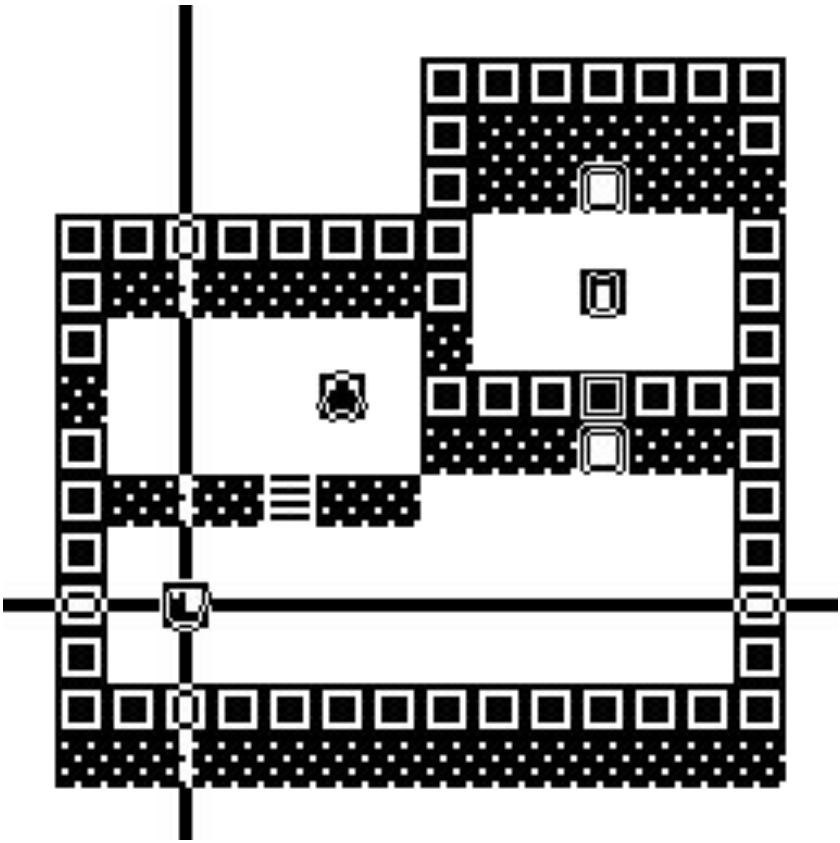

bipsi game maker scripting guide

2021/08/24

intro

bipsi is a tool for making small lo-fi games to play in the browser

this guide continues on from the **bipsi user guide** to help you understand how to achieve for sophisticated effects in with custom javascript code that goes beyond the capabilities of bipsi's user interface

bipsi is available at **kool.tools/bipsi**

if you encounter any bugs or need help using bipsi, you can contact me on twitter at @ragzouken, by email at ragzouken@gmail.com

you can also join the bipsi discord at with the following invite: https://discord.gg/mnARVsgSkc

events

in bipsi, all of the interactive elements are specified as **events**

events sit in a **room** waiting to be **touched** by the player

when **touched**, they run through a set of behaviours such as delivering **dialogue**, moving the player to another **room**, etc

each **event** has a table of data **fields** that are used to specify which behaviours it should perform

the event **templates** are preset collections of **fields** you can use for convenience, but all **events** work the same and you can use any **field** on any **event**

you can, however, override this with your own behaviour

custom scripts

you can redefine an **event's** touch behavior by giving it a **field** called **"touch"** with type **"javascript"**

this is what the "**custom code**" event template contains

when such an event is **touched**, the javascript code will be run instead of the standard event behaviour for touch

the code is regular browser javascript, with some convenient functions and values provided to read and manipulate the current state of the game

this guide won't teach you javascript, but you may be able to copy and modify the examples enough for your purposes

await

you may not have encountered **await** before in javascript. for bipsi it's enough to understand that writing await before a function call pauses the script until the function says it has finished

the **SAY** function is finished when the corresponding dialogue pages are closed

this example changes the player to a mouse after they close the dialogue:

javascript
```
await SAY("you will become a mouse!");
SET_GRAPHIC(AVATAR, FIELD("mouse-tile"));
```

this example changes the player to a mouse as they read the dialogue:

javascript
```
SAY("you are now a mouse!");
SET_GRAPHIC(AVATAR, FIELD("mouse-tile"));
```

event fields

the purpose of event fields is to provide a somewhat convenient way to input constant values for using in event scripts

rather than having to type room numbers, coordinates, dialogue text, etc into javascript code, you can use the existing bipsi picking interfaces to add them as named fields

the "standard" event fields as described in the user guide are special only because they are used by the default event script to decide what to do

when you write a custom **touch** script you can choose your own meaning for the event the fields

but you can also use **DO_STANDARD()** to make the standard fields work as normal

FIELD(S)

get the value of a field name on an event--in this case the "title" dialogue on the player avatar:

code
```
const title = FIELD(AVATAR, "title");
```

get an **array** of the values of a field name on an event:

code
```
const dialogues = FIELDS(EVENT, "say");
```

code
```
SAY_FIELD("example");
```

SAY

the simplest thing in bipsi scripting is to queue dialogue:

code
```
SAY("hello there :)");
```

for longer dialogue, it's more convenient to store it as a field on the event, if we had a dialogue field called "example":

code
```
let dialogue = FIELD(EVENT, "example");
SAY(dialogue);
```

there's a shortcut specifically for SAYing from the current event's fields:

code
```
SAY_FIELD("example");
```

WALK

the **WALK** function lets you move events step by step with a series of directions

the directions understood are:
up, down, left, right, wait: ^ v < > .

this example moves the player avatar to the right twice, hesitates, then moves two more steps to the right:

code

```
await MOVE(AVATAR, ">>.>>");
```

this example moves the touched event around in a circle:

code

```
await MOVE(EVENT, "^^>>vv<<");
```

example: fetch quest

this npc desires collectibles scattered across the world and will thanks you for getting them

each collectible adds one to a common saved value:

javascript
```
await DO_STANDARD();
let prev = GET("apples", 0);
SET("apples", prev + 1);
```

and the npc decides what to say based on the saved value (how many collected):

javascript
```
let count = GET("apples", 0);
if (count > 3) {
    SAY_FIELD("thank-you");
} else {
    SAY_FIELD("desire-apples");
}
```

example: locked door

this door blocks your path--unless you have unlocked it

a key, or a switch, or a friend of the gatekeeper? after being touched, this event saves a shared value marking the door as unlocked

javascript

```
await DO_STANDARD();
SET("door-1-unlocked", true);
```

a door, or gatekeeper, or other blockage? after being touched this event checks for the door unlocked value and if it's there removes itself

javascript

```
await DO_STANDARD();
if (GET("door-1-unlocked")) {
    REMOVE(EVENT);
}
```

example: status report

imagine we've been using other events to track some stat values called "HP" and "MP" for the player

we want to give the player some way to see what their stats actually are

we can write some template text:

field
```
status-text / dialogue / "YOU HAVE [0]/5
  HP AND [1]/5 MP"
```

and use **TEXT_REPLACE** to fill it out with the right values:

javascript
```
let status = FIELD(EVENT, "status-text");
status = TEXT_REPLACE(text, GET("HP"),
  GET("MP"));
SAY(status);
```

example: exodus

guests leaving at the end of the party?
walls tumbling after an earthquake?
something happens that removes several events from the game

add a "leaves" tag to every **event** we want to deal with:

field
leaves / tag

use **FIND_EVENTS** to find all events with the "leaves" tag and javascript's **forEach** to **REMOVE** each one from the game:

javascript
```
await DO_STANDARD();
let leavers = FIND_EVENTS("leaves");
leavers.forEach((leaver) => {
  REMOVE(leaver);
});
```

example: cutscene

a scene where you enter a tavern unannounced. the owner is surprised to hear you enter and walks over to greet you, then walks you over to where they were

this is split into a tavern owner event, tagged with "tavern-owner", who will be moved

and an invisible event at the door that triggers the cutscene when you step on it:

javascript

```
let owner = FIND_EVENT("tavern-owner");
await SAY_FIELD("surprised");
await WALK(owner, "<<<..vv");
await SAY_FIELD("greet-player");
WALK(owner, "^^>>>");
await WALK(AVATAR, "^^^>>");
```

example: menu room

a room you can enter from multiple places but leaving takes you back to where you came from:

entrances to the menu room are events that store the current avatar location and then move them into the room:

javascript
```
SET("return", LOCATION_OF(AVATAR));
MOVE(AVATAR, FIELD(EVENT, "exit"));
```

exits from the menu room are events that move the avatar back to the stored location:

javascript
```
MOVE(AVATAR, GET("return"));
```

example: music

this is the standard script used to play music when a "music" field is present

either the field is a file, or the name of a file field on the **library** event:

javascript

```
let music_file = FIELD(EVENT, "music",
"file");
let music_name = FIELD(EVENT, "music",
"text");

if (!music_file && music_name) {
    let library =
FIND_EVENT("is-library");
    music_file = FIELD(library,
music_name, "file");
}

if (music_file) {
    PLAY_MUSIC(music_file);
}
```

values

PLAYBACK - the playback object, the engine running the bipsi game

AVATAR - the player avatar event

EVENT - the event being touched

PALETTE - the palette of the current room

DIALOGUE - **await** this to pause until there's no dialogue left on screen

functions

SAY(text) - show a normal dialogue with the given text

TITLE(text) - show a title dialogue with the given text

TEXT_REPLACE(text, ...values) - get an updated version of text where occurrences of **[0]**, **[1]**, **[n]** are replaced with the nth value provided

DELAY(seconds) - **await** this to pause for a length of time

MOVE(event, location) - take an event and place it in another location

WALK(event, route) - move an event step by step along a route e.g ".<<.>>^^>>vv" (walking doesn't touch events)

functions

SET(name, value) - set a named value to be used later

GET(name, fallback) - get a named value, or the fallback if named value has never

FIELD(event, name, type) - get the value of the first field on an event with a particular name and type

FIELDS(event, name, type) - get the values of all fields on an event with a particular name and type

SET_FIELDS(event, name, type, ...values) - replace any existing fields of a particular name and type on an event with new values

DO_STANDARD() - do all the standard touch behaviour for the event (title, say, exit, etc)

functions

EVENT_AT(location) - get the event present at a location, if any

LOCATION_OF(event) - get the location of an event

FIND_EVENT(tag) - get an event in the game that has a certain tag

FIND_EVENTS(tag) - get all events in the game that have a certain tag

IS_TAGGED(event, tag) - true if the event has the named tag

TAG(event, tag) - add the named tag to the event

UNTAG(event, tag) - remove the named tag from the event

functions

SET_GRAPHIC(event, tile) - shortcut to set the event's graphic field to another tile (to change its appearance)

REMOVE(event) - remove an event from the game

TOUCH(event) - run the touch behaviour of an event as if the player avatar had touched it

RUN_JS(javascript, event) - run javascript as if it were a **touch** running for a particular event

RESTART() - restart the game

SET_CSS(name, value) - set CSS value on the root e.g "--page-color"

functions

SHOW_IMAGE(id, image, layer, x, y) - replace any image at this id with a new image shown at x,y and on or between the following layers:
0: under everything
1: over tiles but under events
2: over events but under dialogue
3: over everything

HIDE_IMAGE(id) - hide the image being shown with this id

PLAY_MUSIC(file) - switch looping music to a music file

STOP_MUSIC() - stop looping music

functions

SAMPLE(id, mode, ...values) - complex function for stepping through a sequence over multiple **touches**

picks a value from a list of values either shuffled, in a sequence, or cycling

progress is remembered between event **touches** using the given id (this also allows multiple events to use the same sequence)

id - named id to remember progress
mode - "shuffle", "cycle", or "sequence"
values - values to step through

javascript

```
let dialogue = SAMPLE("1", "cycle",
"hello", "heya", "hi");
await SAY(dialogue);
```

notes

about

bipsi is a web tool for making simple browser games in the bitsy style

kool.tools/bipsi

this scripting guide explains how to begin using bipsi at an advanced level, for those willing to mess with writing javascript code

i'm mark wonnacott a.k.a candle, and i created bipsi

kool.tools