vector.

Example:

- Make the armor_stands lock to your orientation

```
# Once
execute as @e[type=armor_stand] at @s run function glib.link:create_link_ata

# In a loop
execute as @e[type=armor_stand,tag=glib.linked] run function glib.link:keep_local_
↪location
```

6.10.5 Reverse location

`reverse_loc`: Allows to determine the displacement made by the parent entity, and reproduce it in the opposite direction.

- This function also has declinations on x, y and z to allow you to reverse the movements that on one or two axes chosen.

Example:

- Make the armor_stands do the opposite of your moves

```
# Once
execute as @e[type=armor_stand] at @s run function glib.link:create_link_ata

# In a loop
execute as @e[type=armor_stand,tag=glib.linked] run function glib.link:reverse_loc
```

6.10.6 Reverse orientation

`reverse_ori`: Allows you to determine the rotation performed by the parent entity, and reproduce it in the opposite direction.

- This function also has variations on h and v to allow you to reverse only the horizontal or vertical rotation.

Example:

- Make the armor_stands mimic your movements

```
# Once
execute as @e[type=armor_stand] at @s run function glib.link:create_link_ata

# In a loop
execute as @e[type=armor_stand,tag=glib.linked] run function glib.link:reverse_ori
```

6.10.7 Update link

`update_link`: This function allows to update the link between entities. If you only use immitation and/or local position keeping functions, this function will not be of any use to you. On the other hand, if you change the position of the child entity automatically, you will have to update the link so that your operation is not cancelled the next time you call the link function.

- The link functions of the lib automatically call the update functions if necessary (example: reverse functions). No need to manage this on your side.

Example:

- Update the armor_stands link

```
# Once
execute as @e[type=armor_stand] run function glib.link:update_link

# See the result
# In a loop
execute as @e[tag=glib.linked] run function glib_debug:link/display_link
```

6.11 Location

`glib.location:`: The "Location" functions allow to manage the position of entities via scores. It is thus possible to detect the position of an entity or to place it at a position defined by a score.

6.11.1 Add up coordinates

`add`: Adds the position passed via the scores `glib.loc[X,Y,Z]` to the one where the command was executed, then teleports the entity to this new position.

Example:

Move Aypierre by 3 blocks on the X axis, -2 on the Y axis and 5 on the Z axis

```
scoreboard players set Aypierre glib.locX 3
scoreboard players set Aypierre glib.locY -2
scoreboard players set Aypierre glib.locZ 5
execute as Aypierre at @s run function glib.location:add
```

`fast_set`: Changes the position of the executing entity to the X,Y and Z coordinates respectively indicated by the scores `glib.loc[X,Y,Z]`. To the user, this function is used in the same way as the `set` function and produces the same results. The differences are:

- This function goes through a succession of teleports and not via NBT modification, which makes it more cumbersome to execute
- It avoids the problem of latency in displaying the position of entities after modifications of their NBT (especially when there is a large number of entities).
- The system is limited to positions between -32000 and +32000 on each axis.

Example:

Place Boblennon at coordinate -5 63 26 (absurd case because the position is hard-coded, so a simple /tp would suffice, but here the scores can be modified unlike the parameters of a /tp command)

```
scoreboard players set Boblennon glib.locX -5
scoreboard players set Boblennon glib.locY 63
scoreboard players set Boblennon glib.locZ 26
execute as Boblennon run function glib.location:fast_set
```

6.11.2 Get location

`get` : Detect the position of the entity (coordinates)

- Stores the values on the scores `glib.loc[X,Y,Z]` with a precision of 1:1.

Example:

Detect and display the position of the nearest spider:

```
# Once
execute as @e[type=spider,limit=1,sort=nearest] run function glib.location:get
tellraw @a [{"text": "X = ", "color": "dark_gray"},{"score":{"name":"@e[type=spider,
↪limit=1,sort=nearest]", "objective": "glib. locX"}, "color": "gold"},{"text":", Y = ",
↪"color": "dark_gray"},{"score":{"name":"@e[type=spider,limit=1,sort=nearest]",
↪"objective": "glib. locY"}, "color": "gold"}{"text":", Z = ", "color": "dark_gray"},{
↪"score":{"name":"@e[type=spider,limit=1,sort=nearest]", "objective": "glib.locZ"},
↪"color": "gold"}]
```

6.11.3 Get distance “as to at”

`get_distance_ata` : Calculates the distance between the source entity and the execution position of the function.

- The result is returned on the score `glib.res0`.
- – Be careful, this function calls `get_distance_squared_ata`, on which it applies the math/sqrt operation. It is therefore relatively heavy and is subject to the same constraint as `get_distance_squared_as_to_at` on integer size.

Example:

Calculate the distance between you and the nearest sheep:

```
# Once
execute as @s at @e[type=sheep,limit=1,sort=nearest] run function glib.location:get_
↪distance_ata
tellraw @a [{"text": "Distance: ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```

6.11.4 Get distance squared “as to at”

`get_distance_squared_ata` : Calculates the squared distance between the source entity and the execution position of the function.

- The result is returned on the score `glib.res0`.

Warning: The scores on Minecraft represent the size of an int variable in java. The latter is huge, but not unlimited. However, calculations involving powers give results that can quickly rise to more than billions, exceeding the size limit of the variable. The game will then have no choice but to “loop” the value (if you exceed the limit of 1, the variable will go negative).

Example:

Calculate the squared distance between you and the nearest sheep:

```
# Once
execute as @s at @e[type=sheep,limit=1,sort=nearest] run function glib.location:get_
↪distance_squared_ata
tellraw @a [{"text": "Distance^2 : ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```

6.11.5 Get relative corrdinates “as to at”

`get_relative_ata` : Allows to obtain the position of the source entity, relative to the execution position of the function.

- The result is then placed on the scores `glib.loc[X,Y,Z]`.

Example:

Get your position relative to the nearest Creeper:

```
# Once
execute as @s at @e[type=creeper,limit=1,sort=nearest] run function glib.location:get_
↪relative_ata
tellraw @a [{"text": "Relative position : X=", "color": "dark_gray"},{"score":{"name":"@s
↪", "objective": "glib.locX"}, "color": "gold"},{"text":", Y=", "color": "dark_gray"},{
↪"score":{"name":"@s", "objective": "glib. locY"},"color":"gold"},{"text":", Z=","color
↪":"dark_gray"},{"score":{"name":"@s","objective":"glib.locZ"},"color":"gold"}]
```

6.11.6 Is in cave?

`is_in_cave`: Allows to know if the location indicated by the execution mosition of the function is located in a cellar.

- Stores the result on glib.res0 (1 if in a cellar, 0 otherwise)

Example:

To know if the skeletons are in cellars or not:

```
# Once
execute as @e[type=skeleton] at @s run function glib.location:is_in_cave

# See the result:
effect give @e[type=skeleton,scores={glib.res0=1}] glowing 1 1 true
```

6.11.7 Set location

`set`: Allows to place the entity at a precise coordinate given via the scores `glib.loc[X,Y,Z]`.

- This function has variations on x, y and z, useful for players, for whom the position can not be changed directly via the /data command.

Example:

Teleport in 15 100 25

```
# Once
scoreboard players set @s glib.locX 15
scoreboard players set @s glib.locY 100
scoreboard players set @s glib.locZ 25
function glib.location:set
```

6.11.8 Spread entity

`spread` : Allows to randomly teleport an entity in a given area.

- The difference with the spreadplayers command is that this function does not teleport to the highest block, it simply does not change the Y position of the entity
- Takes as parameters the scores `glib.var[0,1,2]` corresponding respectively to the X and Z coordinates, as well as to the radius of the area in which the entity will be teleported.

Example:

Teleport to an area with a radius of 10 blocks, having as its center the coordinate X=15, Z=25

```
# Once
scoreboard players set @s glib.var0 15
scoreboard players set @s glib.var1 25
scoreboard players set @s glib.var2 10
function glib.location:spread
```

6.12 MapEdit

No documentation here…

Note: This is a communautary project. Writing docs can take a lot of time and creators don't ever have this time. You can help us by contributing to the project on the Github repository and by joining us on Discord

---

6.13 Math

`glib.math:`: The "Math" functions, as their name suggests, are for doing math. Before you run away remembering your indigestible and incomprehensible lessons that you had to endure at school, you should know that here, you won't need to do any math (in fact, it's the purpose of the lib to simplify your life). Nevertheless, the following math functions are theoretical, but they are also what most other systems are based on. If you don't know what I mean, you can easily create (among other things) raycasting with this lib. This system is essentially based on trigonometry. But again, nothing complicated, everything is already done ;)

6.13.1 Algebra

`glib.math:algebra`: this folder allows you to perform algebra operations

Basic rotation

`basis_rotation_3d`: Allows to obtain the equivalent of the vector passed in parameter in a base with a different orientation. Useful to convert an absolute/relative position into a local position for a given entity.

- Takes in parameter the scores `glib.var[0,1,2]` corresponding to the X, Y and Z compositors of the vector in the starting base
- Takes as parameter the scores `glib.var[3,4]` corresponding to the difference in orientation of the bases, respectively horizontal (Phi) and vertical (Theta)
- Returns the X', Y' and Z' components respectively on the scores `glib.res[0,1,2]`

Examples:

- A block is in ~2 ~5 ~10 from me, I want to have this position in local coordinate (^? ^? ^?)

```
# One time

# Relative coordinates (we multiply by 1000 to have more precision on the result,␣
↪which will also be multiplied by 1000)
scoreboard players set @s glib.var0 2000
scoreboard players set @s glib.var1 5000
scoreboard players set @s glib.var2 10000

# Difference between my orientation (= that of the coondata grid ^X ^Y ^Z) and the␣
↪orientation of the Minecraft blocks grid (~X ~Y ~Z)
function glib.orientation:get
scoreboard players operation @s glib.var3 = @s glib.oriH
scoreboard players operation @s glib.var4 = @s glib.oriV

# Perform the basic rotation
function glib.math:algebra/basis_rotation_3d

# See the result
tellraw @a [{"text": "X = ", "color": "dark_gray"},{"score":{"name":"@s", "objective
↪": "glib.res0"}, "color": "gold"},{"text":", Y = ", "color": "dark_gray"},{"score
↪":{"name":"@s", "objective": "glib. res1"},"color":"gold"},{"text":", Z = ","color
↪":"dark_gray"},{"score":{"name":"@s","objective":"glib.res2"},"color":"gold"}]
```

- I want to have a vector pointing to where I'm looking, but in relative coordinates ~X ~Y ~Z (also called "classical" vector in this library)

```
# Once

# Retrieve a vector ^ ^ ^1 corresponding to a vector directed according to the␣
↪orientation of the entity (we multiply by 1000 to have more precision on the␣
↪result, which will also be multiplied by 1000)
scoreboard players set @s glib.var0 0
scoreboard players set @s glib.var1 0
scoreboard players set @s glib.var2 1000

# Get the orientation
function glib.orientation:get
scoreboard players operation @s glib.var3 = @s glib.oriH
scoreboard players operation @s glib.var4 = @s glib.oriV

# Reversal of the orientation since we want to have the relative orientation of the␣
↪game grid compared to the orientation of the player (unlike the previous example)
scoreboard players operation @s glib.var3 *= -1 glib.const
scoreboard players operation @s glib.var4 *= -1 glib.const

# Perform the basic rotation
function glib.math:algebra/basis_rotation_3d

# See the result
tellraw @a [{"text": "X = ", "color": "dark_gray"},{"score":{"name":"@s", "objective
↪": "glib.res0"}, "color": "gold"},{"text":", Y = ", "color": "dark_gray"},{"score
↪":{"name":"@s", "objective": "glib. res1"},"color":"gold"},{"text":", Z = ","color
↪":"dark_gray"},{"score":{"name":"@s","objective":"glib.res2"},"color":"gold"}]
```

6.13.2 Bitwise

`glib.math:bitwise`: This folder contains various bitwise operators to apply to scores.

logical AND

and: Computes the bitwise conjunction of the two input numbers

- Takes the scores `glib.var0` and `glib.var1` as parameters
- Returns the value of the operation `glib.var0 & glib.var1` on the score `glib.res0`.
- If one of the inputs is negative, the operation will be done between the first operand and the two's complement of the second

Example:

- Calculate and display -9 & 57

```
# Once
scoreboard players set @s glib.var0 -9
scoreboard players set @s glib.var1 57
function glib.math:bitwise/and
tellraw @a [{"text":"-9 & 57 = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```

Get number of bits

`get_number_of_bits`: Calculates the number of bits needed to store the input

- Takes the score `glib.var0` as parameter
- Returns the number of bits needed to store the input
- If the input is negative, returns the number of bits needed to store the absolute value of the number

Example:

- Calculate and display the number of bits of 12

```
# Once
scoreboard players set @s glib.var0 12
function glib.math:bitwise/get_number_of_bits
tellraw @a [{"text": "Number of bits of 12 = ", "color": "dark_gray"},{"score":{
↪"name":"@s", "objective": "glib.res0"}, "color": "gold"}]
```

logical NOT

`not`: Computes the bit by bit negation of the input

- Takes the score `glib.var0` as parameter
- Returns the value of the operation `~glib.var0` on the score `glib.res0`.

Example:

- Calculate and display ~452

```
# Once
scoreboard players set @s glib.var0 452
function glib.math:bitwise/not
tellraw @a [{"text":"~452 = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```

OR logic

`or`: Computes the bit to bit disjunction of the two input numbers

- Takes as parameters the scores `glib.var0` and `glib.var1`.
- Returns the value of the operation `glib.var0 | glib.var1` on the score `glib.res0`.
- If one of the inputs is negative, the operation will be done between the first operand and the two's complement of the second

Example:

- Calculate and display -9 | 57.

```
# Once
scoreboard players set @s glib.var0 -9
scoreboard players set @s glib.var1 57
function glib.math:bitwise/gold
tellraw @a [{"text":"-9 | 57 = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```

Complement to 2

`two_complement`: Computes the two's complement of the input

- Takes the score `glib.var0` as parameter
- Returns the two's complement of `glib.var0` over the score `glib.res0`.

Example:

- Calculate and display the two's complement of 12

```
# Once
scoreboard players set @s glib.var0 12
function glib.math:bitwise/to_complement
tellraw @a [{"text": "Two's complement of 12 = ", "color": "dark_gray"},{"score":{
↪"name":"@s", "objective": "glib.res0"}, "color": "gold"}]
```

OR exclusive

`xor`: Computes the exclusive bit by bit disjunction of the two input numbers

- Takes as parameters the scores `glib.var0` and `glib.var1`.
- Returns the value of the operation `glib.var0 ^ glib.var1` on the score `glib.res0`
- If one of the inputs is negative, the operation will be done between the first operand and the two's complement of the second

Example:

- Calculate and display -9 ^ 57

```
# Once
scoreboard players set @s glib.var0 -9
scoreboard players set @s glib.var1 57
function glib.math:bitwise/xor
tellraw @a [{"text":"-9 ^ 57 = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```

6.13.3 Common

`glib.math:common/`: this folder contains the usual math functions

Rounded division

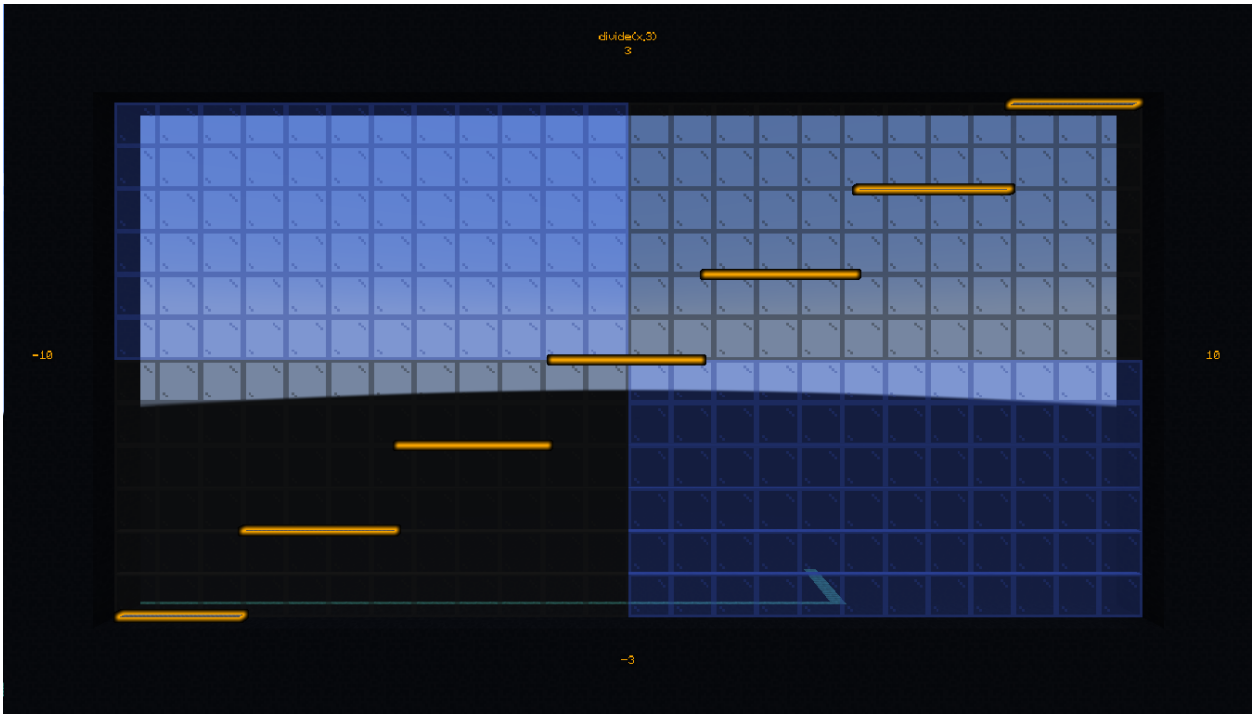
`divide`: Allows you to divide one number by another by rounding the result to the nearest whole number (where Minecraft rounds down to the next whole number).

- Takes as input the scores `glib.var0` and `glib.var1`
- Returns the result on the score `glib.res0`

Example:

- Calculate 9 / 5:

```
# Once
scoreboard players set @s glib.var0 9
scoreboard players set @s glib.var1 5
function glib.math:common/divide
tellraw @a [{"text": "9 / 5 = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```



Exponential

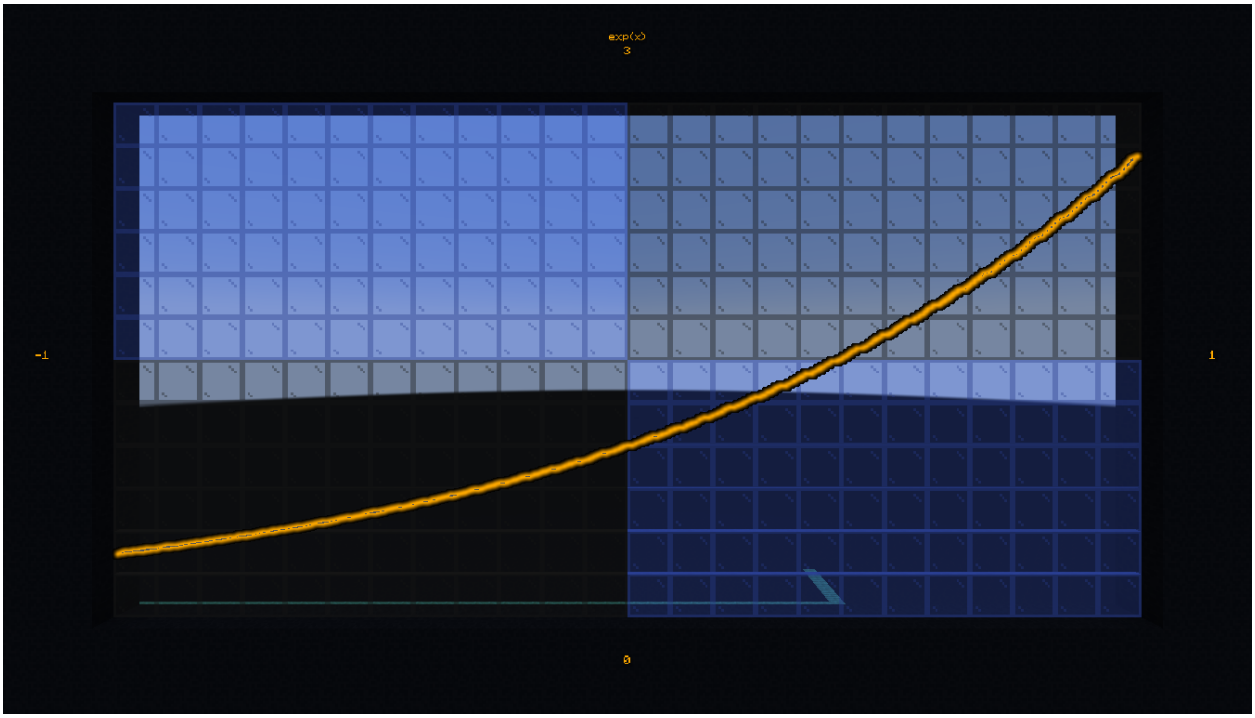
`exp`: Compute the exponential of the number passed in parameter on the score `glib.var0` and return the result on the score `glib.res0`

- In order to take into account a certain number of decimals, `glib.var0` must be multiplied by 100 and `glib.res0` is multiplied by 1000
- Due to technical constraints, this system is limited to a glib.var0 within an interval of `[-6000,12000]` (i.e. `[-6;12]` in real value)

Example:

- Calculate exp(3):

```
# Once
scoreboard players set @s glib.var0 300
function glib.math:common/exp
tellraw @a [{"text":"exp(3)*10^3 = ","color":"dark_gray"},{"score":{"name":"@s",
↪"objective":"glib.res0"},"color":"gold"}]
```



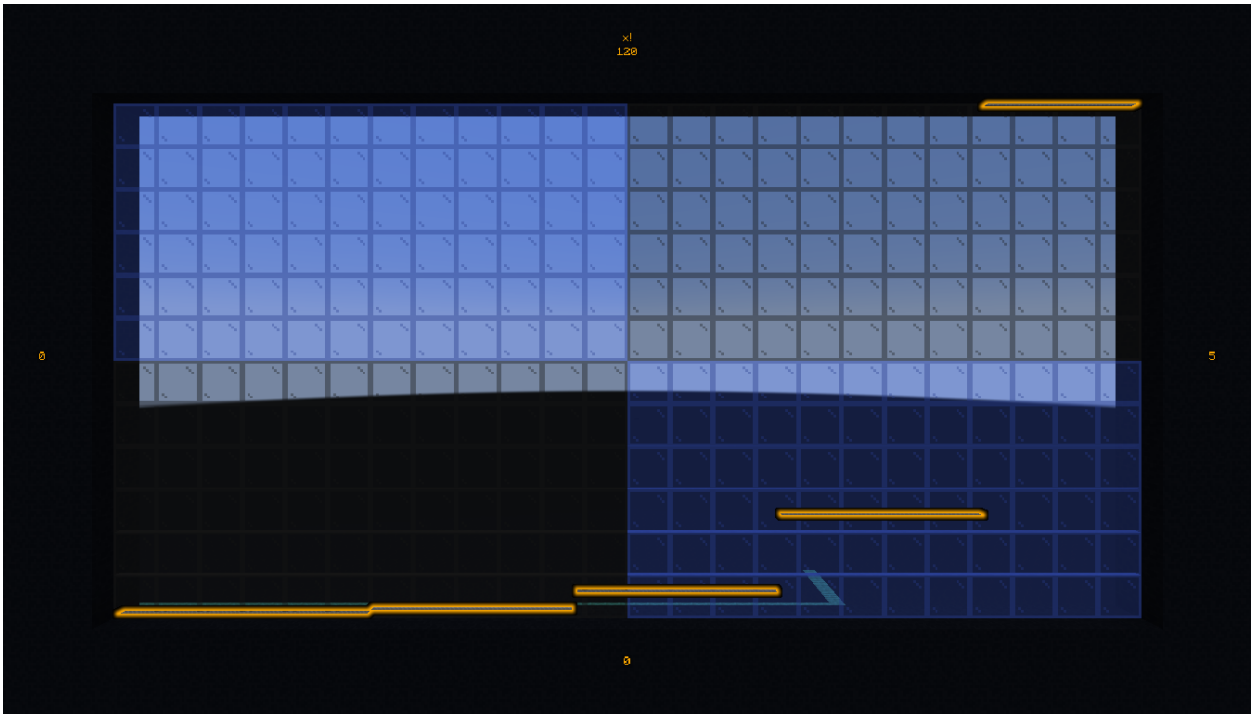
Factorial

`factorial`: Compute the factorial of the number passed in parameter on the score `glib.var0` and return the result on the score `glib.res0`.

Example:

- Compute 3!

```
# Once
scoreboard players set @s glib.var0 3
function glib.math:common/factorial
tellraw @a [{"text": "3! = ","color":"dark_gray"},{"score":{"name":"@s","objective":
↪"glib.res0"},"color":"gold"}]
```



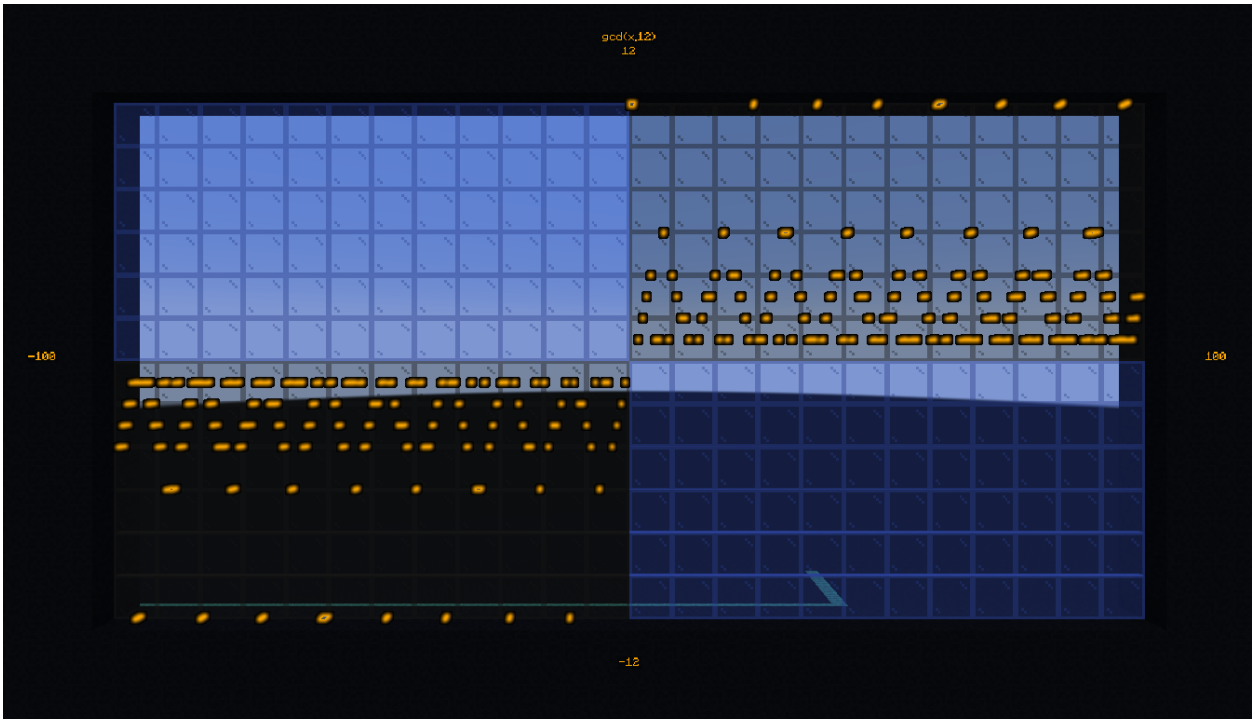
Greatest common denominator

`gcd`: Compute the greatest common denominator of the two numbers passed in parameter on the scores `glib.var0` and `glib.var1` then return the result on the score `glib.res0`.

Example:

- Calculate the greatest common denominator between 16 and 12 :

```
# Once
scoreboard players set @s glib.var0 16
scoreboard players set @s glib.var1 12
function glib.math:common/gcd
tellraw @a [{"text": "gcd(16,12) = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```



Neperian logarithm

`log`: Compute the Neperian logarithm (base e) of the number passed in parameter on the score `glib.var0` and return the result on the score `glib.res0`.

- For precision, the parameters of the function and the returned value are multiplied by 1000 in order to store 3 decimals

Example:

- Calculate ln(28):

```
# Once
scoreboard players set @s glib.var0 28000
function glib.math:common/log
tellraw @a [{"text":"ln(28)*10^3 = ","color":"dark_gray"},{"score":{"name":"@s",
↪"objective":"glib.res0"},"color":"gold"}]
```

Logarithm in base 2

`log2`: Compute the logarithm in base 2 of the number passed in parameter on the score `glib.var0` and return the result on the score `glib.res0`.

- For precision, the parameters of the function and the returned value are multiplied by 1000 in order to store 3 decimals

Example:

- Calculate log2(28):

```
# Once
scoreboard players set @s glib.var0 28000
function glib.math:common/log2
tellraw @a [{"text":"log2(28)*10^3 = ","color":"dark_gray"},{"score":{"name":"@s",
↪"objective":"glib.res0"},"color":"gold"}]
```

Logarithm in base 10

`log10`: Compute the logarithm in base 10 of the number passed in parameter on the score `glib.var0` and return the result on the score `glib.res0`.

- For precision, the parameters of the function and the returned value are multiplied by 1000 in order to store 3 decimals

Example:

- Calculate log10(28):

```
# Once
scoreboard players set @s glib.var0 28000
function glib.math:common/log10
tellraw @a [{"text":"log10(28)*10^3 = ","color":"dark_gray"},{"score":{"name":"@s",
↪"objective":"glib.res0"},"color":"gold"}]
```

Logarithm in base a

`loga`: Computes the logarithm of the number passed in parameter on the score `glib.var0` using as base the name passed in parameter on the score `glib.var1` and returns the result on the score `glib.res0`

- For precision, the parameters of the function and the returned value are multiplied by 1000 in order to store 3 decimals

Example:

- Calculate log4(28):

```
# Once
scoreboard players set @s glib.var0 28000
scoreboard players set @s glib.var1 4
function glib.math:common/loga
tellraw @a [{"text":"log4(28)*10^3 = ","color":"dark_gray"},{"score":{"name":"@s",
↪"objective":"glib.res0"},"color":"gold"}]
```

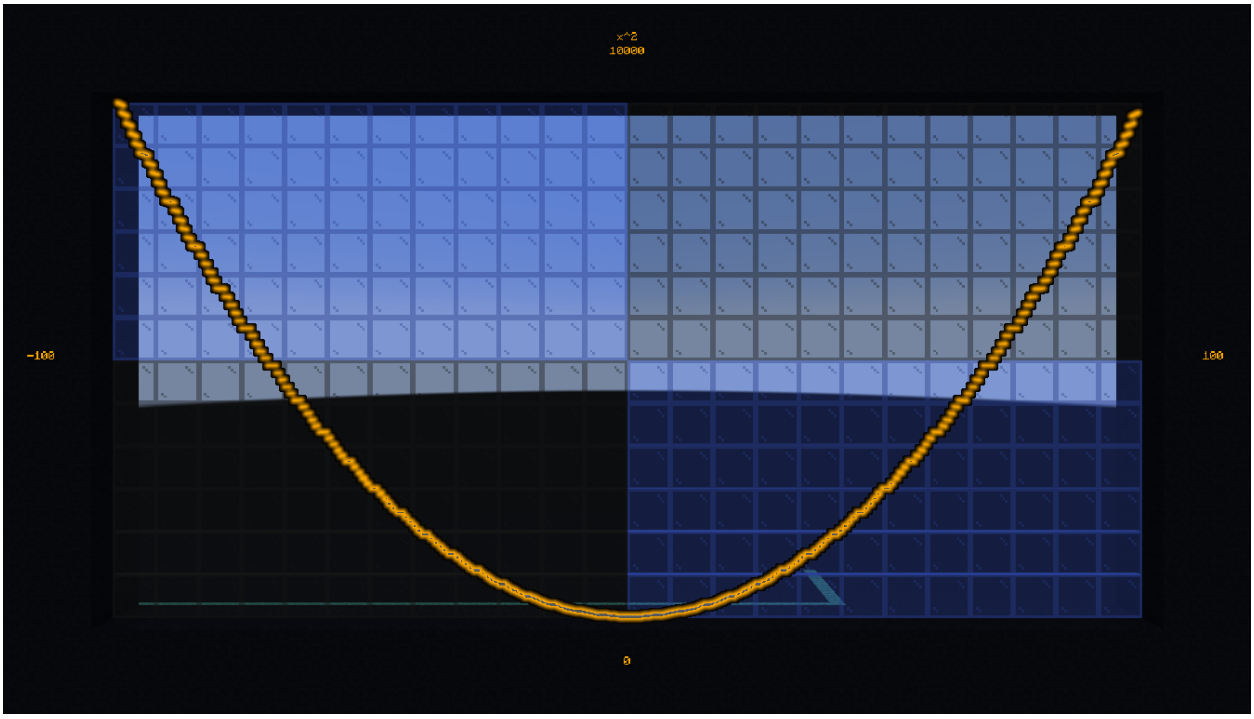
Power

`pow`: Compute the product of the number passed in parameter on the score `glib.var0` raised to the power of the number passed in parameter on the score `glib.var1`, then return the result on the score `glib.res0`

Example:

- Compute 2^6:

```
# Once
scoreboard players set @s glib.var0 2
scoreboard players set @s glib.var1 6
function glib.math:common/pow
tellraw @a [{"text": "2^6 = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```



Square root

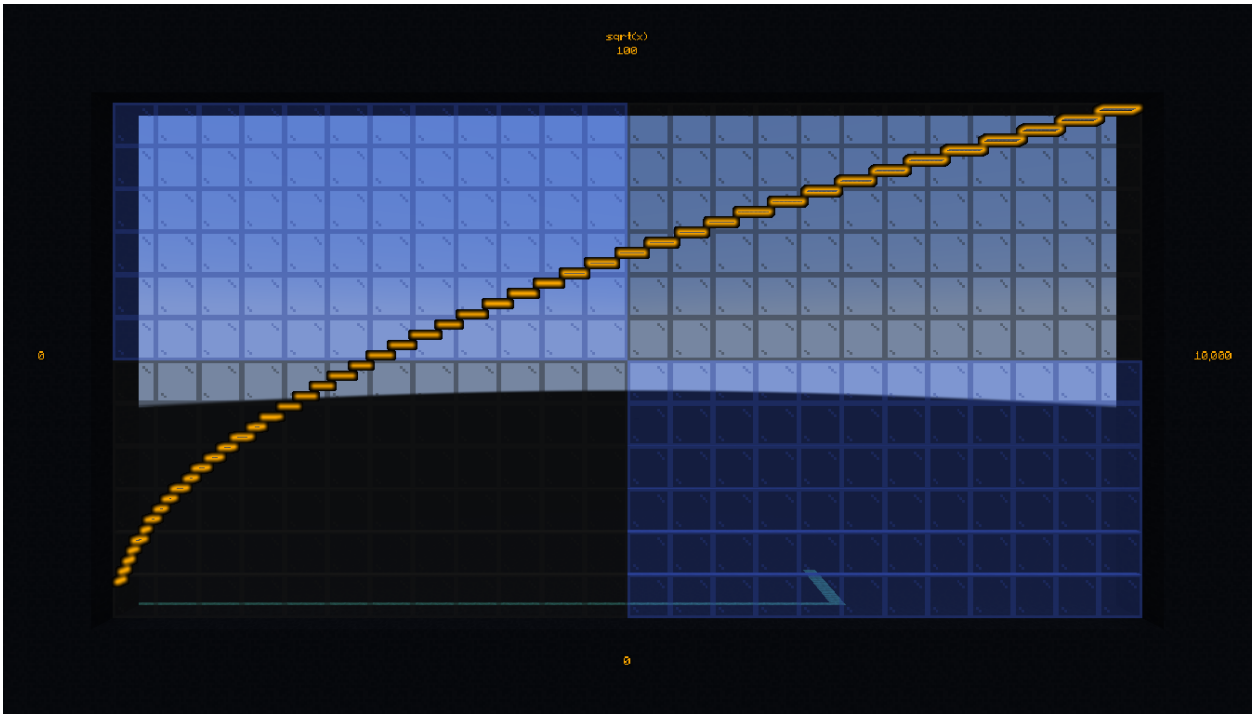
`sqrt`: Compute the square root of the number (ex: Sqrt(16) = 4 because 4^2 = 4x4 = 16)

- Takes as parameter the score `glib.var0` greater than or equal to 0 (corresponding to a value with a precision of 1:1)
- Returns the value of the cosine on the score `glib.res0` greater than or equal to 0 (corresponding to a value with a precision of 1:1)

Example:

- Calculate and display the square root of 42:

```
# Once
scoreboard players set @s glib.var0 42
function glib.math:common/sqrt
tellraw @a [{"text": "sqrt(42) = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```



6.13.4 Special

`glib.math:special/`: this folder contains functions that are of special interest in algortihms (but not or not much in formal mathematics)

Retrieve the next power of 2

`get_next_pow2`: compute the power of 2 directly superior to the number given in parameter on the score `glib.var0` and return the result on `glib.res0`.

Example:

- Find the power of 2 greater than 43

```
# Once
scoreboard players set @s glib.var0 43
function glib.math:special/get_next_pow2
tellraw @a [{"text":"get_next_pow2(43) = ","color":"dark_gray"},{"score":{"name":"@s
↪","objective":"glib.res0"},"color":"gold"}]
```

Random number generator

`random`: Generates a random number and returns the result on the `glib.res0` score

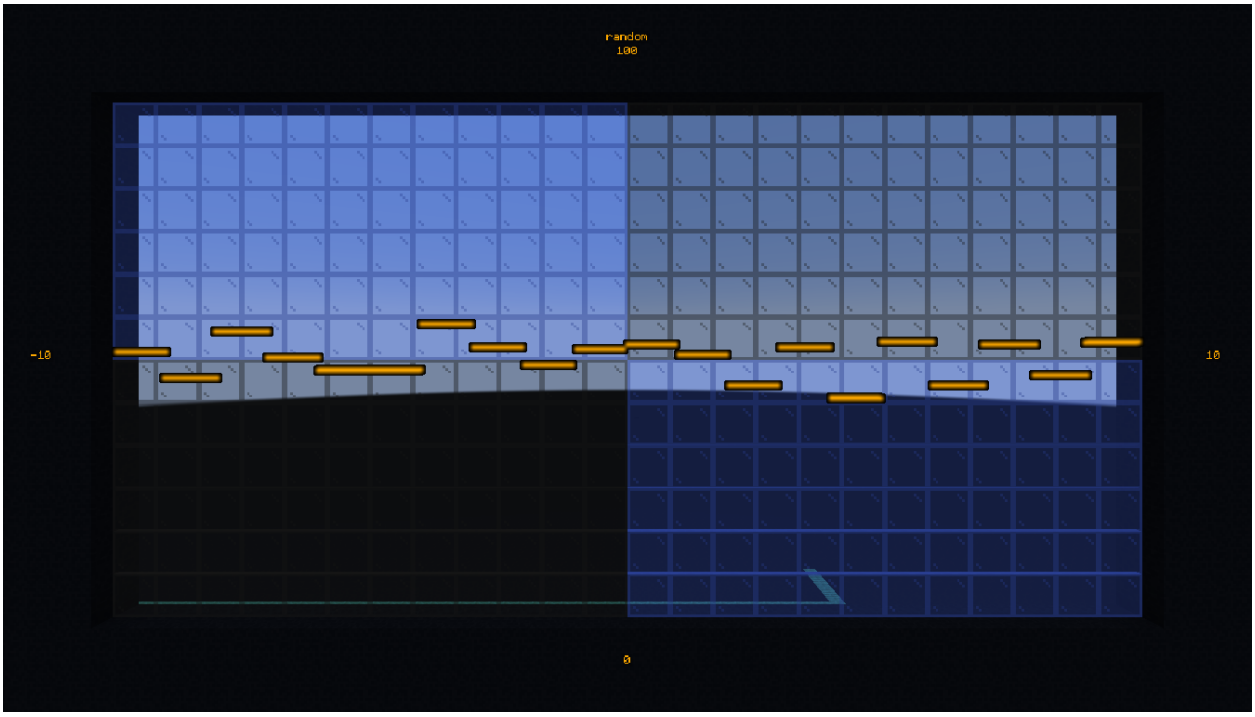
- To reduce this interval, execute the function then do a "modulo" operation on the result (random % 10 -> the random number will be included in the interval [0;9])

Example:

- Get and display a random number between 0 and 100:

```
# Once
function glib.math:special/random
scoreboard players operation @s glib.res0 %= 101 glib.const
tellraw @a [{"text": "random() = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]

Beware: the score `glib.const` does not contain all possible
values. Make sure the value you want to use exists and initialize it
if necessary.
```



6.13.5 Trigonometry

`glib.math:trig/`: this folder contains basic trigonometry functions, opening a lot of doors to creative possibilities in Minecraft.

Arccosinus

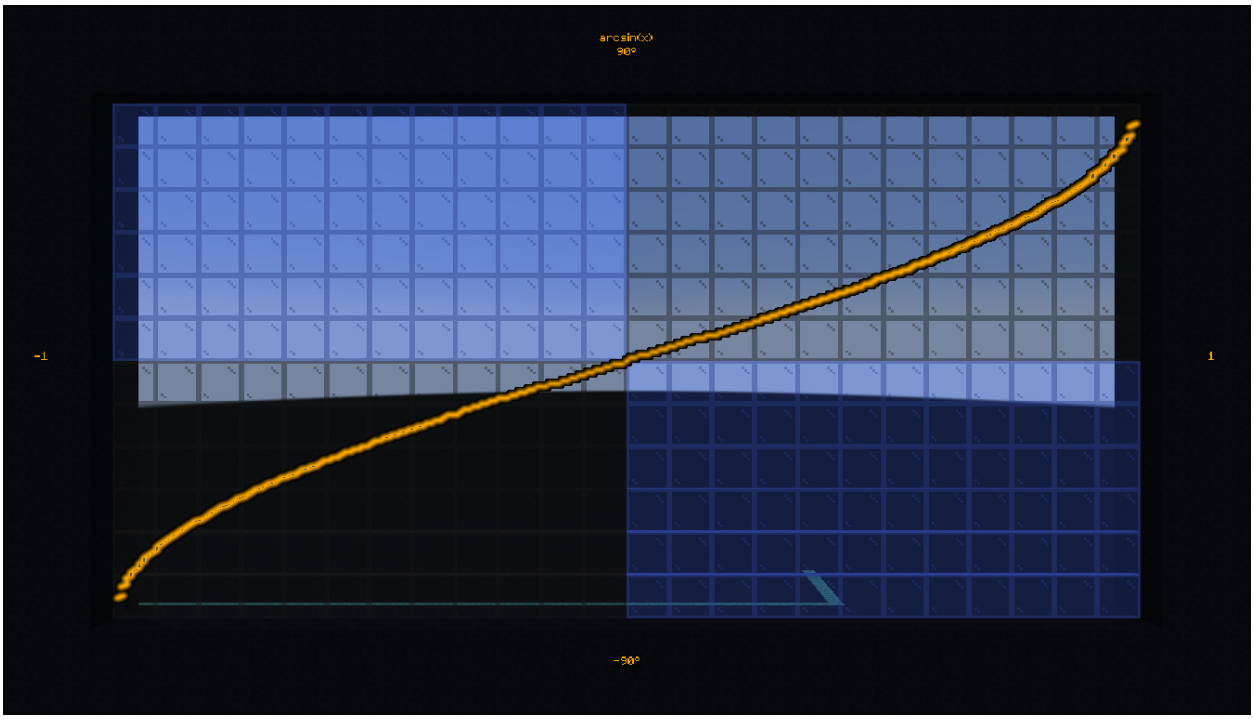
`arccos`: Calculate the arccosinus of a value between -1 and 1

- Takes as parameter the score `glib.var0` between -1000 and 1000 (translating a value from -1 to 1 with a precision of 1:1000)
- Returns the value of the arccosine on the score `glib.res0` (corresponding to an angle with a precision of 1:1 degree)

Example:

- Calculate and display the arccos of 0,42

```
# Once
scoreboard players set @s glib.var0 420
function glib.math:trig/arccos
tellraw @a [{"text":"arccos(0.42) = ","color":"dark_gray"},{"score":{"name":"@s",
↪"objective":"glib.res0"},"color":"gold"}]
```



Arcsinus

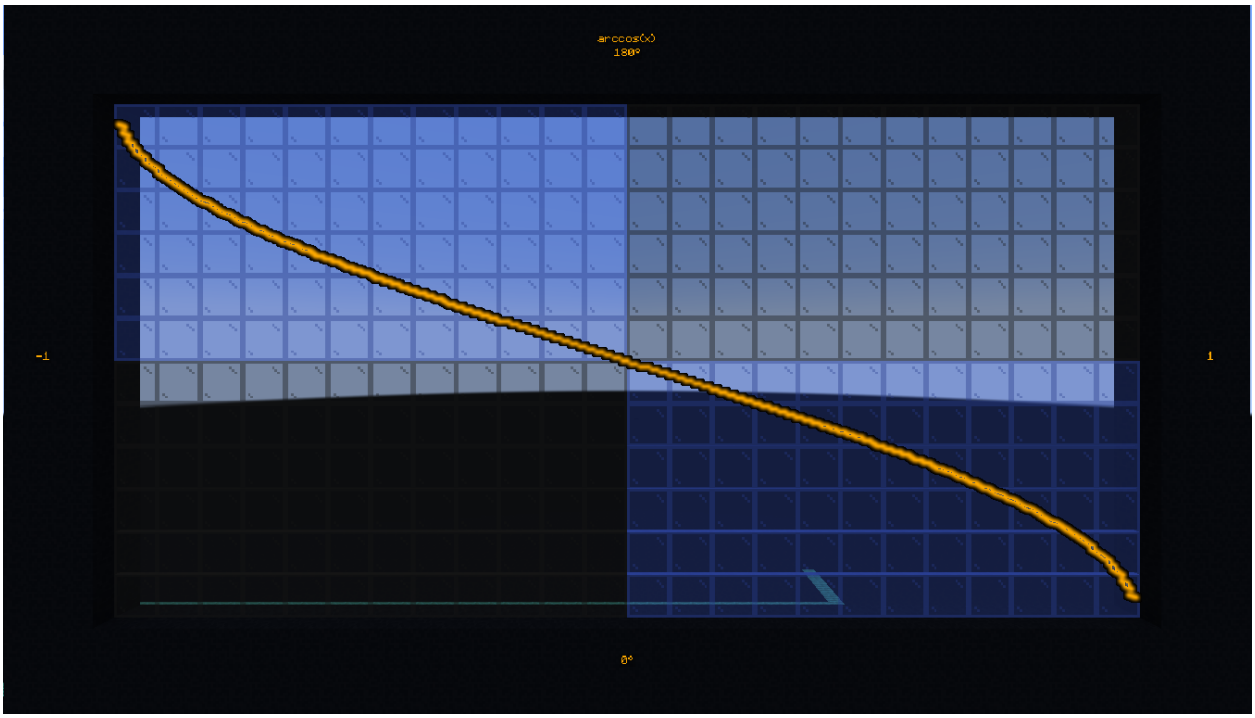
`arcsin`: Compute the arcsinus of a value between -1 and 1

- Takes as parameter the score `glib.var0` between -1000 and 1000 (translating a value from -1 to 1 with a precision of 1:1000)
- Returns the value of the arcsine on the score `glib.res0` (corresponding to an angle with a precision of 1:1 degree)

Example:

- Calculate and display the arcsinus of 0.42

```
# Once
scoreboard players set @s glib.var0 420
function glib.math:trig/arcsin
tellraw @a [{"text":"arcsin(0.42) = ","color":"dark_gray"},{"score":{"name":"@s",
↪"objective":"glib.res0"},"color":"gold"}]
```



Arctangent

`arctan`: Compute the arctangent of a value between -infinite and +infinite

- Takes as parameter the score `glib.var0` (translating a value with a precision of 1:1000)
- Returns the value of the arctangeant on the score `glib.res0` (corresponding to an angle with a precision of 1:1 degree)

Example:

- Calculate and display the arctan of 0.42

```
# Once
scoreboard players set @s glib.var0 420
function glib.math:trig/arctan
tellraw @a [{"text":"arctan(0.42) = ","color":"dark_gray"},{"score":{"name":"@s",
↪"objective":"glib.res0"},"color":"gold"}]
```

Cosine

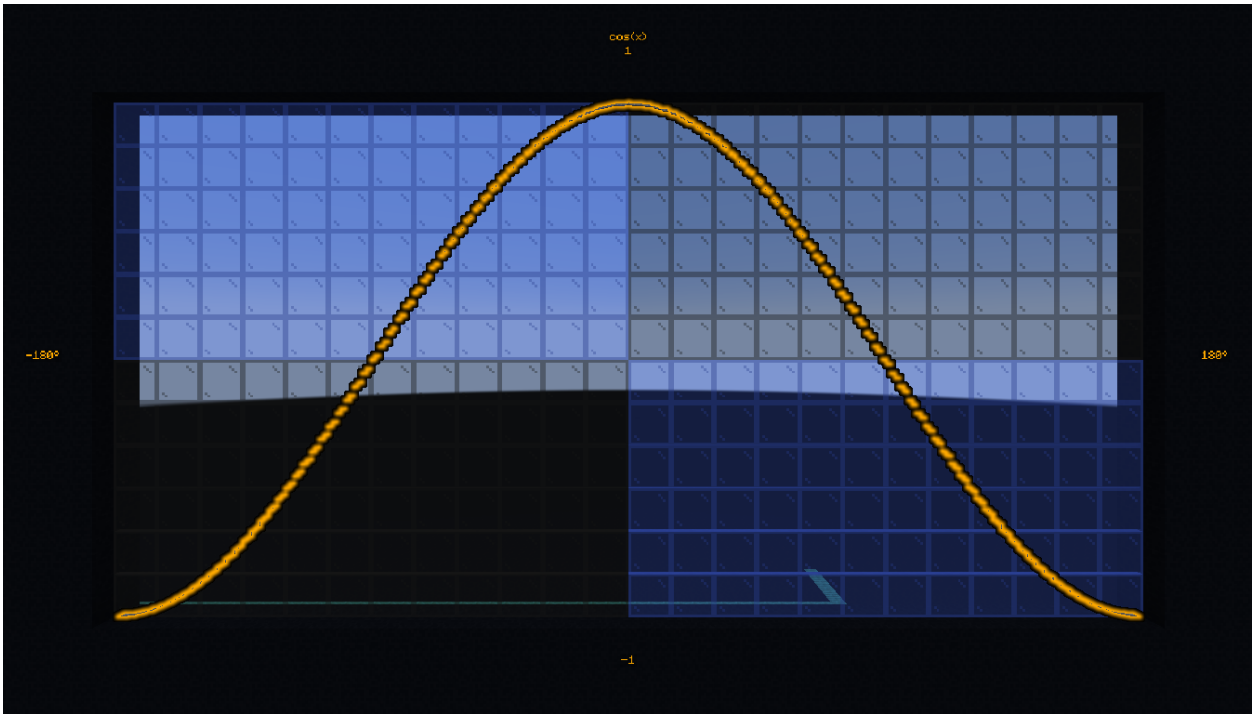
`cos`: Compute the cosine of an angle between 0 and 360

- Takes as parameter the score `glib.var0` between 0 and 360 (corresponding to an angle with a precision of 1:1 degree)
- Returns the value of the cosine on the score `glib.res0` between -1000 and 1000 (translating a value from -1 to 1 with a precision of 1:1000)

Example:

- Calculate and display the cosine of 42

```
# Once
scoreboard players set @s glib.var0 42
function glib.math:trig/cos
tellraw @a [{"text": "cos(42) = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```



Sinus

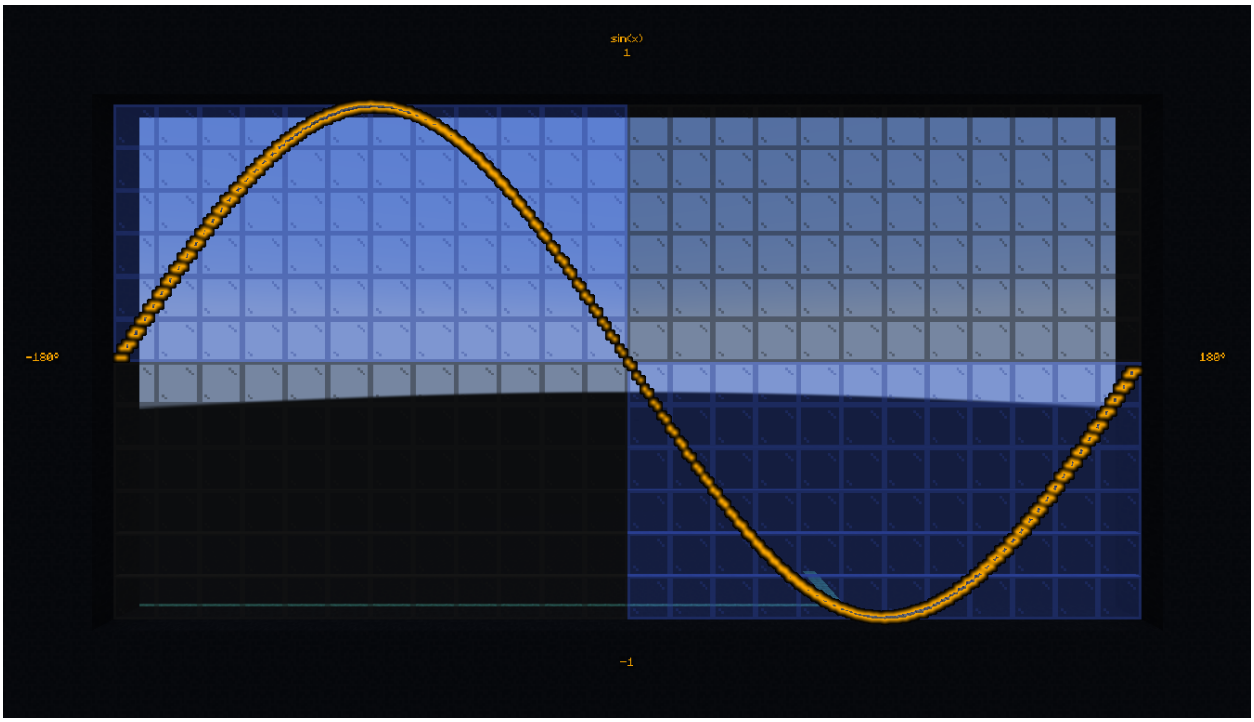
`sin`: Computes the sine of an angle between 0 and 360

- Takes as parameter the score `glib.var0` between 0 and 360 (corresponding to an angle with a precision of 1:1 degree)
- Returns the value of the sine on the score `glib.res0` between -1000 and 1000 (translating a value from -1 to 1 with a precision of 1:1000)

Example:

- Calculate and display the sine of 42

```
# Once
scoreboard players set @s glib.var0 42
function glib.math:trig/sin
tellraw @a [{"text": "sin(42) = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```



Tangent

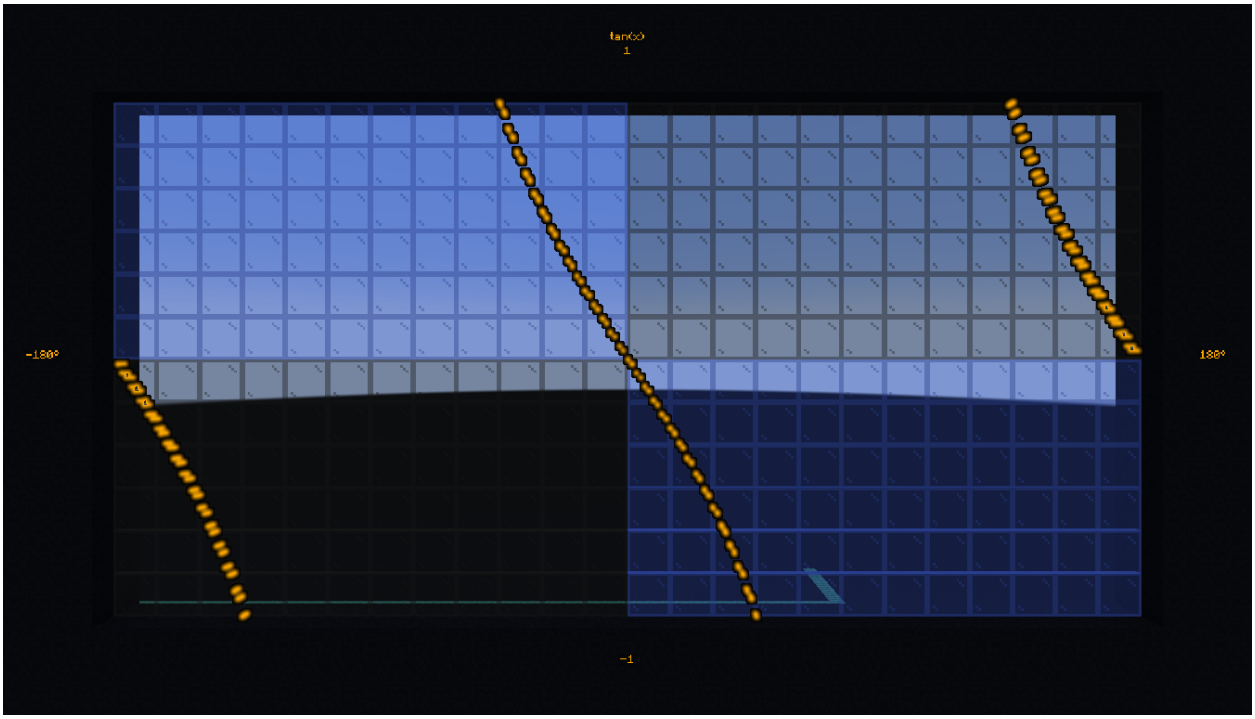
`tan`: Compute the tangeant of an angle between 0 and 360

- Takes as parameter the score `glib.var0` between 0 and 360 (corresponding to an angle with a precision of 1:1 degree)
- Returns the value of the tangeante on the score `glib.res0` between -infinite and +infinite (translating a value with a precision of 1:1000)

Example:

- Calculate and display the tengeante of 42

```
# Once
scoreboard players set @s glib.var0 42
function glib.math:trig/tan
tellraw @a [{"text": "tan(42) = ", "color": "dark_gray"},{"score":{"name":"@s",
↪"objective": "glib.res0"}, "color": "gold"}]
```



6.14 Memory

No documentation here…

Note: This is a communautary project. Writing docs can take a lot of time and creators don't ever have this time. You can help us by contributing to the project on the Github repository and by joining us on Discord

6.15 Move

`glib.move:`: This folder contains all the functions related to the movement of the entity.

6.15.1 Move using local vector

`by_local_vector`: Allows to move the entity according to its vector on each axis of the local reference frame.

- A vector of 1000 on an axis will lead to a movement of a block at each execution of the function.
- The sum of the movements on each axis will give a movement in space (thus in 3 dimensions), corresponding to the global vector of the entity.
- The system takes as input the 3 scores `glib.vector[Left,Up,Front]` (1000 <=> 1 block).

Warning: The system does not include any speed limit. However, the resources consumed by this function are proportional to the number of blocks/tick at which the entity moves.

Example:

- Apply a movement of 0.3 blocks per tick to the left to all boats:

```
# Once
scoreboard players set @e[type=boat] glib.vectorLeft 300
scoreboard players set @e[type=boat] glib.vectorUp 0
scoreboard players set @e[type=boat] glib.vectorFront 0

# In loop
execute as @e[type=boat] run function glib.move:by_local_vector
```

6.15.2 Move by classic vector

`by_vector`: Allows to move the entity according to its vector on each axis of the relative reference frame.

- A vector of 1000 on an axis will lead to a movement of a block at each execution of the function.
- The sum of the movements on each axis will give a movement in space (thus in 3 dimensions), corresponding to the global vector of the entity.
- The system takes as input the 3 scores `glib.vector[X,Y,Z]` (1000 <=> 1 block) as well as the `glib.collision` score.
- This last score allows to manage the behavior. If it is not filled in or equal to 0, the entity will cross all the blocks
- Each behavior is defined via a dedicated file in `glib_config:move/by_vector/`
- It is possible to manage the precision of collision detection by placing the tag `glib.config.override` on the entity and then changing its score `glib.precision` to the desired value (1000 <=> 1 block, 500 <=> 0.5 blocks)
- If the precision is higher than 1 block, the entity will have a certain probability to cross the walls of a block of thickness.

Warning: The system does not include any speed limit. However, the collision accuracy breaks the vector into multiple vectors with a length corresponding to the detection accuracy. The system will then enter a loop to restore the initial vector by successively applying the "vector pieces". Thus, the longer the length of the vector compared to the collision detection accuracy, the more resources the system will require to perform optimally.

Examples:

- Apply a movement of 0.3 blocks per tick in the X direction to all boats (simulating a sea current):

```
# Once
scoreboard players set @e[type=boat] glib.vectorX 300
scoreboard players set @e[type=boat] glib.vectorY 0
scoreboard players set @e[type=boat] glib.vectorZ 0

# In loop
execute as @e[type=boat] run function glib.move:by_vector
```

- Take into account collisions and make the boat stop, with a precision of 0.1 block:

```
# Once
scoreboard players set @e[type=boat] glib.vectorX 300
scoreboard players set @e[type=boat] glib.vectorY 0
scoreboard players set @e[type=boat] glib.vectorZ 0
scoreboard players set @e[type=boat] glib.collision 2
tag @e[type=boat] add glib.config.override
scoreboard players set @e[type=boat] glib.precision 100

# In loop
execute as @e[type=boat] run function glib.move:by_vector
```

6.15.3 Move forward

`forward`: Allows to move the entity according to the direction towards which it looks and its vector `glib.vectorFront`

- A vector of 1000 on an axis will cause a movement of one block at each execution of the function.
- The sum of the movements on each axis will give a movement in space (thus in 3 dimensions), corresponding to the global vector of the entity.
- – The system takes as input the 3 scores `glib.vector[Left,Up,Front]` (1000 <=> 1 block).

Warning: The system does not include any speed limit. However, the resources consumed by this function are proportional to the number of blocks/tick at which the entity moves.

Example:

- Apply a movement of 0.3 blocks per tick forward to all boats:

```
# Once
scoreboard players set @e[type=boat] glib.vectorFront 300

# In a loop
execute as @e[type=boat] run function glib.move:forward
```

6.15.4 Find a path "as to at"

`pathfind_ata`: Allows to determine a path between the position of the source entity and the execution position of the function.

- By default, the function will make 500 tests (defined via the `glib.var1` score). This limit allow to avoid the function taking too many ressources if the path is too complexe or impossible to find.
- The behavior is defined by the variable `glib.var3` which, by default is 0, corresponding to a behavior of a zombie, creeper, skeleton or a player (terrestrial entity of size 121).
  - When it is set to 1, the behavior will be similar to a bat.
  - You can create your own behaviors at any time in the `pathfind/config/` folder and link them in the `main.mcfunction` file in the same folder.
- The path is then defined by a succession of armor_stand with the tag "Glib_Pathfind_Rewind" and "Glib_Pathfind".

Example:

- Find the path to the nearest armor_stand:

```
# Once
execute at @e[type=minecraft:armor_stand,limit=1,sort=nearest] run function glib.
↪move:pathfind_ata
```

6.15.5 Convert vector to motion

`vector_to_motion`: Allows to move the entity according to its vector through a motion (motion system integrated in the game).

- A vector of 1000 on an axis will move a block at each tick of the game.
- The sum of the movements on each axis will give a movement in space (thus in 3 dimensions), corresponding to the global vector of the entity.

Note: This system admits a speed limit corresponding to that of the Motions. Moreover, the entity will have by default a collision system preventing it from crossing blocks. Moreover, adding Marker, NoAI, NoGravity tags can block this system. Collisions are integrated in this system but are not very reliable and therefore not recommended. Only activated when the entity has a Collision score greater than 1 (each value corresponds to a type of collision). You can modify the collision reactions or create your own in the `by_vector/config/collision/` folder. By default, the precision of the collisions, stored on the Var5 score, is 500 (= 0,5 blocks). }}

6.16 Orientation

`glib.orientation:`: This folder contains functions that facilitate the management of the orientation of entities.

6.16.1 Get orientation

`get`: Detects the orientation of an entity and stores it on the scores "OriH" and "OriV", corresponding respectively to the angles formed on the horizontal and vertical plane by the orientation of the entity.

6.16.2 Normalize orientation

`normalize`: Allows to normalize the oriantations (replace the OriH and OriV scores respectively in the interval [0;360[and [0;180[)

6.16.3 Set orientation

`set`: Allows to orientate the entity according to its scores "OriH" and "OriV". This function has variations on h and v, useful for players for whom the orientation cannot be modified directly via a /data.

6.17 Schedule

No documentation here...

Note: This is a communautary project. Writing docs can take a lot of time and creators don't ever have this time. You can help us by contributing to the project on the Github repository and by joining us on Discord

---

6.18 Time

No documentation here...

Note: This is a communautary project. Writing docs can take a lot of time and creators don't ever have this time. You can help us by contributing to the project on the Github repository and by joining us on Discord

---

6.19 Vector

`glib.vector:`: Vectors are used to define the trajectory of an entity. Thus, they allow to manage projectiles easily, offering the possibility to make them undergo collisions, to imitate gravity, wind etc ... They are always defined with a 10^3 factor (1000 = 1 block/tick).

6.19.1 Classic

`glib.vector:classic/`: Classic vectors are vectors that use the default base of the game, corresponding to tilds. Thus, a classic position vector (1000,3000,5000) will correspond to the position ~1 ~3 ~5

Get vector by actual orientation

Note: This function require the Location module.

`get_by_actual_orientation`: Compute the displacement vector of the entity according to its orientation. This vector is composed of 3 elementary vectors stored on the scores `glib.vector[X,Y,Z]` (each between -1000 and 1000).

Example:

Create, for each Creeper, a vector from their respective orientation

```
# Once
execute as @e[type=creeper] run function glib.vector:classic/get_by_actual_orientation
```

Get vector "as to at"

Note: This function require the Location module.

`get_ata`: Compute a vector from the source entity to the execution position of the function.

Example:

Create a vector that connects you to the nearest skeleton:

```
# Once
execute as @s at @e[type=skeleton] run function glib.vector:classic/get_ata
```

Get vector lenght

Note: This function require the Math module.

`get_lenght`: compute the norm of the vector and store it on the score `glib.res0`

Warning: This function calls the sqrt function, it is not recommended to use it frequently for performance reasons.

Get vector leght squared

`get_lenght_squared`: compute the norm of the squared vector and store it on the score `glib.res0`.

Fast normalize

`fast_normalize`: allows to normalize the components of the vector by placing the largest component at 1000 while respecting the proportions linking these components.

> WARNING: For optimization purposes and to avoid doing square root calculations, this function reduces the vector to a cube of 2 blocks centered on the entity (instead of a sphere of radius 1 centered on the entity)

6.19.2 Local

`glib.vector:local/`: Local vectors are vectors that use the base defined according to the orientation of the entity, corresponding to the powers (^). Thus, a position vector in local (1000,3000,5000) will correspond to the position ^1 ^3 ^5

Convert classic to local vector

Note: This function require the Math module.

`get_from_classic_vector`: Allows to convert a "normal" vector (using the relative reference frame) into local coordinates (using the local reference frame) * Takes the 3 scores `glib.vector[X,Y,Z]` as input * Stores the result on the 3 scores `glib.vector[Left,Up,Front]`

Example:

Find the local vector corresponding to the vector X=1000, Y=0, Z=0

```
# Once
scoreboard players set @s glib.vectorX 1000
scoreboard players set @s glib.vectorY 0
scoreboard players set @s glib.vectorZ 0
function glib.vector:classic/get_from_classic_vector

# Display the result
tellraw @a [{"text":"<"},{"selector":"@s"},{"text":">"},{"text":" VectorLeft: ","color":
↪"dark_gray"},{"score":{"name":"@s","objective":"glib. vectorLeft"}, "color": "gold"},{
↪"text": "VectorUp: ", "color": "dark_gray"},{"score":{"name":"@s", "objective": "glib.␣
↪vectorUp"}, "color": "gold"},{"text":" VectorFront: ", "color": "dark_gray"},{"score":{
↪"name":"@s", "objective": "glib.vectorFront"}, "color": "gold"}]
```

6.20 View

`glib.view:`:The view functions allow to get some practical information about what the entity sees or aims.

6.20.1 Get aimed block

`aimed_block`: Places an armor_stand having the tag `glib.aimedBlock` and a score `glib.parentId` corresponding to the Id of the running entity.

Example:

Place in entity on the aimed block:

```
# Once
function glib.view:aimed_block
```

6.20.2 Get aimed entity

`aimed_entity`: Gives the tag `glib.raycastEntity` and a score `glib.parentId` corresponding to the Id of the entity executing at the targeted entity by the player.

Example:

Place in entity on the targeted block:

```
# Once
function glib.view:aimed_entity
```

6.20.3 Can see “as to at”

`can_see_ata`: Allows to know if the entity, from its position, may be able to see the execution position of the command (if no block obstructs its vision). If so, the source entity will get the tag `glib.canSee`.

Example:

Knowing whether an entity sees you:

```
# Once
execute as @e at @s run function glib.view:has_in_front_ata
```

6.20.4 Has in front "as to at"

`has_in_front_ata`: Allows to know if the execution position of the function is in front of the source entity. If it is, the source entity gets the tag `glib.hasInFront`.

Example:

Know if the position 0 5 0 is in front of you:

```
# Once
execute as @s positioned 0 5 0 run function glib.view:has_in_front_ata
```

6.21 XP

`glib.xp:` all function concerning the experience points and levels.

6.21.1 Add levels

`add_levels`: Add levels from a score

- Take the amount of levels to add on the score `glib.var0`

Example:

Add you 123 levels

```
# Once
scoreboard players set @s glib.var0 123
glib.xp:add_levels

# See the result
# look at your XP bar in survival mode
```

6.21.2 Add points

`add_points`: Add XP from a score

- Take the amount of XP to add on the score `glib.var0`

Example:

Add you 1234 XP

```
# Once
scoreboard players set @s glib.var0 1234
glib.xp:add_points

# See the result
function glib.xp:get_total_points
scoreboard objectives setdisplay sidebar glib.res0
# run the add function here
function glib.xp:get_total_points
```

6.21.3 Get bar

`get_bar`: Get the portion of the bar filled

- The percentage of the bar filled is returned on the score `glib.res0`
- Due to the division, the result is rounded to the lowest integer. If you want to round to the nearest integer, use the function *get_bar_rounded*

Example:

Get the portion filled in your XP bar

```
# Once
glib.xp:get_bar

# See the result
scoreboard obective setdisplay sidebar glib.res0
```

6.21.4 Get bar rounded

`get_bar_rounded`: Get the portion of the bar filled

- The percentage of the bar filled is returned on the score `glib.res0`
- This function require the module *glib.math*

Example:

Get the portion filled in your XP bar

```
# Once
glib.xp:get_bar_rounded

# See the result
scoreboard obective setdisplay sidebar glib.res0
```

6.21.5 Get Level Points

`get_level_points`: Get the points required to pass to the next level.

- Returns the number of points required on `glib.res0`

Example:

Get the number of points required to pass from the level 15 to the level 16

```
# Once
scoreboard players set @s glib.var0 15
glib.xp:get_total_points

# See the result
tellraw @a ["",{"text":"I need "},{"score":{"name":"@s","objective":"glib.res0"}},{"text
↪":" to pass this level"}]
```

6.21.6 Get Total Points

`get_total_points`: Get the total amount of points of the player.

- Returns the amount of XP points on the score `glib.res0`

Example:

Get your amount of points

```
# Once (execute on you)
glib.xp:get_total_points

# See the result (execute on you)
tellraw @a ["",{"text":"I have "},{"score":{"name":"@s","objective":"glib.res0"}},{"text
↪":" XP"}]
```

6.21.7 Remove levels

`remove_levels`: Remove levels from a score

- Take the amount of levels to remove on the score `glib.var0`

Example:

Remove you 123 levels

```
# Once
scoreboard players set @s glib.var0 123
glib.xp:add_levels

# See the result
# look at your XP bar in survival mode
```

6.21.8 Remove points

`remove_points`: Remove XP from a score

- Take the amount of XP to remove on the score `glib.var0`

Example:

Remove you 1234 XP

```
# Once
scoreboard players set @s glib.var0 1234
glib.xp:remove_points

# See the result
function glib.xp:get_total_points
scoreboard objectives setdisplay sidebar glib.res0
# run the remove function here
function glib.xp:get_total_points
```

6.21.9 Set bar

`set_bar`: Fill partially the XP bar

- Take the percentage of the bar filled via the `glib.var0` score

Example:

Fill your bar at 50%

```
# Once
scoreboard players set @s glib.var0 50
glib.xp:set_bar

# See the result
# look at your XP bar in survival mode
```

6.21.10 Set levels

`set_levels`: Set levels from a score

- Take the amount of levels to set on the score `glib.var0`

Example:

Set your level to 123

```
# Once
scoreboard players set @s glib.var0 123
glib.xp:set_levels

# See the result
# look at your XP bar in survival mode
```

6.21.11 Set points

`set_points`: Set XP from a score

- Take the amount of XP to set on the score `glib.var0`

Example:

Set your XP to 1234

```
# Once
scoreboard players set @s glib.var0 1234
glib.xp:set_points

# See the result
function glib.xp:get_total_points
scoreboard objectives setdisplay sidebar glib.res0
# run the set function here
function glib.xp:get_total_points
```

6.21.12 Set total points

`set_total_points`: Set XP from a score

- Take the total amount of XP to set on the score `glib.var0`

Example:

Set your total XP amount to 1234

```
# Once
scoreboard players set @s glib.var0 1234
glib.xp:set_total_points

# See the result
function glib.xp:get_total_points
scoreboard objectives setdisplay sidebar glib.res0
```

6.22 Biome Displayer

This system allow to display the biome you're in in the action bar.

To enable this system, you just need to execute the following command in a loop:

```
function gsys.biome_displayer:main
```

6.23 LGdir

This system allow to simply throw a projectile by right-clocking on a `carrot_on_a_stick`.

To enable this system, you just need to execute the following command in a loop:

```
function gsys.lgdir:main
```