The JUICE Manual v1.1.1

by Jon Irons

October 18, 2007

Contents

Chapter 1

Introduction

Welcome to the JUICE utility. Our intention with JUICE is twofold:

1. Editing maps in ways that Forge can, but with greater ease in mass-editing many of the properties that Forge can edit.
2. Allowing for behavior that Forge can't, including certain tricks and settings that were possible, but not allowed by Forge, and support for entirely new things that are added to the Aleph One engine.

We don't intend to replace Forge, or to become a full-featured graphical level editor. Instead, we intend to provide efficient editing of key aspects of map files. This manual exists to help you take full advantage of JUICE.

While JUICE can already do many useful things, we have plans to continue expanding its capabilities. We will allow for editing more aspects of polygons (setting media and lights in particular), and will expand the utility to understand and manipulate the actual data for media and lights. We will also add support for platform editing. In addition, the programmers of JUICE intend to work with Aleph One and its developers to help enable entirely new features; for example, if colored lighting is ever implemented, JUICE will be the first editor capable of manipulating the new map data. So, while JUICE is a specialty tool, it will be the tool of choice for mappers who want to work with future features.

JUICE is, of course, written in Java, and is completely open-source. We want to help others interested in making editors by writing (mostly) clean source code that is also easily ported into other projects. The entire backend ought to be able to do this at some point in the future, and another editor can pick up in the areas where JUICE leaves off—for example, in creating map geometry.

The target userbase of JUICE is the network mapmaker group. As both Watts and I know from experience, netmappers have to deal with many tweaks in rapid succession: playtest, tell your playtest group to wait, make changes, rehost. Many of today's best netmappers do not use the Macintosh Operating system, and with the newer Intel Macs, the "Classic" MacOS (version 9) is no longer accessible directly from OS X. Instead, both groups have to use emulators, which always introduce some level of inconvenience; even if it were running at full speed and bug-free, an emulator usually has to find some way to transfer files to and from its host system.

JUICE runs on many systems, and does not require emulation. It can make map tweaks in a fraction of the time it takes to start up an emulator and go through the workflow (open Forge; adjust parameters, quit Forge, transfer from emulator; shut down emulator). JUICE is not just faster than this method in a general sense; working in it has some powerful functions that speed up tweaking considerably. Without going into too much detail, I can say that working with tables as JUICE does, in conjunction with keystrokes, produces results at incredible speed. Add sorting, and you can find entire groups of things and edit them.

While netmapping is our primary concern, JUICE works fine for other kinds of levels. Because it can operate on merged maps or even merged "unimaps," it is invaluable for last-minute edits that would be a pain to do with the traditional Forge workflow. For example, if you forget to add the Co-operative play mode to a level, or if you forgot to change its name before merging, you can simply open JUICE and edit either parameter quickly.

In short, JUICE is a program by mappers, for mappers. We hope that you find it as useful as we do.

Chapter 2

JUICE Basics

2.1 Running JUICE

Running JUICE can be very easy to do. But first, you need to make sure you have Java 1.4.2 installed (if your Java runtime is up-to-date, you should be fine). If you are on Mac OS X or Windows, there shouldn't be a problem. Linux users, on the other hand, may have more than a few things to deal with. It may be tempting to use the GNU Compiler Collection implementations, but in my experience, they either have many bugs that make using JUICE very inconvenient, or they don't run at all. You will need a "real" Java SDK from Sun (or, if you're on a PPC machine, find some way to install IBM's Blackdown SDK).

Once you are sure Java is installed correctly, running JUICE can be very simple. Go to the directory that was created when you unzipped the JUICE files, find JUICE.jar, and double-click it.

If you're a Linux user, again, things might be different, depending on your configuration. If you can't launch JAR files graphically, or if you're on any operating system and you want to view JUICE's output to stdout (what you might call a "prompt" or "terminal"), use a terminal to change to the JUICE directory and then run JUICE with the following command:

java -jar JUICE.jar

Either way, after a brief wait, you should see a screen like the one in figure 2.1.

From here, you can access some of the basic functions of JUICE.

2.2 Preferences

Before you start delving into the program, you should set up a few preferences to make editing easier.

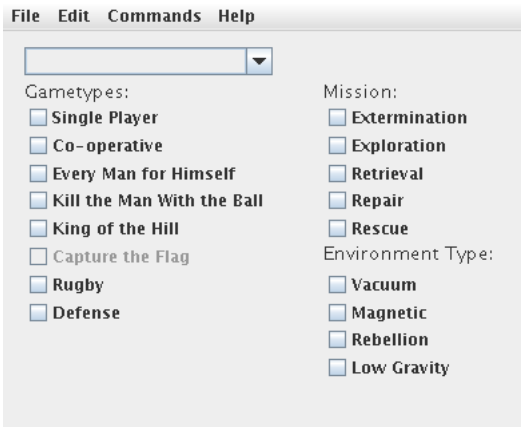

Figure 2.1: The main JUICE window

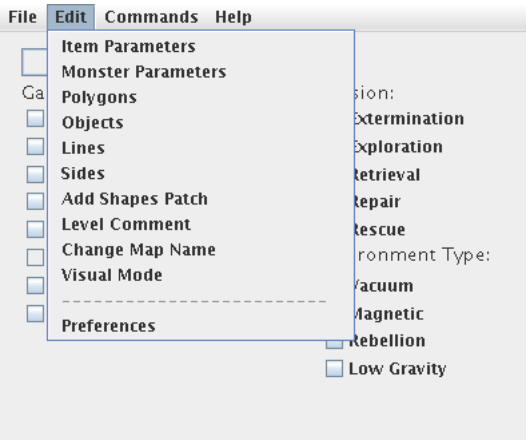

Figure 2.2: The Edit menu

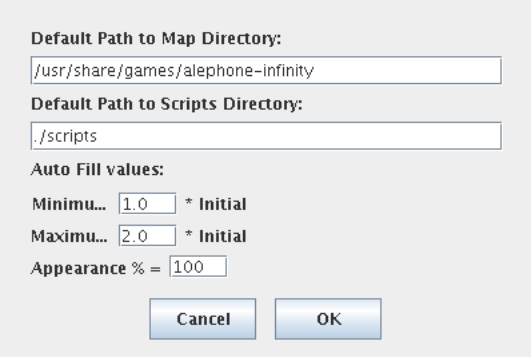

Figure 2.3: The Preferences Menu

First, open the Edit and click the bottom option, Preferences (see figure 2.2). Take a moment to ogle the other options that will be available to you once you begin editing in earnest. Once you have the preferences menu open, you should see a window similar to the one in figure 2.3.

The most important and convenient preference is the top one: the default path to your map directory. Using this, you will set a starting point for JUICE when you decide to open or save a file; by default, it starts somewhere in your user directory ("My Documents" on Windows, and your home directory on most other systems). Take a look at the example provided in my own preferences in figure 2.3.

If you're on Mac OS X, the path will be similar. On Windows, you might need a different approach. For example, if all your maps are in C:\Aleph One\maps\ then you would put that into the path field instead. If you do this successfully, you can save a lot of time you would spend navigating to the place where you usually put your files. Click OK to save the preferences and exit this menu. There will be a new JUICE Preferences file in the same folder as the JUICE.jar file you used to launch JUICE. If you want to keep these preferences, make sure you move them with JUICE.jar.

A second field, added in JUICE v1.1, points to the default directory for scripts, to be used in texture conversions (see Chapter 9). In the example, I have used "./scripts" as the directory, which means that it points to the scripts folder I created in the same place as JUICE.jar. This will make script use a lot faster, so if you plan on using scripts with JUICE, I highly recommend filling in this field.

The other three preferences come into play when using the "Auto fill" feature that comes in handy when you edit monster or item parameters (see Subsection 4.3.5: Auto fill). The autofill feature will automatically fill up certain fields in these paramters, and you set your preferences here. For example, if you wanted to auto-fill your items' parameters so that the minimum number was 2 times greater than the initial number, you would enter 2 into the second field in figure 2.3 (called "Minimum ... * initial"). To make the maximum

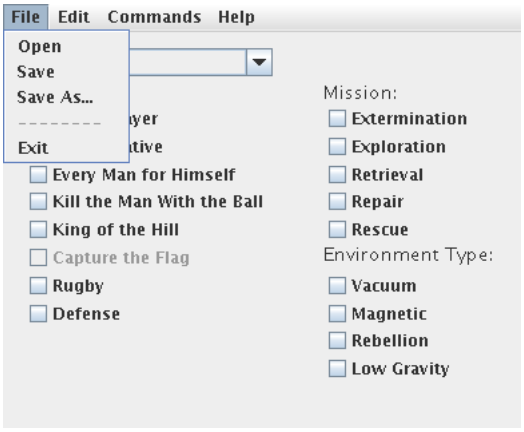

Figure 2.4: The File menu

number four times the initial number, you would enter 4 in the third box. Finally, to make the appearance rate 56%, for example, you would set the last field to 56.

2.3 Opening Map Files

JUICE can open a wider variety of map files than any utility or editor before it. Here's a list of what it can open:

- Unmerged files (created by Forge)
- Merged Forge files
- Marathon map files created by Pfhorte 2, Obed, Pfhorge, or MapEditorOne
- Unimaps; that is, any type of merged map file that has been encoded into MacBinary II or III format to preserve the resource fork, which contains terminal PICTs, chapter screens, and other things

We've done a pretty good job checking for valid map files. If you have a file that you're trying to load, and JUICE doesn't do anything about it, it's probably not a valid map file. We do not recommend loading a Macbinary-encoded unmerged Forge map file; this format has some odd problems that we haven't yet solved.

To open a map file, go to the File menu (shown in figure 2.4) and choose Open. You will be presented with a file choosing dialog (2.5). Navigate to the location of the map file you want to edit. Then click "Open" and the map will load.

When loading maps—in particular, large maps—you might see JUICE appear to "freeze." JUICE is simply taking a little while to load the map file. If this happens, you can make sure that JUICE is not actually frozen by checking the very bottom of the screen. During saving and loading, you will see a "Loading Bar" there. It should look something like figure 2.6. The dark rectangle in the loading bar will slide back and forth while the map loads (or saves), so if you see it, just wait for a little while; it'll probably be done within a few seconds.

Note that, when you've opened a map file, the main JUICE window's title bar is set to the name of that file.

2.4 Saving Map Files

Saving map files is somewhat tricky. You must be careful when you save because you will overwrite the file you are currently working on. If you make a mistake, and you don't have a backup copy, you'll have to try to figure out what you did wrong and change it back in JUICE. Because of this, I highly recommend keeping

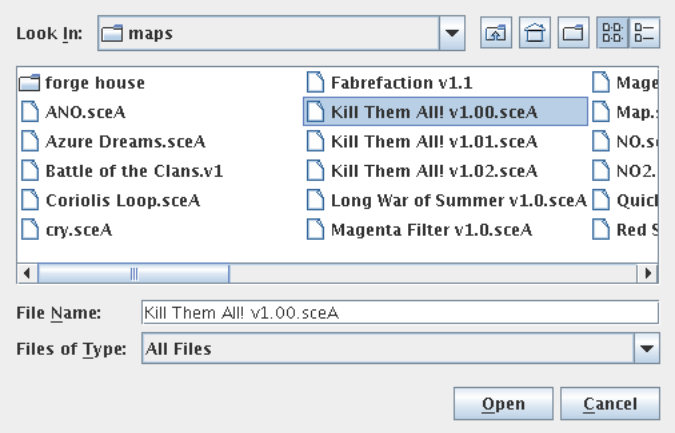

Figure 2.5: The File Chooser

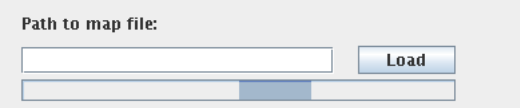

Figure 2.6: Loading Bar

a backup of your map files, unless you're working on some minor tweaks that can easily be changed back. JUICE can save in any of the formats it can read; for example, if you open a merged "unimap" file, you will be able to save a merged unimap file.

In order to save a file in this manner once you have loaded a map file, you need to go back to the File menu and click on Save (fig. 2.7). From here, JUICE will over-write the file you're working on without any further prompting. Depending on the map file size, you may see the loading bar described shown in figure 2.6.

Sometimes, you will want to save an extra copy of the map file under a different name. This is also possible in JUICE. Simply access the File menu and click "Save As..." (fig. 2.8). You will see the familiar File Chooser dialog. Change to a folder that you have permission to write to (let's say, your Desktop folder) and type in the name of the new file you want to save, as shown in figure 2.9.

After a brief time, your new file will be created. *Note:* if you use the Save As... option, JUICE will begin working on the newly saved file instead of the old file you were previously working on. The main window's title bar will reflect this change, and will display the name of the file you just saved to.

After you are finished with your editing, you will want to exit the program.

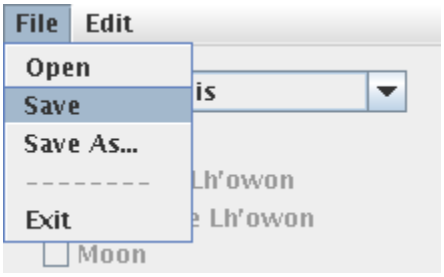

Figure 2.7: Save Menu

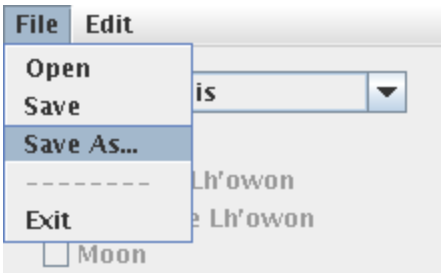

Figure 2.8: Save As... Menu

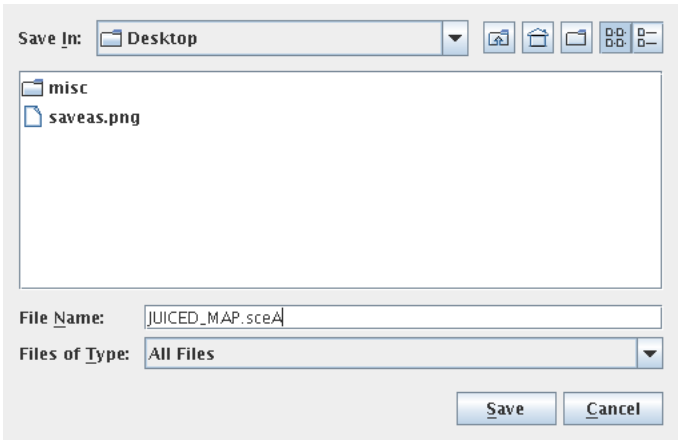

Figure 2.9: Saving to the Desktop

2.5 Quitting JUICE

Quitting is the easiest thing of all to do. Simply choose File and then Exit (fig. 2.10). You may also quit by pressing the close button that most operating systems have at the top of their windows; most often, it has an X icon on it.

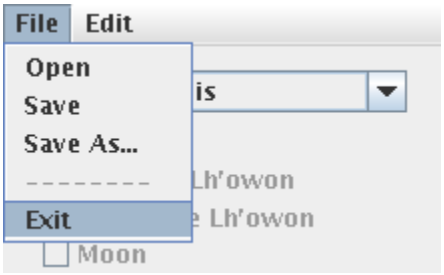

Figure 2.10: Quitting JUICE

Chapter 3

Mission and Level Information

The main JUICE window (shown again in figure 3.1) is used not only as a starting point for all the menus available in the program, but in and of itself serves a function. It is for editing the majority of the mission and level information settings.

In addition, it contains a pull-down menu that allows you to select the level you wish to edit—selection is by the level select name (as you would see in Aleph One when selecting a level in a Gather Network Game dialog or by using the level select cheat).

As you can see from figure 3.1, these settings include the following:

- Landscape type (disabled for the time being, but still displayed so you know which landscape you're using)
- Game type flags (e.g. Single-player, Co-operative, Every Man For Himself)
- Mission flags (Extermination, Repair, etc.)
- Environment type (Vacuum, Rebellion, etc.)

Each of these is easily edited; simply click the checkbox you desire to enable that option. Because all of the editable options are flags, any or all of them may be enabled.

You can perform an interesting trick using the game type flags, as shown later, in Chapter 11: JUICE Tricks. On a similar note, the "Capture the Flag" game type is not selectable because CTF is in fact the product of enabling both "Every Man For Himself" and "Kill the Man with the Ball." Therefore, in order to enable CTF on a map, you must enable those two game types, and if you have both enabled, the map is selectable as a CTF map, even if you don't want it to be. We believe this has to do with Aleph One's

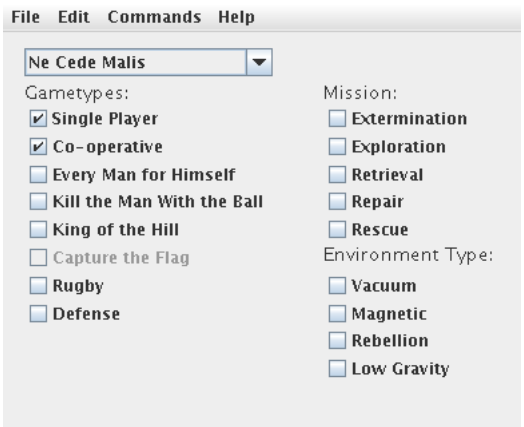

Figure 3.1: The mission info window

implementation of CTF: the "flag" is actually the ball from Kill the Man with the Ball, and the only way for it to show up is to have that game type enabled.

Chapter 4

Item & Monster Parameters

The Item and Monster Parameters settings are in what is called the "plac" section of a level; "plac" is short for "placement," so that is what I'm going to call both of these things together.

Placement settings are the ones that determine such crucial factors as:

- The initial count of an item or monster
- The minimum and maximum allowed number of that item or monster
- Whether or not it spawns in a random location
- The number of random instances available
- The appearance rate of said item or monster

Both monster an item placement settings are exactly the same, except, of course, that one deals with monsters, and one with items. A look at figure 4.1 will reveal the table layout of the Item Parameters window. One immense advantage of the table structure (which is repeated throughout the JUICE editor) is the ability to sort based on field. Click any column title to sort the table by that name or value.

4.1 Non-random settings

These are the most straightforward settings in the placement section. The initial, minimum, and maximum values should be your primary means of causing things to respawn. The "initial" field should be obvious enough: it is how many of that thing are on a map at game start. Generally, this is equal to the number of objects of that type that you have added to a map—like Forge, JUICE adds to or takes away from this value when you add an object—but, if set to a number lower than the number of spawn points for that object, not all of the spawn points will be initially in use. Setting it to a greater number seems to have no effect.

Minimum and maximum are less obvious. In the case of monsters, it's easy: only that few or that many monsters of any type may be alive on the map at any given time. For items, however, it is different. This number is equal to the number of items that may exist at any place: it includes items on the map, and in players' inventories. So if you are playing a network game with eight players, and there is a maximum of 6 SPNKRs on the map, only 6 players can carry one at a time; until one of those carriers dies, no more SPNKRs will spawn. If you have a mimumim number greater than 0, the engine will respawn new instances of that item or monster every ten seconds.

For example, with a minimum of 16 shotguns in a four-player game, you may have eight total shotgun spawn points in the level. If each player picks up two shotguns at the same time as the rest of the players (meaning that eight shotguns are now in play), in ten seconds, eight more will spawn to fulfill the minimum requirement.

Commands

Items	Initial	Minimum	Maximum	Random L...	Random ...	Infinite Av...	Random ...
Knife	0	0	0	☐	0	☐	0
Magnum	0	0	0	☐	0	☐	0
Magnum ...	21	0	0	☐	0	☐	0
Fusion Pis...	0	0	0	☐	0	☐	0
Fusion Ba...	0	0	0	☐	0	☐	0
Assault Ri...	0	0	0	☐	0	☐	0
Assault Ri...	4	0	0	☐	0	☐	0
Grenades	1	0	0	☐	0	☐	0
Missile La...	0	0	0	☐	0	☐	0
Missile 2...	0	0	0	☐	0	☐	0
Invisibility...	0	0	0	☐	0	☐	0
Invincibilit...	0	0	0	☐	0	☐	0
Infravisio...	0	0	0	☐	0	☐	0
Alien Wea...	0	0	0	☐	0	☐	0
Alien Wea...	0	0	0	☐	0	☐	0
Napalm L...	0	0	0	☐	0	☐	0
Napalm ...	1	0	0	☐	0	☐	0
Extravisio...	0	0	0	☐	0	☐	0
Oxygen P...	0	0	0	☐	0	☐	0
Energy P...	0	0	0	☐	0	☐	0
Energy P...	0	0	0	☐	0	☐	0
Energy P...	0	0	0	☐	0	☐	0
Shotgun	1	0	0	☐	0	☐	0
Shotgun ...	9	0	0	☐	0	☐	0
S'pht Doo...	0	0	0	☐	0	☐	0
Uplink Chip	0	0	0	☐	0	☐	0
Slate Ball	0	0	0	☐	0	☐	0

Figure 4.1: The Item Parameters window

4.2 Random settings

To add a little variety to games, you may adjust the random spawn properties of any placement entry. As the name implies, the "Random Location" checkbox will enable an item or monster to spawn in a random location.

This is simple enough. However, since Forge came out, one of the placement settings has been labeled incorrectly: the "Number available" field. Although the field was supposed to limit the total number of an item that could ever spawn on a map, it never seemed to have the desired effect. In looking through the Aleph One source code to understand the Placement section of a level, we found out that the "Number Available" field is actually some Bungie shorthand for "Random Number Available," and we have labeled it as such in JUICE. This field will limit the total number of items or monsters that can spawn in a random location. Let's say you had this field set to 1 for a Major Juggernaut, and you check the "Random Location" checkbox for it. During the entire time that someone plays on the level, only one Major Juggernaut will spawn in a random location. The rest will spawn according to their spawn point objects. You can check the "Infinite Available" box to allow for unlimited random-placement spawns of a given object.

The final field in the placement settings is the Appearance Rate (or often called the "Appearance % Field"). We are still unclear about the exact workings of this number; according to the source code, it only applies to the rate at which something may spawn in a random location; however, years of mapping show that it has an effect. We believe it boils down to this: if an item or a monster has a minimum value that is greater than 0, this field applies only to random spawn locations. However, if the minimum value is 0, it applies to non-random placement as well, which is why many powerups on modern network maps have a minimum of 0 with a rate that controls how quickly the power-up respawns. We will continue working to clear up this small mystery, but, as I said, experience shows that this description is more or less correct.

4.3 Commands

One of the departures of JUICE from Forge and most other editors is its interesting and useful commands. In almost all of JUICE's menus are special commands that can make mass-editing a lot easier. Figure 4.2 shows the commands available in JUICE's Item and Monster Parameters windows.

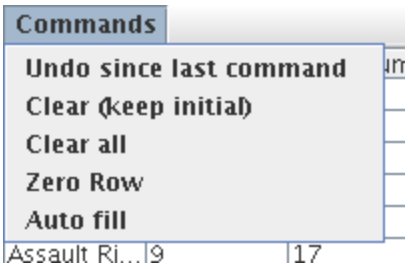

Figure 4.2: The list of commands

4.3.1 Undo since last command

This, like most commands, is more or less self-explanatory. If you use any of the other commands in this menu, you may undo the last one. So, before you panic about setting everything to 0 via a command, relax, and try the Undo command.

4.3.2 Clear (keep initial)

If you use this command, it will reset all items' or monsters' values to 0 *except* the initial number. This is useful if you are, for example, converting a network map (which generally allows items to respawn) to a single-player map, in which mappers often try to limit the numbers of items.

4.3.3 Clear all

This will reset all values to 0 (or, in the case of the Random Location flag, "off"). This is handy if you're planning on completely re-doing the placement and spawn settings of items or monsters.

4.3.4 Zero Row

This will set all values in the selected row to 0. If you have removed all instances of an item or monster from a map, it's a quick way to clear its field to make sure it won't spawn at all.

4.3.5 Auto fill

This is the most interesting new feature in the Commands menu. Using autofill, you can fill all fields according to a formula you can set using preferences (see Section 2.2: Preferences). Note that this will not do anything to rows whose initial values are 0; however, anything with an initial value greater than or equal to 1 will be set up according to the settings you define via preferences.

Chapter 5

Polygons

Ah, yes. Polygons are the basis of all Marathon levels; without polygons, a level can't exist. Before you get too excited, remember that JUICE is not a graphical editor. You can't use it to adjust polygons' positions or to add or remove polygons. We have decided that such features are better left to other kinds of editors. Here's what you *can* edit in JUICE's polygon editor:

- Polygon type (Hill, Base, Teleporter, etc.)
- Permutation (Teleport destination, Automatic Exit destination level, etc.)

We know that JUICE can't edit many aspects of polygons at the moment (only two things), but as we add support for more things, such as editing lights, media, ambient sounds, and random sounds, we will add support for assigning these attributes to polygons.

Notice in figure 5.1 that the polygon editing window is once again in table form; you can sort polygons by their editable fields (obviously, the most useful is sorting by type).

5.1 Polygon Types

A polygon's type is easily edited, and chosen from a pull-down menu as seen in figure 5.1. As an experienced mapper, you should recognize most of the polygon types available in the main menu. However, some are new in the Aleph One engine, or were added in to support Marathon 1 maps (and the M1A1 scenario). I will describe these types to you.

5.1.1 Base polygons

For use in Capture the Flag levels. Honestly, I can't say how the original CTF mode implemented by Benad used them, but I do know how the newer CTF Lua script (presently at version 4) uses them. A base polygon will be the location of a team's flag (see section 5.2, Permutations, for more details).

5.1.2 Minor & Major Damage

Also known as "Ouch" polygons, these were present in Marathon 1, which did not have liquids. Therefore, in order to implement lava or goo, the game needed a kind of polygon that would deal damage. Minor damage polygons are analagous to lava, and Major damage are analagous to Pfhor goo. Note that any position within a damage polygon, be it on the floor or in the air, allows the player to take damage. You might use this to simulate a radioactive room, or to fool players in netmaps—why not make those water-covered polygons deal some damage, too?

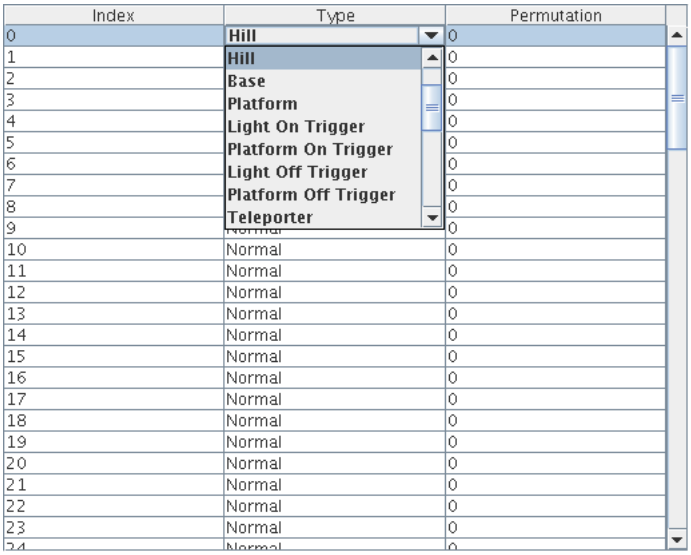

Figure 5.1: The polygon editing window

5.1.3 Glue & Glue Trigger polygons

In spite of the name, these polygons do not cause players or monsters in them to get stuck inside. Instead, the glue and superglue polygons act in conjunction with the glue trigger type, serving as a sort of modified monster trigger.

5.2 Permutations

Editing a polygon's permutation is easy: just click in the number field and type the new permutation number. A simple definition of a polygon's permutation is the target of that polygon. Some polygon types' permutations are obvious, some are not, and some don't even have a use for the permutation field.

5.2.1 Obvious permutations

5.2.1.1 Light on trigger

This type's permutation designates the light that is activated by entering the polygon.

5.2.1.2 Platform on trigger

The permutation of a platform on trigger polygon is the polygon index that the trigger will activate. Obviously, the target polygon should be a platform.

5.2.1.3 Light off trigger

See Subsection 5.2.1.1.

5.2.1.4 Platform off trigger

See Subsection 5.2.1.2.

5.2.1.5 Teleporter

A teleporter's permutation is the index of the teleporter's destination polygon.

5.2.1.6 Goal

Goal permutations may range from 0–10. It's just the kind of goal that this is (if you don't know about goals, read the FORGE Manual).

5.2.1.7 Automatic exit

The first kind of permutation for an automatic exit is the destination level that the automatic exit will lead to. For a novel second kind of permutation, see Chapter 11: JUICE Tricks.

5.2.2 Obscure Permutations

5.2.2.1 Base

A base polygon's permutation is the color of the team that owns the base. The colors are numbers ranging from 0–7:

- Slate team (0)
- Red team (1)
- Violet team (2)
- Yellow team (3)
- White team (4)
- Orange team (5)
- Blue team (6)
- Green Team (7)

5.2.2.2 Platform

What would go in the platform permutation? It turns out that there is another section of the level data that defines the platforms in a level. This permutation is the index of one of the platforms. You probably don't want to mess with this until we allow you to edit platform parameters; however, if you're curious, check out Chapter 11 to see some possible tricks that can go with editing this permutation.

5.2.3 Ineffectual Permutations

The rest of the polygon types appear to have no reliance upon their permutations. We'll keep fiddling around to find out if this is true, but from what we've found so far, editing these poly types' permutations won't do anything.

Chapter 6

Level Comments

Remember that thing I said in the introduction about adding totally new stuff to map files? This is one of those new things. You may add a comment to any level you wish; the maximum length is 512 characters (anything more will be truncated). For the moment, it is only used internally; Aleph One doesn't even read it.

So then, what is its use? Our intention for the level comment (or, as I like to call it, the "JUCE" Chunk) is for things like adding level author information; a small Readme for each level, to describe which engine version it requires, which Lua script, etc.; notes written to yourself or to other mappers—especially if you are taking turns in a Map Making Roulette, and wish to notify the next guy of any changes.

To our knowledge, editing a map and saving it in any program other than JUICE will eliminate any of its levels' comments. This may be inconvenient, but in certain situations, especially during an MMR, it's not so bad; it erases the previous mapper's comment and allows the present mapper to add a new one for the next guy.

There are two ways to add a level comment, by typing directly into the text field, and by importing a text file.

6.1 Typing the comment

As seen in figure 6.1, there's not much to adding a new level comment. You just type it in and press OK to confirm or Cancel to erase the comment. Either button will also return you to the main JUICE window.

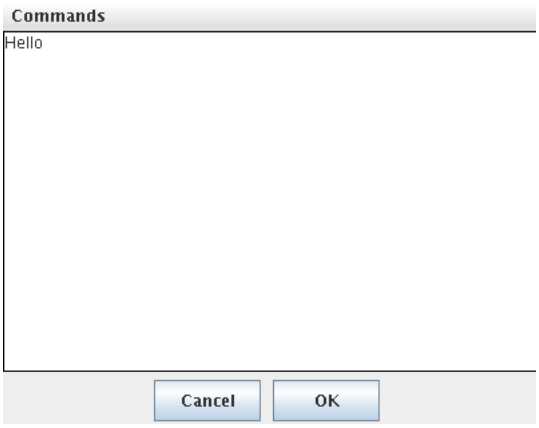

Figure 6.1: Level comment

6.2 Importing from a text file

This method allows you to use a template or to edit and save level comments externally, in case you want to put them back in after you've used another program to edit.

The Commands menu contains the Read from file command. Choosing it will bring up the familiar file chooser window. Pick a file (hopefully a text file), press OK, and you will have the first 512 characters added to the level comment.

Chapter 7

Objects

Here, we come to the heart of JUICE: editing objects. An object refers to any of the following things:

- Monsters
- Scenery
- Items
- Players
- Goals
- Sounds

Take a look at figure 7.1, the Objects editing window. You will see the usual column arrangement; each is easily sorted by clicking the column title. You will also see the little Commands menu that appears in other sections.

Type and Subtype are both pull-down menus; which type you choose affects which subtypes are available.

Polygon, x, y, and z should make sense already. Entering a polygon that does not exist in the level will cause the field to set itself to the highest polygon index available. Z is not an absolute z-coordinate; as in Forge, it is the distance from the polygon's floor (or ceiling, if the "from ceiling" flag is checked).

7.1 Type-specific notes

Each type has some nuances associated with its parameters. Here are some explanations of each type's oddities.

7.1.1 Monsters

Well, I lied. Monsters are monsters, and aren't too complicated. Maybe that's why it's the first type!

7.1.2 Scenery

Scenery is also pretty straightforward. Just note that, like items below, scenery may have a facing value. As of Aleph One version 0.18.0, scenery items associated with any texture set will load in any level, so you can add stuff like water puddles in lava levels.

7.1.3 Items

In case you didn't know, items can have facing values, just like monsters and scenery. This only has an effect if you've modified the shapes file to contain multiple views. Also note that we allow you to use several unusual item subtypes—we allow all ball colors (the regular ball is apparently the "red" ball); Alien Weapon ammunition placement and "knife" item placement.

Commands

Type	Subtype	Facing/volume	Polygon	x	y	z
Monster	Pfhor Fighter...	235	203	3.3125	2.0	0.0
Monster	Pfhor Fighter...	231	92	14.68359375	16.81640625	0.0
Monster	S'pht (minor ...	225	488	7.984375	-2.734375	0.0
Monster	S'pht (major)	112	464	2.26953125	-12.3046875	0.0
Monster	S'pht (major)	87	459	-3.9921875	-7.35546875	0.0
Monster	Pfhor Fighter...	100	513	-2.79296875	-28.043945...	0.0
Monster	Pfhor Fighter...	72	513	-4.125976...	-28.691406...	0.0
Monster	Pfhor Fighter...	90	535	-3.50390625	-30.735351...	0.0
Monster	Bob (crew)	257	520	-2.831054...	-17.214843...	0.0
Monster	Bob (crew)	161	517	-2.611328...	-20.280273...	0.0
Monster	Pfhor Fighter...	46	530	-4.278320...	-13.977539...	0.0
Monster	Bob (crew)	355	345	-10.75292...	6.0556640...	0.0
Monster	S'pht (minor ...	81	374	-9.483398...	-7.2744140...	0.0
Monster	Pfhor Fighter...	88	396	-0.45703125	-8.5947265...	0.0
Monster	Pfhor Fighter...	83	396	-1.47265625	-8.6074218...	0.0
Monster	S'pht (major)	0	318	5.9921875	-0.9775390...	0.0
Monster	S'pht (minor ...	88	167	17.494140...	5.53515625	0.0
Monster	Bob (crew)	159	299	9.8515625	-5.1035156...	0.0
Monster	Pfhor Fighter...	5	298	7.9853515...	-4.3798828...	0.0
Monster	S'pht (minor ...	181	150	17.227539...	-1.6123046...	0.0
Monster	Pfhor Fighter...	88	449	13.012695...	-8.0361328...	0.0
Monster	Pfhor Fighter...	324	112	9.744140625	11.123046...	0.0
Monster	Pfhor Fighter...	221	76	16.65625	11.62890625	0.0

Figure 7.1: Objects editing window

7.1.4 Players

Player objects are mostly straightforward. The most interesting thing to note about them is that the player subtypes are all team colors, so you can use JUICE to create team spawn points for games like Capture the Flag and team King of the Hill.

7.1.5 Goals

Goals' subtypes range from 0–10, just like in Forge. If there is support for more than 11 goals in Aleph One (and it was simply a limit imposed by Forge), we will add more later. But who needs that many?

7.1.6 Sounds

Thanks to some complications caused by the optimizations introduced by Forge's merge process, editing sounds is tricky. Therefore, if an object is not a sound, you can't change it to a sound; otherwise, all types are interchangeable. We intend to find some way to understand the merge process, or, if that fails, we'll simply strip the level of this information.

Sounds also have a confusing convention: their "Facing" value is actually the value of their volume, on a scale from 0–180. If the value is negative, this means that the sound's volume is tied to a light; the formula is $v = -1 \times (l + 1)$, where v is the value entered in the volume field, and l is the light's index. For example, if you want to link the volume to light index 20, you would enter -21 as the value for volume. We will make this process more intuitive when we add support for lights.

7.2 Commands

The commands for Objects are highly useful, and are streamlined from years of mapping experience. Figure 7.2 shows the list of commands.

As you can see, there are quite a few useful ones included in JUICE, and we're likely to add even more. Also notice that we have keyboard shortcuts for most of these essential commands.

7.2.1 Copy, Paste & Duplicate

Each of these does what you would expect: copy puts a copy of the object into memory; select another object and use paste to over-write it with the copied object; use duplicate to add a new object with the properties

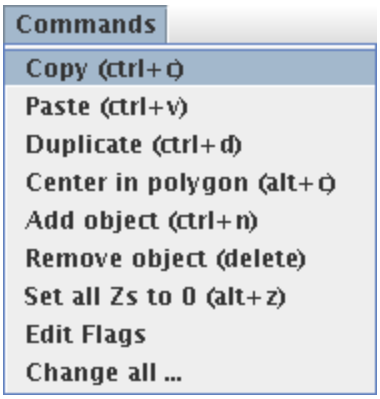

Figure 7.2: Object editing commands

of the selected object.

The only information that copy does *not* copy is the polygon and x-y-z coordinates of the object. Duplicate, on the other hand, creates the new object with exactly the same parameters.

7.2.2 Center in polygon

We added this command because JUICE does not have a graphical way of editing the coordinates of items. We didn't want to risk a player putting an item in impossible coordinates, so we give you the option of centering the object in its polygon. From there, you can change its position bit by bit until you're satisfied. Centering does not change the z coordinate; instead, z is set to 0, since that's where most objects are located anyway.

7.2.3 Add object & remove object

This should be obvious. Adding an object puts a new one at the end of the list; its initial type and subtype are "scenery" and the first subtype of scenery (level-dependent). It is automatically centered in polygon 0.

Removing an object is the reverse of adding one. I hope you knew that.

7.2.4 Set all Zs to 0

This will cause all objects in the level to be set at exactly floor height (or ceiling height, if they are hanging from the ceiling).

7.2.5 Edit flags

This will bring up a window with all of the flags available for a particular object type. Some flags are available for all types, while others are not. Goals have no flags. See figures 7.3–7.7.

7.2.6 Change all...

Use this command to convert all objects of a certain subtype to a different subtype that part of the same type as the original. For a clearer explanation, see figure 7.8.

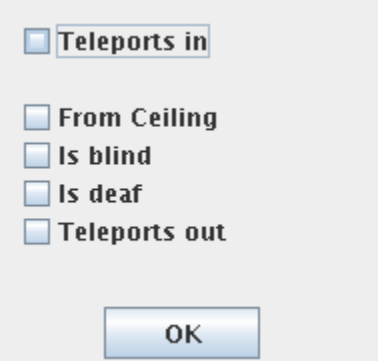

Figure 7.3: Monster flags

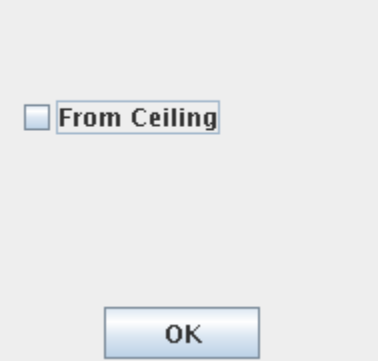

Figure 7.4: Scenery flags

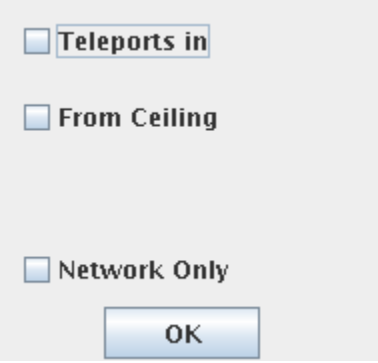

Figure 7.5: Item flags

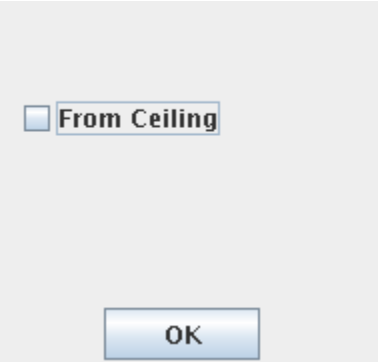

Figure 7.6: Player flags

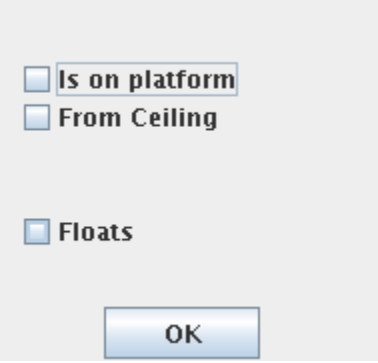

Figure 7.7: Sound flags

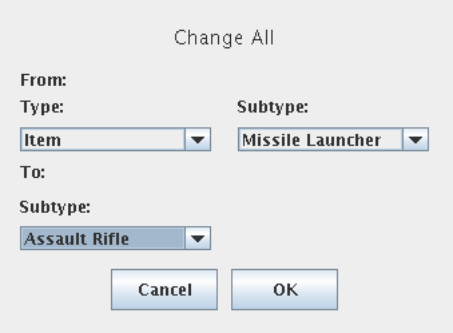

Figure 7.8: Change all

Chapter 8

Lines & Sides

Editing lines (the LINS chunk) and sides (SIDS chunk) is mainly in JUICE for completeness, although we might introduce some novel features with them later. However, there are a few tricks you can pull off by editing these relatively simple things.

8.1 Lines

You can see a screenshot of the Lines editor in figure 8.1.

As you can see, it's in the usual table format. You should be familiar with the "Solid," "Transparent," and "Landscape" flags, but the other options might be new and confusing. Note that the rest of the fields can only be edited if the map file is merged, because unmerged maps don't take advantage of the optimizations that Forge-merged maps create. This includes pre-calculation of lengths and side heights.

8.1.1 Length

Length is straightforward in one sense: it adjusts the supposed length of the line. However, this in no way affects how long a polygon's side will be. Instead, Aleph One uses this value—when found in Forge-optimized maps—in order to calculate the length of the textures rendered on the side. If you are familiar with a Chisel effect that stretches, compresses, or reverses wall textures, this is how the effect is accomplished. For example, let's say the line's length is 1024 (equal to one World Unit in Forge). If you want to stretch that line's texture, reduce the length. Changing it to 512 will render the texture twice as long. Conversely, you can make the length longer to compress the width of the texture; changing this line's length to 2048 will make it twice as short. You can also reverse textures by making the length negative; try this especially on lines that have a "horizontal slide" texture on them. See figure 8.2 for examples.

8.1.2 High Floor/Low Ceiling

These two options are just like the "length" property—they only have an effect if the level is Forge-optimized and merged. These values will adjust the height of the line along the floor or ceiling of that line. With these values, it is possible to create pseudo-railings (see section 11.5).

8.1.3 CW & CCW Poly

CW (Clockwise) and CCW (Counter-clockwise) polygons can't be edited. They are here for sorting purposes, so that, by finding the polygon with the line you want to edit, you can sort by either column and more easily locate the line you want to edit. Note that the "None" value means that the line is part of a wall (it runs along "negative space").

Index	Solid	Transpa...	Landsca...	Length	High Floor	Low Ceili...	CW poly ...	CCW pol...
0	☐	☑	☐	1024	1	1024	2	553
1	☑	☐	☐	256	1	1024	2	None
2	☑	☐	☐	256	1	1024	2	None
3	☑	☑	☐	1024	1	1024	1	2
4	☑	☐	☐	256	0	1024	1	None
5	☐	☑	☐	1024	0	1024	0	1
6	☑	☐	☐	256	0	1024	1	None
7	☑	☐	☐	256	0	1024	3	None
8	☑	☐	☐	256	0	1024	5	None
9	☑	☐	☐	362	0	1024	3	None
10	☑	☐	☐	362	0	1024	5	None
11	☑	☐	☐	181	0	1024	10	None
12	☑	☐	☐	181	563	563	6	None
13	☑	☐	☐	128	563	563	6	None
14	☑	☐	☐	181	563	563	6	None
15	☑	☐	☐	181	563	563	6	None
16	☑	☐	☐	128	563	563	6	None
17	☑	☐	☐	181	563	563	6	None
18	☐	☑	☐	362	512	665	5	18
19	☑	☐	☐	181	0	1024	10	None
20	☑	☐	☐	256	0	1024	5	None
21	☑	☐	☐	256	0	1024	0	None
22	☐	☑	☐	1024	0	1024	0	188
23	☑	☐	☐	256	0	1024	0	None
24	☑	☐	☐	256	0	1024	3	None
25	☐	☑	☐	362	512	665	3	17
26	☑	☐	☐	181	511	511	4	None
27	☑	☐	☐	128	511	511	4	None

Figure 8.1: The Line editor

8.2 Sides

Lines may "own" up to two sides, one in each polygon it constitutes. Sides contain more of the visual properties that you find in a map, including textures. Take a look at the Sides table (fig. 8.3) to see what is initially available to you.

8.2.1 The fields

Only one of the three columns in this table (the "Type" column) may be edited. The other two are for sorting purposes.

What does the "Type" field indicate? This is how the three side segments—the primary (top) segment, the secondary (bottom) segment, and the transparent (middle) segment—are set up.

8.2.1.1 Full

The floors and ceilings on either side are at equal height, indicating that no texture is visible, or that only the transparent texture is visible.

8.2.1.2 High

The primary segment is the one that shows, and whose texture is displayed.

8.2.1.3 Low

The secondary segment is the one that shows, and whose texture is displayed.

8.2.1.4 Composite

A combination of high, low, and full.

8.2.1.5 Split

A combination of high and low.

Figure 8.2: Comparison of texture lengths

Commands

Index	Type	Line Index
0	Full	21
1	Full	23
2	Full	4
3	Full	6
4	Full	1
5	Full	2
6	Full	24
7	Split	31
8	Full	9
9	Full	7
10	Full	26
11	Full	27
12	Full	34
13	Full	30
14	Full	29
15	Full	28
16	Full	20
17	Full	8
18	Full	10
19	Split	32
20	Full	17
21	Full	16
22	Full	15
23	Full	14
24	Full	13
25	Full	12
26	Split	55

Figure 8.3: The Sides editor

8.2.2 Commands

There are two commands for this editor: Properties and Flags.

8.2.2.1 Properties

You may also access this command using the keyboard shortcut Control-P.

As you can see in figure 8.4, Properties control the side permutation, the panel type, the side textures, the side transfer modes, and the side lights.

The only thing that might be confusing is the "permutation." This is simply whatever that side controls if it is a control panel: the light for a light switch, the terminal script for a terminal, the platform for a platform switch, etc.

Also note that all sides have a panel type, but that type only has an effect if "Is Control Panel" is checked in the side's flags.

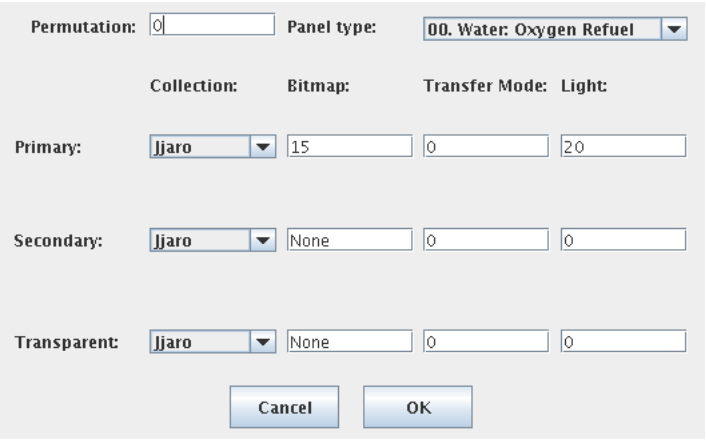

Figure 8.4: Side Properties

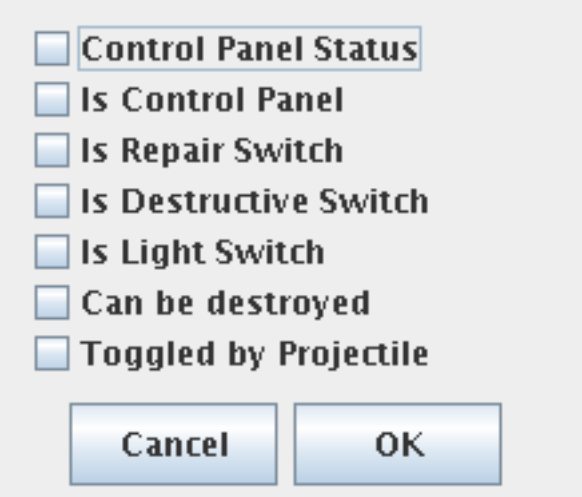

Figure 8.5: Side flags

8.2.2.2 Flags

You may also use the Control-F shortcut to edit flags. A side's flags (seen in fig. 8.5) are mostly self-explanatory, and should be familiar from Forge. The only one that might seem new is the "Control Panel Status." This determines whether or not the control panel is active.

Chapter 9

The Texture Converter

As soon as Aleph One officially introduced support for multiple texture sets without physics hacks like those seen in the Chisel effect called "Texture Munger," work began on a system that would allow JUICE to recurse through all texture-capable surfaces in a map. During recursion, JUICE looks for all textures of a given collection and bitmap (for example, Water bitmap 15) and changes them. This allows for simple operations like changing all surfaces with the water tile texture to the water rock texture, or they can be more complex, extending to include conversion by transfer mode as well. Access the texture converter via the Commands menu in the main JUICE window, as seen in figure 9.1. There are two interfaces for converting textures: the regular Convert Textures dialog and the scripting interface.

9.1 The Dialog Box

The dialog box first appears as shown in figure 9.2. Using the dialog allows for two basic types of texture conversion: those without transfer modes, and those with transfer modes.

9.1.1 Without transfer modes

In order to convert textures without taking into account those surfaces' transfer modes, leave the "Convert transfer modes" box unchecked. You will then have four fields to modify:

- From: Collection—This is simple enough: select the wall texture collection that contains the texture you want to convert.
- From: Bitmap—The bitmap is the image number (starting at #0) in the selected collection. For example, bitmap 19 in the Water collection is the water surface texture. Use an editor such as Anvil or Shapefusion to find the bitmap index.
- To: Collection—This is just what it sounds like: the collection containing the texture to which you want to convert.
- To: Bitmap—As with the To: Collection, this item is the texture bitmap to which you want to convert.

When you have everything as you like it, press "OK" to perform the conversion. The window will remain open and the values unchanged in case you want to perform more than one conversion in a row; if you wish to return to the main window, press "Done."

You can see a complete example in figure .

9.1.2 With transfer modes

If you check the "Convert transfer modes" box at the top of the dialog window, two new areas of the window will become editable. These allow you to be more specific in your conversion, taking into account the transfer mode (e.g. "Normal," "wobble," "horizontal slide") of the texture. Choose which transfer modes you want to convert between. See a complete example in figure .

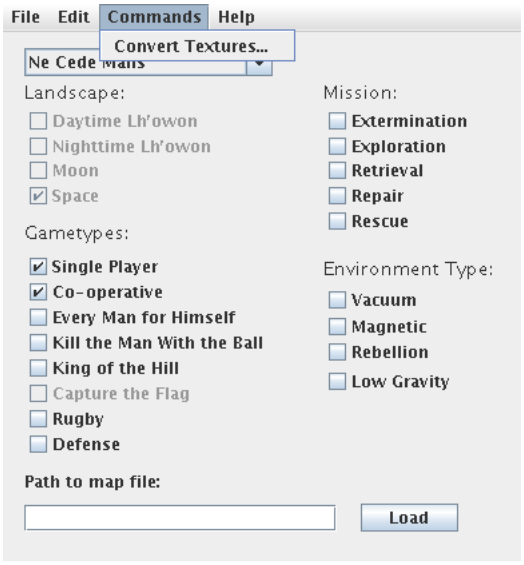

Figure 9.1: The Convert Textures menu

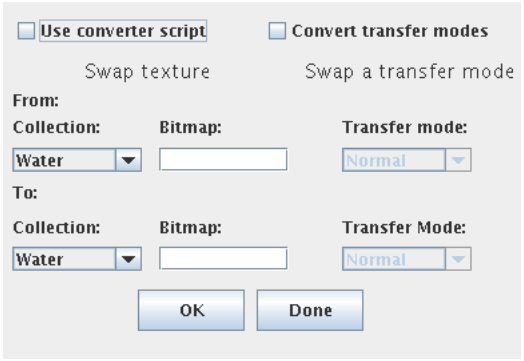

Figure 9.2: The unmodified dialog

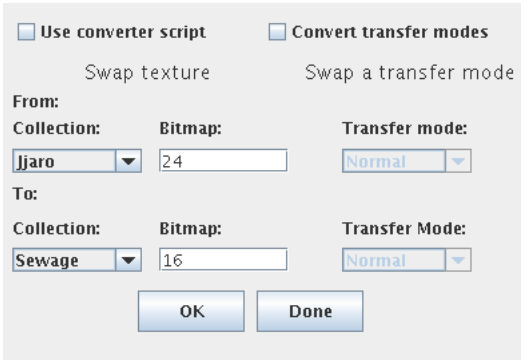

Figure 9.3: Converting without transfer modes

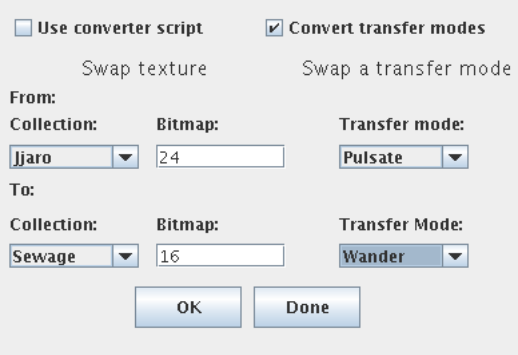

Figure 9.4: Converting with transfer modes

9.1.3 Special numbers

There are special numbers that you can enter into the bitmap fields, each one producing a special result. These numbers range from -1 to -5. If you enter a number that goes below -5, JUICE will set the field equal to -5.

9.1.3.1 No texture (-1)

To convert a texture to "none," enter -1 in the To: Bitmap field. To replace all untextured sides with a texture, enter -1 in the From: Bitmap field.

9.1.3.2 Landscape textures (-2 to -5)

Instead of adding four more collections to both "Collection" menus, we decided to make converting to and from landscapes available by entering a negative number into the bitmap fields. The landscape numbers are as follows:

- Lh'owon Daytime (-2)
- Lh'owon Night (-3)
- Moon (-4)
- Space (-5)

9.1.4 Implications

Not only can you convert between two bitmaps in the same texture set, but you can also introduce textures from other sets into a map that formerly contained none of those sets. While Aleph One is now capable of loading multiple texture sets, the most popular and powerful editor, Forge, does not allow this, and never will. Using the converter, you can get around this editor-based limitation. Consider the following example:

> I want to create a Capture the Flag map with red and blue bases. I can't find the CTF development kit that helps this process, or I don't want to deal with Chisel, the utility that allowed multiple texture sets in a map. Instead, I designate the level as a "water" level and use Forge's visual mode to texture the level as usual, except that where I place texture bitmap 26 (a blue texture) in "Pulsate" transfer mode, I intend to have a lava-set red texure (lava collection, bitmap 7). When the map is finished, I convert all instances of water bitmap 26 "pulsate" to lava bitmap 7 "normal." And now I have a red/blue CTF level!

Now, I hear you saying, "how will I make a map that contains lots of textures from other sets? I don't want to memorize all the transfer modes that I use, and it will take forever to repeatedly convert one texture at a time. This is where the scripting engine comes in.

9.2 Scripting the texture converter

Scripts are essentially lists of conversions to perform in rapid succession. While creating a large script can take a while, it is well worth the effort.

9.2.1 Using a script

In order to use a script, all you have to do is check the "Use converter script" checkbox (see fig. 9.1). This will make all other fields un-editable. Then press "OK." JUICE will present you with a file selection dialog. If you configured a default scripts directory (see section), the chooser will jump straight to that directory.

Make sure you choose a valid JUICE converter script! The converter will do its thing, and the dialog will close. A new script file, with ".undo" added to the end of its filename, will appear in the same directory. As you might guess, this script will allow you to undo the effects of the conversion, in case you wish to edit the level in Forge again.

9.2.2 Making a script

Script syntax is pretty simple: one conversion (or comment) per line. There are presently four different lines that are valid in the script format: comments, bitmap-only conversions, transfer mode conversions, and control panel conversions.

9.2.2.1 Comments

The easiest line of all, a comment is simply a line in whish you want to add a note about something in the script. These are great for showing the beginning or end of a certain conversion section in a script, or for disabling a conversion line without deleting it. Commented lines are designated with the # character at the beginning of the line.

9.2.2.2 Bitmap-only conversions

As discussed in section 9.1.1, the simplest conversion type only takes collection and bitmap into consideration. To specify a conversion of this type, create a line in the script in the form *c1,b1-c2,b2,* where c1 and b1 are the collection and bitmap to convert from, and c2 and b2 are the collection and bitmap to convert to. For an example, see section X.

9.2.2.3 Transfer mode conversions

Again, this type of conversion should be familiar from section 9.1.2. The format for this is similar to the bitmap-only format, but is as follows: *c1,b1,x1-c2,b2,x2,* where c1, b1, and x1 are the collection, bitmap, and transfer mode of the "from" conversion, and c2, b2, and x2 are the collection, bitmap, and transfer mode of the "to" conversion. See the Appendix for a list of the transfer mode index numbers. Again, see section X for an example.

9.2.2.4 Control panel conversions

Because of the way that Aleph One and Forge handle control panels, it was necessary to create a third kind of conversion for use in scripts. This conversion takes a couple more numbers: *c1,b1,x1,p1-c2,b2,x2,p2,* where the c, b, and x values are as in the other types of conversions, and p1 and p2 specify the control panel types to convert from and to, respectively. See the Appendix for a list of control panel types, and see section X for an example.

9.2.3 Example script

```
#This is a comment.
#Here is a bitmap-only conversion:
17,6-18,9
```

```
#(Water bitmap 6 to lava bitmap 9)
#Here is a transfer mode conversion:
20,3,4-21,10,0
#(Jjaro bitmap 3 "pulsate" to
#Pfhor #10 "normal")
#Here is a control panel conversion:
20,2,4,43-17,2,0,1
#(Jjaro 1x charger "pulsate" to
#Water 1x charger "normal")
```

9.3 The Big Picture

All of this stuff comes together when someone creates a script for JUICE and a special shapes file for Forge. Gregory Smith, the primary developer of Aleph One, has already created both for the Infinity textures. Using his example, making a multi-texture-set map goes like this:

1. Create a map in Forge under the Jjaro texture set.
2. When you texture the level, you will see that most textures from the other sets are included in the new Jjaro shape collection. Use these textures normally.
3. Once the map is done, run it through JUICE with the converter script in order to convert it to a map that has all the textures you used, but which is compatible with the default Infinity shapes file (and, therefore, requires no extra shapes file download for players in network games, etc.)
4. If you need to edit the level again, use the .undo script to convert it back to its original form.
5. You can also use Forge to merge this map with others; Forge won't strip away the JUICEd texture configuration.

As you can see, texture conversion is an extremely useful tool. This is the most important feature in JUICE so far, and we hope you find it as useful as we do!

Chapter 10

Shapes Patches

Shapes patches, added in JUICE 1.1.1 in parallel with a new release of Aleph One, allow for something that has long been discussed and dreamed, but never executed: the addition of particular shapes to a level.

Normally, this is achived through manipulation of an external Shapes file. The disadvantage of this normally acceptable method is its externality. Hosting a network game with special shapes requires all players to get and use the external file. This is inconvenient at best, and a failure at worst. It is possible to distribute a patch (produced by the general-purpose editor called ANVIL) instead of an entire shapes file, which usually cuts down on the download size, but it still requires an external application (Anvil) to apply the patch.

JUICE can now import a shapes patch produced by Anvil into a given level, allowing for elegant touches to maps such as new textures or new weapons. Because this data is embedded as a level chunk, it is automatically transmitted across the network during network games. Even for single-player games, there is an advantage to this method: it should now be possible to distribute a level with a few new shapes and custom physics, all using only one map file.

To add a shapes patch to a level, choose the level you wish to patch, then go to the Edit menu (see figure 2.2), and choose "Add Shapes Patch." This will bring up a file chooser starting in the same directory where JUICE is located. Choose a shapes patch exported from Anvil, and JUICE will do the rest. There is a limit to the size of a patch file, 256 kilobytes (KB). For most purposes, this should be enough: it allows for up to 17 custom textures or quite a few sprite changes. Remember that you can add a shapes patch for *each level*, so this should be enough for a netmap pack or a small single-player map file. Aleph One will ignore patches for collections with a "custom" (greater than 8-bit) palette, but JUICE does not prevent these from being embedded.

For now, it is only possible to add a shapes patch to one level at a time, but we intend to add batch-import capabilities in the future. Note that older versions of Aleph One will ignore embedded shapes patches, so be sure to include a warning with your map that players should use the latest version of the engine.

Chapter 11

JUICE Tricks

JUICE's approach to editing allows for a few very interesting tricks. We have decided to hide little or nothing from mappers, instead giving them access to many values as if they were editing the map file's raw hex code (but far safer, of course). I guess you could view these "tricks" as undocumented features; either way, they provide different or new ways of doing things.

Note: steps with chapter or section references are possible (or sometimes only possible) in JUICE.

11.1 Hiding Levels

Have you ever created a great secret that led to a secret level, but realized with a sinking feeling that Aleph One's level select cheat allows the player to enter that level at will? Well, if you are editing mission and level information (see Chapter 3), you can set a level's game types so that none are selected, not even single-player mode. This will hide the level from all level select lists.

Be careful with this feature. You should put all hidden levels after your visible levels, because if you hide a level, and levels after it will be offset in the wrong direction. For example, let's say you hid level 2 of a three-map netpack. If a player gathers a game and selects level 3 from the level list, he will actually select level 2. This problem appears to be remedied if you place hidden levels after normal ones.

11.2 Different Level Select & Overhead Map Names

I had started fiddling with editing a map's hex code and I saw that there were two locations that held a level's name. I didn't think much of it until I made a level that I thought should have two names: one for the level select dialogs (such as the level select cheat or the gather network game selections) and the overhead map screen.

I realized that it was possible when I remembered the extra name reference in a map file. Using the Edit Level Name menu (see Chapter X), you can set a level's two names to different values.

Something to note: the level select name comes into play in another context: on the metaserver. When a player hosts a game on the metaserver, it is the level select name that is printed and sent as a message to other players, telling them what is being hosted.

11.3 Switch-Dependent Teleporters

I admit, I picked this up from some map I don't remember. But I couldn't find any documentation on how it was accomplished, and so I took the time to find out for myself. Follow these steps to have a teleporter that only functions when a certain switch is "on":

1. Go to Mission Info (Chapter 3) and check the "Repair" box.

2. Create a new light. Make it "initially inactive" and make sure it has a different active and inactive setting.

3. Create a light switch somewhere in the level, mark it as a "repair" switch, and set its corresponding light to the one you created.

4. Find the polygon you want to make into a teleporter and be ready to edit it (see Chapter 5). Instead of making it of type "teleporter," make it of type "automatic exit." Then, using JUICE to edit its permutation, change it according to this formula: $p = -1 \times (i+1)$, where p is the permutation number to enter and i is the index of the polygon to which you want the teleporter to lead; to teleport to polygon index 194, you need to enter -195

This will give you a teleporter that only works when the switch is on! You can have as many teleporters with as many destinations as you like, but they will all be either on or off; the "automatic exit" part of the scheme causes this. Using any other mission types will affect the function of these teleporters; all mission goals must be fulfilled for the teleporter to work. For this reason, you should only have the level of mission type "repair" if you want the teleporters to work properly.

11.4 Odd Platform Triggers

If you remember from Subsection 5.2.2.2, it is possible to set a polygon's platform permutation. There are many interesting consequences of this that the automated systems in Forge and other editors did not allow. I'll use one example and let you think of others.

Let's say you want to make a "platform on trigger" polygon that monsters can trigger. This isn't normally possible. However, if you create a platform that is triggered by monsters (maybe a crusher or door, or whatever you want to trigger), you can do this exact thing.

1. Create the polygon you want the monster to activate (let's say it's a player-crusher).

2. Edit that polygon so that the crusher is a platform with your desired attributes. If you don't want the player to activate it when he stands on it, make sure the "activated by player" flag is off. Turn the "activated by monsters" flag on.

3. (This is the hard part, and is not yet easily supported by JUICE.) Try to remember the order in which you've created your platforms. Let's be silly and say that this crusher is the only platform in the level, so we'll say its platform index is 0. You should probably also make the platform "hidden."

4. Use JUICE to edit the polygon that should be the trigger polygon (see Chapter 5). Change the polygon to the platform type, then enter as the polygon's permutation the desired platform index; in our simplified case, platform index 0.

5. This polygon now counts as that platform, but it will not exhibit any of the platform's properties; however, it *will* be triggered by monsters, and will in turn also activate the "real" crusher.

I know it's complicated and not yet possible entirely in JUICE, but think of the cool things you can do; what you essentially have is a trigger that can have different attributes for activation, delay, etc. When we add platform editing to JUICE, this trick will be a whole lot easier.

11.5 Railings

Something that games like Doom have always had, railings are one of those things that could add a nice visual touch to a level. They're somewhat possible in Marathon maps, but with some caveats that will be discussed. Here are the steps to create two different kinds of railings. While you read, note that making the line "solid" will cause players, monsters, and projectiles to be unable to travel through the space above the railing. Sorry. We're trying to find a solution for this.

11.5.1 Railing with a solid texture

These are the more feasible, if less impressive, sort of railing. They are rather easy to create using Forge and JUICE.

1. Create a level, making note of which lines you wish to turn into railings. Raise the floor on one side of this line, texture it with the solid texture you wish to use, then do the same on the other side of the line. After you've done this, make the polygons any heights you wish.
2. Merge the level in Forge (this trick requires Forge's merge optimizations).
3. Open the merged map file in JUICE, access the level to which you wish to add railings, and edit its lines.
4. Find the lines you want to edit by index, then change the "Highest Floor" parameter to something above the actual "Highest Floor" value. Remember that the units in this box are in 1/1024 of a Forge World Unit. For example, to make a railing that was .25 WU off the ground, you'd change this value to $n = o + 256$, where n is the new value and o is the original value.
5. Check out your new railing in Aleph One!

11.5.2 Railing with a transparent texture

The more impressive kind of railing uses a transparent texture (often in the form of grates, etc.).

1. Create the level in Forge. On the lines for which you wish to have the transparent-texture railing, press Shift and texture the middle area of that side (as you do to add them normally).
2. Do not add a texture to either side of the line's ceiling.
3. Save and merge the level with Forge.
4. Open the level with JUICE and edit its lines.
5. Edit the line indexes where you want railings. You should set the line ceiling height according to $h = r - f$, where h is the new height, r is the height at which you wish the railing to be, and f is the floor height.
6. You should now have a transparent railing.

One huge problem with this method is that the space above the railing causes some pretty nasty smearing at odd angles. We're working on a way to solve this problem. Until then, hang in there.