A Guide to Bitsy

Prepared by Jason Boyd

(Version used: 7.2; Guide written March 2021)
(Updated March 2023 using version 8.4) [revised up to and including section 8]

Current version of Bitsy (see https://ledoux.itch.io/bitsy/devlog): 8.12 (May 3, 2024); checked June 13, 2023

1. Introduction

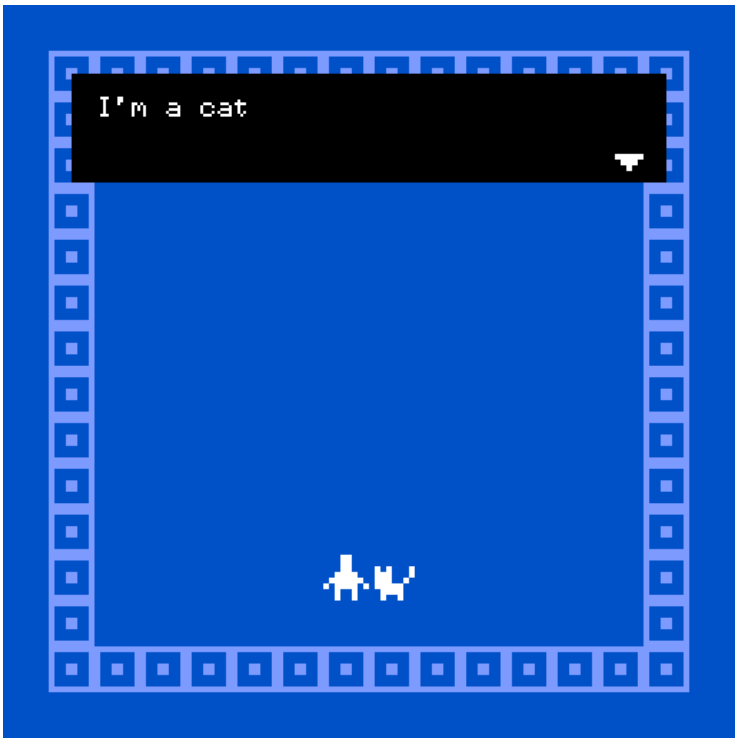

Figure 1. A simple Bitsy game screenshot (this is the game that is built into the Bitsy Editor).

Bitsy is "a little engine for little games, worlds, and stories," created by Adam Ledoux. A Bitsy work uses a combination of user-created graphics (consisting of an avatar, spites, tiles, and items) resembling the 8-bit aesthetic of early computer games, text, and (if desired) sound, and typically consists of one or more square screens depicting spaces that the player's avatar moves within and between (one screen/space at a time) and which contain things (such as non-player characters, structures, and items) that can react when the player avatar comes in contact with them, most typically by producing a pop-up box of text. In the example above (which is taken from Bitsy's built-in sample game, the player has moved the avatar (the human icon) down to

interact with a spite (the cat icon), thereby triggering the pop-up dialog window containing the text, “I’m a cat”.

2. The Bitsy Landing Page and Editor Interface

Bitsy is accessible via https://bitsy.org/. The landing page contains a number of links, including:

- Make a game (links to the Bitsy Editor)
- Play a game (links to a directory of games made with Bitsy on itch.io)
- Read the docs (links to ‘Bitsy Docs’: see section 3)
- Join the forum (links to the Community page of Bitsy on itch.io)
- Friends of Bitsy (links to various resources and tools: see section 12)

The Bitsy Editor consists of a browser-based workspace in which different windows/tools can be opened, closed and moved around. Along the top to the Editor, from left to right, is the Bitsy cat icon (links to the landing page described above), the work’s title editing/display field, and a ‘tools’ button and a ‘play’ button (for the ‘play’ button, see section 4. Rooms). Clicking on the ‘tools’ button displays or hides a menu listing the available windows/tools that can be added to or removed from the workspace, with the windows currently displayed highlighted in a darker colour.

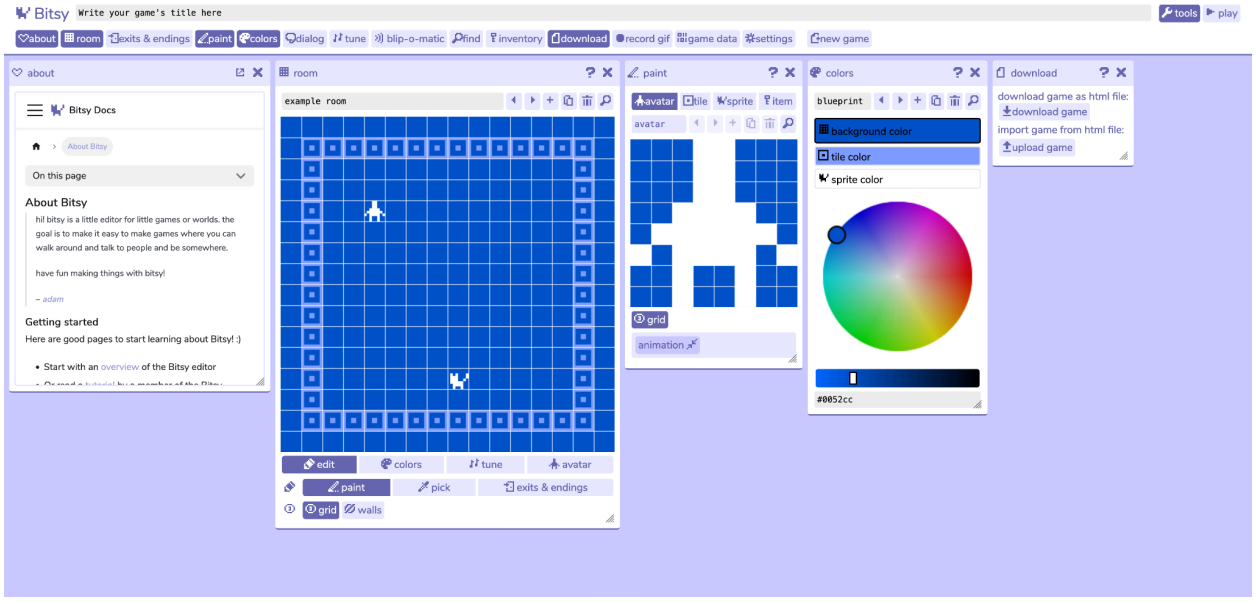

Figure 2. The Bitsy Workspace, with the Tools menu (at the top) toggled on. The tool buttons in the darker colour are the ones currently displayed in the workspace below.

The tools/windows are:

- about (see section 3)
- room (see section 4)
- exits & endings (see section 7)
- paint (see section 5)
- colors (see section 6)
- dialog (see section 9)
- tune (see section 10)
- blip-o-matic (see section 10)
- find (for finding various components you have created, by the name you have given them)
- inventory (see section 8)
- download (covered in this section, below)
- record gif (allows you to make a gif or still image from what is currently being displayed in the room window)
- game data (covered in this section, below)
- settings (not described in this guide)
- new game (covered in this section, below)

Most of the above tools/windows have a '?' icon in the top right, which will open the about window to the relevant section in Bitsy Docs (see section 3, below), offering instructions and guidance about that window. Some of these instructions are fairly extensive, others are minimal or non-existent (therefore this guide).

Note that, as a browser-based engine, Bitsy saves work automatically in the browser cache (*if you delete your browser cache, you will delete your work!*). You should save your work by downloading it using the **download** window (it will download the work as a .html file). Note that when a Bitsy .html file is uploaded in the editor, the 'example room' from the Editor's built-in work may be included; this can be ignored (but not deleted).

The **game data** window displays the raw text/code of your Bitsy work. If you have difficulty uploading a Bitsy .html, open the game data window, then open your previously downloaded .html file in a plain text editor, select all, copy, and paste into the game data window (after deleting all the data currently in the game data window). In the game data window, you can also download/upload your work as a .bitsy game data file using the buttons at the bottom.

If you click on the **new game** button in the tool menu, it will erase any work currently being created in the Editor.

3. About

The About window contains ‘Bitsy Docs,’ which comprise a number of resources (click on the menu icon in the top left to see the table of contents), including a brief ‘Overview’ of the Bitsy editor, and a ‘Tutorial’ section with links to a range of external resources (not that many of these use older versions of Bitsy).

More immediately useful is the ‘Tools’ section, with sections explaining each of the features in the toolbar.. There is also an ‘Advanced Topics’ and a ‘FAQ’ section, but both are currently empty.

The ‘Changelog’ lists changes made to the Bitsy Editor.

4. Room

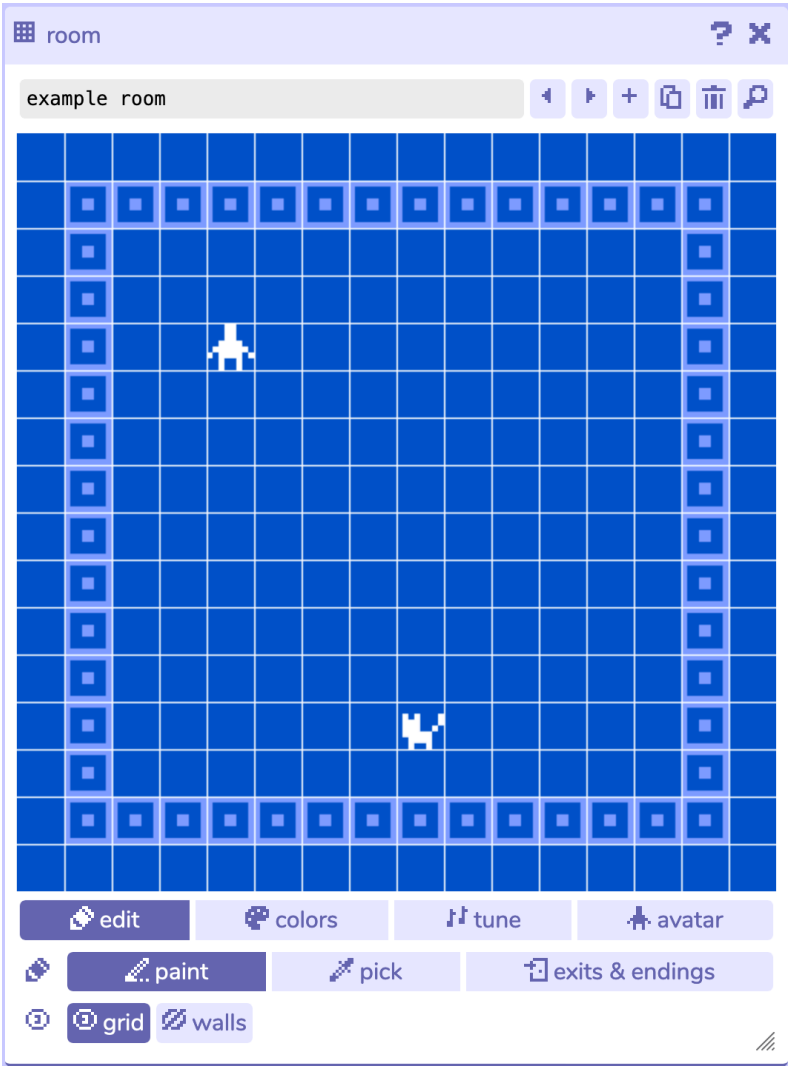

Figure 3. The Bitsy room window/tool.

The *room* window is your canvas (or series of canvases, if your work has multiple rooms/screens). A 'room' is basically a square made up of a 16x16 grid. It is the window where you place *tiles*, *sprites*, *items*, and the *avatar* (which are created using the *paint* window: see section 5).

At the top of the window, there is a text box that displays the automatically generated unique room name (which can be edited), and buttons to move back and forth between existing rooms, add a new room, duplicate an existing room, and delete a room.

Below the room canvas are three rows of buttons. In the first row, the 'edit' button (selected by default) toggles on the edit options in the second row if the user has selected any of the other buttons in the first row.

The edit options (second row when the edit button is selected) are:

- **paint**, which enables you to place/remove in a cell or cells on the grid of the room canvas whatever is currently displaying in the paint window by clicking on the selected cell;
- **pick**, which allows you to click on a cell of the room canvas containing the avatar, a sprite, a tile, or an item and have whatever is in that cell display in the paint window;
- **exits & endings**, which will show where any exits and order are placed in the room. Unlike other objects in the work, which can only be moved by 'picking' them and 'painting' them by clicking on a desired cell in the room canvas, exits & endings can be clicked on and dragged to the desired cell. You can open the exits & endings tool window (see section 7) by clicking on the button that appears to the right when the exits & endings button is clicked.

In the first row, the buttons for colors, tune, and avatar toggles on pull-down menus where you can select amongst different color schemes, tunes and the avatar/sprites you have created in, respectively, the colours (section 6), tune (section 10), and paint (section 5) tool windows.

When the edit button is selected, there is a third row with buttons that enable you to toggle off or on the room grid overlay (useful for placing tiles, etc., and not displayed when the work is played) and tiles that have been designated walls (which, when toggled on, will display walls as solid white or black cells depending on the room's palette). For walls, see section 5.

The room window is also the window in which you can playtest your work (by clicking on the 'play' button in the top right of the Bitsy Editor). [*Note*: if you 'play' your work, remember to 'stop' it before you make any modifications. After stopping a work, you may have to refresh the browser window to reset the work.]

5. Paint

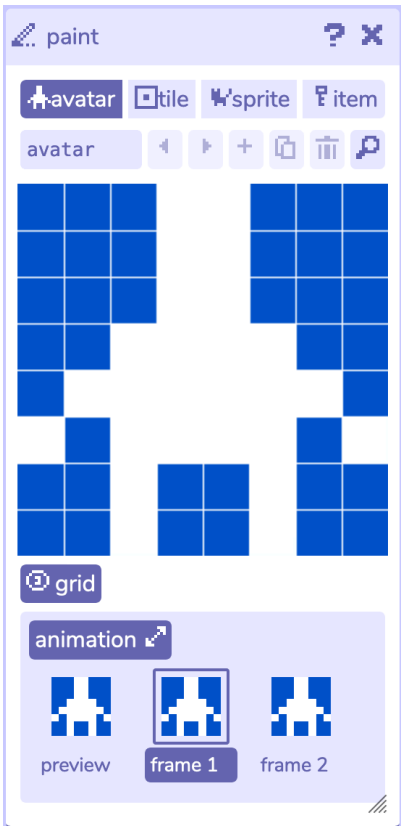

Figure 4. The Paint tool/window, with the avatar option selected and the animation editing window toggled on.

The *paint* window (Figure 4), as can be seen by the buttons along the top, is where you can create/edit the *avatar* (the icon that the player moves with WASD or arrow keys), *tiles* (non-interactive features in the space in which the avatar moves), *sprites* (interactive features in the space in which the avatar moves), and *items* (things which can be 'taken' by the avatar). Each of these buttons can be clicked to access a drawing box where a 'drawing' (i.e., a pixelated outline) can be created/edited and that then can be placed in one square of the room grid (by clicking on the desired square in the room window when the edit→paint button is selected in the room window). Below these buttons, as with the room window, there is a text box that displays the automatically generated unique drawing name (which can be edited), and buttons to move back and forth between drawings, add a new drawing, duplicate an existing drawing, delete a drawing and search for a drawing (opens the find window/tool).

Note about naming tiles/sprites/items: When creating new tiles, sprites, and items, Bitsy will automatically generate a name in the following format: "tile a", "sprite a", "item 1". You should change these automatically generated names to something more descriptive and memorable and avoid (as with these automatically generated names) any spaces in your names: Bitsy code (see section 8) does not recognize such names. A suggested format for names with multiple parts is to use camel case: tileGrass, catMacavity, magicWand, etc.

The paint window drawing box is an 8x8 grid (a grid overlay can be toggled off and on using the grid button below the drawing box) in which you can click on squares in the grid to toggle between two colours: for the avatar, sprites, and items, between the *sprite* and *background* colours; and for tiles, between the *tile* and *background* colours (as chosen in the *colors* window: see section 6).

In the Paint window, the avatar, tiles, sprites and items can be 'animated' by clicking on the **animation button** underneath the drawing box, which will expand the animation creation window. An animation is a continuous circulation between two frames. You can click on the 'frame 1' and 'frame 2' window to edit each frame. The preview window shows how the animation wil look.

4.1 Avatar

There can be only one avatar in a Bitsy work (note that the options after the name box are disabled when ‘avatar’ is selected in the Paint window). The avatar’s initial position in the starting room can be selected/changed by clicking on the desired square in the room window when the edit→paint button is selected in the room window.

4.2 Tiles

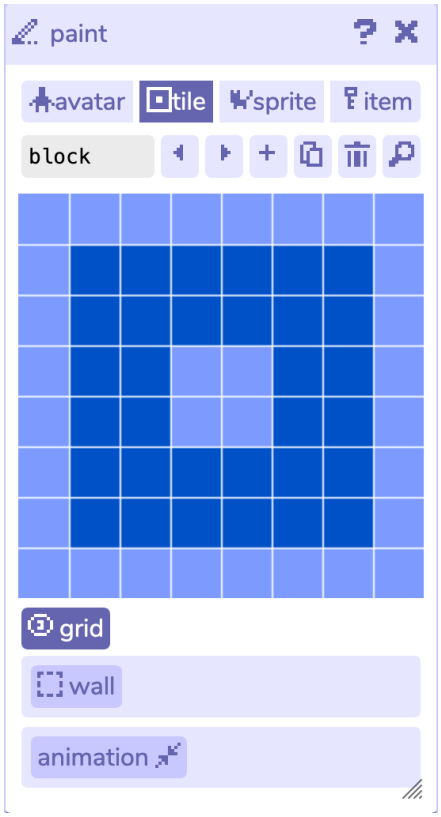

Figure 5. The Paint tool/window, with the tile option selected and the wall option not selected and the animation editing window toggled off.

Multiple tiles of the same name can be placed/removed in a room by clicking on the desired squares in the room window when the relevant tile is selected in the paint window and when the edit→paint button is selected in the room window. Tiles, by default, can be moved over/through by the avatar. To prevent this, click on the *wall* button below the drawing box ***before*** placing the tiles in the room window. If you want to have two tiles that look identical, but have one of them be a wall and one not, then you will have to create two identical looking tiles with different names, and make one a wall.

4.3 Sprites

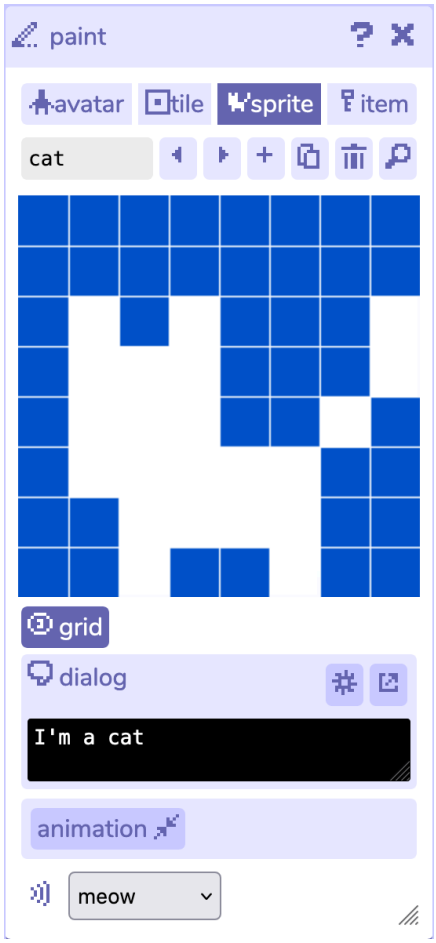

Figure 6. The Paint tool/window, with the sprite option selected and the dialog editing window open.

You can have multiple sprites per room, but each sprite in a given room must have a unique name (even if they look identical). You can duplicate sprites with the 'duplicate drawing' button, which will recreate the sprite, but with a unique name. Sprites can be placed/replaced/erased by clicking on the desired squares in the room window when the relevant sprite is selected in the paint window and when the edit→paint button is selected in the room window.

The avatar cannot move over/through sprites (they are like a tile wall).

For the dialog and sound options that are included in the Sprite view in Paint, see section 4.5.

4.4 Items

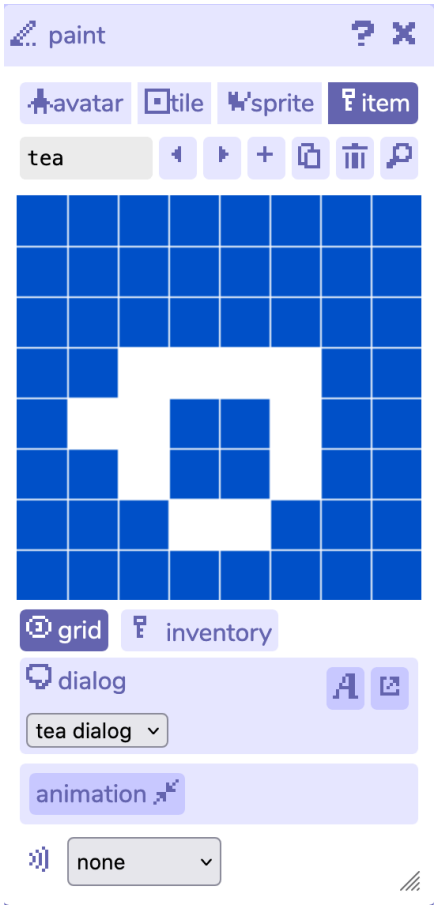

Figure 7. The Paint tool/window, with the item option selected and the dialog editing window closed.

Like tiles, multiple items with the same name can be placed in a room. Items 'disappear' when the avatar moves over them and are added to an Inventory count. Clicking on the inventory button below the drawing box opens the Inventory window (see section 8).

For the dialog and sound options that are included in the Item view in Paint, see section 4.5.

4.5 Dialog and Sound Editing in Paint

The *sprite* and *item* views both contain a dialog section where text can be entered or edited. The sun/letter A button closes and opens the built-in dialog editor, which is an embedded version of the Dialog window/tool (see section 9), which can be opened by clicking on the button with the pop-out/open window icon.

During play, dialog is displayed in a dialog window that appears when the avatar comes in contact with the sprite/item (see Figure 1 for an illustration).

For simple sprite dialog, the paint window dialog editor can be used without opening the Dialog window.

You can also use this dialog editor to provide dialog for an item, but, if you do so, all instances of that item that are placed in rooms will display the same dialog. If you want instances of the same item drawing to display different dialog, use the Dialog window (see section 9).

At the bottom of the sprite and Item views in the Paint window there is the option of associating a sound file with a sprite or item, by selecting a sound from a drop-down menu. To create/edit sound, see section 10.

6. Colors

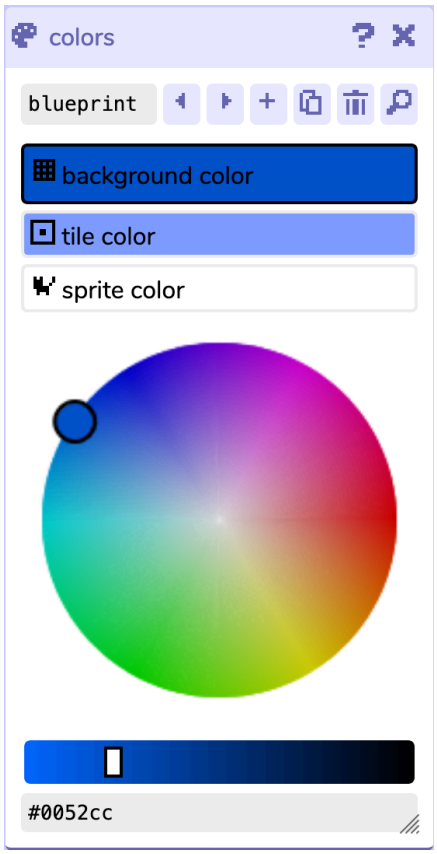

Figure 8. The Bitsy Colors tool/window, with the background color selected.

A Bitsy palette is made up of three colours of your choosing: one for the background, one for tiles, and one for sprites, items and the avatar. The default palette is ‘blueprint’ (seen in Figures 1 and 8).

A room can have only one Bitsy palette, but if your Bitsy work has multiple rooms, each room can have a different palette.

As with the room and paint windows, at the top of the colors window there is a text box that displays the automatically generated unique palette name (which can be edited), and buttons to move back and forth between palettes, add a new palette, duplicate an existing palette, delete a palette, and search for an existing palette.

Starting out, you can modify the default palette ('blueprint') by dragging the small circle over the color picker wheel, dragging the bar in the darker/lighter slider below the wheel, and renaming it, or add a new palette. To associate a palette with a room, choose the desired palette name from the 'colors' pop-down menu at the bottom of the room window. If there is only one palette, it will be selected by default.

If you want to reuse a specific color from one palette in another, you can copy the color hex code that appears in the box below the darker/lighter slider. (Hex codes can be directly entered into this box as well).

7. Exits and Endings

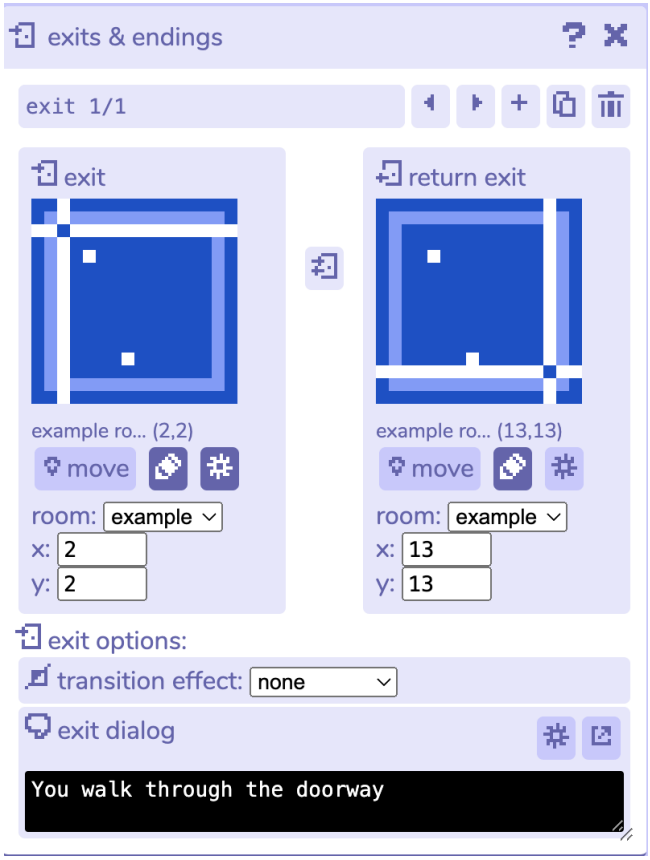

Figure 9. The exits & endings window/tool.

This window/tool is used to create ‘exits’ and ‘endings’ that enable the avatar to ‘move’ or transition from one room to another (or from one spot to another in the same room) and to reach the ‘end’ of the game. At the top of the window there is a text box that displays the automatically generated unique ending/exit name (which can be edited), and buttons to move back and forth between exits/ending, add a new exit/ending, duplicate an existing exit/ending, and delete an exit/ending.

In the middle of the window, there are two panes side by side displaying schematics of two rooms. These show the X and Y coordinates of the exit/destination/ending with white lines and a dark square where they intersect. By default, a schematic of the room currently being displayed in the room window will appear in a pane on the left and the next room in sequence will be displayed in a pane on the right (if only one room has been created, that room will appear in both panes).

A door icon with arrows between the panes can be clicked to change the type of connection between rooms: a one-way exit from the room in the pane on the left to the room on the right (the destination), or the reverse (a one-way exit from the room on the right to the room on the right), or a two-way exit.

Below each pane, there is a button to **move** the location of exits/destinations/endings. First, the ‘exits & endings’ button in the room window should be selected in order to show the placement of the exits/destinations/endings (these will display in black and white, and blink). In the room window, You can click on these and drag them to the desired locations. Alternatively, in the exits & endings window, you can click on the ‘move’ button and then click on the desired square in the room window.

The **pencil button** next to the move button opens and closes an edit pane where you can select a room from a drop-down window as well as specify the numerical X/Y coordinates of the exit/destination/ending in a room.

Clicking on the **sun buttons** next to the pencil buttons in each pane will open separate option editors for each room at the bottom of the window/tool. Here, you can choose from a pull-down menu a **transition effect** for when the avatar moves between exits and destinations; these are: none, fade (white), fade (black), wave, tunnel, slide up, slide down, slide left, slide right.

There is also an embedded dialog editor (similar to the one in the Paint window/tool), where one can **add narration** or **add a lock** (the latter automatically opens the dialog window/tool). See section 9.

Endings

8. Inventory

The Inventory window/tool is primarily useful for tracking the starting number of all created items (automatically added to the inventory) or what the current count of items are in the avatar's inventory (i.e., items the avatar has interacted with) when you are playtesting the work (as the item inventory updates automatically during a playtest). Although there is a variables button that can be toggled to access an '+add variable' button, it makes more sense to create variables in the Dialog window (see section 9).

9. Dialog

Figure 10. The Dialog window/tool, showing the options presented when clicking on the 'add' button.

The Dialog window is where both the text that the player reads when playing a Bitsy work (called the "dialog text" in this guide) and the coding that determines *what* text is read by the player is created. The generation of dialog through the avatar's interaction with sprites, items, and exits and endings is the primary form of interaction in a Bitsy work.

At the top of the window there is a text box that displays the automatically generated unique dialog name (which can be edited), and buttons to move back and forth between dialogs, add a new dialog, duplicate an existing dialog, and delete a dialog.

Beneath the Dialog text display/editor (see section 9.1), there is an 'add button' that open ups a menu with the options: dialog (see section 9.2), lists (see section 9.3), room actions (see section 9.4), sound actions, an item and variable actions.

The Dialog window offers two views of (and two ways to create/edit) the same content: what can be called the *template code editor view* and the *raw code editor view*. The default view is the template view, which by default displays the black dialog editing pane. It is essentially a snippet version of the raw code view, which can be switched to (or opened fully) by toggling the 'show code' button at the bottom of the Dialog window/tool.

As there is no reference guide for Bitsy's code, it is difficult (if not impossible) to use the raw code editor exclusively, and sometimes the template code editor version is more complicated, less functional, and less intuitive/transparent than the raw code editor version. The best method is to periodically switch between the template code editor and the raw code editor to see the raw code editor version of what you have created in the template code editor (and sometimes vice versa). The raw code editor is also useful to finesse the (mis)placement of the coding that sometimes happens when using the template code editor.

In the template code editor, clicking on a black box will open a text effects menu below it (this menu can be hidden/revealed by clicking on the 'AB' icon above the black box). These text effects (for text that the player will read) include changing the colour of the text to one of the three colours in a palette [***Note: this does not seem to be functioning in version 7.2***], or creating wavy, shaky, or rainbow-coloured text. To implement these text effects, select the desired text in the black box, and click on the desired text effect button. You will see that the selected text is enclosed or bracketed, depending on your choice, in {wvy}…{wvy}, {shk}…{shk}, or {rbw}…{rbw}.

You can also add small versions of a drawing (i.e, the avatar, or a created tile, sprite or item) in the text that players will read. Place your cursor in the black box where you want to insert the drawing, choose from the pull-down next to the '+insert drawing' button and press the button. You will see that, depending on your choice, a {printSprite "[lower case letter/upper case letter]"}, {printTile "[lower case letter]"}, or {printItem "[number]"} is inserted.

These text effects can be combined within a single dialog box.

Annoyingly, when using the template code editor, the resulting code in the raw code editor does not use (as the template code editor does) the names given to tiles, sprites and items, but instead uses automatically assigned single-letter names (for sprites and tiles, "a", "b", "c", etc.; for items, "1", "2", "3" etc.). Depending on how much you use the raw code editor, you may want to change these names to the ones you have created using the raw code. If you do so you

will notice that the small icon for the tile/sprite/item that appears after its name in the template code editor disappears.

9.1 Dialog Text Effects Editor

Clicking on the black dialog display/editing pane opens a **text effects editor** (which can be toggled open or closed using the 'AB' button above the pane. This allows you to apply animation effects to the text in the editing pane by highlighting (by clicking and dragging) the desired text and clicking on one of the buttons, which will add Bitsy markup to the text:

- Wavy text: encloses the selected text in {wvy}...{/wvy} markup
- Shaky text: encloses the selected text in {shk}...{/shk} markup
- Rainbow text: encloses the selected text in {rbw}...{/rbw} markup
- You can also colour the text with a background ('ground'), tile, or sprite colour, by selecting from the pull-down menu to the right of the palette button, highlighting the desired text, and clicking on the palette button. This will enclose the selected text in either {clr 0}...{/clr 0} (background), {clr 1}...{/clr 1} (tile), or {clr 2}...{/clr 2} (sprite) markup.
- You can also add small icons of the avatar, sprites, tiles, and items to the dialog text by selecting from the pull-down menu to the right of the pen button, highlighting the desired text, and clicking on the pen button. This will enclose the selected text in either {drws "A"} (avatar), {drws "[name of sprite]"}, {drwt "[name of tile]"}, or {drwi "[name of item]"} markup.

You can also apply multiple text effects to the same selection of dialog (e.g., rainbow and wavy). Make sure that the code is properly nested, e.g., {rbw}{wvy}...{/wvy}{/rbw} (the wavy markup is nested within/enclosed by the rainbow markup).

You can manually add/edit markup in the dialog window instead of using the buttons descibed above.

A required dialog is the **'title' dialog** (the 'title' name cannot be changed and the dialog cannot be deleted). While it cannot be deleted, it can be left blank, which will result in the Bitsy work starting with the initial room (rather than with the usual 'title page').

9.2 Dialog

This contains two options (beside the 'back' option):

+dialog: in the template view, this adds another black dialog box to the current spite/item dialog/ending/exit narration. If there is no intervening code (like pagebreak [discussed below] or a conditional statement) between multiple black boxes, they will be treated as the same text window/code block (i.e., no different than if they were in a single dialog pane).

+pagebreak: adds a page break (markup: {pg}), after which a new dialog (+dialog) can be added. When the work is played, the text in additional dialog panes will be displayed in a 'fresh' window after the player selects the down/next key (which can be useful for pacing longer dialogue).

As +pagebreak shows, the template code editor is intended to offer a more descriptive and visually structured way of working with the Bitsy code. The +pagebreak button adds a box after the black box reading "pagebreak / start a new page of dialog". A downward arrow above shows that it is executed after the initial black box (and presumably before a second dialog pane to be added). In the raw code editor, everything in the template code editor appears together (with the pagebreak box counterpart being "{pg}"). While this makes the template code editor seem excessive, for longer code with multiple sections, the template editor can make it easier to write/parse the code.

9.3 Lists

This contains four options which are applicable to **dialog associated with sprites and items**. When an avatar interacts with a sprite or takes an item, you can vary what dialog is provided using these options:

+sequence list: will proceed once through each dialog in the list in the order provided. The last option on the list will continue to be repeated when the sequence is run through once;

+cycle list: will loop through each dialog in the order provided;

+shuffle list: will randomly provide a dialog option;

The raw code format for the three lists above is:

```
A key! What does it open?
{cycle [or sequence, or shuffle]
 - A door?
 - A chest?
 - A can of sardines?
}
```

Note that by default, the dialog in a list is *added to* the initial dialog text (which repeats).

+branching list: this enables the creation of if/else if/else statement (conditional statements). By clicking on one of these statements, one can edit the conditional statement, by selecting an item from a pull-down menu, specifying the number of items, indicating whether the condition should be true if the number is equal, greater than or equal, less than or equal, greater than, or less than the specified number of items, and print a dialog.

Example (raw code):

```
{
```

```
 - {item "widget"} == 0 ?
   You found a widget!
 - {item "widget"} == 1 ?
   You found another widget!
 - {item "widget"} == 2 ?
   One more widget to go!
 - {item "widget"} == 3 ?
   You found all the widgets!
}
```

Note that to count inventory like this requires starting at 0, not 1.
Note that 'is equal to' is represented with '==' ('=' is the assignment operator).

The following branching code example checks to see if a condition is true (i.e., if there are four keys in inventory) and prints out an associated message, or, if the condition is false, a different message:

```
{
 - {item "key"} == 4 ?
   You unlock the door.
 - else ?
   You need four keys!
}
```

9.4 Room Actions

This contains three options:

+ exit: enables you to specify which room and the x/y coordinates you want to move the avatar to (alternately, you can use the 'move destination' button and select the desired square in the room window). You can also choose one of eight transition effects: fade (white), fade (black), wave, tunnel, slide up, slide down, slide left, slide right.
+ end: like an exit, enables one to place an end point, and display an end message.
+ lock/unlock: enables creating a conditional statement using the property "locked" and true/false (which is evaluated on, for example, whether the avatar had an item or items in its inventory).

Example:

```
{
 - {item "key"} >= 1 ?
   {property locked false}
   The key opens the door!
 - else ?
```

```
   {property locked true}
   The door is locked...
}
```

9.5 Sound Actions

9.6 Item and Variable Actions

This contains five fairly self-explanatory options:

+ set item count: code {item “name” #}
+ increase item count: code {item "name" {{item "name"} + #}}
+ decrease item count: code {item "name" {{item "name"} - #}}
+ set variable value
+ change variable value

Clicking on the last two will create new boxes with boilerplate code (“a = 5” and “a = a + 1” respectively). Clicking on these boxes will open editable versions. Clicking on the sun icon provides additional editing options that allow you to change the categories of what is on either side of the assignment operator (=): number, text, bool, variable, function, expression.

Clicking on the math symbols button opens up a fuller editing window, where you can delete some or all of this boilerplate code (using the delete button above the AC button) and start from scratch.

Example: for a dog sprite, a variable called hasTalkedToDog was created and assigned a value of true:

```
{hasTalkedToDog = true} [alternately we could use a number, “1”]
```

This was placed in the dialog associated with the dog sprite, so that when the avatar interacts with the dog sprite, the variable is executed (i.e., hasTalkedToDog becomes true). Then, in the dialogue for a cat sprite, the following conditional statement was created:

```
{
 - hasTalkedToDog == true ?
   I saw you talking to the dog.
 - else ?
   Don’t talk to the dog.
}
```

If the avatar interacts with the cat before the dog, they will get the ‘else’ message; if the reverse, the first message.

10. Sound (Tune and Blip-o-matic)

For a single soundtrack that will play throughout your entire Bitsy work, use Mark Wonnacott’s Bitsy Audio tool: https://candle.itch.io/bitsy-audio

If you want your rooms to have different soundtracks, use Wonnacott’s Bitsy Muse UI for David Mowatt’s bitsymuse hack:

https://kool.tools/bitsy/tools/bitsymuse-ui/
https://github.com/seleb/bitsy-hacks/blob/main/dist/bitsymuse.js

The Tune window allows you to create music for your Bitsy game. (*Guide instructions pending.*)

The blip-o-matic allows you to create sound effects. For more on the blip-o-matic, see https://ledoux.itch.io/bitsy/devlog/381983/meet-the-bitsy-sound-effects-tool-blip-o-matic- (*Guide instructions pending*)

11. Publishing

Use the Download window to download an .html file. This is the file that can be used to publish your Bitsy work on itch.io.

12. Other Resources

Borksy has a range of various hacks: https://ayolland.itch.io/borksy
Many more hacks: https://github.com/seleb/bitsy-hacks
Tools on itch.io tagged with ‘Bitsy’: https://itch.io/tools/tag-bitsy (see also related collections in sidebar)
Pixsy: https://ruin.itch.io/pixsy (turns an image file into a Bitsy room)
Fontsy: https://seansleblanc.itch.io/fontsy
Multicolor Bitsy; https://aurysystem.itch.io/multicolorbisty
Bitsy 3D: https://bitsy3d.xyz/
Bitsy Savior: https://aloelazoe.itch.io/bitsy-savior

Tutorials:
Dan Cox, Bitsy 6 Video Tutorial Playlist (7 videos, 2019)
Rob Duarte, Advanced Bitsy Tutorial Playlist (12 videos, March 2021)
Haraiva, Bitsy [Bitsy 5.3] Workshop http://bit.ly/nywf-bitsy
Night City Academy, "Bitsy conditional dialog tutorial with variables" (23 February 2021) https://youtu.be/Mrt0tk6HSvI
"Game Making with Bitsy with the National Videogame Museum" (12 May 2020): https://youtu.be/j4bPhG_RGo8
Cephalopodunk, "Time for some rambling about dialog"

Bitsy FAQ
Bitsy Wiki: https://bitsy.fandom.com/wiki/Bitsy_Wiki

13. Index of Bitsy Code

Branching lists (that is, if/else statements) do not have a specific function word: they are simply dashed lists of {item "#"}s with evaluative statements like "== 1 ?", which are treated as if/else if/else statements.

{clr1}
{clr2}
{clr3}
{cycle}
- else ?
{end}
{exit}
{item "#"}
{pg}
{printItem "1"}
{printSprite "A"}: the Avatar
{printSprite "a"}
{printTile "a"}
{property locked false}
{property locked true}
{rbw}
{sequence}
{shk}
{shuffle}
{wvy}