Early Preview Branch

Mod support is included with Shovel Knight Pocket Dungeon's Paradox Pack DLC, which is currently only available to Steam users who are on the Steam Early Preview branch! So if you would like to begin creating mods of your own, you need to switch to the Early Preview branch to test and play your mod.

Follow this link for further instructions on how to switch branches!

We also have created a Mod Support specific channel called `#mod-support-skpd` in the Yacht Club Games Discord! As we receive feedback and make changes, we'll often post updates in that channel and we'll be updating our Early Preview Update Notes once a week.

Also as you are developing your Mods, please keep in mind our Modding Guidelines that we posted as a Steam Guide!

Locating the Mods Folder

Local mods are located in the `mods` folder next to your save.

On Windows: `\AppData\Roaming\Yacht Club Games\Shovel Knight Pocket Dungeon\{steamID}\mods`

On Mac: `/Library/Application Support/Yacht Club Games/Shovel Knight Pocket Dungeon/{steamID}/mods`

where `{SteamID}` is the ID of your Steam account.

However, the mods you have subscribed to via Steam Workshop are located in the `Steamapps` folder.

On Windows: `\Program Files (x86)\Steam\steamapps\workshop\content\1184760`

On Mac: `/Library/Application Support/Steam/steamapps/workshop/content/1184760`

Note: If your Steam library is installed in an area that's not the default, you can search for the Shovel Knight Pocket Dungeon App ID (1184760) to locate it.

Note: remote mod folders are always named with their unique upload IDs, while local mod folder names are unrestricted.

Note: don't edit remote files directly; Steam will automatically reload them when the game starts. Make a copy of the downloaded mod in the local mods folder instead.

Mod Structure

A mod is essentially just a folder with files (and other folders).

Any mod should contain a `mod_info.ini` file in the root that packs basic information about it.

```
[general]
url="1875062006" // in case you want to update an item that is already uploaded to Steam Workshop (instead of creating a new one), provide a link to it here. Include only the ID part of the link. You can only update an item if you are its author. Leave blank for a new item.
name="Mod Name" // 20 characters max (remaining will get cropped)
description="Mod Description" // no limit here :)
major_version="1" // the versions are automatically added together, i.e. "v1.1"
minor_version="1" // minor version is also automatically increased by 1 when you push the mod update, i.e. "v1.1" -> "v.1.2"
icon="mod_icon.png" // .png image, your mod's icon. Dimensions should be of a square.
[tags] // mark features your mod uses as "1" to apply corresponding Steam tags to your item:
units="0"
enemies="0"
blocks="0"
playable_characters="0"
bosses="0"
items="1"
stages="1"
prefabs="0"
siderooms="0"
camp="0"
relics="0"
quandaries="0"
modifiers="0"
hats="0"
game_modes="0"
cosmetics="0"
tools="0"
```

Depending on what features your mod uses, you'll want it to include certain types of files and folders.

Templates

We have templates and assets available to help start your modding journey! You can download those zip files here:

- Mod Templates
- Stage Sprites
- Unit Sprites
- Workshop Icon Template

Special note about the Mod Templates:
Most of the `.gml` files in the templates are empty and only contain a few comments! If you create a mod that does not use that .gml template, it's best to delete that file

Testing Mods

To test a local mod, navigate to the in-game "Mod Support" menu and ensure it's enabled.

Note: the menu will only be visible if the tutorial is completed for the current save slot.

If there was an error compiling or executing your scripts, it'll be printed to the mod log. Press `Ctrl+T` to show/hide full output, and `Num+`/`Num-` to scroll through it. Press `Ctrl+Delete` to clear the log.

You can use `Ctrl+R` to quickly reload all the mods and `Ctrl+X` to clear the current grid.

Submitting and Updating Mods

Submitting and updating mods is done through the same in-game "Mod Support" menu.

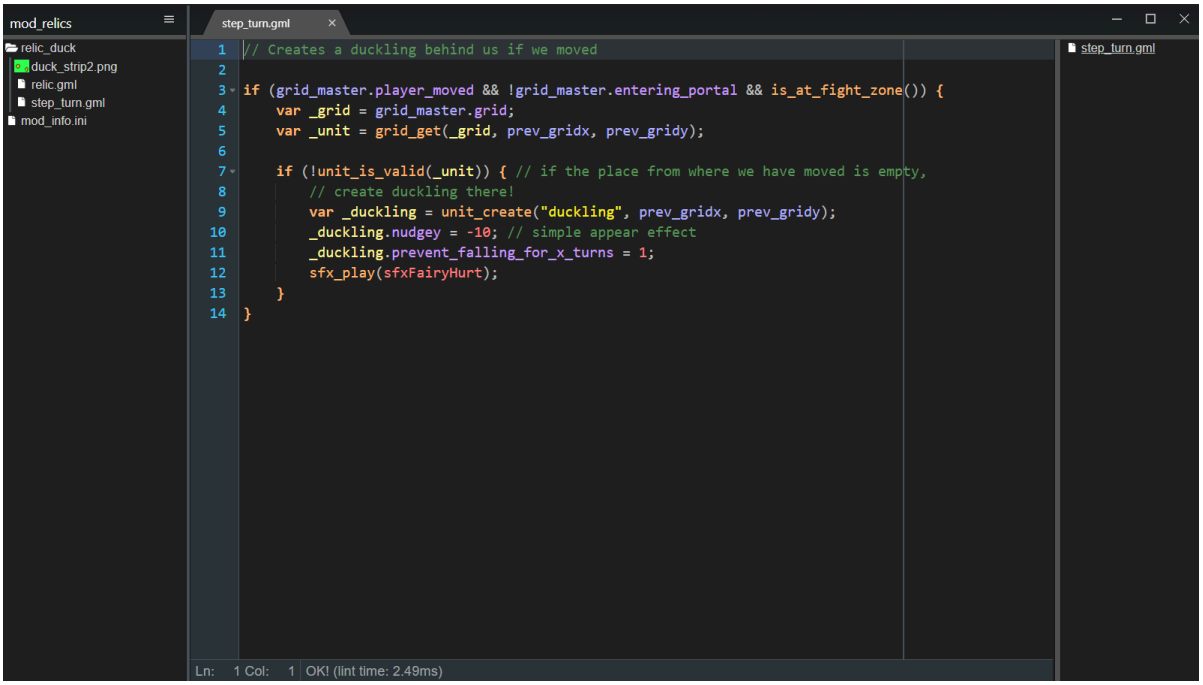
If you encounter an error when uploading your item, refer to Steam's error code list to narrow down the issue.

If the upload is successful, the new item's page will open in Steam overlay, so make sure you have it enabled.

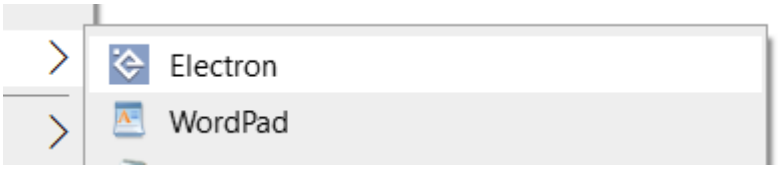
Tools

GMEdit

GMEdit is an external GML editor you can use to create custom scripts. Shovel Knight Pocket Dungeon has its own GML dialect you can install (use this guide!) for the editor to highlight and autocomplete all the Mod Support's functions and macros.



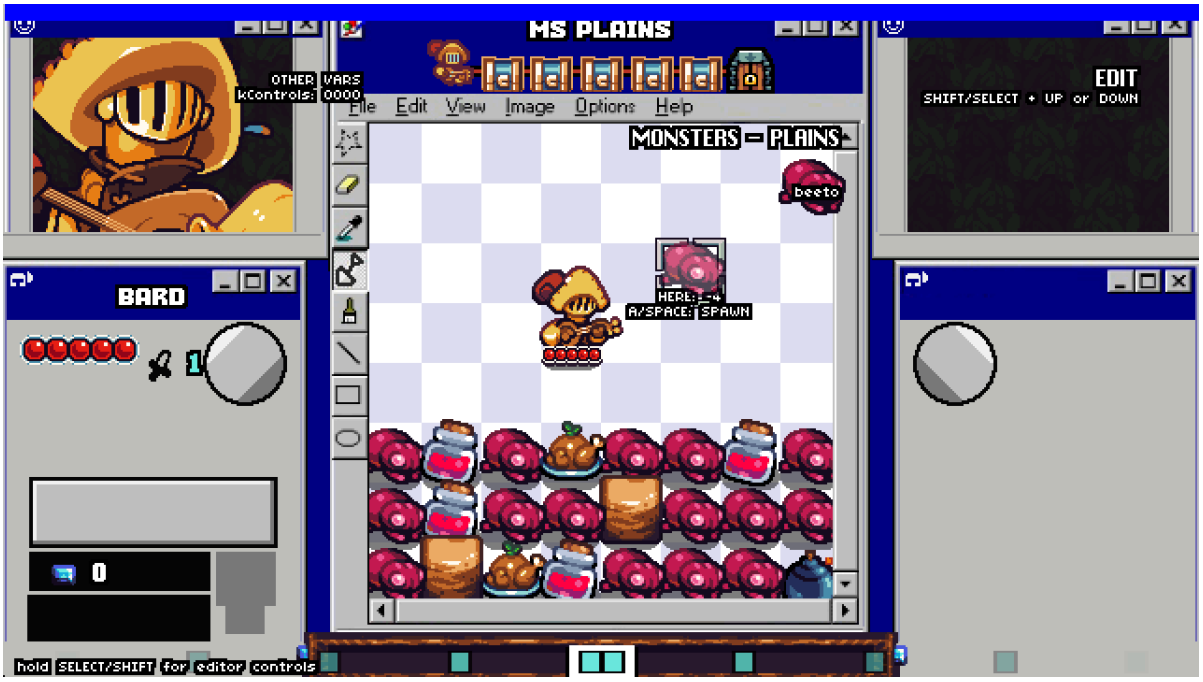
To open the mod in GMEdit, right-click the `mod_info.ini`, select Open As, and choose Electron.



IGGE

IGGE stands for the In-Game Grid Editor. It can be toggled by `Ctrl+E` from any Adventure prefab. The editor lets you freely spawn, delete, and inspect units, items, and traps (including the ones added by mods!). The controls can be found in the editor's UI.

NOTE: IGGE can only be accessed if at least one mod is enabled.



Pallette Swapper Tool

[This tool](#) can be used to easily generate knight's [palatable sprites](#).

VFX/SFX/Prefab/Stage Viewer

With this tool you can see the in-game VFX, SFX, stages and prefabs list. Toggle it by pressing `Ctrl+V`. Use the input field to filter assets and `up`/`down` to navigate the list.

Portrait Tool

This tool by MajorasCask is designed to make the process of creating/editing of units' portraits much easier.

- Drag and drop your portrait layers and see how it animates!

- Change the origin point to fine-tune its position;
- Test the portrait with the game's boards;
- Zoom in to simulate higher resolutions;
- Drop images onto the layers to change them on the fly, toggle their visibility, or delete them altogether.

Unit Mods

You can think of a unit as a game object. Technically, everything in the game is a unit, but the most common use case for unit mods is playable characters and enemies.

Note: items and bosses are units as well, but they have their own modding templates, and it's recommended you use them.

Unit mod should be packed inside a folder named "unit_{name}", which should also include a `unit.gml` file. Inside `unit.gml` you define what your unit is using pre-built variables.

Unit Variables

- `unit_id`: If your mod is an edit of an existing unit, this variable should include its ID. If you want to create a new unit from scratch, this variable should be a unique string instead.

```
unit_id = "shovel knight"; // this mod overrides shovel knight unit
```

```
unit_id = "cool custom enemy"; // this mod creates new enemy unit
```

- `unit_template:` If your mod is a new unit, you can define here a different unit you would like to copy as a template.

```
unit_template = "beeto"; // this mod uses beeto as its starting point
```

- `behaviors`: a list of a unit's special traits (or tags!), packed into a string separated by commas.

```
behaviors = "big,undead,jumps:8,collapses:6" // for example, these are traits used by Super Skeletons
```

If you are modifying a unit, you can make it so that your behaviors are applied on top

of the behaviors the unit already has:

```
add_behaviors = true; // behaviors you specified will be added on top of existing behaviors
```

- `name`: displayed name (max 15 characters).
- `description`: displayed description (max 150 characters).
- `hp`: default is `2`.
- `atk`: default is `1`.
- `is_knight`: if set to `true`, will add `knight` behavior.
- `is_big`: if set to `true`, will add `big` behavior.
- `is_enemy`: if set to `true`, will add `enemy` behavior.
- `is_boss`: if set to `true`, will mark the unit as boss.
    - `boss_ending`: if set, defeating the boss will trigger ending cutscene: `0` = good ending, `1` = bad ending.
- `is_grapps`: if set to `true`, will mark the unit as grapps.
    - `body_description`: displayed description for the body sub-unit (max 150 characters).
- `dialogue`: dialogue line the unit will play if bumped at hub (max 200 characters)
    - `dialogue_loop`: an array of dialogue lines (an array of strings). If set, overrides `dialogue` property. Each time the player bumps into the unit, a new line will play. When the array end is reached, circles back to the first one.
    - `dialogue_beep`: the sound to play while the dialogue line is displayed (see SFX loading).
- `sfx_hurt`: the sound effect played when the unit takes damage
    - `sfx_hurt_pitch`: the pitch of the hurt sound. Default is `1` (no pitch change). A value of less than `1` will lower the pitch and greater than `1` will raise the pitch. It is best to use small increments for this function as any value under `0` or over `5` may not be audible anyway. It is worth noting that the total pitch change permitted is clamped to (`1/256`) - 256 octaves, so any value over or under this will not be registered.
- `sfx_dead`: the sound effect played when the unit dies
    - `sfx_dead_pitch`: the pitch of the death sound.
- `hub_spawn` and `hub_spawn2` : are array definitions of what prefabs this unit will spawn at when in Camp. This is so you can more easily find and access your modded Knights to play as:

```
hub_spawn = ["hub", 2,4]; // this unit will spawn in hub at grid 2,4
```

Note: This will only work for the base unit and not for the B unit.

- `mod_portal`: whether the unit should spawn in the Mod Hub (accessed from the portal in the main area). You might want to set it to `false` for secret units or units that should be unlocked first. Default is `true`.

- `joustus_card`: array defining this unit's joustus card if defeated by King Knight B. If you don't provide one the game will generate a default one based on tags and HP. Customize it like so:

```
 // one bomb arrow up, one arrow right and the power "slam"
joustus_card = [0,"bomb",1,0,"slam"];
 // the array is [left, up, right, down, power]
```

  - **Arrows:**
    - `0 , 1, 2, 3`, amount of arrows.
    - `"conveyor"` , a geared arrow.
    - `"bomb",` a bomb arrow.
  - **Powers**
    - `""` , no power.
    - `"slam"`, pushes units when attacking.
    - `"cascade",` flips units upside down.
    - `"switch h"`, switches arrows horizontally every few turns.
    - `"switch v"`, switches arrows vertically every few turns.
    - `"switch"`, switches arrows every few turns.
    - `"heal"`, heals for 2HP.
    - `"rock"`, spawns a block that will fall at your position.
    - `"bomb"`, causes an explosion.
    - `"freeze"`, freezes units around.
    - `"poison"`, poisons units around.
    - `"grave"`, can attack the top most row (normally unreachable)

Sprite Tags

The following variables define sprites to be used by your unit mod. See VFX loading section for more info on how to handle them.

Note: the origin (i.e. the center for drawing and mirroring/rotating) for all sprites is `48x64`, and the recommended size is `96x96px`. If the unit is `big`, the origin is `97x100` instead (recommended size is `192x192px`).

Note: if you don't define any of the sprite tags (i.e. `sprite_hub`), it will automatically refer to `sprite_idle`. This will not happen if you are overriding an existing unit though. To enable this feature for an existing unit, set `force_idle_sprite` to `true` from `unit.gml`.

- `sprite_idle`
  - `sprite_idle_frames` (default `4`)
- `sprite_hub`: overrides `sprite_idle` if the unit is at hub.
  - `sprite_hub_frames` (default `4`)

- `sprite_head`: overrides `sprite_idle` if the unit is a grapps head.
  - `sprite_head_frames` (default `4`)
- `sprite_body`: overrides `sprite_idle` if the unit is a grapps body.
  - `sprite_body_frames` (default `4`)
- `sprite_conceal`: overrides `sprite_idle`. For example, memmecs start with a different sprite to fool you into being chests.
  - `sprite_conceal_frames` (default `4`)
- `sprite_fly`: unit is tossed and is on the way up.
  - `sprite_fly_frames` (default `4`)
- `sprite_fall`: unit is tossed and falling down.
  - `sprite_fall_frames` (default `4`)
- `sprite_map`: used during map transitions.
  - `sprite_map_frames` (default `4`)

Behavior-Specific tags

These are normally set by various behaviors:

- `sprite_special`
  - `sprite_special_frames` (default `4`)
- `sprite_charge`
  - `sprite_charge_frames` (default `4`)
- `sprite_aim`
  - `sprite_aim_frames` (default `4`)
- `sprite_skill`
  - `sprite_skill_frames` (default `4`)
- `sprite_skill_up`
  - `sprite_skill_up_frames` (default `4`)
- `sprite_skill_down`
  - `sprite_skill_down_frames` (default `4`)
- `sprite_other`
  - `sprite_other_frames` (default `4`)

Knight-Specific Tags

- `sprite_emote`: triumphing animation used on level start.
  - `sprite_emote_frames` (default `4`)
- `sprite_emote_out`: transition animation from `sprite_emote` to `sprite_idle`.
  - `sprite_emote_out_frames` (default `2`)
- `sprite_weak`: overrides `sprite_idle` if HP is 2 or less.
  - `sprite_weak_frames` (default `4`)
- `sprite_dying`: transition animation from `sprite_idle` to `sprite_dead`.
  - `sprite_dying_frames` (default `2`)
- `sprite_dead`
  - `sprite_dead_frames` (default `4`

- `sprite_mini`: used in the HUD to indicate level progress. Origin is `16x16px`, recommended size is `32x32px`.
    - `sprite_mini_frames` (default `2`)

- `sprite_moonland`: used during the ending cutscene. Origin is `24x24px`, recommended size is `48x48px`.
    - `sprite_moonland_frames` (default `2`)

Flip tags

These should only be set if you want a knight to display different sprites while facing left and right. Random Knight and Shuffle Knight apply them by default:

- `sprite_idle_flip` (default `-1`)
    - `sprite_idle_flip_frames` (default `4`)
- `sprite_emote_flip` (default `-1`)
    - `sprite_emote_flip_frames` (default `4`)
- `sprite_emote_out_flip` (default `-1`)
    - `sprite_emote_out_frames` (default `4`)
- `sprite_dying_flip` (default `-1`)
    - `sprite_dying_flip_frames` (default `4`)
- `sprite_dead_flip` (default `-1`)
    - `sprite_dead_flip_frames` (default `4`)

Portraits

Portraits are `160x160px` sprites that can contain up to 6 frames. Each subsequent frame is drawn on top of the previous ones, to create a "layer" effect.

- `portrait_idle`
  - `portrait_idle_frames` (default `6`)
- `portrait_hurt`
  - `portrait_hurt_frames` (default `6`)
- `portrait_win` (knights-only)
  - `portrait_win_frames` (default `6`)

Knight-only Variables

- `relics_start`: an array of relics the knight starts with. See relic IDs.

```
relics_start = ["gem beet", "frosty fauld"]; // this knight mod will start runs with Gem Beet and Frosty Fauld relics
```

- `unit_b_id`: name (`unit_id`) of the knight's refract ability. B abilities are optional, but if you decide to do one, it should be a separate unit with its name, packed inside its own folder.
- `unit_b_locked`: if the B ability should be locked (`true`) or not (`false`). If set to `true`, the ability will be unlocked by beating the Refraction Realm quandary.
- `unit_a_id`: if this unit is refract (B) ability, this variable should contain the name (`unit_id`) of the original (A) unit.
- `boss_unit`: unit ID of the boss unit to remove from random pools if this knight is chosen (similar to how, for example, King Knight boss won't appear when playing as King Knight).
- `palette_sprite`: the palette to apply to all the knight's sprites. More info.
- `palette_portrait`: the palette to apply to all the knight's portraits.
- `guidebook_statue`: the sprite to display in the Guidebook when all quandaries are beaten.
- `monster_unit`: unit ID of the unit to turn into when wearing Monstrous Montera. Default is `"beeto"`.
- `hat_x_delta`: the x offset (in pixels) for drawing equipped hats. Default is `0`. Can be an array.
- `hat_y_delta`: the y offset (in pixels) for drawing equipped hats. Default is `0`. Can be an array.
- `stache_x_delta`: the x offset (in pixels) for drawing stache/eyes. Default is `0`. Can be an array.

- `stache_y_delta`: the x offset (in pixels) for drawing stache/eyes. Default is `0`. Can be an array.
- `stache_x_prt_delta`: the x offset (in pixels) for drawing stache on the portrait. Default is `60`.
- `stache_y_prt_delta`: the y offset (in pixels) for drawing stache on the portrait. Default is `80`.
- `crown_x_delta`: the x offset (in pixels) for drawing the Versus Mode victory crown. Default is `0`.
- `crown_y_delta`: the y offset (in pixels) for drawing the Versus Mode victory crown. Default is `25`.
- `emote_crown_x_delta`: the x offset (in pixels) for drawing the Versus Mode victory crown during emote animation. Default is `0`.
- `emote_crown_y_delta`: the y offset (in pixels) for drawing the Versus Mode victory crown during emote animation. Default is `25`.

Palettes

Palettes are used to render knight's costumes (that can be changed when bumping into the costume pile in-game).

If you want to allow costumes for your knight mod, you need to make sure the color channel values for your sprites are equal and a multiple of 3.

I.e.:

```
rgb(0, 0, 0)
rgb(18, 18, 18)
rgb(3, 3, 3)
```

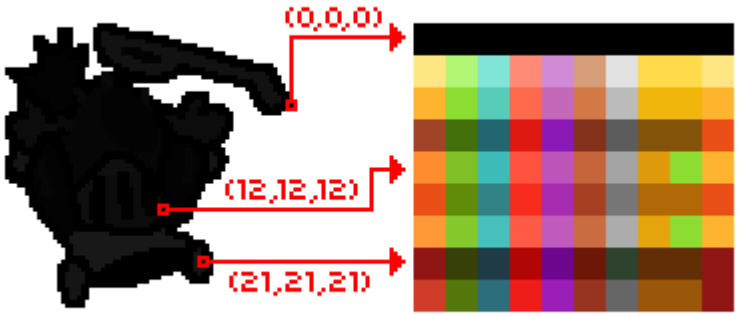
are valid colors, while

```
rgb(2, 2, 2)
rgb(3, 6, 9)
rgb(24, 0, 0)
```

are not.

With these rules applied, your sprites will end up containing only different shades of black, but in game they will be recolored by a shader using your palette sprite.

Each column in the palette is the costume index, and each row is the color for the group of pixels. The firstrow color will be applied to all the `RGB (0, 0, 0)` pixels, the second row - to the `RGB (3, 3, 3)` pixels, and so on.



Note: there's a standalone tool we developed to make creating palletable sprites easier!

You can set up to 10 costumes per unit.

If you are editing an existing unit, providing a palette will override the one it had (if it had any). For instance we can change Shovel Knight's palette by doing:

```
unit_id = "shovel knight" // unit we want to mod
palette_sprite = "new_palette_sprite.png"; // custom palette
palette_portrait = "new_palette_portrait.png"; // custom palette
```

Unit Scripts

These are the heart and soul of your mod! Scripts contain your custom code which is executed if certain conditions are met. Place them in the mod's folder next to `unit.gml`.

- `"create.gml"`: runs when the instance of the unit is created.
- `"input.gml"`: runs every frame right after checking inputs and before any turn or player logic (move/bump) happens.
- `"step.gml"`: runs every game frame (tick).
- `"step_turn.gml"`: runs every turn.
- "`step_act.gml`": runs every turn the player unit attacks or moves. This only works on the player unit.
- `"ability_active.gml"`: runs when the knight unit uses an ability.
- `"ability_aim.gml"`: runs when the knight unit is using an ability that can be aimed. Use `ability_start` to check for requirements for actually using the ability, and `ability_dx` / `ability_dy` to get the direction of the player input:

```
if (ability_start) {// check if can we use ability
      if (hp > 1) { // we can:
```

```
            ability_aim(emote_sprite_index); // enable aiming
      } else { // nay:
            screenshake(5);
            sfx_play(sfxCantDoThat);
            return; // do not forget to exit the script
      }
} else { // a direction was pressed, use ability!
      var _x = ability_dx;
      var _y = ability_dy;
      // ability_dx and ability_dy indicate the direction of the button
press! (if none is pressed, this script will be ignored)

      //...
}
```

- `"ability_aim_draw.gml"`: used for drawing while the knight unit is aiming an ability.
- `"draw.gml"`: should be only used for drawing, runs every game frame (tick).
- `"draw_end.gml"`: used to draw sprites other than the unit's sprite. Everything will be drawn in front of the unit. Runs every game frame (tick).
- `"draw_hud.gml"`: used to draw on the HUD layer. Runs every game frame (tick).
- `"draw_ui.gml"`: used to draw on the layer that is on top of everything else, but below certain popups like the examine one. Useful for drawing information on special abilities similar to Tinker Knight's or Propeller B's. Runs for modded knights only. Runs every game frame (tick).
- `"draw_map.gml"`: used to draw during map transitions. Runs every game frame (tick).
- `"attack.gml"`: called when bumping into other units. Executed once per every unit in a chain. Before the script is run, these local variables are set and can be referenced in your custom code:
    - `victim`: instance ID of the unit being attacked.
    - `attacker`: instance ID of the attacking unit.
    - `is_bumped_unit`: is `true` if the script is run for the first attacked unit in a chain (`victim`), otherwise is `false`.
    - `is_indirect`: is `true` if damage is indirect (damage_indirect), otherwise is `false`. Indirect damage is generally damage not caused by directly bumping into an enemy and it won't pick items or chests.
- `"defend.gml"`: same as `"attack.gml"`, but is run on the victim side. Also provides access to `victim`, `attacker`, and `is_bumped_unit` variables.
- `"hurt.gml"`: same as `"defend.gml"`, but is only executed if damage is received. You can use `damage_taken` variable to reference it. Also provides access to `victim`, `attacker`, and `is_bumped_unit` variables. Will not run if the unit is destroyed by damage, use `"defeat.gml"` for that case.
- `"defeat.gml"`: called when the unit is destroyed by damage.
- `"destroy.gml"`: called when the unit is destroyed, regardless of damage. This is

different from `"defeat.gml"` and works on things like opening chests.
- `"post_attack_loner.gml"`: called after attack.gml when bumping into a single unit. You can use `victim` variable to access its id.
- `"post_attack_chain.gml"`: called after `attack.gml` when bumping into a chain. You can use `victim` variable to access the id of the attacked unit and `chain_size` to access the chain size.
- `"boss_event.gml"`: exclusive to boss units, runs every turn. You can use the following built-in variables inside the script:
    - `boss_timer`: what turn in the boss fight cycle we are in.
    - `boss_phase`: what phase of attacks are we in, starts at `1`.
    - `boss_idling`: is `true` when the boss is idling, when dealt 5 damage it will jump away.

IMPORTANT: using `input.gml` or any variant of `step.gml` scripts for anything outside of pure visual effects might cause desyncs in online mode. Use rpc_call to overcome that.

Unit Functions

unit_is_valid(unit:ref)
unit_is_foe(unit:ref)
unit_info(unit_type)
:unit_create(unit:string, x:int, y:int, …values)
unit_destroy(unit:ref)
:unit_transform(unit:ref, unit_type:string)
:player_transform(unit:ref, unit_type:string)
:unit_get_grid()
unit_freeze(unit:ref, value:int)
unit_poison(unit:ref, value:int)
unit_has_behavior(unit:ref, data:string)
unit_type_has_behavior(data:int, behavior:string)
unit_get_type_int(unit_type:string)
unit_get_type_string(unit_type:int)
:unit_draw()
:unit_draw_end()
:unit_damage(victim:ref, attacker:ref, ...values)
:damage_indirect(victim:ref, attacker:ref, ...values)
:damage_simple(victim:ref, …values)
:damage_player(enemy:ref, …values)
damage_chain_start()
damage_chain_end()
damage_chain_get()
unit_toss(unit:ref, from_unit:ref, ...values)
unit_heal(unit:ref, value:int)
unit_move(unit:ref, x:int, y:int)
:unit_bump(x:int, y:int)
unit_is_in_chain(unit:ref)

unit_get_chain(unit:ref, …values)
unit_get_chain_size(target_unit:ref, max_count:int)
:ability_aim(sprite:int)
:sprite_default()
unit_change_costume(unit:ref, costume:int, ...values)
:unit_animate(sprite, ...values)
unit_jump(unit:ref, …values)
:lob_attack(attacker:ref, gridx:int, gridy:int, damage:int, sprite, …values)

Item Mods

Items are equipables that provide passive or activatable effects if picked up. Unlike relics, only one item can be equipped at a time (Chester B can have 2), and they typically break after a certain amount of uses.

An item mod should be packed inside a folder named "item_{name}" and defined from `item.gml`.

Item Variables

- `item_id`: the ID of the item (a string). You can use the name of one of the existing items to override it, or a unique name for a new item. IDs are better explained in the unit section. For functions like unit_get_type_int and unit_info, custom item IDs have to start with `"item_"`. So, for example, if you've added a new item with an ID of `"super bomb"`, you'd have to use `"item_super bomb"` as an argument for those functions.
- `name`: displayed name (max 15 characters).
- `description`: displayed description (37 characters, the max is 150, but it won't fit the item description when equipped).
- `behaviors`: a list of an item's special traits (or tags!), packed into a string separated by commas.

```
behaviors = "fire_touch:3,attack:0,passive_item" // for example, these are the behaviors of the Flame Axe
```

- `durability`: how many uses it takes for the item to break. Default is `24`.
- `actionable`: if set to `true`, the item will have to be activated with a button press. Default is `false`.
- `sprite`: base item's sprite. See VFX loading. Origin is `48x64px`, size is `96x96px`.
- `portrait`: portrait sprite.
- `chest_rarity`: a number between 1 and 10, defines how often the item will appear in chests relative to the other items in the mod. Set to `0` if you don't want it to appear at all. Default is `1`.

- `king_chest_rarity`: a number between 1 and 10, defines how often the item will appear in king chests relative to the other items in the mod. Set to `0` if you don't want it to appear at all. Default is `1`.
- `reshuffle_chance`: a number between 1 and 10, defines how often the item will appear as a result of reshuffle (when using Juggler's Cap or Transmuting Tammie). Set to `0` if you don't want it to appear at all. Default is `1`.

Item Scripts

All items' scripts are executed from the player's instance.

- `"pickup.gml"`: runs when the player obtains the item.
- `"use.gml"`: runs when the active item is used.
    - `item_aiming`: False by default, if you set it to true it will aim the item. See example below.

```
if(!item_aiming) // begin aiming
{
	item_aiming = true;
}
else
{
	// direction pressed, what to do
	// item_dx, item_dy
}
```

    - `item_dx`: When aiming, what direction we pressed horizontally.
    - `item_dy`: When aiming, what direction we pressed vertically.
- `"bump.gml"`: runs when the player bumps into another unit with the item equipped. Before the script is run, these local variables are set and can be referenced in your custom code:
    - `victim`: instance ID of the unit being attacked.
    - `victim_unit_type`: unit ID of the unit being attacked.
    - `attacker`: instance ID of the attacking unit.
    - `move_x`: indicates the direction of the bump. Can be either `0`, `1` or `-1`. `1` means the player is bumping into something to their right, `-1` - to their left. `0` means the movement is vertical instead (see `move_y`).
    - `move_y`: indicates the direction of the bump. Can be either `0`, `1` or `-1`. `1` means the player is bumping into something to their bottom, `-1` - to their top. `0` means the movement is horizontal instead (see `move_x`).

The rest of the scripts are similar to units' ones and are also executed from the player's instance. You can check the unit scripts section for the detailed description of each:

- `"step.gml"`

- `"step_turn.gml"`
- `"draw.gml"`
- `"draw_end.gml"`
- `"draw_hud.gml"`

When an item is used, you have to decrease the amount of durability in the variable `item_durability`, once it reaches zero it will automatically be unequipped.

Item Functions

:item_create(item:string, x:int, y:int)
unit_is_item(unit_type)
:ban_item(item)
:unlock_item(item)

Relic Mods

Relics are equipable items with passive effects, most commonly obtained from shops.
A relic mod should be packed inside a folder named "relic_{name}" and defined from `relic.gml`.

Relic Variables

- `relic_id`: the ID of the relic (a string). You can use the name of one of the existing relics to override it, or a unique name for a new relic. IDs are better explained in the unit section.
- `name`: displayed name (max 15 characters).
- `description`: displayed description (max 150 characters).
- `description_store`: chester's dialogue line when the player is observing the relic (max 150 characters).
- `cost`: base relic's price (can be further changed by Chester's ability, Extravagant Gem Hat, etc.)
- `weight`: a number between 1 and 10, defines the chance of a relic appearing in Chester's shop (the higher the more chances of it spawning). Default is `10`. Most relics average a weight of `7` or `8`.
- `sprite_icon`: displayed both in shops (frame 0) and on the HUD (frame 1). See VFX loading. Origin is `16x32px`, size is `76x64px` (2 frames).

Relic Scripts

- `"pickup.gml"`: runs when the player obtains the relic.

The rest of the scripts are similar to units' ones and are even executed from the player's instance. You can check the unit scripts section for the detailed description of each:

- `"step.gml"`
- `"step_turn.gml"`
- `"draw.gml"`
- `"draw_end.gml"`
- `"draw_hud.gml"`
- `"attack.gml"`
- `"defeat.gml"`
- `"defend.gml"`
- `"hurt.gml"`
- `"post_attack_loner.gml"`
- `"post_attack_chain.gml"`

Relic Functions

:get_relic_data(relic:string)
:has_relic(relic)
give_relic(relic, ...values)
:ban relic(relic)
:unlock_relic(relic)
:relic_set_price(price:int)
:save_purchase_info()
:shop_item_set(relic:string, shop_item_number:int)
:relic_remove(relic:string)
:relic_remove_all()

Trap Mods

Traps are a special object type different from units. They typically don't move and disappear over time.

A trap mod should be packed inside a folder named "trap_{name}" and defined from `trap.gml`.

Trap Variables

- `trap_id`: the ID of the trap (a string). You can use the name of one of the existing traps to override it, or a unique name for a new trap. IDs are better explained in the unit section.
- `sprite`: idle sprite. See VFX loading. Origin is `16x16px`, size is `768x96px` (8 frames).
- `portrait`: portrait sprite.
- `name`: displayed name (max 15 characters).
- `description`: displayed description (max 150 characters).

- `turns`: number of turns it takes for the trap to disappear. Set to `infinity` if you want it to be persistent. Default is `3`.

Trap Scripts

- `"create.gml"`: runs when the instance of the trap is created.
- `"step.gml"`: runs every game frame (tick).
- `"step_turn.gml"`: runs every turn.
- `"draw.gml"`: should be only used for drawing, runs every game frame (tick).
- `"draw_end.gml"`: used to draw sprites other than the trap's sprite. Everything will be drawn in front of the trap. Runs every game frame (tick).
- `"destroy.gml"`: runs when the instance of the trap is destroyed.
- `"activate.gml"`: runs after any unit steps on the trap. `victim` variable holds the ID of the instance that has activated the trap.

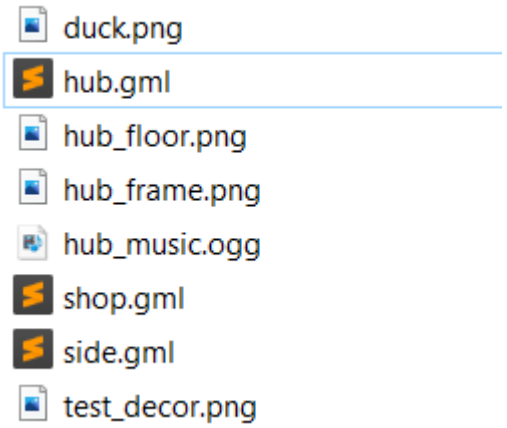
Trap Functions

:trap_create(trap:string, gridx:int, gridy:int)
trap_damage(unit:ref, damage:int)

Prefab Mods

A prefab is a custom-made game screen. Siderooms, bossrooms, hub rooms and shops are all prefabs.

A prefab is defined by a single "{name}.gml" script. Since it's just one file, there's no need to pack it inside a folder, unlike most other mod types. All prefabs are loaded from the "prefabs" folder instead.



This folder includes 3 prefabs: "hub", "shop" and "side".

Prefab script consists of two parts: the first one defines its values, and the second builds the actual prefab when called from the game code.

```
// First call of this script defines the prefab on game start,
consequent build the actual prefab in game
if (prefab_loading) {
      // set prefab properties
      prefab_name = "hub";
      // ...
      return; // exit this section of the script
}

// else create units & decor
unit_create("tree", 1, 1);
// ...
```

As you can see, the first part is wrapped inside the `if (prefab_loading)` block and has a `return;` at the end of it.

You can use the Prefab Viewer to test prefabs you edited or created.

Prefab Variables

- `prefab_name`: the ID of the prefab (a string). You can use the name of one of the existing prefabs to override it, or a unique name for a new prefab. IDs are better explained in the unit section.
- `prefab_is_first`: set to `true` if you want this prefab to take place of the main hub area. Default is `false`.
- `prefab_sideroom`: whether this prefab is a sideroom. Custom siderooms have a chance of appearing when entering portals. Default is `false`.
- `prefab_for_versus`: whether this prefab should appear in Versus Mode. Default is `false`.
- `prefab_shop`: whether this prefab is a shop. Shops have a built-in saving system that prevents already purchased items from spawning again. Default is `false`.
    - `prefab_shop_slots`: the number of items sold in the shop. Default is `0`.
- `prefab_frame`: the HUD sprite to use in the prefab. See VFX loading. Origin is `0x0px`, size is `780x480px`. Default is `"sFrameAdventure"` (See image).
    - prefab_frame_index: image index of the frame sprite. Default is `0`.
- `prefab_floor`: the sprite to tile prefab floor with. Origin is `0x0px`, size is `64x64px`. Default is `"sFloor"` (See image).
    - `prefab_floor_index`: image index of the floor sprite. Default is `0`.
    - `prefab_floor_x` & `prefab_floor_y` : Edits the center of the sprite. By default at 32x32 (top-right corner).
- `prefab_music`: the music to play when entering the prefab. See SFX loading. Default is `"musSideroom"`.

- `prefab_music_is_side`: defines if a music track is “side music” or not. If set to `true`, on exiting the prefab the game will return to the previous track.

Prefab Functions

:prefab_do(prefab_name:string)
:prefab_create_quandary_portal(quandary:int, x:int, y:int)
:prefab_set_player_position(x:int, y:int)
:prefab_create_portal(x:int, y:int, goto:string, …values)
:prefab_decor(sprite, x:int, y:int)
:prefab_decor_ext(sprite, x:int, y:int, depth, ...values)
:prefab_wall(sprite, x:int, y:int)
:prefab_hidden_wall(x:int, y:int)
:prefab_door_light(x:int, y:int, …values)
:sideroom_return_portal_random()
:sideroom_return_portal(x:int, y:int)
:hub_dialogue_reset(unit:ref)

Stage Mods

A stage is essentially a game level. A stage mod can specify what enemies can spawn during the playthrough and how often, level length, siderooms, etc.

A stage mod is defined by `stage.gml` and should be packed inside a “lvl_{name}” folder. The folder should also include a .gml file with spawner code.

Stage Variables

- `lvl_id`: stage ID (a string). You can use the ID of one of the existing stages to override it, or a unique name for a new stage. IDs are better explained in the unit section.
- `lvl_name`: a string, 15 characters max. Displayed on the adventure map and at the top of the HUD when playing a level.
- `lvl_frame`: the HUD sprite to use in the level. See VFX loading. Origin is `0x0px`, size is `780x480px`. Default is `"sFrameAdventure"` (See image).
    - `lvl_frame_index`: image index of the frame sprite. Default is `0`.
- `lvl_frame_vs`: the HUD sprite to use in the level when playing Versus Mode. See VFX loading. Origin is `0x0px`, size is `320x360px`. Default is `"sFrameVS"`.
    - `lvl_frame_vs_index`: image index of the frame sprite. Default is `0`.
- `lvl_floor`: the sprite to tile level floor with. Origin is `0x0px`, size is `64x64px`. Default is `"sFloor"` (See image).
    - `lvl_floor_index`: image index of the floor sprite. Default is `0`.

- `lvl_music`: the music to play when entering the level. See SFX loading. Default is `"musPlains"`.
- `lvl_length`: how long the stage is. Default is `350`.
- `lvl_chests`: how many chests should spawn on the stage. Default is `3`.
- `lvl_sideroom`: the type of sideroom that is guaranteed to spawn during the level. Can be either `"chester"`, `"shrine"` or `"noone"`.
- `lvl_tiles`: the custom tiles sprite used to define the look of the stage on the adventure map. Origin is `0x0px`, size is `1216x96px`.
- `lvl_tiles_row`: the index of the tile row in the tileset. Only applied if the mod is a unique stage and not an override of the existing one. See custom tileset section for more info. You don't need to set this if you've already provided `lvl_tiles`.
- `lvl_map_enemy_sprites`: an array of unit IDs. These define the 4 enemies that will appear on the adventure map around the stage entrance.
- `lvl_map_progress_icon`: icon displayed on the bottom of the Adventure map HUD. See VFX loading. Origin is `16x40px`, size is `43x56px`. Default is `"sPercyBlockLit"` (See image).
    - `lvl_map_progress_icon_index`: image index of the icon. Default is `0`.
- `lvl_map_progress_icon_visited`: displayed if the level is beaten. Origin is `16x40px`, size is `43x56px`. Default is `"sPercyBlockUnlit"` (See image).
    - `lvl_map_progress_icon_visited_index`: image index of the icon. Default is `0`.
- `lvl_spawner`: the name of the .gml file with spawner code.
- `lvl_hazard`: the name of the .gml file with hazard code. This is a script that will run after the stage is generated to spawn custom objects. If you want to use existing hazards, you can set `lvl_hazard` to a corresponding string instead.
- `lvl_hazard_is_legendary`: if the hazards from this stage only spawn in The Legendary Path.

Spawner Code

This code defines what enemies should spawn during the level and their spawn rate. It's recommended you break the file into 3 sections: the last one would be for normal mode, the second one - for the Legendary Path version of the level, and the first one - for both. So enemies defined at the start of the code file will appear for both modes, everything inside `if (lvl_advanced_spawns)` will only be applied to Hardmode, and everything inside the last `else` will take place for Normal mode.

```
// First to declare are bombs
lvl_bomb(0.1);
// other units for both modes
// ...

if (lvl_advanced_spawns) // Legendary Path:
{
      // Foes
```

```
	lvl_unit3("beeto", 2, 3, 1);
	lvl_unit3("boneclang", 1, 0, 2);
	lvl_unit3("blorb", 1, 1, 1);
	// Big unit
	lvl_big_unit2("dozedrake", 1, 0, 0.25);
}
else // Normal mode:
{
	// Heals
	lvl_unit("turkey", 0.1);
	lvl_unit("potion", 1);
	// Blocks
	lvl_unit("dirt block", 0.5);
	// Foes
	lvl_unit("beeto", 2);
	// Grapps unit
	lvl_head_unit("custom grapps", 1, 2);
}
```

If the player has set the level order to “Shuffle All”, you can balance out your stage by checking the `lvl_tier` variable like so:

```
// Tiers are -1, 1, 2 and 3
// Tier -1 is the default stage balance, used when the level order isn't shuffled
// Tier 1,2,3 represent the 3 stages before each boss fight
// Tier 3 is the highest tier
switch (lvl_tier)
{
	case 1: case -1: // Tier 1 and default balance
		// Your spawner code
	break;

	case 2: // Tier 2
		// Your spawner code
	break;

	case 3: // Tier 3
		// Your spawner code
	break;
}
```

Hazards

You can code your own custom hazards in a .gml script, however if you want to use an existing one, you set `lvl_hazard` to any of these presets:

“`cauldrons`”: Pridemoor Keep’s cauldrons.
“`lights` out”: Lich Yard’s blackouts.
“`magic` landfill”: Magic Landfill setups (bombable blocks) and magic floor.
“`water`”: Iron Whale’s water
“`lava`”: Lost City’s lava
“`crystal` mirror”: Crystal Caverns legendary path hazard.
“`conveyor` belts”: Clockwork Tower’s conveyor belts.
“`spikes`”: Stranded Ship’s spikes.
“`wind`”: Flying Machine’s turbines.
“`burners`”: Explodatorium’s burners.

Stage Functions

:stage_get_unit_list()
:stage_get_random_unit_type()
:stage_skip_turn()
:lvl_get_spawner_code(level)
:lvl_remove_unit(unit)
:lvl_bomb(rate:float)
:lvl_unit(unit:string, rate:float)
:lvl_unit2(unit:string, rateA:float, rateB:float)
:lvl_unit3(unit:string, rateA:float, rateB:float, rateC:float)
:lvl_big_unit(unit:string, limit:int, rate:float)
:lvl_big_unit2(unit:string, limit:int, rateA:float, rateB:float)
:lvl_big_unit3(unit:string, limit:int, rateA:float, rateB:float, rateC:float)
:lvl_grapps_unit(unit:string, limit:int, rate:float)
:lvl_grapps_unit2(unit:string, limit:int, rateA:float, rateB:float)
:lvl_grapps_unit3(unit:string, limit:int, rateA:float, rateB:float, rateC:float)

Modifier & Hat Mods

A modifier is a set of rules applied to a game run, activated by either a quandary or by wearing a hat.

A modifier mod should be packed inside a folder named "modifier_{name}" and defined from `modifier.gml`.

Modifier Variables

- `modifier_id`: the ID of the modifier (a string). You can use the name of one of the existing modifiers to override it, or a unique name for a new modifier. IDs are better explained in the unit section. This ID can be further used in quandary mods.
- `is_hat`: whether a hat should be created on top of this modifier. If set to `true`, the hat will have a chance to appear for sale in Mr. Hat's shop. Default is `false`.

The rest of the variables are only applied if `is_hat` is `true`:

- `sprite`: hat sprite. See VFX loading. Origin is `24x24px`, size is `48x48px`.
- `name`: displayed name (max 15 characters).
- `description`: displayed description (max 150 characters).
- `chaos`: how much the hat changes the chaos meter if equipped. Can be negative. Default is `0`.
- `height`: how tall the hat is in pixels, needed so that other hats can draw properly on top of it. Default is `16`.
- `hat_y`: how much to offset the hat vertically when drawing it. Default is `0`.
- `price`: hat's price in Mr. Hat's shop. Default is `20000`. If set to `0`, the hat will be auto-unlocked from the start.
- `vs_banned`: if the hat should be banned in VS mode. Default is `true`.
- `helmet_unit`: you can provide the unit ID of any knight if you want the hat to change player appearance (similar to "Helm of Shovelry" and others). Default is `noone`.

Modifier Scripts

These scripts are called form the grid object:

- `"grid_turn.gml"`: runs every turn.
- `"move.gml"`: runs after the player has moved (attacked/using an ability).
- `"idle.gml"`: runs after the turn was skipped due to the timer.
- `"grid_step.gml"`: runs every game frame (tick).
- `"grid_draw.gml"`: should be only used for drawing, runs every game frame (tick).
- `"grid_draw_hud.gml"`: used to draw on the HUD layer. Runs every game frame (tick).

These scripts are called from every unit instance in the game:

- `"unit_draw.gml"`: should be only used for drawing, runs every game frame (tick).
- `"unit_draw_end.gml"`: used to draw sprites other than the unit's sprite. Everything will be drawn in front of the unit. Runs every game frame (tick).

These scripts are similar to units' ones and called from the player instance. You can check the unit scripts section for the detailed description of each:

- `"attack.gml"`
- `"defend.gml"`
- `"defeat.gml"`
- `"hurt.gml"`
- `"post_attack_loner.gml"`
- `"post_attack_chain.gml"`

Modifier Functions

:hat_find(hat)
:ban_hat(hat)
:unlock_hat(hat)
:has_modifier(modifier:string, …values)
:has_hat(modifier:string)
:add_modifier(modifier:string, …values)
:remove_modifier(modifier:string, …values)

Quandary Mods

A quandary is a custom gamemode set up by a list of modifiers. Quandaries can be only accessed from portals in the `"bean palace"` prefab.

A quandary is defined by a single “{name}.gml” script. Since it’s just one file, there’s no need to pack it inside a folder, unlike most other mod types. All quandaries are loaded from the “quandaries” folder instead.

Quandary Variables

- `name`: quandary name displayed when entering its portal (max 15 characters).
- `description`: quandary description displayed when entering its portal (max 150 characters).
- `skill_a_only`: if set to `true`, forces character’s base ability (A skin). Default is `false`.
- `quandary_knight`: unit ID of the knight this quandary is for.
- `quandary_modifiers`: an array of modifier IDs the quandary will apply.
- `music`: custom music to play during the quandary (Marathon only).

```
quandary_modifiers = ["double HP", "pandemic"]; // modifiers used in
Plague Knight's Character Quest
```

- `quandary_index`:
  - `0`: Legendary Path
  - `1`: Refraction Realm
  - `2`: Character Quest
  - `3`: Trial of Offerings
  - `4`: Marathon

  You can use values above 4 for custom quandaries. To create hub portals for these new quandaries, call `prefab_create_quandary_portal`:

```
if (prefab_loading) { // define the prefab (don't build it yet)
      prefab_name     = "bean palace";
      prefab_is_first = true;
      prefab_sideroom = false;
      return; // exit
}

prefab_do("bean palace"); // make the game build original bean palace

// add new quandary portal at x: 2, y: 6
prefab_create_quandary_portal(5, 2, 6);
```

To check if the game is currently in your modded quandary, you can call get_quandary_index:

```
controller_run_start.gml:
if (get_quandary_index() == 5) give_relic("divine liquid"); // give the
player relic on quandary start
```

Controller Object

Each mod comes with a custom controller object that gets created as soon as the mod is loaded and is persistent throughout the whole game. You can use this object to save and load data, grant the player relics on run start, track the number of deaths, etc.

Controller's scripts have to be placed in the core folder of the mod, next to `mod_info.ini`.

- `"controller_game_start.gml"`: runs as soon as the mod is loaded.
- `"controller_run_start.gml"`: executed when the Adventure run starts.
- `"controller_input.gml"`: runs every frame right after checking inputs and before any turn or player logic (move/bump) happens.
- `"controller_begin_step.gml"`: runs every game frame (tick) before most of the other code.
- `"controller_step.gml"`: runs every game frame (tick).
- `"controller_step_turn.gml"`: runs every turn.
- `"controller_end_step.gml"`: runs every game frame (tick) after most of the other code.
- `"controller_draw.gml"`: should be only used for drawing, runs every game frame (tick).
- `"controller_draw_end.gml"`: should be only used for drawing, runs every game frame (tick). This runs on the draw end event.
- `"controller_draw_hud.gml"`: should be only used for drawing, runs every game frame (tick). Draws on the HUD layer.

- `"controller_draw_gui.gml"`: should be only used for drawing, runs every game frame (tick). Draws on top of the normal HUD layer that is unaffected by the view position, scale or rotation.
- `"controller_draw_gui_end.gml"`: should be only used for drawing, runs every game frame (tick). Runs after normal draw GUI, use this script to ensure you're drawing on top of *everything.*
- `"controller_stage_start.gml"`: runs when the stage (level) starts.
- `"controller_stage_end.gml"`: runs when the stage (level) ends.
- `"controller_player_death.gml"`: runs on player's death.
- `"controller_good_ending.gml"`: runs if the good ending is triggered.
- `"controller_bad_ending.gml"`: runs if the bad ending is triggered.
- `"controller_custom.gml"`: can be run from other mods using mod_call().

Modifying Level Order

You can change the order of levels and bosses in Adventure by including `stage_order.gml` file in your mod's core folder.

- `lvl_order` is an array of Stage IDs in the order they'll appear in game. You can modify the whole array, but if you just want to add your level to the default order, you can do

```
lvl_order[_position] = "your custom stage";
```

- `lvl_order_vs` is an array of Stage IDs for Versus mode.
- `lvl_bosses` is an array of Boss IDs, also ordered. `-1` sets no boss for current stage.
- `banned_siderooms` is an array of Sideroom IDs that you don't want to spawn during Adventure. You can also use ban_sideroom and unlock_sideroom functions to change this array during gameplay.
- `shortcuts` is an array of Stage IDs that will appear in the Percy's cannon room. Only 10 first entries will be processed. The array can be shorter than 10, though - in that case remaining stages will stay default.
- `disable_shortcuts` is a boolean that determines whether you can use Percy's cannon or not when using this mod.
- `adaptive_levels` is a boolean that determines if levels should adapt in difficulty based on their position in the level order (base game does this if "Shuffle All" setting is enabled).

Note: you can include arrays inside both `lvl_order` and `lvl_bosses`. In that case, the game will choose a random item from the array for that position.

```
lvl_order = [
```

```
      "plains"
      ,["pridemoor keep", "explodatorium"] // the game will choose
random one from the array
      ,"lich yard"
      ,"magic landfill"
      ,"iron whale"
      ,"crystal caverns"
      ,"clockwork tower"
      ,"stranded ship"
      ,"scholars sanctum"
      ,"tower of fate"
];

// this replaces the first level in Versus with a custom stage:
lvl_order_vs[0] = "my custom stage";

lvl_bosses = [
      -1
      ,-1
      ,-1
      ,-1
      ,-1
      ,["tinker boss", "mole boss", "treasure boss", "scrap boss"] //
the game will choose random one from the array
      ,-1
      ,["propeller boss", "polar boss", "prism boss"]
      ,"ycg boss"
      ,"enchantress boss"
];

// siderooms that won't appear:
banned_siderooms = ["sideroom2", "sideroom ghost"];

// shortcuts that will appear in the percy's room:
shortcuts = ["my custom stage"]; // here we replace plains with a custom
stage

// can't use percy's cannon, by default is false
disable_shortcuts = true;
```

You can also alter level order by calling level_set.

Custom Tileset

You can modify the look of the Adventure map by changing the tileset. Here's the original one. You can place your custom version of it in the mod's core folder and it'll be loaded automatically.

If you want to set what set of tiles a stage uses (i.e. make your custom stage look like Plains), use the `lvl_tiles_row` property. If you want to load custom tiles for your stage, use the `lvl_tiles` variable instead.

Note: `lvl_tiles_row` is not exactly the number of the tile row, but rather the number of the pair of rows (since every level set takes 2 rows). You can use the numbers below level names in the file as a reference (to access plains' set of tiles you would use `lvl_tiles_row` of `1`, for pridemoor keep - of `2`, etc.)

Boss Mods

A boss is technically a unit with a `boss` behavior. It has to be defined from a `unit.gml` file, but unlike a unit mod, should be packed inside a "boss_{name}" folder. The folder should also include a prefab of the boss room named `prefab.gml`. Make sure the prefab spawns the boss unit!

To make this boss room actually appear during adventure, you need to add it to the `lvl_bosses[]` array inside `stage_order.gml` file. See modifying level order.

These `unit.gml` variables are boss-specific:

- `is_boss`: if set to `true`, will mark the unit as boss.
- `boss_second_phase`: a unit ID of the second stage boss unit. If set, this variable allows to create boss fights that consist of multiple stages (similar to Tinker Knight's Mech). This unit should be a proper boss unit, packed inside the "boss_{name}" folder with its own prefab (and it can even have its own `boss_second_phase`!).
- `boss_ending`: if set, defeating the boss will trigger ending cutscene: `0` = good ending, `1` = bad ending. `boss_second_phase` will be ignored if the cutscene is triggered.
- `sprite_mini`: this sprite is used in the progression bar to indicate that the level has a boss. Origin is `16x16px`, recommended size is `32x32px`.

Boss units also have access to a `boss_event.gml` script. See unit scripts for more info.

Current in-game boss IDs.

Boss Functions

boss_create_intro(unit:ref, dialogue)
boss_get_for_level()

Custom Grapps

Grapps are special because they consist of two units: a head and a body.

Head instance:

- has a `"head"` behavior;
- `is_head` variable is `true`.

Body instance:

- has `"body"` and `"bombproof"` behaviors;
- `is_body` variable is `true`;
- `head` variable holds the instance ID of the head unit.

To create a new custom grapps unit, make sure you set `is_grapps = true;` from `unit.gml`. You might also want to set `body_description`, `sprite_head` and `sprite_body` variables. The body unit will be defined automatically for you - its unit ID is the unit ID of the head unit + `" body"`. So if your unit ID is `"cool grapps"`, the body can be accessed by the `"cool grapps body"` ID. Another way to access it is to use `unit_type + 1`:

```
var _tail = unit_create(unit_type + 1, gridx, gridy);
```

Grapps spawn rate can be defined by lvl_grapps_unit functions.

Custom Shops

To create a custom shop, first mark a prefab as one and specify the number of items sold:

```
prefab_shop       = true;
prefab_shop_slots = 3;
```

Then you can use `item_create()` to occupy those slots.

```
with (item_create("shop relic", 4, 4)) { // base relic
      it = get_relic_data("meal ticket");
      shop_item_number = 0;
      shop_item_price = relic_set_price(20000);
}
```

`"shop relic"` is a specific unit ID designed for sold relics. Let's look at the variables it's assigned:

- `it` - relic ID, `get_relic_data()` is used to set it. This can be a modded relic!
- `shop_item_number` - the index of a shop slot this item occupies, starting at `0`.
- `shop_item_price` - the price of the item. For relics, it's set using `relic_set_price()` so the number can be further changed by chester's ability, Extravagant Gem Hat and other modifiers.

Shop can also sell items:

```
var _item = choose("steel blade", "wood blade"); // random base item
with (item_create(_item, 5, 4)) {
      shop_item_number = 1;
      shop_item_price = 2000;
      is_sold = true;
}
```

You can use choose() and other random functions inside the shop prefab - the item will be only picked once at the first run of the script and saved between the shop visits. Also note that item ID is directly tossed as a function argument. And the `is_sold` variable has to be set to `true` (since items are created as free pickups by default).

Finally there's a third type of objects the shop can sell:

```
with (item_create("shop upgrade", 3, 4)) { // non-relic non-item
      shop_item_number = 2;
      shop_item_price = "2 HP";
      shop_purchase_condition = function() {
            return true; // always purchasable
```

```
	}
	on_purchase = function() {
		unit_damage(get_grid_master().player, me(), 2); // damage
the player
		add_gems(2000);
		sfx_play(sfxBlessingBought);
		save_purchase_info(); // make sure the item doesn't appear
again when the player re-visists the shop
		instance_destroy();
	}
	name = "Blood-Sucking Duck";
	desc = "Get 2000 gems, but take 2 damage";
	sprite = "duck";
}
```

Shop upgrades are designed to give certain effects as they are purchased, and they are neither relics nor items. To set them up, you can use `shop_purchase_condition` and `on_purchase` variables. Both are functions: the first one is called to check if the upgrade can be purchased, and the second one is called after it's bought by the player.

Note that we call `save_purchase_info();` inside the `on_purchase()` function - it's used to save the fact the upgrade was purchased and to make sure it doesn't appear on the second and consequent shop visits.

Saving & Loading

Mods can store data between game launches! This can be used to track mod-specific stats, create unlockable relics or even build your own progression system.

All custom data is stored inside the original save file, so it's synced between devices using Steam Cloud, but can be reset by the player at any time. Everything is saved per mod, meaning only your mod can read and write to its own save file section.

It's recommended you use a controller object to manage saving and loading since it's persistent throughout the whole game. Here's an example on how to track player's deaths:

`controller_game_start.gml`:

```
global.deaths = savedata_read("deaths", 0); // 0 is the default value in
case no data is found
```

`controller_player_death.gml`:

```
if (is_adventure()) savedata_save("deaths", ++global.deaths);
```

You can also read and write to any number of text files inside your mod's folder. Here's an example of saving the same death data to a text file:

```
var _file = file_text_open("data\\save.txt", false); // opens the file
from the "data" folder for writing
// the whole path would look something like
// \AppData\Roaming\Yacht Club Games\Shovel Knight Pocket
Dungeon\{your_steam_ID}\mods\{your_mod_ID}\data\save.txt
if (_file != -1) { // if the file was opened successfully
      file_text_write_real(_file, global.deaths);
      file_text_close(_file); // close the file to prevent memory leaks
}
```

Text Formatting

You can use scribble's text formatting for some strings:

- Mod description
- Unit description
- Relic description
- Dialogue lines
- Strings drawn with scribble_draw and scribble_draw_outline

Scribble Button Tags

These tags will be automatically replaced with corresponding button icons:

`[ActionButton]`
`[AbilityButton]`
`[ExamineButton]`
`[ItemButton]`
`[TurboButton]`
`[EscapeButton]`
`[UpButton]`
`[LeftButton]`
`[RightButton]`
`[DownButton]`

VFX and SFX loading

Certain variables from init scripts (i.e. `unit.gml`) are used to define sprites and sound effects (i.e. `sprite_idle` or `dialogue_beep`). You can do that in two ways:

- If you want to use sprite or sound asset already present in the base game, you can just provide its name:

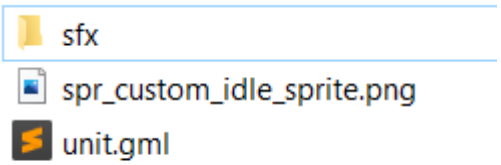
```
sprite_idle = "spr_mona_b_idle";
dialogue_beep = "sfx_text_reveal";
```

  You can use the in-game VFX/SFX viewer tool for a list of sprites and sounds assets available.

- If you want to use a custom sprite or sound, provide a relative path to it, including the extension:

```
sprite_idle = "spr_custom_idle_sprite.png";
dialogue_beep = "sfx/custom_beep.ogg";
```

  In this example we're in the unit mod folder:



  (`custom_beep.ogg` file if located inside the `sfx` folder).

Note: all sprites should be in `.png` format and all sounds in `.ogg` format.

Note: if the sprite you are loading is animated, you can place the subimages in a strip next to each other. You can then specify the number of frames by changing the `sprite_{animation_name}_frames` variable from the init script (i.e. `unit.gml`):

```
sprite_idle = "spr_googly_beeto";
sprite_idle_frames = 4;
```

Note: as you can see in the above image, the sprite is placed on a green background, which is then removed when the sprite is loaded to the game. This works by checking the color of the bottom left pixel. The alpha channel of the source image is also ignored, meaning all pixels other than the "background color" ones become fully opaque.

Objects

oUnit

Object for all units in the game and handles all the logic for each type of unit (foes, blocks, knights…). This is the most common object in the game.

Status variables

- `hp (int)` - current HP of the unit.
- `hpmax` (`int`) - current max HP of the unit.
- `atk` (`int`) - current attack power of the unit. This resets after each attack, so you will need to adjust this inside damage scripts.
- `item_dx (int)` - additional attack power for this unit's next attack.
- `frozen` (`int`) - How many turns this unit will be frozen. Frozen units don't fight back, fall or do their behaviors.
- `poison` (`int`) - How many turns this unit will be poisoned. Each turn loses 1HP.
- `invincible` (`int`) - How many turns this unit is immune to damage.
- `invincible_side` (`array`) - How many bumps this unit is invincible from damage in a certain direction. This is the goldarmor behavior. (i.e. `[1,0]` means you can't attack it from the left, while `[-1,0]` means you can't attack from the right)
- `electrified` (`int`) - How many turns this unit is electrified. Electrified units are immune to damage.
- `has_balloon` (`bool`) - Make this unit float and avoid hazards. This is the effect from Nimbus Balloon.
- `unit_id` (`string`) - unit ID of this unit.
- `unit_type (int)` - unit ID, but converted into a number. To get an int from a string, you can use unit_get_type_int.

Useful variables

- `gridx` (`int`) - current x position in the grid.
- `gridy` (`int`) - current y position in the grid.
- `grid_master` (`id`) - reference to the grid that holds this unit.
- `static_in_grid (bool)` - prevents unit from falling down.
- `prevent_falling_for_x_turns` (`int`) - temporarily prevents unit from falling down for X turns.
- `make_money_for_player` (`bool`) - Whether the unit gives gems or not. Foes mark this as false and the player marks it as true. It is true by default.
- `make_money_for_player_ever` (`bool`) - Whether the unit gives gems or not, regardless of who took it down.
- `dialogue` (`string`) - Dialogue to play when this unit is bumped. A unit with dialogue won't be hurt.
- `dialogue_loop` (`array`) - Array of dialogues to loop through when bumped. A unit with dialogue won't be hurt. It is noone by default.
- `keys` (`int`) - How many keys this unit has picked up.
- `item` (`int`) - Equipped item, if no item is equipped it is set to -1.
    - `item2` - for Chester B's second item.
- `item_durability` (`int`) - Durability of equipped item. If it reaches zero the item is dropped.
    - `item_durability2` - for Chester B's second item.
- `item_aim_anyway` (`bool`) - If set to `true`, the player will ignore common checks for if you can or can't use an item.
- `push_attack` (`bool`) - If set to `true`, simulates Goo Bumpin' Bonnet effect on the player.

Logic variables

- `player` (`int`) - Whether this unit is a player or not. If it is 1, it is player 1 and if it is 2, it is player 2.
- `stun` (`int`) - How many turns the player's controls are stunned.
- `hits_back` (`bool`) - Whether this unit fights back when attacked.
- `is_modded` (`bool`) - Whether this unit is made from a mod or edited from a mod.

Behavior variables

- `is_grapps` (`bool`) - Whether this unit is a "grapps type" of unit. Whether it is a head or a body.
    - `is_head` (`bool`) - If it is the head of a "grapps type".
    - `is_body` (`bool`) - If it is the body of a "grapps type".
    - `head` (`id`) - Reference to the head of a "grapps type".
- `is_enemy` (`bool`) - Whether this unit is an enemy or not.
- `is_knight` (`bool`) - Whether this unit is a knight or not. Knights are the units the player can play as.
- `is_big` (`bool`) - Whether this unit is big or not.

- `is_boss` (`bool`) - Whether this unit is a boss or not.
- `is_block` (`bool`) - Whether this unit is a block or not.
- `is_wall` (`bool`) - Whether this unit is a wall or not.
- `is_explodes` (`bool`) - Whether this unit is an explosive or not.
- `is_tangle` (`bool`) - Whether this unit sticks together with other units of the same type.
- `is_inert` (`bool`) - Whether this unit is inert or not.
- `is_corpse` (`bool`) - Whether this unit is a corpse or not.
- `is_heal` (`bool`) - Whether this unit heals on pickup.
    - `heal_value` (`int`) - Amount of heal when picked.
- `is_moveable` (`bool`) - Whether this unit can be moved or not.
- `is_npc` (`bool`) - Whether this unit is an NPC or not.
- `is_bombproof` (`bool`) - Whether this unit is immune to explosions or not.
- `is_bomb` (`bool`) - Whether this unit is a bomb or not. Ratsploders for instance are explosive but don't count as bombs.

Sprite variables

- `idle_sprite_index` (`sprite`)
- `weak_sprite_index` (`sprite`)
- `emote_sprite_index` (`sprite`)
- `out_sprite_index` (`sprite`) - Emote Out sprite.
- `dying_sprite_index` (`sprite`)
- `dead_sprite_index` (`sprite`)
- `special_sprite_index` (`sprite`)
- `charge_sprite_index` (`sprite`)
- `aim_sprite_index` (`sprite`)
- `skill_sprite_index` (`sprite`)
- `skill_up_sprite_index` (`sprite`)
- `skill_down_sprite_index` (`sprite`)
- `other_sprite_index` (`sprite`)
- `fly_sprite_index` (`sprite`)
- `fall_sprite_index` (`sprite`)

Portrait variables

- `portrait` (`sprite`) - Current portrait.
- `portrait_idle` (`sprite`) - Standard portrait for this unit.
- `portrait_hurt` (`sprite`) - Portrait when unit is hurt.
- `portrait_win` (`sprite`) - Portrait when unit wins in Versus, beats a level or used in boss intros.

Drawing variables

- `nudgex` (`float`) - X offset that lerps to zero.

- `nudgey` (`float`) - Y offset that lerps to zero.
- `z` (`float`) - Z position. A different Y offset that has gravity.
- `zspd` (`float`) - Z speed.
- `zgrav` (`float`) - Z gravity, by default 0.75.
- `zbounce` (`int`) - How many times the Z offset will do a bounce effect when reaching zero.
- `name_override` (`string`) - Changes the unit's name.
- `desc_override` (`string`) - Changes the unit's description.
- `whiteflash_unit` (`int`) - For how many turns this unit will be drawn entirely white.
- `is_sparkle` (`bool`) - If set to true, the unit emits sparkles.
- `shadow_draw` (`bool`) - Whether a shadow is drawn or not for this unit. This can be reset back to true when doing certain actions.
- `shadow_draw_ever` (`bool`) - Whether a shadow is drawn ever or not. It doesn't get reset.
- `depth_add` (`int`) - Added depth to this unit.

oGrid

Object that controls the current level/room and its grid. The grid is where all units reside and where turn logic and player input is processed.

Important variables

- `grid` (`ds_grid`) - Data structure holding data for each cell in the game. It is a 8x9 grid.
    - **IMPORTANT:** The size of the grid is fixed, attempting to increase or decrease its size won't work well.
- `player` (`ref`) - Reference to the player's oUnit.
- `turn_timer` (`int`) - How many steps/frames left for the next turn to trigger naturally. When it reaches zero, it resets.
- `turn_timer_base` (`int`) - How long a turn is in steps/frames. By default it is 60, however Adventure settings and some hats can change this.
- `next_spawn_counter` (`int`) - How many turns for the next unit spawn.
- `progress` (`float`) - Current progress in the stage, defeating units increases this by 3 by default.
- `progress_required` (`float`) - Total amount of progress needed to beat the level.
- `spawn_enemies` (`bool`) - Whether units are spawning or not.

Informative variables

- `level_id` (string) - This is the stage ID of the current level.
- `level_idx` (int) - This is the current stage number we are in. This refers to the stage order and is not related to the stage id.

- `specific_prefab_loaded` (`string`) - The current prefab ID.
- `previous_room_loaded` (`string`) - The previous prefab ID. “” if none.
- `turn_number` (`int`) - What turn are we in.
- `player_id` (`int`) - Whether this grid is Player 1 or 2.
- `versus_mode` (`bool`) - whether we are playing versus or not.
- `player_is_respawning` (`bool`) - whether the player is respawning or not. This is true during the entire duration of the player floating to the top of the grid.
- `loop_count` (`int`) - How many times we looped this run. Keep in mind that changing this will only affect the current oGrid and not the overall run.
- `relicless` (`bool`) - Whether relics can spawn. False by default, meaning relics can spawn.
- `can_spawn_chester` (`bool`) - Whether Chester’s chest can spawn or not.
- `row_spawner` (`bool`) - Whether units will spawn in rows.
- `down_spawner` (`bool`) - Whether units will spawn from below.
- `buried_mode` (`bool`) - Whether the grid is full at all times or not.
- `traffic_mode` (`bool`) - Whether this mode is active or not.
- `traffic_red_light` (`bool`) - If red light is active, meaning you cannot move but units start spawning.
- `is_arcade_mode` (`bool`) - Whether you are playing Marathon or not.
- `ceiling_crusher` (`id`) - Reference to the Ceiling Crusher object. Noone by default.
- `cpu` (`bool`) - Whether this grid is controlled by the CPU.
- `entering_portal` (`bool`) - Whether we are entering a portal or not.
- `time_frozen` (`int`) - How long the grid will be frozen. (Chronos Coin or Chronos Glass)

Input variables

- `kUp` (`bool`) - Pressed up.
- `kUpHold` (`bool`) - Holding up.
- `kDown` (`bool`) - Pressed down.
- `kDownHold` (`bool`) - Holding down.
- `kLeft` (`bool`) - Pressed left.
- `kLeftHold` (`bool`) - Holding left.
- `kRight` (`bool`) - Pressed right.
- `kRightHold` (`bool`) - Holding right.
- `kItem` (`bool`) - Pressed item.
- `kSpecial` (`bool`) - Pressed ability button.
- `kSpecialHold` (`int`) - How many frames we’ve held the ability button.
- `kSpeed` (`bool`) - Holding turbo.
- `kSpeedPressed` (`bool`) - Pressed turbo. When pressed it is set to 15 as a cooldown before `kSpeed` can work.

Drawing variables

- `floor_sprite` (`sprite`) - Sprite for the grid's floor.
- `floor_index` (`int`) - Index of floor_sprite.
- `frame_sprite` (`sprite`) - Sprite for the grid's frame.
- `frame_index` (`int`) - Index of frame_sprite.

oTrap: trap object

Object that controls all floored hazards (i.e. puddles, spikes, conveyors…). Unlike oUnit, oTrap isn't assigned a spot in the grid but does read the grid to see which oUnit they affect.

Variables

- `trap_id` (`string`) - Trap ID of this trap.
- `grid_master` (`id`) - Reference to the board's `oGrid`
- `base_image_speed` (`float`) - Overwrites `image_speed`. Use this to adjust a trap's image speed instead.
- `turns` (`int`) - How many turns will this trap live for.
- `delay` (`int`) - How many turns before this trap becomes active. This is useful if you spawn a trap below you and do not want to be immediately affected by it.
- `active_trap` (`bool`) - Whether the trap is active or not.
- `expiring` (`bool`) - Whether the trap is expiring or not. When this is true the trap fades out for 5 frames before getting destroyed.
- `offscreen` (`bool`) - Whether the trap is offscreen or not. This happens when the player enters a sideroom and the trap is moved out of the screen temporarily.

oModController: controller object

Each mod creates its own `oModController` that you can use to make more advanced mods. This object persists during the entirety of the game's session and is completely blank by default.

Other objects

These objects control other areas of the game and aren't as common. You can access and edit them, however they serve very specific functions so the lengths you can edit them are very limited.

- **oGib:** Particle effect with gravity. Calling gib_play makes an instance of this object.
- **oPop**: Static VFX or particle effect. Calling vfx_play makes an instance of this object.

- **oProjectile**: Object used in some boss fights to make projectile attacks (i.e. snowballs, fireballs…).
- **oJumper**: Object used when oUnit jumps to a different spot in the grid. It temporarily desync the unit from the grid to place it at the end of the jump.
- **oLobber**: Object used when an attack is launched in an arc pattern (similar to units jumping). Calling lob_attack makes an instance of this object. Examples of this are Tinker Knight's wrenches and Polar Knight's spinwulfs.
    - `completion` (function) - Assign a function here if you want something to happen at the end of the attack.
    - `stepping` (function) - Assign a function here if you want something to happen each step.
    - `hit_sfx` (sound) - Assign the sfx to play when hitting something. sfxSoilSteelDie by default.
    - `end_sfx` (sound) - Assign the sfx to play when the attack ends, whether it hit or not. Noone by default.
    - `gib_sprite` (sprite) - Assign a sprite to play when the attack ends, sNoone by defuault.
- **obj_terrorpin**: used for terrorpin drawing and logic.

Macros

Note: you can use all the-built in GML variables and constants in your mods.

`"GRID_WIDTH"`: the width of the grid in tiles (8)
`"GRID_HEIGHT"`: the height of the grid in tiles (9)
`"c_combo_blue"`: 0xdd5e4b
`"c_combo_green"`: 0x407b23
`"c_combo_yellow"`: 0x2733b9
`"c_combo_pink"`: 0x721a9e
`"c_combo_blue_light"`: 0xe4ef6b
`"c_combo_green_light"`: 0x5eedb9
`"c_combo_yellow_light"`: 0x50c3ed
`"c_combo_pink_light"`: 0xc2a4fe
`"font_main"`: the main, bold font. Set by default when calling draw_text functions.
`"font_tiny"`: the smaller font. You can set it by calling draw_set_font.

Functions

Note: you can use all the built-in GML functions in your code.

`":"` before the function name means it can access local variables of the instance it's called from.

`"…values"` means the function can further take optional arguments.

Argument Types

`:bool`	Boolean
`:int`	64-bit integer
`:float`	64-bit double-precision floating point value
`:string`	String
`:ref`	Reference to the instance (instance ID)

If no type is provided, an argument can be any type.

Saving & Loading

:savedata_save(key:string, value:int)
:savedata_read(key:string, ...values)
file_text_open(filename:string, read:bool)

Grid Functions

:grid_pixel_x(x:int)
:grid_pixel_y(y:int)
:grid_get(x:int, y:int)
:grid_set(x:int, y:int, value)
cell_is_valid(x:int, y:int)
trigger_grid_death(grid:ref)

VFX Functions

sprite_add(sprite_name:string, filename:string, image_number:int, ...values)
sprite_get(sprite_name:string)
sprite_replace(sprite_name:string, filename:string, image_number:int, …values)
sprite_get_og(sprite_name:string)
:vfx_play(sprite, unit:ref)
:vfx_play_at(sprite, x:int, y:int)
:gib_play(sprite, unit:ref)

SFX Functions

sfx_add(sfx_name:string, filename:string)
sfx_get(sfx_name:string)
:sfx_play(sfx)
:sfx_play_ext(sfx, …values)

:sfx_stop(sfx)
:sfx_is_playing(sfx)
:sfx_pause(sfx)
:sfx_resume(sfx)
:sfx_is_paused(sfx)
:sfx_set_volume(sfx, volume:float, time:int)
:sfx_get_volume(sfx)

Unit Functions

Item Functions

Relic Functions

Prefab Functions

Stage Functions

Modifier Functions

Other Helpers

print(...values)
rpc_call(both_parties:bool, function:string, ...values)
:mod_call(file_id:string)
:mod_is_enabled(file_id:string)
file_text_open(filename:string, read:bool)
log_clear()
get_seed()
draw_text_outline(x:int, y:int, text:string, …values)
scribble_draw(x:int, y:int, text:string, …values)
scribble_draw_outline(x:int, y:int, text:string, …values)
scribble_set_wrap(max_box_width:int, max_box_height:int, ...values)
draw_key(x:int, y:int, input:string, …values)
:draw_sprite_palette(sprite, image_index:int, x:int, y:int, skin:int)
:draw_sprite_palette_ext(sprite, image_index:int, x:int, y:int, skin:int, …values)
:dialogue_define_start()
:dialogue_define_end()
:dialogue_define(unit:int, accent:int, text:string, ...values)
:dialogue_play(...values)
:dialogue_proceed()
is_hub()
is_adventure()
is_versus()

is_title()
:is_shop()
:is_sideroom()
:is_at_fight_zone()
:is_minigame()
game_is_paused()
menu_get()
:timed_chest_tick()
:frozen_time_tick()
:check_gems()
:add_gems(gems:int, …values)
:remove_gems(gems:int, ...values)
quandary_get_status(unit_type, quandary:int)
get_quandary_index()
:explosion_create(unit_exploding:ref, range:int, damage:int)
:explosion_create_ext(unit_exploding:ref, damage:int, left:int, top:int, right:int, bottom:int, ...values)
:explosion_freeze(unit_exploding:ref, range:int, freeze:int, affects_player:bool)
:explosion_fire(unit_exploding:ref, range:int, damage:int)
:create_warning_tile(x:int, y:int, unit:ref)
remove_warning_tiles(unit:ref)
screenshake(value:int)
:chain_meter_fix()
:reset_adventure_settings()
make_checkpoint()
set_last_hit(unit:ref)
set_killer_sprite(sprite)
call_later(period:int, callback, …values)
call_cancel(handle)
call_pause(handle)
call_resume(handle)
input_check(input:string, …values)
input_check_pressed(input:string, …values)
input_check_released(input:string, …values)

Unit IDs

Knights

`"shovel knight"`
`"shovel knight b"`
`"treasure knight"`
`"treasure knight b"`
`"mole knight"`
`"mole knight b"`
`"shield knight"`

"shield knight b"
"king knight"
"king knight b"
"specter knight"
"specter knight b"
"plague knight"
"plague knight b"
"tinker knight"
"tinker knight b"
"propeller knight"
"propeller knight b"
"polar knight"
"polar knight b"
"black knight"
"black knight b"
"scrap knight"
"scrap knight b"
"prism knight"
"prism knight b"
"puzzle knight"
"puzzle knight b"
"random knight"
"shuffle knight"
"shuffle knight b"
"chester knight"
"chester knight b"
"mona"
"mona b"
"enchantress"
"enchantress b"
"monster" (assigned if wearing Monstrous Montera)

Other Knight-Related Units

"beefto"
"guard knight"
"stache"
"donovan"
"polar spinwulf"
"mobile gear"

Enemies

Plains

"beeto"
"blorb"

"boneclang"
"divedrake"
"dozedrake"
"bubble"

Pridemoor Keep

"goldarmor"
"blitzsteed"
"griffoth"

Lich Yard

"tadvolt"
"invisishade"
"tombstone"
"zamby"
"super skeleton"

Explodatorium

"simulacra"
"kettleleg"
"plagueclang"

Magic Landfill

"wizzem"
"floorb"
"ratsploder"
"magic book"
"pot genie"

Iron Whale

"floatsome"
"hermittack"
"serprize"
"grapps"
"grapps body"
"teethalon"
"teethalite"
"teethalon no chest"

Lost City

"moler"
"birder"
"hoppicles red"
"charflounder"
"mole minion"

Crystal Cave

`"wasp"`
`"crabstal"`
`"mini mollusk"`
`"blastshell"`
`"blastshell spent"`
`"nautilus"`

Clockwork Tower

`"blue beeto"`
`"electrodent"`
`"fairy"`
`"tinkerbot"`
`"cogslotter"`

Stranded Ship

`"tundread"`
`"spinwulf"`
`"freezorb"`
`"hoppicles"`
`"yeti"`
`"yeti2"`

Flying Machine

`"propeller rat"`
`"blue floatsome"`
`"hoverhaft"`
`"airship"`

Scholar Sanctum

`"firefleeto lead"`
`"firefleeto part"`
`"tadlock"`

Tower of Fate

`"lavalorb"`
`"samurai sword"`
`"samurai arrow"`
`"samurai claws"`
`"red boneclang"`
`"dark grapps"`
`"dark grapps body"`
`"black griffoth"`

Others

`"shrinemaster"`

```
"memmec"
"growthclang"
"growthclang 2"
"growthclang 3"
"growthclang 4"
"prism clone"
"fated shadow chase"
"fated shadow attack"
"big creep chase"
"big creep attack"
"terrorpin"
"lumber knight"
```

Bosses

```
"king boss"
"specter boss"
"plague boss"
"treasure boss"
"tinker boss"
"tinker mech"
"mole boss"
"scrap boss"
"propeller boss"
"polar boss"
"prism boss"
"puzzle boss"
"enchantress boss"
"megaboss"
"black knight boss"
"shovel knight boss"
"chester boss"
"mr hat boss"
```

NPCs

```
"npc troupple king"
"npc chester"
"npc percy"
"npc glitzem"
"npc mona"
"npc tief"
"npc puzzle knight"
"npc tipp"
"npc kee"
"npc hedge farmer"
"npc fleeto"
"npc sage"
"npc beefto"
```

"npc sledge farmer"
"npc bricks"
"npc plagueclang"
"npc mr hat"
"npc shrinemaster"
"npc bean"
"npc adult bean"
"npc tip giver"
"npc armorer"
"npc cloaked figure"

Chests

"chest"
"king chest"
"chester chest"
"angler chest"
"timed chest"

Heals

"potion"
"expired potion"
"turkey"
"turkey tray"
"teacup"

Bombs

"bomb"
"bomb fire"
"bomb ice"
"bomb ultimate"
"bomb poison"
"bomb shuffle"
"poison vat"
"mona bomb mini"
"mona bomb big"

Items

"zap wand"
"smoke bomb"
"war horn"
"red troupple chalice"
"blue troupple chalice"
"wildcard"
"chronos coin"
"spear"

`"ice axe"`
`"fire axe"`
`"shield"`
`"wood blade"`
`"steel blade"`
`"hero blade"`
`"glitzemizer"`
`"anchor"`
`"copycard"`
`"magic wand"`
`"super spear"`
`"poison dagger"`
`"flipwand"`
`"exploder X"`
`"exploder T"`

VS items

`"flood fish"`
`"bomb cannon"`
`"shuffle top"`
`"monster potion"`

Blueprints

`"shop upgrade"`
`"shop relic"`

Blocks

`"dirt block"`
`"stone block"`
`"steel block"`
`"snow block"`
`"bomb block"`
`"goo block"`
`"crusher block"`
`"secret dirt block"`
`"secret stone block"`
`"secret snow block"`
`"secret steel block"`

Other Units

`"key"`
`"door"`
`"boss door"`

`"moneybag"`
`"myriad"`
`"costume pile"`
`"bohto bugle"`
`"sideroom portal"`
`"junk block"`
`"growth gem"`
`"corpse"`
`"campfire"`
`"blessing"`
`"hidden wall"`
`"reserved space"`
`"tree"`
`"beanstalk"`
`"puzzle protector"`

Unit Behaviors

Enemy Behaviors

Name	Used By	Description

`wall`	`"npc bean"`
	`"beanstalk"`
	`"npc fleeto"`
	`"campfire"`
	`"hidden wall"`
	`"tree"`
	`"reserved space"`
	`"timed chest"`
		Unit behaves like a wall. This sets it to:
		• `immortal`
		• `immune`
		• `unmoveable`
		• `bombproof`
		• `inert`
		• `levitate`
		• `ignore_chains`
		• `untouchable`
		• `NOT hits_back`

`big`	`"npc glitzem"`
	`"beefto"`
	`"griffoth"`
	`"black griffoth"`
	`"yeti"`
	`"dozedrake"`
		Occupies 4 squares (2x2). This also sets it to:
		• `ignore_chains`
		• `unmoveable`

	`"super skeleton"`
	`"teethalon"`
	`"teethalon no chest"`
	`"yeti2"`
	`"nautilus"`
	`"cogslotter"`
	`"airship"`
	`"pot genie"`
	`"kettleleg"`
	`"mole minion"`
	`"npc shrinemaster"`
	`"shrinemaster"`	

`mega`	`"puzzle boss"`
	`"enchantress boss"`
	`"megaboss"`
		Occupies 9 squares (3x3). Adds a 5 damage cap when attacked. This is ignored if damage is superior to 20.

`hits_back`		Whether the unit retaliates if hit. Is applied to all units by default.

`bombproof`	`"key"`
	`"moneybag"`
	`"grapps body"`
	`"dark grapps body"`
	`"crusher block"`
	`"king boss"`
	`"specter boss"`
	`"plague boss"`
	`"treasure boss"`
	`"tinker boss"`
	`"mole boss"`
	`"propeller boss"`
	`"prism boss"`
	`"puzzle boss"`
	`"scrap boss"`
	`"chester boss"`
	`"enchantress boss"`
	`"megaboss"`
	`"tinker mech"`
	`"black knight boss"`
	`"shovel knight boss"`
	`"npc shrinemaster"`
	`"shrinemaster"`
	`"blessing"`
		Unaffected by bombs.

`levitate`	`"crusher block"`
	`"lava spout"`
	`"prism clone"`
		Ignores gravity. Does not fall down every turn.

	`"fated shadow chase"`
	`"fated shadow attack"`
	`"big creep chase"`
	`"big creep attack"`
	`"mr hat boss"`
	`"terrorpin"`
	`"npc shrinemaster"`
	`"shrinemaster"`
	`"mona bomb mini"`
	`"mona bomb big"`	

`unmoveable`	`"crusher block"`
	`"angler chest"`
	`"terrorpin"`
	`"timed chest"`
		Can't be moved by unit abilities. For instance Mole Knight or Prism Knight's abilities.
		
		Behaviors like `"wall"` or `"big"` already set this automatically.

`moveable`	`"chester chest"` `"npc bean"`	Can be moved by unit abilities. It is on by default, so you would only call this in case you want to make a wall to be moveable. Order matters, i.e. `"wall,moveable"` or `"big,moveable".`
`immortal`	`"terrorpin"`	Can't be damaged.
`untouchable`	`"bomb block"`	Bombable Block's behavior. Can't be attacked unless you have Obsidian Drill.

`npc`	`"costume pile"`
	`"myriad"`
	`"npc troupple king"`
	`"npc glitzem"`
	`"shop relic"`
	`"shop upgrade"`
	`"npc chester"`
	`"npc percy"`
	`"npc mona"`
	`"npc tief"`
	`"npc puzzle knight"`
	`"npc tipp"`
	`"npc tip giver"`
	`"npc kee"`
	`"npc hedge farmer"`
	`"npc sage"`
	`"npc adult bean"`
	`"npc beefto"`
	`"npc sledge farmer"`
	`"npc bricks"`
	`"npc plagueclang"`
	`"npc mr hat"`
	`"npc cloaked figure"`
	`"npc shrinemaster"`
		Marks unit to be an NPC, it will ignore attacks. This also sets:
		• `levitate`
		• `unmoveable`

`inert`	`"key"`
	`"moneybag"`
	`"blessing"`
	`"turkey tray"`
	`"crusher block"`
	`"bomb"`
	`"bomb fire"`
	`"bomb ice"`
	`"bomb ultimate"`
	`"bomb poison"`
	`"bomb shuffle"`
		Health pips are hidden for this unit.

	`"prism clone"`
	`"bubble"`
	`"angler chest"`
	`"timed chest"`
	`"mona bomb mini"`
	`"mona bomb big"`	

`heal:{X}`	`"potion"` `"turkey"`	How many HP the unit will heal if consumed.
`potion_persists`		Makes it so a potion won't transform into a turkey or diluted potion when playing in certain modes.

`block`	`"secret dirt block"`
	`"secret stone block"`
	`"secret snow block"`
	`"secret steel block"`
	`"tombstone"`
	`"dirt block"`
	`"stone block"`
	`"snow block"`
	`"steel block"`
	`"bomb block"`
	`"goo block"`
		Tags unit as block. Blocks are affected by certain relics (i.e. Obsidian Drill) and give Tinker Knight metal.

`spawn_portal:{sideroom_name}`	`"sideroom portal"`	Spawns a portal that leads to a sideroom. `":???"` makes a random one.

`chain_tag:{tag}`	`"potion"`
	`"grapps"`
	`"grapps body"`
	`"dark grapps"`
	`"dark grapps body"`
	`"nautilus"`
	`"mini mollusk"`
	`"blastshell"`
	`"blastshell spent"`
	`"samurai sword"`
	`"samurai arrow"`
	`"samurai claws"`
	`"firefleeto lead"`
	`"firefleeto part"`
		Makes two or more different unit types chain together (i.e. Liquid Samurais all use the tag "samurai").

`ignore_chains`	`"bomb"`
	`"bomb fire"`
	`"bomb ice"`
	`"bomb ultimate"`
		Can't chain with other units of the same type.

	`"mona bomb mini"`
	`"mona bomb big"`
	`"bomb shuffle"`
	`"bomb poison"`	

`garbage`	`"junk block"`	Spawns a random unit from the stage on dead.

`potion`	`"potion"`
	`"expired potion"`
	`"teacup"`
		Tags unit as potion. Potions are affected by some relics (i.e. Citrus Slice) and are collected by Plague Knight B or Mona.
		
		Note: this isn't the behavior that makes potions heal, that'd be `"heal:{X}"`.

`potion_persist`	`"teacup"`	Makes it so this unit can't be transformed into Turkeys or diluting potions. Note: this behavior does nothing if not paired with the behavior `"potion"`

`drop:{unit}`	`"turkey tray"`
	`"blorb"`
	`"lavalorb"`
	`"freezorb"`
	`"memmec"`
	`"angler chest"`
	`"teethalon"`
		Creates the specified unit on death.

`moneybag`	`"moneybag"`	Behavior used for the money bag dropped by the player on defeat. When destroyed it gives you back the `gem_value` set inside. Note: you can set it from `create.gml` of your unit mod.

`ice_touch:{X}`	`"bomb ice"`
	`"ice axe"`
	`"yeti"`
	`"yeti2"`
		Freezes units for `{X}` turns when attacking.

`fire_touch:{X}`	`"bomb fire"` `"fire axe"`	Ignites units for `{X}` turns when attacking.

`poison_touch:{X}`	`"poison dagger"`
	`"plague knight"`
	`"plagueclang"`
	`"bomb poison"`
		Poisons units for `{X}` turns when attacking.

`chest`	`"chest"`
	`"king chest"`
	`"chester chest"`
		Chest behavior, needs a key to be opened and drops a random item.

`king`	`"king chest"`	King chest behavior used in Versus. You need to be the player in the lead to open it for a random item.
`chester_chest`	`"chester chest"`	Chest that when opened leads to prefab `goto` (`"chester"` by default, which stands for chester's shop prefab). Note: you can set the `goto` value from `create.gml` of your unit mod.
`timed_chest`	`"timed chest"`	Chest behavior used inside some siderooms. Opens if you get a level clear before the count gets to zero for a random item.

`explodes:{X}`	`"bomb"`
	`"bomb fire"`
	`"bomb ice"`
	`"bomb ultimate"`
	`"mona bomb mini"`
	`"mona bomb big"`
	`"ratsploder"`
	`"bomb shuffle"`
	`"fated shadow chase"`
	`"big creep chase"`
	`"bomb poison"`
		Unit that will explode in `{X}` radius when attacked (a radius of 1 will result in a 3x3 explosion, 2 will be 5x5, etc…). Use `"attack:{X}"` to set the damage of the explosion.
		
		`"sprite_charge"` is used for the "preparing to explode" state.
		
		Note: by default it will take 3 turns to explode, unless you change this with `"timer:{X}"` behavior.

`timer:{X}`	`"bomb"`
	`"bomb fire"`
	`"bomb ice"`
	`"bomb ultimate"`
	`"mona bomb mini"`
	`"mona bomb big"`
	`"ratsploder"`
	`"bomb shuffle"`
	`"fated shadow chase"`
	`"big creep chase"`
	`"bomb poison"`
		Changes how long an `"explodes:{X}"` unit will take to explode.

`friendly`	`"bomb fire"`
	`"bomb ice"`
	`"bomb ultimate"`
		Can't damage the player.

`sprite_explosion:{sprite}`	`"bomb shuffle"`	Sets the explosion sprite (default is `sExplosion`). Only applied if `explodes` behavior is set.

`shuffle_explosion`	`"bomb shuffle"`	When a unit explodes, shuffles units around in its range.

`enemy`	`"mobile gear"`
	`"blorb"`
	`"lavalorb"`
	`"wizzem"`
	`"boneclang"`
	`"red boneclang"`
	`"beeto"`
	`"blue beeto"`
	`"floorb"`
	`"goldarmor"`
	`"griffoth"`
	`"black griffoth"`
	`"invisishade"`
	`"tadvolt"`
	`"moler"`
	`"spinwulf"`
	`"ratsploder"`
	`"electrodent"`
	`"fairy"`
	`"wasp"`
	`"crabstal"`
	`"hermittack"`
	`"grapps"`
	`"grapps body"`
	`"dark grapps"`
	`"dark grapps body"`
	`"propeller rat"`
	`"lava spout"`
	`"floatsome"`
	`"prism clone"`
	`"hoppicles"`
	`"freezorb"`
	`"yeti"`
	`"memmec"`
	`"dozedrake"`
	`"super skeleton"`
	`"tinkerbot"`
	`"magic book"`
	`"fated shadow chase"`
	`"fated shadow attack"`
	`"big creep chase"`
	`"big creep attack"`
	`"teethalon no chest"`
	`"teethalon"`
		Tags unit as enemy.

	"teethalite" "serprize" "yeti2" "nautilus" "mini mollusk" "blastshell" "blastshell spent" "cogslotter" "tundread" "blue floatsome" "hoverhaft" "airship" "samurai sword" "samurai arrow" "samurai claws" "divedrake" "blitzsteed" "tombstone" "zamby" "growthclang" "growthclang 2" "growthclang 3" "growthclang 4" "pot genie" "firefleeto lead" "firefleeto part" "tadlock" "mr hat boss" "terrorpin" "simulacra" "kettleleg" "plagueclang" "hoppicles red" "birder" "charflounder" "mole minion" "king boss" "specter boss" "plague boss" "treasure boss" "tinker boss" "mole boss" "propeller boss" "polar boss" "prism boss" "puzzle boss" "scrap boss"	

	`"enchantress boss"`
	`"megaboss"`
	`"tinker mech"`
	`"black knight boss"`
	`"shovel knight boss"`
	`"chester boss"`
	`"junk knight"`
	`"shrinemaster"`	

`poison_vat`	`"poison vat"`	Poisons target when attacked and splashes poison all around in a “+” pattern.
`poison_immune`	`"plague knight"` `"plagueclang"`	Poison immunity.
`protector:{unit}`	`"puzzle protector"` `"tadlock"`	If present on board, prevents damage to the `{unit}`. Setting the property to `:around` will make the unit protect 8 surrounding units instead.
`protector_loop:{X}`	`"tadlock"`	Makes `protector` behavior active only for a `{X}` turns.

`undead`	`"boneclang"`
	`"red boneclang"`
	`"invisishade"`
	`"super skeleton"`
	`"big creep chase"`
	`"big creep attack"`
	`"zamby"`
	`"growthclang"`
	`"growthclang 2"`
	`"growthclang 3"`
	`"growthclang 4"`
	`"plagueclang"`
	`"specter boss"`
	`"shrinemaster"`
		Tags unit as undead. Undead units are affected by the Divine Liquid relic.

`raise_shield`	`"goldarmor"`
	`"hoppicles"`
	`"hoppicles red"`
		Raises shield when struck.
		
		The following sprite tags are used for the shield sides:
		
		● `"sprite_other"` : the shield itself
		● `"sprite_skill"` : unit + shield sprite looking left
		● `"sprite_skill_up"` : unit + shield looking up
		● `"sprite_skill_down"` : unit + shield looking down

`slams`	`"griffoth"`
	`"black griffoth"`
	`"yeti"`
		Attacks by slamming the ground.
		
		The following sprite tags are used for animating the attack:
		
		• `"sprite_charge"` : attack charge
		• `"sprite_skill"` : actual attack

`slams_side`	`"mole minion"`	Attacks in two full rows. The following sprite tags are used for animating the attack: • `"sprite_charge"` : attack charge • `"sprite_skill"` : actual attack
`ghost`	`"invisishade"`	Hides until you hit something else. `"sprite_charge"` is used for the invisible state.
`electric_loop:{X}`	`"tadvolt"` `"blue floatsome"`	Electrifies every `{X}` turns. The following sprite tags are used to animate the attack: • `"sprite_charge"` : preparation • `"sprite_special"` : electric sprite

`jumps:{X}`	`"memmec"`
	`"super skeleton"`
	`"yeti2"`
	`"samurai sword"`
		Leaps around the board every `{X}` turns.
		
		The following sprite tags are used to animate the jump:
		
		• `"sprite_fly"` : on the way up
		• `"sprite_fall"` : on the way down

`collapses:{X}`	`"super skeleton"`	Collapses every `{X}` jumps and becomes invulnerable. Use `"sprite_special"` for collapse animation.
`stomps:{X}`	`"yeti2"`	Deals `{X}` damage to all surrounding units when landing after a jump.
`wake:{unit}`	`"tombstone"`	Transforms into `{unit}` if not attacked as part of a chain. Use `"sprite_special"` for transform animation.
`teleport:{X}`	`"wizzem"`	When at `{X}` HP or lower, teleports away when hit.
`hide_corpse`	`"magic book"` `"fated shadow chase"`	Used to make a unit's corpse invisible.

	`"fated shadow attack"`
	`"big creep chase"`
	`"big creep attack"`	

`explode_charge`	`"magic book"` `"pot genie"`	Forces a unit to execute its ability when hit by an explosion.
`lancer_jump:{X}`	`"pot genie"`	Floats in the air every `{X}` turns, then slams the ground. The following sprite tags are used in the process: • `"sprite_charge"` : start floating • `"sprite_skill_up"` : in air • `"sprite_aim"` : start dropping • `"sprite_skill"` : slam • `"sprite_skill_down"` : stop spinning

`start_turns:{X}`	`"memmec"`
	`"magic book"`
	`"fated shadow attack"`
	`"big creep attack"`
	`"blastshell"`
	`"cogslotter"`
	`"airship"`
	`"blitzsteed"`
	`"pot genie"`
	`"expired potion"`
		How many turns should pass before the start of the level for the unit to be able to use their ability. Default is `12+irandom(2)`.
		
		Note: only behaviors from this list can use attack variables.

`cooldown:{X}`	`"dozedrake"`
	`"cogslotter"`
	`"airship"`
	`"pot genie"`
	`"expired potion"`
		Turns between subsequent ability uses. Default is `12`. You can also directly modify `attack_cooldown` from any script of your unit mod.
		
		Note: only behaviors from this list can use attack variables.

`load:{X}`	`"dozedrake"`
	`"fated shadow attack"`
	`"big creep attack"`
		You can think of this value as max “ammo” capacity. Every use of the unit’s ability decreases `attack_ammo` by 1 and sets `attack_timer` to `attack_refire`. If `attack_ammo` is at 0, `attack_timer` is instead set to `attack_cooldown`, and `attack_ammo` is reset back to the “load” value. Default is `1`. You can also directly modify both `attack_load` and

		`attack_ammo` from any script of your unit mod. Note: only behaviors from this list can use attack variables.

`refire:{X}`	`"dozedrake"`
	`"fated shadow attack"`
	`"big creep attack"`
		See `load` behavior. Default is `1`. You can also directly modify `attack_refire` from any script of your unit mod.
		
		Note: only behaviors from this list can use attack variables.

`camouflage:{X}`	`"serprize"`	Turns invisible after `{X}` turns. The following sprite tags are used to animate the behavior: • `"sprite_skill_up"` : disappear • `"sprite_skill_down"` : appear
`pushable`	`"hermittack"`	Gets pushed horizontally when attacked.

`tangle`	`"floatsome"`
	`"mini mollusk"`
	`"blue floatsome"`
	`"firefleeto lead"`
	`"firefleeto part"`
		Units stick to each other.
		
		Optional, sprite tags for connecting units together (i.e. floatsome links):
		
		• `"sprite_skill_up"` : vertical link
		• `"sprite_skill_down"` : horizontal link

`mob_rage:{X}`	`"wasp"`	Deals more damage in group (+1 for each group member). `{X}` is the damage limit. The following sprite tags are used to animate the behavior: • `"sprite_charge"` : at least 2 units in chain • `"sprite_special"` : max rage
`crab`	`"crabstal"`	Falls diagonally if it cannot fall down.
`hit_run:{X}`	`"electrodent"`	Starts moving when hit for `{X}` turns.
`hit_electric:{X}`	`"electrodent"`	Becomes electrified when hit for `{X}` turns. Use `"sprite_special"` for the electrified state.
`lowHP_rage`	`"fairy"`	Deals extra damage when the player is at 2 HP or less. Use `"sprite_special"` for

		raging sprite.
`side_shield:{side}`	`"tinkerbot"`	Invincible when attacking from `{side}`. Use `"sprite_special"` for when preventing damage.
`wind_touch`	`"yeti"`	Throws the player away on attack.
`pierce`	`"tundread"`	Retaliates with a spear when struck. Use `"sprite_skill"` for retaliation animation.
`hit_refresh`	`"hoppicles"` `"hoppicles red"`	Able to change raised shield position.
`hit_follow:{X}`	`"hoppicles"` `"hoppicles red"`	Starts following the player for `{X}` turns after being hit.
`weak_flight`	`"propeller rat"`	Flies up until it can't, then drops down and repeats. Use `"sprite_skill_down"` for falling animation.
`chases`	`"fated shadow chase"` `"big creep chase"`	Chases the player around the board.
`proximity_prime`	`"fated shadow chase"` `"big creep chase"`	Forces the unit to explode when near the player. The unit should have `"explodes"` behavior and, optionally, `"timer"` behavior.
`simulacra`	`"simulacra"`	Mirrors horizontal movement, if they aren't above you.
`kettleleg`	`"kettleleg"`	Shoots purple acid when attacked or falling. Use `"sprite_special"` for shooting animation.
`diagonal`	`"birder"`	Moves diagonally and bounces from the level borders. The following sprite tags are used to animate the behavior: • `"sprite_skill_up"` : on the way up • `"sprite_skill_down"` : on the way down
`geothermal`	`"charflounder"`	Gains +1 attack if on top of lava. Use `"sprite_charge"` for when ability is active.
`wild`	`"lumber knight"`	Makes the unit chain with anything.

Knight Behaviors

Name	Used By	Description

`knight`	`"shovel knight"`
	`"shovel knight b"`
	`"treasure knight"`
	`"treasure knight b"`
	`"mole knight"`
	`"mole knight b"`
	`"shield knight"`
	`"shield knight b"`
	`"king knight"`
	`"king knight b"`
	`"specter knight"`
	`"specter knight b"`
	`"plague knight"`
	`"plague knight b"`
	`"tinker knight"`
	`"tinker knight b"`
	`"propeller knight"`
	`"propeller knight b"`
	`"polar knight"`
	`"polar knight b"`
	`"black knight"`
	`"black knight b"`
	`"scrap knight"`
	`"scrap knight b"`
	`"prism knight"`
	`"prism knight b"`
	`"puzzle knight"`
	`"puzzle knight b"`
	`"random knight"`
	`"shuffle knight"`
	`"shuffle knight b"`
	`"chester knight"`
	`"chester knight b"`
	`"mona"`
	`"mona b"`
	`"enchantress"`
	`"enchantress b"`
		Marks the unit as a playable character.

`damage_on_kill`	`"shovel knight"`	Fatal blows do extra damage to chains. Use `"sprite_special"` for when ability has activated.
`extra_pots`	`"shovel knight"`	Potions are more common.
`active_bash`	`"king knight"`	Active bash ability. Use `"sprite_charge"` for bashing animation.
`lifesteal`	`"specter knight"`	Killing enemies recovers 2 HP.
`pot_weak`	`"specter knight"`	Potions damage you. Use `"sprite_skill_down"` for the hurt animation.
`poison_touch`	`"plague knight"` `"plagueclang"`	Attacking foes poisons them, including chains.
`poison_bomb`	`"plague knight"`	Bombs poison enemies.
`poison_immune`	`"plague knight"` `"plagueclang"`	Poison immunity.
`uppercut`	`"treasure knight"`	Deals more damage when striking foes from below. Use `"sprite_skill_up"` for the uppercut animation.
`collect:{behavior}`	`"tinker knight"` `"tinker knight b"`	Gathers metal from `{behavior}`. For example `collect:block.`
`active_mech`	`"tinker knight"`	Can build a mech. Use `"sprite_charge"` for mech building animation and `"sprite_special"` for the mech itself.
`25%steel`	`"tinker knight"` `"tinker knight b"`	Steel blocks are 25% more common.
`active_burrow`	`"mole knight"`	Can burrow and swap positions. The following sprite tags are used to animate the behavior: • `"sprite_skill_down"` : dig in • `"sprite_special"` : burrowed • `"sprite_skill_up"` : dig out
`propelled_hit`	`"propeller knight"`	+1 attack for each lone foe defeated.

`chain_recoil`	`"propeller knight"`	Chains reset your attack and deal you 1 more damage.
`ice_kill`	`"polar knight"`	Fatal blows freeze chains.
`ice_extra_dmg`	`"polar knight"`	Deal 1 more damage to frozen foes.
`extra_ice`	`"polar knight"`	Find more ice items.
`active_gempower`	`"black knight"`	Spend Gem Meter to enhance attack. Use `"sprite_charge"` for when ability has activated.
`active_steal`	`"scrap knight"`	Can bag stuff and release it elsewhere.
`bonus_durability`	`"scrap knight"`	Items have more durability.
`active_prism`	`"prism knight"`	Can swap or teleport.
`last_shield`	`"shield knight"`	Survive 1 fatal blow per level.
`hit_shield:{X}`	`"shield knight"`	Can nullify `{X}` instances of non-hazard damage of any strength.
`spin_units`	`"puzzle knight"`	Can rotate units around. Use `"sprite_special"` for the rotating animation.
`shuffle`	`"shuffle knight"`	Transforms into a random knight when starting a new level.
`random`	`"random knight"`	Transforms into a random knight when starting a new run.
`active_pogo`	`"shovel knight b"`	Active shovel drop ability. The following sprite tags are used to animate the behavior: • `"sprite_special"` : idle • `"sprite_skill_up"` : hit
`active_joustus`	`"king knight b"`	Can earn cards from chains and use them. The following sprite tags are used to animate the behavior: • `"sprite_aim"` : card choosing • `"sprite_skill"` : card play • `"sprite_other"` : idle, no cards • `"sprite_skill_down"` : no cards, trying to use ability

`active_slice`	`"specter knight b"`	Can slice through foes, as long as there's an open space behind them. The following sprite tags are used to animate the behavior: • `"sprite_aim"` : slice start • `"sprite_skill_down"` : slice end
`potion_burst`	`"plague knight b"`	Can explode a potion and jump away. The following sprite tags are used to animate the behavior: • `"sprite_aim"` : aiming • `"sprite_fly"` : jump up • `"sprite_fall"` : fall down
`active_hook`	`"treasure knight b"`	Can hook units towards you or pull himself to walls. The following sprite tags are used to animate the behavior: • `"sprite_aim"` : aiming • `"sprite_skill"` : hook side • `"sprite_skill_up"` : hook up • `"sprite_skill_down"` : hook down
`active_gear`	`"tinker knight b"`	Can build a Mobile Gear. The following sprite tags are used to animate the behavior: • `"sprite_skill"` : ejecting out of a gear • `"sprite_skill_up"` : riding gear • `"sprite_skill_down"` : riding gear when weak
`passive_burrow`	`"mole knight b"`	Move freely under units. `"sprite_skill_up"` is used for when under a unit.
`less_durability`	`"mole knight b"`	Reduced item durability.
`propel`	`"propeller knight b"`	Active dive ability. The following sprite tags are used to animate the behavior: • `"sprite_skill_up"` : rising • `"sprite_skill_down"` : diving
`puppies`	`"polar knight b"`	Can release a Spinwulf that attacks foes

		and fetches items. `"sprite_skill_up"` is used for aiming animation.
`meteor_shower`	`"black knight b"`	Spend Gem Meter to summon meteors. The following sprite tags are used to animate the behavior: • `"sprite_aim"` : aiming • `"sprite_skill_down"` : summoning
`special_hold`	`"black knight b"`	Can increase meteors power.
`active_bag`	`"scrap knight b"`	Can toss a bag to then swap positions with it. The following sprite tags are used to animate the behavior: • `"sprite_skill_up"` : teleport up • `"sprite_skill_down"` : teleport down • `"sprite_skill"` : teleport left • `"sprite_other"` : teleport right • `"sprite_special"` : idle, no bag • `"sprite_charge"` : idle, no bag, weak
`active_clone`	`"prism knight b"`	Can spend half HP to create clones. `"sprite_aim"` is used for aiming animation.
`active_throw`	`"shield knight b"`	Can throw a shield. Catching it boosts the next attack or absorbs a hit. The following sprite tags are used to animate the behavior: • `"sprite_skill"` : throw • `"sprite_special"` : idle, no shield • `"sprite_charge"` : idle, no shield, weak
`identity_crisis`	`"shuffle knight b"`	Shapeshift to a random knight each time you drink a potion.
`active_rubik`	`"puzzle knight b"`	Can shift the rows or columns of the board. The following sprite tags are used to animate the behavior: • `"sprite_skill"` : shifting

		• `"sprite_skill_up"` : shifting upwards
`less_pots`	`"puzzle knight b"` `"beefto"`	Potions are less common.
`active_potion_boom`	`"mona"`	Can detonate nearby potions. The following sprite tags are used to animate the behavior: • `"sprite_aim"` : aiming • `"sprite_skill"` : detonating
`collect_potions`	`"mona b"`	Can collect potions to then heal or use them as bombs. The following sprite tags are used to animate the behavior: • `"sprite_aim"` : aiming • `"sprite_skill"` : throw
`caltrop`	`"donovan"`	Can toss caltrops that become a hazard when thrown on the ground.
`fast_move`	`"polar spinwulf"`	Moves one extra space per turn.
`active_chester`	`"chester knight"` `"chester knight b"`	Can enter shop at any time. The following sprite tags are used to animate the behavior: • `"sprite_skill_down"` : entering shop • `"sprite_skill_up"` : exiting shop
`cheap_relics`	`"chester knight"`	Relics are cheaper.
`more_relics`	`"chester knight"`	Can carry more relics.
`more_chests`	`"chester knight b"`	Chests are more common.
`dual_wield`	`"chester knight b"`	Can dual wield items.
`active_boom`	`"enchantress"`	Spend 2 HP to attack in all directions at once. `"sprite_skill"` is used for aiming animation.
`active_fireball`	`"enchantress b"`	Spend 1 HP to shoot a fireball that deals extra damage the farther it travels. The following sprite tags are used to animate the behavior:

		• `"sprite_aim"` : aiming • `"sprite_skill"` : shoot
`spear`	`"guard knight"`	Always attacks with spear.
`relicless`	`"guard knight"`	Cannot find relics.
`evil_ending`	`"enchantress"` `"enchantress_b"`	No gameplay effect. Applies the Enchantress special cutscene when beating the game.

Item Behaviors

Name	Used By	Description

`passive_item`	`"spear"`
	`"ice axe"`
	`"fire axe"`
	`"shield"`
	`"wood blade"`
	`"steel blade"`
	`"hero blade"`
	`"magic wand"`
	`"super spear"`
	`"poison dagger"`
		Doesn't have an activatable ability.

`shield`	`"shield"`	Blocks 1 damage.
`spear`	`"spear"` `"super spear"`	Extends attack range.
`magicwand`	`"magic wand"`	Triggers Magic Wand.

`ice_touch:{X}`	`"bomb ice"`
	`"ice axe"`
	`"yeti"`
	`"yeti2"`
		Freezes units for `{X}` turns when attacking.

`fire_touch:{X}`	`"bomb fire"` `"fire axe"`	Ignites units for `{X}` turns when attacking.

`poison_touch:{X}`	`"poison dagger"`
	`"plague knight"`
	`"plagueclang"`
	`"bomb poison"`
		Poisons units for `{X}` turns when attacking.

Attack Behaviors

These are behaviors that use `attack_timer`, `attack_ammo`, `attack_refire` and `attack_load` variables.

- `lancer_jump`
- `jumps`
- `collapses`
- `slams`
- `slams_side`

Relic IDs

`"meal ticket"`
`"premium ticket"`
`"diamond dust"`
`"obsidian drill"`
`"snoutin charm"`
`"shockproof socks"`
`"fizzle wand"`
`"bomb seed bag"`
`"nimbus balloon"`
`"potion napkin"`
`"dynamallet"`
`"power pail"`
`"desperation talon"`
`"gem chain"`
`"gem beet"`
`"gold leaf clover"`
`"swift dagger"`
`"third amulet"`
`"single glove"`
`"divine liquid"`
`"five string"`
`"fenix feather"`
`"chronos glass"`
`"too big bomb"`
`"toad totem"`
`"mentor manual"`
`"fury horns"`
`"zesty slice"`
`"chester chart"`
`"super skeleton key"`
`"ram rod"`
`"conjuring cupcake"`
`"underdog collar"`

`"coin of carnage"`
`"coin of cover"`
`"bountiful bugle"`
`"time capsule"`
`"pocket portal"`
`"explosion jar"`
`"wicked whetstone"`
`"frosty fauld"`
`"thorny tambourine"`
`"hearty wand"`
`"lumber 1"`
`"lumber 2"`
`"lumber 3"`
`"lumber 4"`
`"meal coupon"`
`"swift stick"`
`"toad tadpole"`
`"dubious dust"`
`"cheaters manual"`
`"apple"`
`"lumber cube"`
`"scholar sextant"`
`"painter kit"`
`"portal tracker"`

Trap IDs

`"water"`
`"spikes"`
`"poison"`
`"lava"`
`"evil fire"`
`"hole"`
`"conveyor"`
`"oil"`
`"ice"`
`"fire"`
`"caltrop"`
`"burner"`
`"acid"`

Prefab IDs

Note: you can use the Prefab Viewer to quickly go to any prefab.

`"game"`
Going into this prefab loads the last stage you've been to.

Camp prefabs

`"hub"` (See image)
Campfire : Initial prefab of the Camp area.

`"hub2"` (See image)
Table Room : Top area of the Camp and where you go to start an adventure run.

`"hub3"` (See image)
Hangout : Bottom area of the Camp and where you go to start an endless run.

`"hub4"` (See image)
Hedge Farm : Bottom-right area of the Camp and where you grow the beanstalk to enter Castle Quandary.

`"unlocks"` (See image)
Chester's Shop (camp) : Shop of the Camp area where you unlock relics to be found in future runs.

`"practice"` (See image)
Percy's Cannon : Right area of the Camp and where you unlock and use shortcuts.

`"minigame"` (See image)
Grotto : Left area of the Camp and where you play Mona's minigame.

`"puzzle_knights_room"` (See image)
Puzzle Knight's Room : Secret area inside Camp.

`"hedgeroom"` (See image)
Hedge Farmer's Room : Secret area inside Hedge Farm.

Castle Quandary prefabs

`"bean entry"` (See image)
Cloud Gate : Entrance to the Castle Quandary.

`"bean hub"` (See image)
Hallowed Hall : Main area of the Castle Quandary.

`"bean palace"` (See image)

Chamber of Quandaries : Top area of the Castle Quandary where you access Quandary runs.

`"bean arcade"` (See image)
Marathon : Starts a Marathon run, waves of enemies will spawn and you have to survive as long as possible. **IMPORTANT:** must be called/entered as a quandary portal.

`"bean garden"` (See image)
`"bean garden2"` (See image)
`"bean garden3"` (See image)
`"bean garden4"` (See image)
`"bean garden fall"` (See image)
Sculpture Gardens : Right area of the Castle Quandary, here you can find statues for completing Quandaries and a way to fall back to Camp (`"bean garden fall"`).

`"hat shop"` (See image)
Mr.Hat's shop : Left area of the Castle Quandary, here you can buy and equip hats to spice up your runs.

Adventure prefabs

All of these are siderooms that can be found at random in each stage. With the exception of Chester's shop and Shrines that have a fixed spawn rate.

`"chester"` (See image)
Chester's Shop : Shop found during runs, here you can buy relics.

`"shrineroom"` (See image)
Shrine room : Collect a part of the key needed for the true ending. A reward is given based on how many parts you've collected (to a max of 4).

`"preshrine"`
Shrine Door room (general) : Door before entering a Shrine, depending of how many key parts you've collected for the true ending it will spawn a different door:

- `"shrine sacrifice"` (See image) (0 parts) : Door that opens with 3 keys or the Super Skeleton Key
- `"shrine money"` (See image) (1 part) : Door that opens by sacrificing 1 max HP
- `"shrine 3keys"` (See image) (2 parts) : Door that opens when you have 20.000 gems
- `"shrine fight"` (See image) (3 parts) : Door that opens by defeating the Shrinemaster

`"boss1"` (See image)
Triggers this stage's boss fight.

`"tinker mech"` (See image)
Triggers the second phase of Tinker Knight's boss fight.

`"megaboss"` (See image)
Triggers the final boss of the game.

Adventure prefabs - Shop siderooms

These siderooms are rare and on average you should get 1 or 2 in a run. 3 if you are lucky.

`"sideroom gamble"` (See image)
Glitzzem's sideroom : Talk to Glitzzem to spend money for a chance at a free relic.

`"sideroom troupple"` (See image)
Troupple King's sideroom : Meet Troupple King and pick one of two Troupple Chalices.

`"sideroom tief shop"` (See image)
Tief's shop : Tief sells 6 relics, out of which some are unique to Tief's shop and others are regular relics bundled together with Lumber.

`"sideroom armorer"` (See image)
Armorer's shop : Armorer offers 3 random items for trade, he can also increase your equipped item's durability for a price.

Adventure prefabs - Event siderooms

These siderooms happen after specific conditions are met and only once.

`"sideroom hedgefarmer"` (See image)
Hedge Farmer sideroom : After enough progress is made in the game, this sideroom will appear. Rescuing the Hedge Farmer from a horde of beetos is the first step to unlocking Castle Quandary.

`"sideroom mrhat"` (See image)
Mr.Hat fight : If you wear a lot of hats, there's a chance for Mr.Hat to fight you for them.

`"sideroom loop10"` (See image)
Beefto sideroom : Found when reaching loop 10 in endless in plains.

`"sideroom chester"` (See image)
Chester sideroom : Unlock method for Chester, found by progressing enough through the game. There is also a low chance of showing up naturally or when you are lagging behind in relics.

`"sideroom valentines"` (See image)
Valentines sideroom: Exclusive sideroom to the valentine's event, where a bunch of beetos are hiding.

Adventure prefabs - Random siderooms

Random siderooms that generally have a gimmick or enemies in it.

`"sideroom2"` (See image)
Random enemy sideroom 1 : Spawns random enemies from the next stage.

`"sideroom4"` (See image)
Random enemy sideroom 2 : Spawns random enemies from the next 2 stages.

`"sideroom powerup"` (See image)
Powerup sideroom : Spawns four random items of which, at random, some will trigger an ambush on pickup (the entire room fills with random enemies).

`"sideroom egg stash"` (See image)
Egg Stash : Spawns a lot of Growth Gems, some Crabstals and a chest.

`"sideroom chest stash"` (See image)
Chest Stash : Spawns four chests and keys.

`"sideroom falling blocks"` (See image)
Falling blocks : Spawns random blocks and a chest.

`"sideroom shield"` (See image)
Shield sideroom : Spawns a lot of shielded enemies and a Shield item.

`"sideroom ghost"` (See image)
Ghost sideroom : Spawns a lot of Invisishades and a Hero Blade.

`"sideroom ice"` (See image)
Ice sideroom : Snow/Ice themed sideroom. Spawns an Ice Axe.

`"sideroom fire"` (See image)
Fire sideroom : Fire/Lava themed sideroom with a lot of lavalorbs. Spawns a Fire Axe.

`"sideroom chrono coin"` (See image)
Chrono Coin : Starts frozen, spawns random enemies and a Chronos Coin.

`"sideroom bomb"` (See image)
Bomb sideroom : Survival room, dodge falling bombs to get a free item.

`"sideroom cauldron"` (See image)
Cauldron sideroom : Survival room, dodge cauldron lava to get a free item.

`"sideroom firing"` (See image)
Firing range : Survival room, dodge projectiles to get a free item.

`"sideroom glitzemizer"` (See image)
Glitzemizer : Find the rare Glitzemizer item on a pedestal. Taking it out has a high chance of triggering an ambush (room filled with enemies).

`"sideroom traveler"` (See image)
Wandering Traveler: Find a random relic laying around… However if you pick it up a Wandering Traveler will pick a fight with you!

`"sideroom random puzzle"` (See image)
Timed Chest : Clearing this room of enemies within a certain amount of attacks will open a chest.

`"sideroom poison"` (See image)
Poison sideroom: Poison themed sideroom with blorbs and Plague Minions. Spawns Poison Dagger.

`"sideroom miniboss"` (See image)
Miniboss sideroom: Spawns a random miniboss (big unit) in the middle with +2HP.

Stage IDs

`"plains"`
`"pridemoor keep"`
`"lich yard"`
`"magic landfill"`
`"iron whale"`
`"crystal caverns"`
`"clockwork tower"`
`"stranded ship"`
`"flying machine"`
`"scholars sanctum"`
`"tower of fate"`
`"explodatorium"`
`"lost city"`

Modifier IDs

Note**:** Modifiers marked as Unused are old experiments that remain in code but never made it into the final game. They may not work fully.

`"advanced mode"` (Legendary Gold Helm)
Activates Legendary Path.

`"cursed mode"`
Choose one offering before entering a stage.

`"skill A mode"`
Forces character's base ability (A skin).

`"skill B mode"`
Forces character's refract ability (B skin).

`"longer levels"` (Sluggish Beeto Beret)
Levels are 20% longer.

`"relic price up"` (Extravagant Gem Hat)
Relic prices are increased by 25%.

`"boss speed up"`
Boss fights have turns move 35% faster.

`"turkey trays"` (Platter Hatter)
Turkeys appear on platters. Slightly increases turkey spawn rate as well.

`"advanced spawns"` (Legendary Foe Helm)
Adds foes from the Legendary Path.

`"advanced hazards"` (Legendary Hazard Helm)
Adds hazards from the Legendary Path.

`"hidden chestkey"` (Mining Helmet)
Some chests and keys must be found inside blocks.

`"hidden chests"`
Unused. Hides all chests inside blocks.

`"hidden keys"`
Unused. Hides all keys inside blocks.

`"gem skeletons"` (Growthclang Wig)
Growth Gems become Growthclangs.

`"lights out"` (Jack-o'-Hattern)

Blackouts occur periodically.

`"small wallet"` (Pauper's Cap)
No more than 20,000 Gems can be carried.

`"cauldrons"` (Cauldron Cap)
Cauldrons always appear.

`"crystal mirror"` (Wizzem Hat)
Rotates or flips the board occasionally.

`"throwback"` (Boneclang Skull)
All levels end with bosses.

`"short stages"` (Fleeto Beret)
Levels are 30% shorter.

`"double HP"` (Troupple Acolyte's Hat)
Foes have 50% more Health. Your Health is doubled.

`"pandemic"` (Toad Totem Topper)
Start with a Toad Totem and poisoned. Poison is permanent but acts slowly.

`"moonlight"`
Specter Knight's Quest. Turns into Donovan periodically.

`"buried mode"` (Moler Hat)
The board is always full, and will collapse if you act too slowly.

`"super long levels"` (Protracted Beeto Beret)
Levels are twice as long.

`"tax hike"` (Deflated Beret)
10% fewer Gems from all sources.

`"gems halved"` (Half Top Hat)
50% fewer Gems from all sources.

`"row spawner"` (Wall Hat)
Units drop as a full row.

`"down spawner"` (Upside-Crown)
Units appear from below.

`"delayed fall"`
Unused. When a unit would start falling (spot below became free), it will take one extra turn to fall.

`"mech forever"`
Tinker Knight's Quest. Makes it so you are always in Mech Mode and running out of metal kills you.

`"no gravity"` (Propeller Pith Helm)
A strong wind prevents units from falling.

`"full shuffle"`
Shuffle Knight's Quest. Randomizes your character every now and then.

`"random hazards"` (Rainbowlorb Hat)
Level hazards are randomized.

`"items change on use"` (Transmuting Tammie)
Items transform when used.

`"total bash"`
King Knight's Quest. Makes it so you bash when moving and replaces your ability with a way to move 1 step at a time.

`"boss spawn phase forever"`
Makes it so units keep spawning at all times during boss fights.

`"memmec chests"` (Memmec's Kin Cap)
Some chests become Memmecs.

`"spikeless crusher"`
King Knight's Quest. Removes the spikes from the Ceiling Crusher.

`"pop threes"`
Prism Knight's Quest. Makes chains automatically pop when formed.

`"no chains"`
Removes the ability to chain damage. Used on some Quandary Quests and the cheat "NOTAPUZZLE".

`"swap attack"`
Unused. Your attacks swap your position with the bumped unit.

`"pop squares"`
Puzzle Knight's Quest. Makes chains automatically pop when a square is formed.

`"pop lines"`
Puzzle Knight's Quest. Makes chains automatically pop when a line is formed.

`"double boss spawns"`
Puzzle Knight and Prism Knight's Quest. Makes bosses spawn double the units.

`"avalanches"` (Snow Pelt Cap)
Occasionally triggers an avalanche, pushing you down and freezing blocks and foes.

`"bag spawner"`
Scrap Knight's Quest. Spawns something inside your bag periodically. If there's not a free spot it will push out whatever was inside.

`"enchantress ghost"` (Invisishade Cap)
A malevolent force haunts you. If you are Shield Knight the Enchantress will haunt you, otherwise it will be The Big Creep.

`"rewardless solo kills"` (Herder's Hat)
Lone foes award no Gems when defeated.

`"terrorpin"` (Terrorpin's Toque)
A spinning, hulking turtle joins the battle.

`"more potions"`
Mona's Quest. Levels spawn more potions.

`"bomb mayhem"`
Mona's Quest. Levels spawn more bombs, however they have shorter range.

`"random hp"` (Broken Heart Hat)
Some foes will spawn with more or less HP than normal.

`"text is q"`
Turns text into question marks.

`"push attack"` (Goo Bumpin' Bonnet)
Bump foes to push them, pin them to damage.

`"necromancy"` (Honcho's Headdress)
Defeated foes sometimes leave behind other units.

`"tag team"` (Super Skeleton Skull)
Bosses appear as a duo.

`"traffic mode"` (Red Alert Beret)
Units fall in when the light turns red.

`"floor poison"` (Blorb Hat)
Poison Puddles fill some of the board.

`"floor ice"` (Freezorb Hat)
Ice Patches fill some of the board.

`"floor water"` (Aqualorb Hat)
Water Puddles fill some of the board.

`"floor lava"` (Lavalorb Hat)
Lava Puddles fill some of the board.

`"floor spikes"` (Needleorb Hat)
Spikes fill some of the board.

`"floor conveyors"` (Conveyorb Hat)
Conveyor Belts fill some of the board.

`"out of stock"` (Meager Bowler)
Chester sells only one Relic per level.

`"buy weapons"` (Inverted Chest Hat)
Chester sells items instead of Relics.

`"chaos"` (Juggler's Cap)
All equipped Relics are shuffled each level.

`"relic snare"` (Gremlin's Cap)
Temporarily disables one Relic every level.

`"half durability"` (Shoddy Helmet)
All items have half durability.

`"reinforcements"` (Goldarmor Crest)
The board starts with 2 more rows filled.

`"stampede"` (Stampede Stetson)
Units drop in clusters, but are less frequent.

`"low gravity"` (Floatsome Fez)
Units take more turns to fall.

`"player gravity"` (Stonefall Ushanka)
Every turn you don't move, you fall.

`"birder"` (Birder Bicorne)
Inactivity causes your last action to repeat.

`"no backtracks"` (Hoppicles Helmet)
Leaves a damaging trail that affects only you.

`"turbo"` (Haste Helm)

Turns no longer pass when you move, but speed is increased.

`"faster turns"` (Swift Spinwulf Shapka)
Turn speed increased by 30%.

`"damage buff"` (Wrathful Tyrolean)
All attacks deal +1 damage.

`"all poisonous"` (Toxic Turban)
All attacks become poisonous.

`"enemy autoheal"` (Foegenerative Cap)
Damaged foes heal over time.

`"underdog"` (Taming Topper)
Foes with 2HP get reduced to 1HP.

`"more hp enemies"` (Toque Blanche)
Foes have +1HP.

`"less hp player"` (Withering Turkey Skull)
Player has -2 max HP.

`"expiration date"` (Diluting Skullcap)
Potions lose effectiveness after a few turns.

`"hp fog"` (Myopic Top Hat)
Everyone's remaining HP is invisible.

`"dark fog"` (Darkness Veil)
Only nearby units are totally visible.

`"choice fog"`
Trial of Offerings. Obscures future Offering choices.

`"hazards always"` (Hazardous Headgear)
Hazards will appear wherever possible.

`"advanced biggies"`
Unused. Spawns big foes from the Legendary Path.

`"poison long"`
Unused. Poison takes an extra turn to trigger.

`"exit relic"` (Present Hat)
Receive a free Relic at the end of each level.

`"gacha relics"` (Glitzem's Top Hat)
Relics become Time Capsules.

`"turkey pots"` (Turkey Toque)
Potions become Turkeys.

`"emergency exit"` (Scout's Cap)
The exit appears earlier.

`"hp all"` (Troupple Crown)
Player gains +5HP. Foes gain +1HP.

`"discount 25"` (Connoisseur's Cap)
Relics are 25% cheaper.

`"autoslice"` (Dash Slash Cap)
Dash Slash through foes, if possible.

`"waltz walk"` (Bard's Waltzing Hat)
Every third turn, your action is repeated.

`"weapon rain"` (Armorer's Bandana)
Items can also fall from above. Item durability reduced.

`"all tangle"` (Magnetic Morion)
All adjacent units stick to one another.

`"more meal tickets"` (Bohto Boater)
Player gains +1HP. Find more Meal Tickets in Chester's shop.

`"start with 5 relics"`
Start a run with 5 relics.

`"magic floors"` (Teleporter's Trapper)
Magic floors will appear in each level.

`"character quest"` (Quester's Hat)
Levels behave as if on the selected character's Quandary Quest.

`"offering trial"` (Cornucopia Hat)
Levels behave as if on the Trial of Offerings. This is the hat version of the `"cursed mode"` modifier.

`"hat shuffler"` (Shuffler's Hat)
Worn hats, except for this one, will be randomly shuffled after each stage.

`"pickles"` (Picklehaube)

Some potions are replaced with pickle jars that inflict 2 poison damage.

`"random units"` (Summoning Slouch)
Random units occasionally appear.

`"monster"` (Monstrous Montera)
Transform into your character's monster.

`"beeto breach"` (Dungstalker)
Beetos invade each level.

`"bombs away"` (Bomb Bowler)
Bombs appear more often.

`"forever overtime"`
Start game in overtime.

`"spinwulf spirit"` (Spinwulf Snood)
Move two spaces at a time.

`"no UI"` (No-Peek Nightcap)
Parts of the interface are hidden.

`"clearance"` (Bugle Hat)
Start with Bountiful Bugle. Relics are cheaper.

`"no loners"` (Extrovert Hat)
Can't attack lone foes or blocks.

`"double garbage"`
Double damage and junk blocks.

`"no relics"` (Shop Lock Shako)
Relics will not appear.

`"no items"` (Chest Lid.)
Items will not appear.

`"single stock"`
One stock.

Functions Reference

:savedata_save(key:string, value:int)

Saves specified value for the specified key in the current save file under current mod's section. See saving & loading for more info.

Argument	Type	Description
`key`	`string`	The key in the save file.
`value`	`int`	The value to save.

Returns: undefined

:savedata_read(key:string, …values)

Returns a value for the specified key in the save file. The mod can only access its own section. See saving & loading for more info.

Argument	Type	Descriptiond
`key`	`string`	The key in the save file.
`?default`	`any`	[optional, default is `undefined`] The value that is returned if the key is not found.

Returns: any

file_text_open(filename:string, read:bool)

Opens a text file for reading/writing and returns its ID for future reference or `-1`. The file path is relative to the mod's folder. See saving & loading for more info.

Argument	Type	Descriptiond
`filename`	`string`	The name of the file to open (can also include a relative path).

`read`	`bool`	Whether to open the file for reading (`true`) or writing (`false`).

Returns: any

The rest of the file functions are identical to default Game Maker ones:

- file_text_read_real(file)
- file_text_read_string(file)
- file_text_readln(file)
- file_text_write_real(file, value:int)
- file_text_write_string(file, value:string)
- file_text_writeln(file)
- file_text_eoln(file)
- file_text_eof(file)
- file_text_close(file)

:rpc_call(both_parties:bool, function:string, …values)

Runs a specified function for both you and your opponent in online play.

Usually your custom code is simultaneously applied to both local and remote representations of the character (item/modifier/etc.), but certain conditions like the button press only happen locally and will cause the game state to desync. You can use `rpc_call` to avoid this type of scenario and make your mods fully compatible with online.

The function you call has to be defined inside your mod. You can use the `rpc_sender_is_local`, `rpc_sender_is_remote`, `netcode_local_player_id` and `netcode_remote_player_id` variables inside it to get more data about the environment of the function call.

Argument	Type	Description
`both_parties`	`bool`	If set to `true`, the function will be executed for both players. If set to `false`, it will run for the remote player only.
`function`	`string`	The name of the function to run.
`?args`	`any`	Any number of arguments for the function.

Returns: n/a

Example:

```
if (grid_master.kSpecial) {
    rpc_call(true, "player_heal", 2); // run the script for both parties
}

function player_heal(_amount) {
    var _player_id;
    // get the player_id of the grid to apply the script to:
    if (rpc_sender_is_local) { // if we are the sender,
        _player_id = netcode_local_player_id; // use local player
    } else {
        _player_id = netcode_remote_player_id; // else, use remote
player
    }
    var _player;
    with (oGrid) if (player_id == _player_id) _player = player; // get
the instance id of the player
    // and apply heal:
    unit_heal(_player, _amount);
}
```

:mod_call(file_id:string)

Calls `controller_custom.gml` script from another mod. The mod has to be remote and enabled. Returns if the script was run successfully (`true`) or not (`false`).

Argument	Type	Descriptiond
`file_id`	`string`	The file id of the mod to run the script for. It's stored in `mod_info.ini` under the `url` field (i.e. `"3077703330"`).

Returns: bool

:mod_is_enabled(file_id:string)

Returns if the specified mod is enabled (`true`) or not (`false`). The mod has to be remote.

Argument	Type	Descriptiond

`file_id`	`string`	The file id of the mod to run the script for. It's stored in `mod_info.ini` under the `url` field (i.e. `"3077703330"`).

Returns: bool

sprite_add(sprite_name:string, filename:string, image_number:int, ...values)

Analog of the default Game Maker's sprite_add. Loads an external sprite and returns its index to be further referenced in code. The newly created sprite can also be retrieved using sprite_get.

Argument	Type	Description
`sprite_name`	`string`	The name of the sprite to be retrieved by sprite_get. If you provide an empty string, `filename` will be used as a name instead.
`filename`	`string`	The relative path to the sprite file. Should be a `.png` image. If the sprite is not found, `sOccupiedCell` will be returned.
`image_number`	`int`	The number of sub-images in the sprite.
`?removeback`	`bool`	[optional, default is `true`] Indicates whether to make all pixels with the background color (left-bottom pixel) transparent.
`?smooth`	`bool`	[optional, default is `false`] Indicates whether to smooth the edges if transparent.
`?xorig`	`int`	[optional, default is `0`] Indicates the x position of the origin of the sprite.
`?yorig`	`int`	[optional, default is `0`] Indicates the y position of the origin of the sprite.
`?palette`	`string|sprite asset`	[optional, default is `noone`]

		The palette used to recolor the sprite. If this is set, the newly added sprite can be drawn via draw_sprite_palette and draw_sprite_palette_ext.

Returns: sprite asset

sprite_get(sprite_name:string)

Can be used to get the sprite index of any sprite: either built-in or loaded by your mod via sprite_add. You can see the list of built-in sprite names using the VFX/SFX Viewer.

Argument	Type	Description
`sprite_name`	`string`	The name of the built-in sprite or the one previously loaded by sprite_add. If no sprite is found by this name, `sOccupiedCell` will be returned.

Returns: sprite asset

sprite_replace(sprite_name:string, filename:string, image_number:int, ...values)

Analog of the default Game Maker's sprite_replace. Replaces one of the built-in sprites with an external one. **Warning:** replacing too many sprites or too large sprites with it will eventually lead to memory leak!

Argument	Type	Description
`sprite_name`	`string`	The name of the built-in sprite (you can use the VFX Viewer to see the list of all in-game sprites).
`filename`	`string`	The relative path to the sprite file. Should be a `.png` image.
`image_number`	`int`	The number of sub-images in the sprite.
`?removeback`	`bool`	[optional, default is `true`]

		Indicates whether to make all pixels with the background color (left-bottom pixel) transparent.
`?smooth`	`bool`	[optional, default is `false`] Indicates whether to smooth the edges if transparent.
`?xorig`	`int`	[optional, default is `0`] Indicates the x position of the origin of the sprite.
`?yorig`	`int`	[optional, default is `0`] Indicates the y position of the origin of the sprite.

Returns: bool

sprite_get_og(sprite_name:string)

Returns the original sprite that was previously replaced using sprite_replace.

Argument	Type	Description
`sprite_name`	`string`	The name of the sprite.

Returns: sprite asset

sfx_add(sfx_name:string, filename:string)

Analog of the default Game Maker's audio_create_stream. Loads an external sound and returns its index to be further referenced in code. The newly created sfx can also be retrieved using sfx_get.

Argument	Type	Description
`sfx_name`	`string`	The name of the sound to be retrieved by sfx_get. If you provide an empty string, `filename` will be used as a name instead.

`filename`	`string`	The relative path to the sound file. Should be in `.ogg` format. If the sound is not found, `sfxCantDoThat` will be returned.

Returns: sound asset

sfx_get(sfx_name:string)

Can be used to get the sound index of any SFX: either built-in or loaded by your mod via sfx_add. You can see the list of built-in sound and music names using the VFX/SFX Viewer.

Argument	Type	Description
`sfx_name`	`string`	The name of the built-in SFX or the one previously loaded by sfx_add. If no sfx is found by this name, `sfxCantDoThat` will be returned.

Returns: sound asset

:sfx_play(sfx)

Plays the specified sound. Can be built-in or previously loaded by the mod.

Argument	Type	Description
`sfx`	`string|sound asset`	The sound asset (use sfx_get to retrieve it) or the name of the SFX to play.

Returns: sound instance ID

:sfx_play_ext(sfx, …values)

Analog of the default Game Maker's audio_play_sound. Plays the specified sound. Can be built-in or previously loaded by the mod.

Argument	Type	Description
`sfx`	`string|sound asset`	The sound asset (use sfx_get to retrieve it) or the name of the sfx to play.

`?volume`	`float`	[optional, default is `1.3`] The volume for the sound.
`?pitch`	`float`	[optional, default is `1`] The pitch multiplier.
`?priority`	`int`	[optional, default is `0`] The channel priority for the sound.
`?loop`	`bool`	[optional, default is `false`] Sets the sound to loop or not.

Returns: sound instance ID

:sfx_stop(sfx)

Stops the specified sound. Can be built-in or previously loaded by the mod.

Argument	Type	Description
`sfx`	`string|sound asset|audio instance ID`	The sound asset (use sfx_get to retrieve it), the name of the SFX or the audio instance ID to stop.

Returns: n/a

:sfx_is_playing(sfx)

Returns if the specified sound is currently playing (`true`) or not (`false`). Can be built-in or previously loaded by the mod.

Argument	Type	Description
`sfx`	`string|sound asset|audio instance ID`	The sound asset (use sfx_get to retrieve it), the name of the SFX or the audio instance ID to get the status of.

Returns: bool

:sfx_pause(sfx)

Pauses the specified sound. Can be built-in or previously loaded by the mod.

Argument	Type	Description
`sfx`	`string|sound asset|audio instance ID`	The sound asset (use sfx_get to retrieve it), the name of the SFX or the audio instance ID to pause.

Returns: n/a

:sfx_resume(sfx)

Resumes the specified sound. Can be built-in or previously loaded by the mod.

Argument	Type	Description
`sfx`	`string|sound asset|audio instance ID`	The sound asset (use sfx_get to retrieve it), the name of the SFX or the audio instance ID to resume.

Returns: n/a

:sfx_is_paused(sfx)

Returns if the specified sound is currently paused (`true`) or not (`false`). Can be built-in or previously loaded by the mod.

Argument	Type	Description
`sfx`	`string|sound asset|audio instance ID`	The sound asset (use sfx_get to retrieve it), the name of the SFX or the audio instance ID to get the status of.

Returns: bool

:sfx_set_volume(sfx, volume:float, time:int)

Sets the volume of the specified sound during the specified amount of time.

Argument	Type	Description
`sfx`	`string|sound asset|audio instance ID`	The sound asset (use sfx_get to retrieve it), the name of the SFX or the audio instance ID to set the volume of.
`volume`	`float`	The volume to set.
`time`	`int`	The length for the volume change in milliseconds.

Returns: n/a

:sfx_get_volume(sfx)

Returns the volume of the specified sound. Can be built-in or previously loaded by the mod.

Argument	Type	Description
`sfx`	`string|sound asset|audio instance ID`	The sound asset (use sfx_get to retrieve it), the name of the SFX or the audio instance ID to get the status of.

Returns: int

:vfx_play(sprite, unit:ref)

Creates oPop object at the specified unit's location and returns its instance ID. This object can be assigned any sprite and it will be destroyed after the sprite's animation ends. Useful for displaying VFX like the slash effect when the unit gets attacked. Returns `noone` if the object couldn't be created.

Argument	Type	Description
`sprite`	`string|sprite asset`	The sprite asset (use sprite_get to retrieve it) or the name of the sprite to assign to the VFX.

`unit`	`ref`	The instance ID of oUnit to play the VFX on.

Returns: instance ID or noone

:vfx_play_at(sprite, x:int, y:int)

Creates oPop object at the specified location and returns its instance ID. This object can be assigned any sprite and it will be destroyed after the sprite's animation ends. Unlike vfx_play, this function is not tied to an instance of oUnit. Returns `noone` if the object couldn't be created.

Argument	Type	Description
`sprite`	`string|sprite asset`	The sprite asset (use sprite_get to retrieve it) or the name of the sprite to assign to the VFX.
`x`	`int`	The x-position to create the object at. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position to create the object at. Should be between 0 and GRID_HEIGHT.

Returns: instance ID or noone

:gib_play(sprite, unit:ref)

Creates oGib object at the specified unit's location and returns its instance ID. This object can be assigned any sprite and will move, affected by gravity. Useful for creating effects like an item being tossed out when it runs out of durability. Returns `noone` if the object couldn't be created.

Argument	Type	Description
`sprite`	`string|sprite asset`	The sprite asset (use sprite_get to retrieve it) or the name of the sprite to assign to the gib.
`unit`	`ref`	The instance ID of oUnit to create the gib on.

Returns: instance ID or noone

print(...values)

Takes any number of arguments, and packs them together in a string. Outputs the string to the mod log and also returns it. Useful for debugging.

Argument	Type	Description
`…values`	`any`	The part of the output string.

Returns: string

Example:

```
print("my hp is: ", hp, ", my max hp is: ", hmpax);
// output: my hp is: 5, my max hp is: 5
```

log_clear()

Clears the mod log. This is a code equivalent of `Ctrl+Delete`.

Returns: n/a

get_seed()

Returns seed for the current run. If the game is not currently in Adventure mode, returns an empty string.

Returns: string

is_hub()

Checks if the player is currently in the hub area (including Camp and Castle Quandary). For custom prefab the function will return `true` if `prefab_sideroom` is `false`.

Returns: bool

is_adventure()

Checks if the game is currently in Adventure mode.

Returns: bool

is_versus()

Checks if the game is currently in Versus mode.

Returns: bool

is_title()

Checks if the game is currently on the title screen or main menu.

Returns: bool

:is_shop()

Checks if the player is currently in the shop (this includes `"sideroom armorer"`, `"sideroom tief shop"` and `"sideroom troupple"`). For custom prefab the function will return `true` if it is marked as `prefab_shop`.

Returns: bool

:is_sideroom()

Checks if the game is currently in the sideroom. For custom prefab the function will return `true` if it is marked as `prefab_sideroom`.

Returns: bool

:is_at_fight_zone()

Checks if the game is currently in the fight zone (level, sideroom or boss).

Returns: bool

is_minigame()

Checks if the player is currently playing Mona's minigame in the camp.

Returns: bool

game_is_paused()

Checks if the game is currently paused (includes IGGE being active).

Returns: bool

menu_get()

Gets the name of the currently displayed menu. Returns `undefined` if no menu is active.

Returns: string or undefined

:timed_chest_tick()

Increases timed chest's timer by 1 and returns its current value.

Returns: int

:frozen_time_tick()

Increases freeze time counter by 1 and returns its current value.

Returns: int

:check_gems()

Returns the current number of player's gems.

Returns: int

:add_gems(gems:int, …values)

Gives the specified amount of gems to the player. Returns the new gem amount. Will return -1 if the oGrid object is not found.

Argument	Type	Description
`gems`	`int`	The amount of gems to add.
`?camp`	`bool`	[optional, default is `false`] Whether to modify the camp bank instead of the run-specific gems.

Returns: int

:remove_gems(gems:int, …values)

Removes the specified amount of gems from the player (gems can't go into the negative). Returns the new gem amount. Will return -1 if the oGrid object is not found.

Argument	Type	Description
`gems`	`int`	The amount of gems to remove.
`?show_popup`	`bool`	[optional, default is `true`] Whether to show money lost popup.
`?camp`	`bool`	[optional, default is `false`]

		Whether to modify the camp bank instead of the run-specific gems.

Returns: int

quandary_get_status(unit_type, quandary:int)

Returns the status of the specified quandary:

`-1`: not unlocked
`0`: not beaten
`1`: bad ending
`2`: good ending

Argument	Type	Description
`unit_type`	`string|int`	The unit ID to check for.
`quandary`	`int`	The quandary index of the quandary to check for.

Returns: int

get_quandary_index()

Returns the quandary index of the currently applied quandary. Returns `-1` for non-quandary runs.

Returns: int

:item_create(item:string, x:int, y:int)

Creates an item at the specified position and returns its instance ID. Mostly designed for custom shops, but you can use it to spawn items during other parts of the game as well. Returns `noone` if the item couldn't be created.

Argument	Type	Description

`item`	`string`	The item ID of the item to spawn.
`x`	`int`	The x-position to spawn the item at. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position to spawn the item at. Should be between 0 and GRID_HEIGHT.

Returns: instance ID or noone

:unit_create(unit:string, x:int, y:int, …values)

Creates a unit of the specified unit ID and returns the instance ID of the created oUnit. If the unit ID is an empty string, the game will fetch a random unit from level spawns. If the unit cannot be created at the specified coordinates, the game will try to find another position within the same row. Returns `noone` if the unit couldn't be created.

Argument	Type	Description
`unit`	`string`	The unit ID of the unit to create.
`x`	`int`	The x-position to spawn the unit at. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position to spawn the unit at. Should be between 0 and GRID_HEIGHT.
`?skip_check`	`bool`	[optional, default is `false`] Whether to bypass checks on unit creation. If set to `true`, the game will not try to spawn the unit in the same row if it could not be spawned at the initial coordinates.

Returns: instance ID or noone

unit_destroy(unit:ref)

Removes a unit from the grid without executing events tied to its death (no corpse is created, no on-death behavior is triggered, etc.). Returns if the unit was destroyed successfully (`true`) or not (`false`).

Note: If you want all those events to trigger, use `instance_destroy()`.

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to destroy.

Returns: bool

unit_is_valid(unit:ref)

Returns if the specified unit exists (`true`) or not (`false`). It is good practice to call this before editing a unit (i.e. after getting it using grid_get).

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to check for.

Returns: bool

Example:

```
var _unit = grid_get(2, 2);

if (unit_is_valid(_unit)) { // if the unit exists
    // do something!
}
```

unit_info(unit_type)

Returns a struct with information about the specific unit type.

Argument	Type	Description
`unit_type`	`string|int`	The unit ID to get the info for.

Returns: struct

Example:

```
var _info = unit_info(item);
if (_info != noone) { // if data is valid:
      var _item_sprite = _info.sprite; // get the sprite of the
currently equipped item
}
```

The struct holds these variables:

- `unit_type`
- `unit_id`
- `unit_id_string`
- `text_name`
- `text_description`
- `dialogue`
- `dialogue_loop`
- `base_hp`
- `base_attack`
- `skill_a`
- `skill_b`
- `has_behaviors`
- `behaviors_string`
- `relics_start`
- `is_modded`
- `has_skins`
- `bombproof`
- `immune`
- `untouchable`
- `moveable`
- `levitate`
- `immortal`
- `inert`
- `hits_back`
- `diagonal`
- `ignore_chains`
- `sprite` (sprite_idle)
- `sprite_emote`
- `sprite_emote_out`
- `sprite_hub`
- `sprite_dead`
- `sprite_weak`
- `sprite_dying`
- `sprite_map`
- `sprite_aim`

- `sprite_charge`
- `sprite_special`
- `sprite_body`
- `sprite_skill_up`
- `sprite_other`
- `sprite_skill_down`
- `sprite_conceal`
- `sprite_skill`
- `sprite_moonland`
- `sprite_head`
- `sprite_fly`
- `sprite_fall`
- `mini_sprite`
- `portrait` (portrait_idle)
- `portrait_hurt`
- `portrait_win`
- `helm_portrait_hurt`
- `helm_portrait_win`
- `helm_sprite_emote`
- `helm_sprite_emote_out`
- `helm_sprite_dead`
- `helm_sprite_dying`
- `helm_sprite_weak`
- `sfx_hurt`
- `sfx_hurt_pitch`
- `sfx_death`
- `sfx_death_pitch`
- `crown_offset`
- `emote_crown_offset`
- `hat_x_delta`
- `hat_y_delta`
- `stache_x_delta`
- `stache_y_delta`
- `stache_x_prt_delta`
- `stache_y_prt_delta`
- `boss_second_phase`

unit_get_type_int(unit_type:string)

Converts unit ID from a string into an int. Unit IDs are stored as integers for oUnit in the `unit_type` variable, so this function might be useful for doing comparison of different sorts. Returns `noone` if the unit ID couldn't be converted.

Argument	Type	Description
`unit_type`	`string`	The unit ID to convert to an integer.

Returns: int or noone

Example:

```
if (unit_type == unit_get_type_int("beeto")) { // if the unit is beeto
    // do something!
}
```

unit_get_type_string(unit_type:int)

Converts unit ID from an int to a string. An inverse to unit_get_type_int. Returns `"?"` if the unit ID couldn't be converted.

Argument	Type	Description
`unit_type`	`int`	The unit ID to convert to a string.

Returns: string

unit_is_foe(unit:ref)

Returns if the specified unit is an enemy (`true`) or not (`false`).

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to check for.

Returns: bool

unit_is_item(unit_type)

Returns if the specified unit is an item (`true`) or not (`false`).

Argument	Type	Description
`unit_type`	`int|string`	The unit ID to check for.

Returns: bool

unit_transform(unit:ref, unit_type:string)

Transforms the specified oUnit into a different unit type. Returns if the unit was transformed successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to transform.
`unit_type`	`string`	The unit ID to transform to.

Returns: bool

player_transform(unit:ref, unit_type:string)

Similar to unit_transform, but to change the player's character. You need to provide the player unit. Returns if the unit was transformed successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to transform.
`unit_type`	`string`	The unit ID to transform to.

Returns: bool

unit_get_grid()

Gets a unit's grid. Should be only called from within oUnit. This is the same as referencing `grid_master.grid`. Returns `noone` if the object couldn't be found.

Returns: ds_grid or noone

get_grid_master()

Finds the currently active oGrid. If called inside an oUnit it will return `grid_master`.
Returns `noone` if the object couldn't be found.

Note: calling this in Versus outside a unit will only return one of the grids.

Returns: instance ID or noone

unit_is_in_chain(unit:ref)

Checks if the specified unit forms a chain of 3 or more units.

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to check for.

Returns: bool

unit_get_chain(unit:ref, …values)

Returns an array of units that chain together from a given unit.

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to check the chain for.
`?max_count`	`int`	[optional, default is `72`] The maximum chain size.

Returns: array

unit_get_chain_size(unit:ref, max_count:int)

Returns the chain size for the specified unit. If the instance is invalid, returns `0`.

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to check for.
`max_count`	`int`	The chain size to stop the check at.

Returns: int

:ability_aim(sprite:int)

Enables ability aim. This function is designed to be called from within `ability_aim.gml`.

Argument	Type	Description
`sprite`	`string|sprite asset`	The sprite asset (use sprite_get to retrieve it) or the name of the sprite to use for the aiming animation.

Returns: n/a

:unit_damage(victim, attacker:ref, …values)

Deals damage to a unit. Here you specify who the attacker is and who the victim is: damage and chain will be calculated automatically.

Argument	Type	Description
`victim`	`ref`	The instance ID of oUnit to deal damage to. You can provide an array of units instead to attack multiple at once and count as the same chain.
`attacker`	`ref`	The instance ID of oUnit or any other object that is dealing damage.
`?damage`	`int`	[optional, default is `undefined`] The specific damage to force. By default the damage equals the attacker's attack.

`?chains`	`bool`	[optional, default is `true`] If the damage should chain or not.
`?process_chain`	`bool`	[optional, default is `true`] Whether this damage is considered a chain. If not, the chain won't give any gem bonus or trigger certain relics.

Returns: n/a

:damage_indirect(victim, attacker:ref, …values)

Deals damage to a unit. Unlike unit_damage, indirect damage does not trigger certain actions like collecting keys, opening chests and doors, etc.

Argument	Type	Description
`victim`	`ref`	The instance ID of oUnit to deal damage to. You can provide an array of units instead to attack multiple at once and count as the same chain.
`attacker`	`ref`	The instance ID of oUnit or any other object that is dealing damage.
`?damage`	`int`	[optional, default is `undefined`] The specific damage to force. By default the damage equals the attacker's attack.
`?prevent_chain`	`bool`	[optional, default is `false`] If the damage should prevent a chain, or not.
`?process_chain`	`bool`	[optional, default is `true`] Whether this damage is considered a chain. If not, the chain won't give any gem bonus or trigger certain relics.

Returns: n/a

:damage_simple(victim:ref, …values)

Deals damage to the specified unit without calculating chain or applying relics.

Argument	Type	Description
`victim`	`ref`	The instance ID of oUnit to deal damage to.
`?damage`	`int`	[optional, default is `1`] The damage to deal.

Returns: n/a

:damage_player(enemy:ref, …values)

Damages the player, can be used for custom enemy counterattacks. Returns if the player was damaged successfully (`true`) or not (`false`).

Argument	Type	Description
`enemy`	`ref`	The instance ID of oUnit that deals damage.
`?dir_h`	`int`	[optional, default is `0`] The x component of the attack impulse. Used to animate the player nudge (no animation if set to `0`).
`?dir_v`	`int`	[optional, default is `0`] The y component of the attack impulse. Used to animate the player nudge (no animation if set to `0`).
`?exrta_dmg`	`int`	[optional, default is `0`] The extra damage (on top of enemy's attack value) to apply.

Returns: bool

damage_chain_start()

Makes it so that all further calls of unit_damage and damage_indirect count towards one chain.

Returns: n/a

damage_chain_end()

Processes the chain after all unit_damage and damage_indirect calls.

Returns: n/a

damage_chain_get()

Returns an array of instance IDs of the units currently in the chain. Only functional after damage_chain_start is called.

Returns: array

Example:

```
damage_chain_start();
damage_indirect(target, id);
var _enemies = damage_chain_get();

for (i = 0; i < array_length(_enemies); i++) { // for every enemy in the chain
      var _enemy =_enemies[i];
      with (_enemy) print(hp); // print its hp
}
damage_chain_end();
```

:dialogue_define_start()

Should be called before the dialogue defining sequence.

Returns: n/a

:dialogue_define_end()

Should be called after the dialogue defining sequence.

Returns: n/a

:dialogue_define(unit:int, accent:int, text:string, ...values)

Defines a dialogue line for the specified unit. The lines should be defined in the order you want them to appear. Call `dialogue_define_start()` before the sequence and `dialogue_define_end()` after it.

Argument	Type	Description
`unit`	`int`	The instance ID of oUnit to speak the line.
`accent`	`int`	The accent of the dialogue window: • `-1` : accent on the left • `1` : accent on the right • `0` : no accent
`text`	`string`	The dialogue line. Can include scribble formatting.
`?nudge`	`bool`	[optional, default is `true`] If the speaking unit should jump in place before saying the line (`true`) or not (`false`).

Returns: n/a

dialogue_play(...values)

Plays the dialogue sequence defined previously.

Argument	Type	Description
`?pause`	`bool`	[optional, default is `true`] Whether to pause the game during dialogue (`true`) or not (`false`).
`?instant`	`bool`	[optional, default is `false`] Whether the text should appear instantly (`true`) or with a typewriter effect (`false`).

Returns: n/a

Example:

```
dialogue_define_start();
dialogue_define(id, 1, "Can you stop that? My eyes hurt!");
with (get_grid_master().player) {
      dialogue_define(id, -1, "[shake]NO![/shake]");
}
dialogue_define_end();
dialogue_play();
```

:dialogue_proceed()

Instantly moves to the next line in the current dialogue sequence.

Returns: n/a

unit_toss(unit:ref, from_unit:ref, …values)

Tosses a unit into the air to a random available position in the grid. For animating the effect, an instance of oJumper will be created and returned by the function. Returns `noone` if the unit couldn't be tossed.

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to be tossed.

`from_unit`	`ref`	The instance ID of oUnit to toss the other unit from.
`?range`	`int`	[optional, default is `0`] The range in tiles to launch the unit in. If set to `0`, any tile can be the destination.
`?damage`	`int`	[optional, default is `0`] The damage the tossed unit will take.
`?xoffset`	`int`	[optional, default is `0`] The x-offset relative to the `from_unit`.
`?yoffset`	`int`	[optional, default is `0`] The y-offset relative to the `from_unit`.

Returns: instance ID or noone

unit_jump(unit:ref, …values)

Makes a unit jump to a random available position in the grid. For animating the effect, an instance of oJumper will be created and returned by the function. Returns `noone` if the jump couldn't happen.

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to jump.
`?horizontal_range`	`array`	[optional, default is `[0,7]`] The range in tiles for the unit to jump to at random. You can set it to something like `[2,2]` if you want a specific tile.
`?vertical_range`	`array`	[optional, default is `[1,8]`] The range in tiles for the unit to jump to at random. You can set it to something like `[2,2]` if you want a specific tile.
`?damage`	`int`	[optional, default is `0`] The damage the unit will take when landing a jump.
`?height`	`int`	[optional, default is `1`] The height of the jump.

`?speed`	`int`	[optional, default is `1`] The speed of the jumping animation.

Returns: instance ID or noone

unit_move(unit:ref, x:int, y:int)

Moves a unit to the specified spot in the grid. Returns `true` if the move was successful, otherwise returns `false`.

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to move.
`x`	`int`	The x-position to move to. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position to move to. Should be between 0 and GRID_HEIGHT.

Returns: bool

:unit_bump(x:int, y:int, …values)

Bumps a spot in the grid (whether it's empty or not) and returns the bumped target, or `noone` if the target wasn't found. Unlike unit_damage, it also triggers certain actions like activating shrines and portals. The function is mainly designed to be called from weapon mods to implement custom attack patterns (see Whip mod for reference).

Argument	Type	Description
`x`	`int`	The x-position to attack (relative to the attacker). Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position to attack (relative to the attacker). Should be between 0 and GRID_HEIGHT.
`?attacker`	`ref`	If not used from a weapon mod in the bump event, you can provide the attacker here instead.

Returns: instance ID or noone

Example (called from `bump.gml` of the item mod):

```
if (move_x != 0) { // attack horizontally
      unit_bump(move_x, -1); // attack a cell to the left
      unit_bump(move_x,  1); // attack a cell to the right
} else { // attack vertically
      unit_bump(-1, move_y); // attack a cell to the top
      unit_bump(1,  move_y); // attack a cell to the bottom
}
```

unit_heal(unit:ref, value:int)

Heals a unit for the specified amount of HP. HP can't get past the `maxhp` in the process. Returns the amount of HP restored.

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to heal.
`value`	`int`	The amount of HP to heal.

Returns: int

unit_freeze(unit:ref, value:int)

Freezes a unit for the specified amount of turns. Block units can't be frozen. Returns if the unit was successfully frozen (`true`) or not (`false`).

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to freeze.
`value`	`int`	The amount of turns to freeze the unit for.

Returns: bool

unit_poison(unit:ref, value:int)

Poisons a unit for the specified amount of turns. Certain units like blocks or those with the `"poison_immune"` behavior can't be poisoned. Returns if the unit was successfully poisoned (`true`) or not (`false`).

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to poison.
`value`	`int`	The amount of turns to poison the unit for.

Returns: bool

:unit_animate(sprite, …values)

Call this inside oUnit to play a sprite. Once the animation is over it will revert back to its idle.

Argument	Type	Description
`sprite`	`string|sprite asset`	The sprite asset (use sprite_get to retrieve it) or the name of the sprite to play.
`?times`	`int`	[optional, default is `1`] The amount of times to loop through the sprite.
`?index`	`int`	[optional, default is `0`] The frame to start the animation on.
`?speed`	`float`	[optional. default is `0.15`] The animation speed (frames per game tick).

Returns: n/a

unit_has_behavior(unit:ref, behavior:string)

Returns if the unit has the specified behavior (`true`) or not (`false`).

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit to check.

`behavior`	`string`	The behavior to check for.

Returns: bool

unit_type_has_behavior(unit_type:int, behavior:string)

Returns if the unit has the specified behavior (`true`) or not (`false`). Unlike unit_has_behavior, this function is based on the unit type, not the instance of oUnit.

Argument	Type	Description
`unit_type`	`int`	The unit type to check. Use unit_get_type_int to get it from the unit ID.
`behavior`	`string`	The behavior to check for.

Returns: bool

unit_draw()

Calls a unit's draw event. Only call this within a draw event (i.e. `draw.gml`).

Returns: n/a

unit_draw_end()

Calls a unit's draw end event. Only call this within a draw event (i.e. `draw.gml`).

Returns: n/a

draw_text_outline(x:int, y:int, text:string, ...values)

Draws text similar to the default Game Maker's draw_text, but with a black outline.

Argument	Type	Description

`x`	`int`	The x-position to draw the text at.
`y`	`int`	The y-position to draw the text at.
`text`	`string`	The text to draw.
`?scale`	`int`	[optional, default is `1`] The scale of the text.
`?outline`	`int`	[optional, default is `2`] The thickness of the outline in pixels.

Returns: n/a

scribble_draw(x:int, y:int, text:string, …values)

Draws text using scribble.

Argument	Type	Description
`x`	`int`	The x-position to draw the text at.
`y`	`int`	The y-position to draw the text at.
`text`	`string`	The text to draw.
`?flip_textures`	`bool`	[optional, default is `false`] Set this to `true` if you plan to use button tags inside the string.

Returns: n/a

scribble_draw_outline(x:int, y:int, text:string, ...values)

Draws text using scribble with a black outline.

Argument	Type	Description
`x`	`int`	The x-position to draw the text at.
`y`	`int`	The y-position to draw the text at.

`text`	`string`	The text to draw.
`?outline`	`int`	[optional, default is `1`] The thickness of the outline in pixels.
`?flip_textures`	`bool`	[optional, default is `false`] Set this to `true` if you plan to use button tags inside the string.

Returns: n/a

scribble_set_wrap(max_box_width:int, max_box_height:int, ...values)

Sets scribble's text wrapping state.

Argument	Type	Description
`max_box_width`	`int`	The maximum line width for each line of text. Use a negative number for no limit.
`max_box_height`	`int`	The maximum height for the whole textbox. Use a negative number for no limit.
`?wrap_characters`	`bool`	[optional, default is `false`] Whether to wrap text per character (rather than per word).

Returns: n/a

draw_key(x, y, input, …values)

Draws a key prompt at the specified position. Returns the sprite of the button (`noone` if it couldn't be drawn). If you want to insert button prompts inside the scribble string, use scribble_draw and scribble_draw_outline instead.

Argument	Type	Description
`x`	`int`	The x-position to draw key at.
`y`	`int`	The y-position to draw key at.

`input`	`string`	The input to draw key for: • `"up"` • `"down"` • `"left"` • `"right"` • `"confirm"` • `"item"` • `"cancel"` • `"pause"` • `"ability"` • `"turbo"` • `"examine"`
`?size`	`int`	[optional, default is `1`] The size of the key: • `0` - 24x24 • `1` - 16x16 • `2` - 16x16, without any extra descriptive text (e.g. "Share" on PS4)
`?player`	`int`	[optional, default is `1`] The player index to get the key for.
`?outline`	`int`	[optional, default is `0`] The thickness of the black outline for the key.

Returns: sprite asset or `noone`

:draw_sprite_palette(sprite, image_index:int, x:int, y:int, skin:int)

Draws sprite with the palette colors.

Argument	Type	Description
`sprite`	`string|sprite asset`	The sprite asset (use sprite_get to retrieve it) or the name of the sprite to draw.
`image_index`	`int`	The image index of the sprite to draw.
`x`	`int`	The x-position to draw the sprite at.
`y`	`int`	The y-position to draw the sprite at.

`skin`	`int`	The palette skin to use (0 to 10).

Returns: n/a

:draw_sprite_palette_ext(sprite, image_index:int, x:int, y:int, skin:int, …values)

Draws sprite with the palette colors. Is an extended version of draw_sprite_palette.

Argument	Type	Description
`sprite`	`string|sprite asset`	The sprite asset (use sprite_get to retrieve it) or the name of the sprite to draw.
`image_index`	`int`	The image index of the sprite to draw.
`x`	`int`	The x-position to draw the sprite at.
`y`	`int`	The y-position to draw the sprite at.
`skin`	`int`	The palette skin to use (0 to 10).
`?image_xscale`	`int`	[optional, default is `1`] The image_xscale to draw the sprite with.
`?image_yscale`	`int`	[optional, default is `1`] The image_yscale to draw the sprite with.
`?image_angle`	`int`	[optional, default is `0`] The image_angle to draw the sprite with.
`?image_blend`	`int`	[optional, default is `c_white`] The image_blend to draw the sprite with.
`?image_alpha`	`float`	[optional, default is `1`] The image_alpha to draw the sprite with.

Returns: n/a

unit_change_costume(unit:ref, costume:int, ...values)

Changes a unit's costume (palette).

Argument	Type	Description
`unit`	`ref`	The instance ID of the oUnit you want to change the costume for.
`costume`	`int`	The costume to set.
`?total`	`int`	[optional, default is `10`] The total amount of costumes. **WARNING:** Changing the total to a number higher than 10 could cause issues since the game only uses a max of 10 palettes.

Returns: n/a

sprite_default()

Returns the unit's sprite to its default state, either idle sprite or weak sprite. Should be only called from within oUnit. Returns if the unit sprite was reset successfully (`true`) or not (`false`).

Returns: bool

:hub_dialogue_reset(unit:ref)

Resets the unit's dialogue inside the hub. This only resets dialogue if it has been specified inside `unit.gml` or it is an existing unit that had dialogue.

Argument	Type	Description
`unit`	`ref`	The instance ID of the oUnit to reset dialogue for.

:ban_item(item)

Prevents the item from appearing in shops or as a reshuffle result. The effect persists between runs and is only canceled by calling unlock_item. Returns if the ban was successful (`true`) or not (`false`).

Argument	Type	Description
`item`	`int|string`	The item ID of the item to ban.

Returns: bool

:unlock_item(item)

Unbans the item previously banned by ban_item. Returns if the unban was successful (`true`) or not (`false`).

Argument	Type	Description
`item`	`int|string`	The item ID of the item to unban.

Returns: bool

:get_relic_data(relic:string)

Returns data for the specified relic that can be further assigned to a shop slot when creating a custom shop or be used in has_relic and give_relic functions.

Argument	Type	Description
`relic`	`string`	The relic ID of the relic to get data for.

The data is packed into an array:

Array Index	Type	Data
`0`	`string`	name
`1`	`float`	store description
`2`	`sprite asset`	sprite
`3`	`int`	relic ID
`4`	`int`	price

`5`	`string`	description
`6`	`bool`	indicates if the player has spent relic's effect or not; currently used only by `"fenix feather"` If set to 2 or higher, the relic is considered "snared" and becomes inactive.
`7`	`int`	unlock price
`8`	`int`	weight

Returns: array

:has_relic(relic)

Checks if the player has the specified relic equipped, and returns how many instances of it (useful since meal tickets are stackable).

Argument	Type	Description
`relic`	`array|string`	The relic to check. Can be a relic ID or an array returned by get_relic_data.

Returns: real

give_relic(relic, ...values)

Grants a relic to the player. Returns whether it was successfully granted (`true`) or not (`false`).

Argument	Type	Description
`relic`	`array|string`	The relic to give. Can be a relic ID or an array returned by get_relic_data.
`?player`	`int`	[optional, default is `1`] The player index to give the relic to.
`?show_popup`	`bool`	[optional, default is `true`]

		Whether to show relic' acquire popup.

Returns: bool

give_key_fragment()

Grants a key fragment to the player.

Returns: n/a

:ban_relic(relic)

Prevents the relic from appearing in shops. The effect persists between runs and is only canceled by calling unlock_relic. Returns if the ban was successful (`true`) or not (`false`).

Argument	Type	Description
`relic`	`int|string`	The relic ID of the relic to ban.

Returns: bool

:unlock_relic(relic)

Unbans the relic previously banned by ban_relic. Returns if the unban was successful (`true`) or not (`false`).

Argument	Type	Description
`relic`	`int|string`	The relic ID of the relic to unban.

Returns: bool

:disable_relic(relic, ...values)

Disables the relic for the duration of one level. Returns if it was successfully disabled (`true`) or not (`false`).

Argument	Type	Description
`relic`	`int|string`	The relic ID of the relic to disable.
`?player`	`int`	[optional, default is `1`] The player to target. If you want it to be the second player in Versus, use `2`.

Returns: bool

:enable_relic(relic, ...values)

Enables the relic that was previously disabled using disable_relic. Returns if it was successfully enabled (`true`) or not (`false`).

Argument	Type	Description
`relic`	`int|string`	The relic ID of the relic to disable.
`?player`	`int`	[optional, default is `1`] The player to target. If you want it to be the second player in Versus, use `2`.

Returns: bool

:relic_set_price(price:int)

This function is designed to be called from within the shop item when setting up a custom shop. Use it if you want the relic's price to be affected by other relics, hats or knight's abilities (i.e. chester's ability, Extravagant Gem Hat, etc.). Returns the new price after all the modifiers are applied.

Argument	Type	Description
`price`	`int`	The price to set.

Returns: int

Example:

```
with (item_create("shop relic", 0, 0))
      shop_item_price = 2000); // this relic's price is set in stone
}
with (item_create("shop relic", 0, 1))
      shop_item_price = relic_set_price(20000); // this relic's price
will be further modified
}
```

:save_purchase_info()

This function is designed to be called from a custom purchase function. It's used to save the fact the shop item was purchased and to make sure it doesn't appear on second and consequent shop visits.

Returns: n/a

:trap_create(trap:string, gridx:int, gridy:int)

Creates a trap at the specified position and returns the instance ID of the created oTrap object.

Argument	Type	Description
`trap`	`string`	The trap ID of the trap to create.
`x`	`int`	The x-position to spawn the trap at. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position to spawn the trap at. Should be between 0 and GRID_HEIGHT.

Returns: instance ID or noone

:trap_damage(unit:ref, damage:int)

Deals damage to a unit when called from a trap.

Argument	Type	Description
`unit`	`ref`	The oUnit of the unit to damage.
`damage`	`int`	The damage to deal.

Returns: n/a

:prefab_do(prefab_name:string)

Builds the specified prefab. Returns if it was built successfully (`true`) or not (`false`). Useful if you want to base your custom prefab on the existing one.

Argument	Type	Description
`prefab_name`	`string`	The prefab ID of the prefab to build.

Returns: bool

Note: going into a sideroom, then going to another sideroom inside of it should be done by returning to the main level first:

```
prefab_do("game");
prefab_do("next sideroom");
```

:prefab_create_quandary_portal(quandary:int, x:int, y:int)

Creates a quandary portal at the specified position. Returns if it was created successfully (`true`) or not (`false`).

Argument	Type	Description
`quandary`	`int`	The quandary index of the quandary to create.
`x`	`int`	The x-position to create the portal at. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position to create the portal at. Should be between 0 and GRID_HEIGHT.

Returns: bool

:prefab_set_player_position(x:int, y:int)

Defines the starting player position for the prefab. Returns if the position was set successfully (`true`) or not (`false`).

Argument	Type	Description
`x`	`int`	The x-position for the player. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position for the player. Should be between 0 and GRID_HEIGHT.

Returns: bool

:prefab_create_portal(x:int, y:int, goto:string, …values)

Creates a portal that leads to another prefab. Returns if it was created successfully (`true`) or not (`false`).

Argument	Type	Description
`x`	`int`	The x-position for the portal. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position for the portal. Should be between 0 and GRID_HEIGHT.
`goto`	`string`	The prefab ID of the prefab the portal should lead to.
`?name`	`string`	[optional, default is empty string] The text that will be displayed when the player approaches the portal (150 characters max).

Returns: bool

:prefab_decor(sprite, x:int, y:int)

Creates a decoration object that doesn't collide with other units. Returns if it was created successfully (`true`) or not (`false`).

Argument	Type	Description
`sprite`	`string|sprite asset`	Sprite asset (use sprite_get to retrieve it) or the name of the sprite to assign to the object.
`x`	`int`	The x-position for the object. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position for the object. Should be between 0 and GRID_HEIGHT.

Returns: bool

:prefab_decor_ext(sprite, x:int, y:int, depth, …values)

Creates a decoration object that doesn't collide with other units. This function is an extension of prefab_decor and allows more customization for the object. Returns if it was created successfully (`true`) or not (`false`).

Argument	Type	Description
`sprite`	`string|sprite asset`	Sprite asset (use sprite_get to retrieve it) or the name of the sprite to assign to the object.
`x`	`int`	The x-position for the object. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position for the object. Should be between 0 and GRID_HEIGHT.
`?depth`	`int|noone`	[optional, default is `noone`] The depth for the object. If set to `noone`, the object will be created on top of everything.

`?xscale`	`int`	[optional, default is `1`] The xscale for the object.
`?yscale`	`int`	[optional, default is `1`] The yscale for the object.
`?image_index`	`int|noone`	[optional, default is `noone`] The image index for the object's sprite. If set to `noone`, the sprite will loop through its frames instead.

Returns: bool

:prefab_wall(sprite, x:int, y:int)

Creates an obstacle at the specified position and returns the instance ID of the created object. The object is an instance of oUnit with unit ID of `"tree"`.

Argument	Type	Description
`sprite`	`string|sprite asset`	Sprite asset (use sprite_get to retrieve it) or the name of the sprite to assign to the object.
`x`	`int`	The x-position for the object. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position for the object. Should be between 0 and GRID_HEIGHT.

Returns: instance ID or noone

:prefab_hidden_wall(x:int, y:int)

Creates a hidden wall at the specified position and returns the instance ID of the created object. The object is an instance of oUnit with unit ID of `"tree"`.

Argument	Type	Description
`x`	`int`	The x-position for the wall. Should be between 0 and GRID_WIDTH.

`y`	`int`	The y-position for the wall. Should be between 0 and GRID_HEIGHT.

Returns: instance ID or noone

:prefab_door_light(x:int, y:int, …values)

Creates a visual effect that is used to indicate doorways inside the hub.

Argument	Type	Description
`x`	`int`	The x-position for the light in pixels.
`y`	`int`	The y-position for the light in pixels.
`?depth`	`int`	[optional, default is `0`] The depth of the light.
`?angle`	`int`	[optional, default is `0`] The angle of the light.

Returns: n/a

:sideroom_return_portal_random()

Creates a portal that leads to the main level at a random position. Should be only called from within a sideroom prefab. Returns if it was created successfully (`true`) or not (`false`).

Returns: bool

:sideroom_return_portal(x:int, y:int)

Creates a portal that leads to the main level at the specified position. Should be only called from within a sideroom prefab. Returns if it was created successfully (`true`) or not (`false`).

Argument	Type	Description
`x`	`int`	The x-position for the portal. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position for the portal. Should be between 0 and GRID_HEIGHT.

Returns: bool

:stage_get_unit_list()

Returns a ds_list with information on every unit that can spawn on the level. Each position in the list is an array:

Array Index	Type	Data
`0`	`int`	The unit ID of the unit (this ID is an integer, not a string. You can use unit_get_type_int to convert string IDs to integers)
`1`	`float`	The unit's spawn rate.

Returns: ds_list

Example:

```
var _enemies = ds_list_create(); // let's populate this list with enemy types that can spawn on the level
var _spawn_list = stage_get_unit_list();
var _spawn_list_size = ds_list_size(_spawn_list); // get list size to not ask for it every time in the loop

for (var i = 0; i < _spawn_list_size; i++) {
var _array = _spawn_list[| i]; // array containing information on the unit
	var _utype = _array[0]; // unit ID
	if (unit_type_has_behavior(_utype, "enemy")) {
		ds_list_add(_enemies, _utype); // if the unit is enemy, add it to our list
```

```
        }
}
```

:stage_get_random_unit_type()

Randomly selects a unit from the stage spawn pool and returns its unit ID. If no unit can be picked, returns beeto's ID.

Note: the returned ID is an integer, not a string. You can use unit_get_type_int to convert string IDs to integers.

Returns: real

:stage_skip_turn()

Forces a turn skip. Returns if it was done successfully (`true`) or not (`false`).

Returns: bool

:level_set(position:int, level:string)

Changes level at the specified position. Returns if level order was updated successfully (`true`) or not (`false`). Calling this function while on the map won't update it instantly - do it beforehand.

Argument	Type	Description
`position`	`int`	The position in the level order to update.
`level`	`string`	The stage ID of the level.

Returns: bool

:boss_set(position:int, boss:string)

Changes boss at the specified position in the level order. Returns if it was updated successfully (`true`) or not (`false`). Calling this function while on the map won't update it instantly - do it beforehand.

Argument	Type	Description
`position`	`int`	The position in the level order to update.
`boss`	`string`	The unit ID of the boss.

Returns: bool

:ban_sideroom(sideroom:string)

Prevents the specified sideroom from appearing in Adventure. The effect persists between runs and is only canceled by calling unlock_sideroom. Returns if the ban was successful (`true`) or not (`false`).

Argument	Type	Description
`sideroom`	`string`	The sideroom ID of the sideroom to ban.

Returns: bool

:unlock_sideroom(sideroom:string)

Unbans the sideroom previously banned by ban_sideroom. Returns if the unlock was successful (`true`) or not (`false`).

Argument	Type	Description
`sideroom`	`string`	The sideroom ID of the sideroom to unlock.

Returns: bool

:lvl_get_spawner_code(level)

Gets default spawn data for the specified level, so that you can modify it further. Returns if the data was retrieved successfully (`true`) or not (`false`).

Argument	Type	Description
`level`	`string`	The stage ID of the level to get the spawner code from.

Returns: bool

:lvl_remove_unit(unit)

Prevents the unit from spawning in the level. Useful when you are modifying the default spawner code (use lvl_get_spawner_code to retrieve it). Returns if the unit was removed successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`string|int`	The unit ID of the unit to remove.

Returns: bool

:lvl_bomb(rate:float)

Sets the bombs spawn rate for the level. The rate is relative to the other units' spawn rate and defines how often the unit will spawn. Returns if it was set successfully (`true`) or not (`false`).

Argument	Type	Description
`rate`	`float`	The spawn rate to set.

Returns: bool

:lvl_unit(unit:string, rate:float)

Sets the specific unit's spawn rate for the level. The rate is relative to the other units' spawn rate and defines how often the unit will spawn. Returns if it was set successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`string`	The unit ID of the unit to set the spawn rate for.
`rate`	`float`	The spawn rate to set.

Returns: bool

:lvl_unit2(unit:string, rateA:float, rateB:float)

Sets the specific unit's spawn rate for the level. This function allows you to separately define the rate for the first third of the level (`rateA`) and for the last part of it (`rateB`). The rate is relative to the other units' spawn rate and defines how often the unit will spawn. Returns if it was set successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`string`	The unit ID of the unit to set the spawn rate for.
`rateA`	`float`	The spawn rate to set for the first third of the level.
`rateB`	`float`	The spawn rate to set for the last part of the level.

Returns: bool

:lvl_unit3(unit:string, rateA:float, rateB:float, rateC:float)

Sets the specific unit's spawn rate for the level. This function allows you to separately define the rate for the first third of the level (`rateA`), second third (`rateB`) and the last (`rateC`). The rate is relative to the other units' spawn rate and defines how often the unit will spawn. Returns if it was set successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`string`	The unit ID of the unit to set the spawn rate for.
`rateA`	`float`	The spawn rate to set for the first third of the level.
`rateB`	`float`	The spawn rate to set for the second third of the level.
`rateC`	`float`	The spawn rate to set for the last part of the level.

Returns: bool

:lvl_big_unit(unit:string, limit:int, rate:float)

Sets the specific big unit's spawn rate for the level (big unit is the one with the `"big"` behavior). The rate is relative to the other units' spawn rate and defines how often the unit will spawn. Returns if it was set successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`string`	The unit ID of the unit to set the spawn rate for.
`limit`	`int`	The number of big units there can be on the level simultaneously.
`rate`	`float`	The spawn rate to set.

Returns: bool

:lvl_big_unit2(unit:string, limit:int, rateA:float, rateB:float)

Sets the specific big unit's spawn rate for the level (big unit is the one with the `"big"` behavior). This function allows you to separately define the rate for the first third of the level (`rateA`) and for the last part of it (`rateB`). The rate is relative to the other units' spawn rate and defines how often the unit will spawn. Returns if it was set successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`string`	The unit ID of the unit to set the spawn rate for.
`limit`	`int`	The number of big units there can be on the level simultaneously.
`rateA`	`float`	The spawn rate to set for the first third of the level.
`rateB`	`float`	The spawn rate to set for the last part of the level.

Returns: bool

:lvl_big_unit3(unit:string, limit:int, rateA:float, rateB:float, rateC:float)

Sets the specific big unit's spawn rate for the level (big unit is the one with the `"big"` behavior). This function allows you to separately define the rate for the first third of the level (`rateA`), second third (`rateB`) and the last (`rateC`). The rate is relative to the other units' spawn rate and defines how often the unit will spawn. Returns if it was set successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`string`	The unit ID of the unit to set the spawn rate for.
`limit`	`int`	The number of big units there can be on the level simultaneously.
`rateA`	`float`	The spawn rate to set for the first third of the level.
`rateB`	`float`	The spawn rate to set for the second third of the level.
`rateC`	`float`	The spawn rate to set for the last part of the level.

Returns: bool

:lvl_grapps_unit(unit:string, limit:int, rate:float)

Sets the specific grapps unit's spawn rate for the level. The rate is relative to the other units' spawn rate and defines how often the unit will spawn. Returns if it was set successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`string`	The unit ID of the unit to set the spawn rate for.
`limit`	`int`	The number of grapps units there can be on the level simultaneously.
`rate`	`float`	The spawn rate to set.

Returns: bool

:lvl_grapps_unit2(unit:string, limit:int, rateA:float, rateB:float)

Sets the specific grapps unit's spawn rate for the level. This function allows you to separately define the rate for the first third of the level (`rateA`) and for the last part of it (`rateB`). The rate is relative to the other units' spawn rate and defines how often the unit will spawn. Returns if it was set successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`string`	The unit ID of the unit to set the spawn rate for.
`limit`	`int`	The number of grapps units there can be on the level simultaneously.
`rateA`	`float`	The spawn rate to set for the first third of the level.
`rateB`	`float`	The spawn rate to set for the last part of the level.

Returns: bool

:lvl_grapps_unit3(unit:string, limit:int, rateA:float, rateB:float, rateC:float)

Sets the specific grapps unit's spawn rate for the level. This function allows you to separately define the rate for the first third of the level (`rateA`), second third (`rateB`) and the last (`rateC`). The rate is relative to the other units' spawn rate and defines how often the unit will spawn. Returns if it was set successfully (`true`) or not (`false`).

Argument	Type	Description
`unit`	`string`	The unit ID of the unit to set the spawn rate for.
`limit`	`int`	The number of grapps units there can be on the level simultaneously.
`rateA`	`float`	The spawn rate to set for the first third of the level.
`rateB`	`float`	The spawn rate to set for the second third of the level.
`rateC`	`float`	The spawn rate to set for the last part of the level.

Returns: bool

:hat_find(hat:string)

Returns hat ID for the specified modifier name or hat name. This ID can be further used in ban_hat and unlock_hat functions.

Argument	Type	Description
`hat`	`string`	The modifier ID or the hat name.

Returns: int

:ban_hat(hat)

Prevents the hat from appearing in Mr. Hat's shop. The effect persists between runs and is only canceled by calling unlock_hat. Returns if the ban was successful (`true`) or not (`false`).

Argument	Type	Description
`hat`	`int|string`	The hat ID (use hat_find to retrieve it), modifier ID or hat name to ban.

Returns: bool

:unlock_hat(hat)

Unbans the hat previously banned by ban_hat. Returns if the unban was successful (`true`) or not (`false`).

Argument	Type	Description
`hat`	`int|string`	The hat ID (use hat_find to retrieve it), modifier ID or hat name to ban.

Returns: bool

:get_hat_data(hat)

Returns data for the specified hat.

Argument	Type	Description
`hat`	`string|int`	The hat ID (use hat_find to retrieve it), modifier ID or hat name to get data for.

The data is packed into a struct:

Struct Field	Type	Data
`chaos`	`int`	how much the hat changes the chaos meter if equipped
`hat_display_name`	`string`	name

`text_description`	`string`	description
`hat_sprite`	`sprite asset`	sprite
`hat_height`	`int`	height of the hat in pixels
`hat_price`	`int`	price
`hat_vs_banned`	`bool`	if the hat is banned in Versus Mode

Returns: struct or undefined

:has_modifier(modifier:string, …values)

Returns if the game has the specified modifier applied.

Argument	Type	Description
`modifier`	`string`	The modifier ID to check for.
`?player`	`int`	[optional, default is `1`] The player to check the modifier for. If you want to target the second player in Versus, use `2`.

Returns: bool

:has_hat(modifier:string)

Same as has_modifier, but can be also used in hub.

Argument	Type	Description
`modifier`	`string`	The modifier ID to check for.

Returns: bool

:add_modifier(modifier:string, …values)

Applies the specified modifier.

Argument	Type	Description
`modifier`	`string`	The modifier ID to add.
`?player`	`int`	[optional, default is `1`] The player to add the modifier for. If you want to target the second player in Versus, use `2`.
`?is_offering`	`bool`	[optional, default is `false`] Ifr modifier should be added as an offering.

Returns: n/a

:remove_modifier(modifier:string, …values)

Removes the specified modifier. Returns if it was removed successfully (`true`) or not (`false`).

Argument	Type	Description
`modifier`	`string`	The modifier ID to remove.
`?player`	`int`	[optional, default is `1`] The player to remove the modifier for. If you want to target the second player in Versus, use `2`.

Returns: bool

boss_create_intro(unit:ref, dialogue, ?random)

Creates boss intro cutscene when the prefab loads (that is inside of prefab.gml when making a modded boss).

Argument	Type	Description
`unit`	`ref`	The instance ID of the boss oUnit object.

`dialogue`	`string|array`	The text to display during the boss intro cutscene. If this argument is an array, a random entry will be picked from it.
`?random`	`bool`	Default is true. If the dialogue argument is an array, it will pick one dialogue at random. If set to false, it will play all dialogues in the array in order.

Returns: n/a

Example:

```
var _boss = unit_create("my custom boss", 0, 0);
// create boss intro:
boss_create_intro(_boss, ["Wow, this DLC sure is big!", "Wow, this DLC
sure is free!", "Wow, this mod sure works!"]);
```

boss_get_for_level()

Returns the unit ID of the boss unit for the current level, or `undefined` if the level doesn't have a boss.

Returns: int or undefined

:grid_pixel_x(gridx:int)

Returns the position in pixels of the given x-position in the grid. This is useful to know where we should draw to the screen. Returns `noone` if the grid couldn't be found.

```
var _x = grid_pixel_x( gridx);
var _y = grid_pixel_y( gridy);

draw_rectangle(_x - 10, _y - 10, _x + 10, _y + 10, false);
```

Argument	Type	Description

`gridx`	`int`	The x-position in the grid to transform to pixels.

Returns: int

:grid_pixel_y(gridy:int)

Returns the position in pixels of the given y-position in the grid. This is useful to know where we should draw to the screen. Returns `noone` if the grid couldn't be found.

```
var _x = grid_pixel_x( gridx);
var _y = grid_pixel_y( gridy);

draw_rectangle(_x - 10, _y - 10, _x + 10, _y + 10, false);
```

Argument	Type	Description
`gridy`	`int`	The y-position in the grid to transform to pixels.

Returns: int

:grid_get(x:int, y:int)

Gets the instance ID of oUnit object at the specified grid coordinates. Returns `0` if the cell is empty. Returns `noone` if the value couldn't be retrieved.

Argument	Type	Description
`x`	`int`	The x-position in the grid.
`y`	`int`	The y-position in the grid.

Returns: instance ID or noone

Example:

```
var _unit = grid_get(5, 5); // get the unit at x: 5, y:5
if (unit_is_valid(_unit)) { // if it exists
      unit_freeze(_unit, 1); // freeze it!
}
```

:grid_set(x:int, y:int, value)

Updates the specified grid coordinates with a new unit reference. If the value is set to `0`, the unit will be deleted from the grid instead, and won't interact with other objects. Returns if the grid was updated successfully (`true`) or not (`false`).

Argument	Type	Description
`x`	`int`	The x-position in the grid.
`y`	`int`	The y-position in the grid.
`value`	`ref`	The instance ID of the oUnit to assign to the grid position, or `0`, if you want to clear the cell instead.

Returns: bool

:cell_is_valid(x:int, y:int)

Checks if a position in the grid is inbounds or not. The game runs a fixed grid of 8x9.

Argument	Type	Description
`x`	`int`	The x-position in the grid.
`y`	`int`	The y-position in the grid.

Returns: bool

:trigger_grid_death(grid:ref)

Triggers game end by the board filling up with garbage blocks for the specified grid.

Argument	Type	Description
`grid`	`ref`	The instance ID of the grid to end the game.

Returns: n/a

:explosion_create(unit_exploding:ref, range:int, damage:int)

Creates an explosion around the specified unit. You can define its range and damage, but if you need more control over it you can use explosion_create_ext. Returns if the explosion was created successfully (`true`) or not (`false`).

This function does not make any sounds or screenshake, you will have to add that yourself.

Note: This function makes an explosion unaffected by Too Big Bomb. Use explosion_create_ext instead.

Argument	Type	Description
`unit_exploding`	`ref`	The instance ID of the oUnit triggering the explosion. This unit is immune to its own explosion.
`range`	`int`	The range of the explosion. This is its “radius”, meaning a range of 1 will result in a 3x3 explosion, 2 will be 5x5, etc…
`damage`	`int`	The damage the explosion deals to units.

Returns: bool

:explosion_create_ext(unit_exploding:ref, damage:int, left:int, top:int, right:int, bottom:int, …values)

Creates an explosion. You can define the dimensions of the explosion, its damage and other features. Returns if the explosion was created successfully (`true`) or not (`false`).

This function does not make any sounds or screenshake, you will have to add that yourself.

Argument	Type	Description
`unit_exploding`	`ref`	The instance ID of the oUnit triggering the explosion.
`damage`	`int`	The damage the explosion deals to units.

`left`	`int`	The left position of the explosion in the grid.
`top`	`int`	The top position of the explosion in the grid.
`right`	`int`	The right position of the explosion in the grid.
`bottom`	`int`	The bottom position of the explosion in the grid.
`?self_damage`	`bool`	[optional, default is `false`] Whether the unit exploding receives damage as well or not.
`?too_big_bomb`	`bool`	[optional, default is `false`] Whether the unit is affected by the Too Big Bomb relic.
`?poison_touch`	`bool`	[optional, default is `false`] Whether the explosion causes poison.
`?electric_touch`	`bool`	[optional, default is `false`] Whether the explosion is electric (increases damage by 1).
`?bombproof_ignore`	`bool`	[optional, default is `false`] Whether the explosion should ignore bombproof units or not.

Returns: bool

:explosion_freeze(unit_exploding:ref, range:int, freeze:int, affects_player:bool)

Creates an explosion that freezes units. You can define its range, amount of freeze and whether the player can be frozen by it or not. Returns if the explosion was created successfully (`true`) or not (`false`).

This function does not make any sounds or screenshake, you will have to add that yourself.

Argument	Type	Description
`unit_exploding`	`ref`	The instance ID of the oUnit triggering the explosion.

`range`	`int`	The range of the explosion. This is its “radius”, meaning a range of 1 will result in a 3x3 explosion, 2 will be 5x5, etc…
`freeze`	`int`	The amount of turns units will be frozen for.
`affects_player`	`bool`	Whether this explosion can freeze the player or not.

Returns: bool

:explosion_fire(unit_exploding:ref, range:int, damage:int)

Creates an explosion that additionally sets everything within range on fire. Returns if the explosion was created successfully (`true`) or not (`false`).

This function does not make any sounds or screenshake, you will have to add that yourself.

Argument	Type	Description
`unit_exploding`	`ref`	The instance ID of the oUnit triggering the explosion.
`range`	`int`	The range of the explosion. This is its “radius”, meaning a range of 1 will result in a 3x3 explosion, 2 will be 5x5, etc…
`damage`	`int`	The damage the explosion deals to units.

Returns: bool

:lob_attack(attacker:ref, gridx:int, gridy:int, damage:int, sprite, …values)

Creates a lobbed attack similar to Tinker Knight’s wrenches or Mona B’s potion bombs. Returns an instance of oLobber that you can customize further (for example, customize what happens after the attack lands).

Argument	Type	Description

`attacker`	`ref`	The instance ID of the oUnit attacking. Its position will be used to determine where this is thrown from.
`gridx`	`int`	The x-position for the attack to land. Should be between 0 and GRID_WIDTH.
`gridy`	`int`	The y-position for the attack to land. Should be between 0 and GRID_HEIGHT.
`damage`	`int`	The damage to deal when the attack lands.
`sprite`	`sprite|string`	The sprite asset (use sprite_get to retrieve it) or the name of the sprite to set as an attack sprite.
`?height`	`float`	[optional, default is `1`] Multiplier for high the projectile should fly.
`?speed`	`float`	[optional, default is `1`] Multiplier for how fast the projectile should travel.

Returns: Instance ID or noone

:create_warning_tile(x:int, y:int, unit:ref)

Creates a warning tile on the floor (i.e. the flashing warning tiles a bomb makes before exploding) at the specified grid coordinates. Returns if it was created successfully (`true`) or not (`false`).

Argument	Type	Description
`x`	`int`	The x-position for the warning tile. Should be between 0 and GRID_WIDTH.
`y`	`int`	The y-position for the warning tile. Should be between 0 and GRID_HEIGHT.
`unit`	`ref`	The instance ID of the oUnit that creates/holds the warning tiles.

Returns: bool

remove_warning_tiles(unit:ref)

Argument	Type	Description
`unit`	`ref`	The instance ID of the oUnit to remove the tiles for.

Removes all warning tiles created by the specified unit (the unit argument provided in create_warning_tile). Returns `true` if at least one tile was removed, otherwise returns `false`.

Returns: bool

screenshake(value:int)

Shakes the screen!

Argument	Type	Description
`value`	`int`	How strong the screenshake is. Normally this value ranges between 2 to 10 and it roughly is the amount of pixels of the initial shake.

Returns: n/a

:chain_meter_fix()

When gaining Gem Meter, it will take some time for the Gem Meter to finish increasing in value. If you were to move or gain more Gem Meter before it stops going up, you would be losing in some meter that you would have won if you had just waited.

This function fixes this by instantly giving you any Gem Meter that was still in the process of being awarded. Generally, you don't need to use this ever since the game automatically handles the Gem Meter for you.

Returns: n/a

:reset_adventure_settings()

Resets adventure settings to match those of Daily or Weekly mode. That is:

- 1 stock
- 1 Atk
- 0 extra HP
- Items ON
- Relics ON
- Bosses ON
- Random levels OFF
- Normal speed
- No added full board grace

For this to work properly, it is best to call it at the beginning of a stage.

Returns: n/a

:shop_item_set(relic:string, shop_item_number:int)

Replaces a relic on every active oGrid. Use this when you want a specific relic to appear inside Chester's shop. Returns `true` if at least one relic was replaced, otherwise returns `false`.

Argument	Type	Description
`relic`	`string`	The relic ID of the relic to be placed in the shop.
`shop_item_number`	`int`	The shop slot to place the relic at, a number between 0 and 2. All chester shops have 3 slots for relics.

Returns: bool

make_checkpoint()

Creates a checkpoint at the current stage. This works the same way as using Percy's cannon to shortcut into a specific stage, making quick restarting cost 1000 gems to continue at the same stage the checkpoint is placed at.

Note: only works in singleplayer.

Returns: n/a

:relic_remove(relic:string)

Removes all relics of the specific type from the player's inventory. Returns `true` if at least one relic was removed, otherwise returns `false`.

Argument	Type	Description
`relic`	`string`	The relic ID of the relic to be removed from the inventory.

Returns: bool

:relic_remove_all()

Removes all relics from the player's inventory. Returns `true` if at least one relic was removed, otherwise returns `false`.

Returns: bool

set_last_hit(unit:ref)

Sets the last hit unit, which is the unit shown on the right of the UI. Returns `true` if the UI was updated, otherwise returns `false`.

Argument	Type	Description
`unit`	`ref`	The instance ID of oUnit or oTrap to set on the UI.

Returns: bool

set_killer_sprite(sprite)

When you get a game over, the last sprite set as the killer sprite will be shown as what took you down. Normally the game sets this for you already, however if you want to modify it, use this function.

Argument	Type	Description
`sprite`	`string|sprite asset`	The sprite asset (use sprite_get to retrieve it) or the name of the sprite to set as killer sprite.

Returns: n/a

call_later(period:int, callback, ...values)

Triggers a callback function after the specified amount of seconds. Useful for when you want a certain action to be tied to the time in seconds, not in frames.

Argument	Type	Description
`period`	`int`	The time in seconds to run the callback after.
`callback`	`function`	The callback function to run. Note that it can't take any arguments!
`?loop`	`bool`	[optional, default is `false`] Whether to loop the function call.

Returns: time source ID

Example:

```
var _my_func = function() {
      print("5 seconds passed!");
}

var _callback = call_later(5, _my_func, true); // the function will be
called every 5 seconds
```

call_cancel(handle)

Cancels a callback function that was started earlier by call_later.

Argument	Type	Description

`handle`	`time source ID`	The handle to a Time Source returned by call_later.

Returns: n/a

call_pause(handle)

Pauses a callback function that was started earlier by call_later.

Argument	Type	Description
`handle`	`time source ID`	The handle to a Time Source returned by call_later.

Returns: n/a

call_resume(handle)

Resumes a callback function that was paused earlier by call_pause.

Argument	Type	Description
`handle`	`time source ID`	The handle to a Time Source returned by call_later.

Returns: n/a

input_check(input:string, …values)

Checks if a certain input is being pressed (held) or not. Inputs are:

- `"up"`
- `"down"`
- `"left"`
- `"right"`
- `"item"`

- `"special"` (ability button)
- `"speed"` (turbo button)
- `"action"` (confirm)
- `"escape"` (cancel)
- `"examine"`
- `"pause"`

Argument	Type	Description
`input`	`string`	The input to check for.
`?player`	`int`	[optional, default is `0`] Player 1 is `0` and Player 2 is `1`.

Returns: bool

input_check_pressed(input:string, …values)

Checks if a certain input was pressed or not.

Argument	Type	Description
`input`	`string`	The input to check for.
`?player`	`int`	[optional, default is `0`] Player 1 is `0` and Player 2 is `1`.

Returns: bool

input_check_released(input:string, …values)

Checks if a certain input was released or not.

Argument	Type	Description
`input`	`string`	The input to check for.
`?player`	`int`	[optional, default is `0`] Player 1 is `0` and Player 2 is `1`.

Returns: bool