Ozmoo

Ozmoo

A Z-machine interpreter for the Commodore 64 and similar computers

Release 14.56: 6 August 2025

Ozmoo was conceived and designed by Johan Berntsson and Fredrik Ramsberg. Special thanks to howtophil, Retrofan, Paul van der Laan, Jason Compton, Alessandro Morgantini, Thomas Bøvith, Eric Sherman, Paul Gardner-Stephen, Steve Flintham, Bart van Leeuwen, Karol Stasiak and Dennis Holmes for testing, code contributions and bug fixes.

Contents

Overview

Ozmoo is a a redistributable interpreter of Z-code games - Infocom games and games written in Inform, ZIL or Dialog. Ozmoo can be used for new interactive fiction works on the Commodore 64, Commodore 128, Commodore Plus/4, Commander X16 and MEGA65. There is also a fork of Ozmoo for the Acorn computers (BBC Micro and other variants). While the old Infocom interpreters are still available, the license situation is not clear so it is risky to use in new work, especially commercial. Furthermore, some of the newer Inform-based games use features which the old Infocom interpreters can't handle. Ozmoo is written to provide a free alternative that doesn't have these risks and limitations.

Features

Ozmoo supports:

- Z-code version 1, 2, 3, 4, 5, 7 and 8. Essentially this covers all games except for the Infocom games with graphics.
- Fitting a lot more text on screen than Infocom's interpreters - This is done by using all 40 columns, smart wordwrap and a MORE prompt which uses a single character.
- Embedding a custom font. Currently two fonts are included in the distribution, plus some versions for Swedish, Danish, German, Italian, Spanish and French. And you can supply your own font.
- Custom alphabets in Z-machine version 5, 7 and 8.
- Custom character mappings, allowing for games using accented characters. Comes with predefined mappings for Swedish, Danish, German, Italian, Spanish and French.
- Custom colour schemes.
- A fully configurable secondary colour scheme (darkmode) which the player can toggle by pressing the F1 key.
- A configurable splash screen which is shown just before the game starts.
- Up to ten save slots on a save disk.
- Writing a name for each saves position.
- Building a game for play on systems with dual 1541 or 1571 drives, which

allows for larger games. Even games that could be played on a single drive can benefit, as using two drives means the read heads need to move less.

- Building a Z-code game without virtual memory (C64 and Plus/4 only). This means the whole game must fit in RAM at once, imposing a size restriction of about 50-52 KB. A game built this way can then be played on a C64 without a diskdrive. This far, save/restore does require a diskdrive, but there may be a version with save/restore to tape in the future. Also, a game built in this mode doesn't support RESTART.
- Building a game as a d81 disk image. This means there is room for any size of game on a single disk. A d81 disk image can be used to create a disk for a 1581 drive or it can be used with an SD2IEC device or, of course, an emulator. Ozmoo uses the 1581 disk format's partitioning mechanism to protect the game data from being overwritten, which means you can safely use the game disk for game saves as well, thus eliminating the need for disk swapping when saving/restoring.
- Using an REU (Ram Expansion Unit) for caching. The REU can also be used to play a game built for a dual disk drive system with just one drive.
- Adding a loader which shows an image while the game loads (C64, Plus/4 and MEGA65 only).
- Undo support (requires an REU on C64. See separate chapter on Undo for details).

Limitations

Ozmoo should be able to run most Z-code games, regardless of size (A Z-code game can be up to 512 KB in size). However, there are some limitations:

- A Z-code file always starts with a section called dynamic memory. Ozmoo on the Commodore 64 can't handle games with more than roughly 35 KB of dynamic memory.
- If you want to run Ozmoo on a system with a single 1541 drive (or an emulation of one), the part of the game file that is not dynamic memory can be no larger than 170 KB. This typically means the game file can be about 190 KB in size.
- Some Inform 6 games and pretty much all Inform 7 games are too slow to be much fun on a Commodore 64. In general Infocom games, PunyInform games and modern-day ZIL games work the best. Inform 5 games and early Inform 6 games (typically using library 6/1 or 6/2) often work well too.

Quickstart

The simplest option is to use Ozmoo Online, a web page where you can build games with Ozmoo without installing anything on your computer. It supports most of the options Ozmoo has. Ozmoo online is located at: http://ozmoo.online

The other option is to install Ozmoo on your computer. This can be done on Windows, Linux and Mac OS X. To build a game, you run something like "ruby make.rb game.z5" Add -s to make the game start in Vice when it has been built.

Dependencies

You need to install:

- Acme cross-assembler
- Exomizer file compression program (tested with 3.0.0, 3.0.1 and 3.0.2)
- Ruby (Tested with 2.4.2 and 3.3.5, but any version in between should work fine)
- The Vice emulator to test C64, C128 and Plus/4 builds on virtual hardware
- The xemu-xmega65 emulator if you want to test MEGA65 builds on virtual hardware
- A zip program if you want to build games for X16
- The Commander X16 emulator if you want to test X16 builds on virtual hardware

Windows

Acme can be downloaded from SourceForge: https://sourceforge.net/projects/acme-crossass/

Exomizer can be downloaded from Bitbucket. The download includes binaries for Windows: https://bitbucket.org/magli143/exomizer/wiki/browse/downloads

Get WinVice from SourceForge: http://vice-emu.sourceforge.net/windows.html

You can get Ruby from RubyInstaller: https://rubyinstaller.org/

Download the MEGA65 emulator from https://github.lgb.hu/xemu/, and read the instructions on setting it up at https://github-wiki-see.page/m/lgblgblgb/xemu/wiki/MEGA65-quickstart

Get 7-Zip from https://www.7-zip.org/

The Commander X16 emulator is available at https://github.com/X16Community/x16-emulator

Linux

Acme is available on Debian/Ubuntu with:

```
> sudo apt install acme
```

Exomizer can be downloaded from Bitbucket and compiled:

```
> cd src
> make
```

Vice is available on Debian/Ubuntu with:

```
> sudo apt install vice
```

Note that you have to supply the ROM images (kernal, basic, chargen, dos1541) under /usr/lib/vice to make x64 (the C64 emulator) run. See VICE instructions for more details.

Ruby is available on Debian/Ubuntu with:

```
> sudo apt install ruby
```

Download the MEGA65 emulator from https://github.lgb.hu/xemu/, and read the instructions on setting it up at https://github-wiki-see.page/m/lgblgblgb/xemu/wiki/MEGA65-quickstart

The zip program that ships with Linux is all you need for zipping Ozmoo games for X16.

The Commander X16 emulator is available at https://github.com/X16Community/x16-emulator

Customizing the make script

Edit the file make.rb. At the top of the file, you need to specify paths to the Acme assembler, Exomizer, the Vice C64 emulator, and the program "c1541" which is also included in the Vice distribution. If you are using Windows, you can ignore the section on Linux and vice versa. Another option is to create a .ozmoorc file, see the following section.

Creating a .ozmoorc File

If you sometimes update Ozmoo to a new version, you may grow tired of updating the paths to different programs in make.rb. What you can do instead is create a file called ".ozmoorc" where you specify the paths you'd otherwise need to edit. make.rb will look for such a file in three locations, in this order: * the folder specified by the environment variable OZMOO_HOME, if any * current working directory (cwd) * HOME directory (on Windows, this is typically something like "C:\Users\MyName").

The first file found is the only one used.

The file can contain any number of lines. Each line consists of a path identifier, equal character + greater than character ("=>"), and the path value. E.g. to set a path to your local copy of X16emu, you look at the beginning of make.rb and find that the identifier for this path is "X16", and so you might put this in the ".ozmoorc" file:

```
X16  =>  C:\MyEmulators\x16emu\x16emu.exe
```

View all commandline options for make.rb

At a command prompt, type "ruby make.rb"

The basic way to build a game

At a command prompt, type "ruby make.rb mygame.z5"

Build a game which will only consist of a single file

At a command prompt, type "ruby make.rb -P mygame.z5" to build a game which will only consist of a single file. A game created in this way does not require a disk drive to play.

Build a game with optimized preloaded virtual memory data

Ozmoo has the option of optimizing which virtual memory blocks are loaded when the game starts. This is done to make the game as fast as possible in the beginning.

Typically, this is a step you want to do when you have the release version of the game. If you do this optimization and then change the story file, adding or removing more than a few bytes, you should redo the optimization.

Use these steps:

1. At a command prompt, type `ruby make.rb -o -s mygame.z5`. If you plan to release the game for the Commodore 128, add -t:c128 to this command. The text file produced in step 4 can then be used for all other Ozmoo platforms as well in step 5.
2. Play the game, performing the actions you think the player is likely to do first. Keep playing until the game halts, printing a report with lots of numbers. If you have done a reasonable amount of moves (let's say 10-20 moves) and you don't feel like spending more time on this, type `xxx` to end the optimization session and print the results.
3. Vice should now show a lot of hexadecimal numbers (and maybe some game text) on screen. In Vice, select `Edit -> Copy` from the menu to copy all of the screen contents.
4. Create a new text file (let's say you call it `mygame_optimization.txt`), paste the complete text you just copied from Vice into the file and save it.
5. At a command prompt, type `ruby make.rb -c mygame_optimization.txt mygame.z5`. You can use `-cf` instead of `-c` to have Ozmoo fill up any free slots of RAM with more virtual memory blocks. This will pick the lowest blocks which haven't already been loaded.

Repeat step 5 for all platforms you want to build the game for.

Build a game in Benchmark Mode

```
ruby make.rb -bm hollywood_hijinx.z3
```

In this mode, Ozmoo loads a walkthrough for the game from the file benchmarks.json. When launched, the interpreter will play through the game automatically. The pseudo-random-number-generator gets seeded so the same walkthrough will work on every playthrough. On the first and last move, the interpreter prints the number of jiffies (1/60th seconds) elapsed since the computer was powered on, according to the system clock.

This functionality can be used to measure performance gains when tweaking the interpreter code, or to just check that some select games can still be played through from start to finish.

Open the file benchmarks.json in a text editor to see the title, release and serial numbers for the games that the walkthroughs are for. If the serial and release for the current walkthrough don't match one of the walkthroughs in the json file, an error message is printed when using -bm to build Ozmoo.

Targets

Ozmoo was originally written for the Commodore 64, but has been adapted for some other computers as well. make.rb takes a -t:target argument to build for other computers, and currently supports these platforms:

Target	Comment
-t:c64	Build Ozmoo for the Commodore 64 (default)
-t:c128	Build Ozmoo for the Commodore 128
-t:plus4	Build Ozmoo for the Commodore Plus/4
-t:mega65	Build Ozmoo for the MEGA65
-t:x16	Build Ozmoo for the Commander X16

Note that not all build options are supported for every platform. If an option isn't supported, the make.rb script will stop with an appropriate error message, and no Ozmoo files will be produced.

Commodore 64

The Commodore 64 version is the default build target, and supports all build options. A game can have about 35 KB of dynamic memory. Games will need to do more disk access the more dynamic memory they have, so more than about 30 KB may not be advisable. An REU can be used for caching if present. A loader image can be used, see Loader image.

Commodore 128

The Commodore 128 version automatically detects if it is started from 40 or 80 columns mode, and adjusts to the screen size. When run in 80 column mode, the CPU runs at 2 MHz, making for quite responsive games. It makes use of the additional ram available compared to the Commodore 64 version, and allows for games with up to 44 KB dynamic memory. An REU can be used for caching if present.

The Commodore 128 version does not allow a loader image, and build mode -P is not supported. The default build mode for Commodore 128 is -71. For large z8 games, mode -71D is also available (requiring dual 1571 drives).

Commodore Plus/4

The Commodore Plus/4 version makes use of the simplified memory map compared to the Commodore 64 version, allowing for games with up to 46 KB dynamic memory. Games will need to do more disk access the more dynamic memory they have, so more than about 30 KB may still not be advisable. A loader image can be used, see Loader image.

MEGA65

The MEGA65 version is very similar to the C64 version of Ozmoo. It runs in C64 mode on the MEGA65, but uses the 80 column screen mode, extended sound support, higher clockspeed, and the extra RAM of the MEGA65. There is no limitation on dynamic memory size. The only supported build mode is -81. Undo is enabled by default for games that support it. A loader image can be used, see Loader image.

Commander X16

The Commander X16 version is using the extended RAM fully to preload the story file by default. It also adapts automatically to the screen resolution used when starting the game. Undo is supported. Unlike the other platforms, scrollback buffer is currently not supported on the X16. The only supported build mode is ZIP.

Other targets

A fork of Ozmoo targeting the Acorn computers (BBC Micro and other variants) can be found at https://github.com/ZornsLemma/ozmoo/tree/acorn. Note that this fork is using a different build script called make-acorn.py.

Build Modes

Drives and devices

A game built using Ozmoo is placed on one or more disks. These disks can then be used in different disk drives attached to the C64. The device numbers which can be used are 8, 9, 10, 11. If the game has two story disks (meaning it was built using mode D2 or D3), the player will need a computer with at least two disk drives OR one disk drive and an REU to play it.

File name

The story is started by loading and running a boot file which is called "STORY" by default. It is possible to change this file name by using -fn. For example,

```
make.rb -fn temple temple.z5
```

Make sure that the -fn argument follows the naming for the drive you are creating disks for.

List of build modes

Notes:

- Preloading means some or all of memory is filled with suitable parts of the story file, by loading this content from a file as the game starts. Using preloading speeds up game start for many players since this initial loading sequence can use any fastloader the user may have enabled. It also means gameplay is as fast as it gets, right from the start.
- Less RAM available for virtual memory system: This means a smaller part of C64 memory can be used for virtual memory handling, which means the game will need to load sectors from disk more often. This will of course slow the game down.

Modes not requiring a disk drive for play:

P: *Program file*

- Story file size < ~51 KB: Using full amount of RAM.

Disks used:

- Boot / Story disk. This contains a single file, which may be moved to any other medium, like another disk image or a tape image.

Modes requiring a single 1541 drive for play:

S1: *Single 1541 drive, one disk*

- Story file size < ~150 KB: Full preloading. Full amount of RAM available for virtual memory system.
- Story file size < ~170 KB: Less preloading the larger the story file. Full amount of RAM available for virtual memory system.

Disks used: - Boot / Story disk

S2: *Single 1541 drive, two disks*

- Story file size < ~190 KB: Full preloading. Full amount of RAM available for virtual memory system.

Disks used:

- Boot disk
- Story disk

Modes requiring two 1541 drives for play:

D2: *Double 1541 drives, two disks*

- Story file size < ~330 KB: Full preloading. Full amount of RAM available for virtual memory system.
- Story file size < ~360 KB: Less preloading the larger the story file. Full amount of RAM available for virtual memory system.

Disks used:

- Boot disk / Story disk 1
- Story disk 2

D3: *Double 1541 drives, three disks*

- Story file size < ~370 KB: Full preloading. Full amount of RAM available for virtual memory system.

Disks used:

- Boot disk

- Story disk 1
- Story disk 2

Modes requiring a 1571 drive for play:

71: *Single 1571 drive, one disk*

- Story file size < ~320 KB: Full preloading. Full amount of RAM available for virtual memory system.
- Story file size < ~340 KB: Less preloading the larger the story file. Full amount of RAM available for virtual memory system.

Disks used:

- Boot / Story disk

Modes requiring two 1571 drives for play:

71D: *Double 1571 drives, two disks*

Any story size: Full preloading. Full amount of RAM available for virtual memory system.

Disks used:

- Boot disk / Story disk 1
- Story disk 2

Modes requiring a 1581 drive for play:

81: *Single 1581 drive, one disk*

Any story size: Full preloading. Full amount of RAM available for virtual memory system.

Thanks to the partitioning available on the 1581, the story data is protected even in the event of a validate command. Thus, the user can safely use the story disk as a save disk as well.

Disks used:

- Boot / Story disk

Archive Modes

ZIP: *Compressed archive with game and story data*

Any story size: Full preloading. Full amount of RAM available for virtual memory system.

Creates a ZIP archive containing a folder with two files, the game executable and the zcode story file. This is currently only used for X16 targets.

Splash Screen

By default, Ozmoo will show a splash screen just before the game starts. At the bottom of the screen is a line of text stating the version of Ozmoo used and instructions to use F1 to toggle darkmode. After three seconds, or when the player presses a key, the game starts.

You can use the following commandline parameters add up to four lines of text to the splash screen:

```
-ss1:"text"
-ss2:"text"
-ss3:"text"
-ss4:"text"
```

```
-sw:nnn
```

This sets the number of seconds that Ozmoo will pause on the splash screen. The default is three seconds if no text has been added, and ten seconds if text has been added. A value of 0 will remove the splashscreen completely.

Example:

```
ruby make.rb supermm.z5 -ss1:"Super Mario Murders" -ss2:"A coin-op mystery" \
    -ss3:"by" -ss4:"John \"Popeye\" Johnsson" -sw:8
```

Colours

Ozmoo lets you pick two different colour schemes for your game. We refer to these two colour schemes as normal mode and darkmode. The idea is that you may want lighter text on a dark background when playing at night, while dark text on a light background has proven to be easier to read, in well-lit conditions. Ozmoo will always start in normal mode, and the player can switch between normal mode and darkmode using the F1 key. When switching modes, Ozmoo will change the colour of all onscreen text which has the default foreground colour *or* which has the same colour as the background colour in the mode it's switching to and thus would otherwise become invisible.

A word of caution

The C64 has severe problems showing certain colours next to each other, e.g. brown text on blue background is typically impossible to read. As a rule of thumb, when two colours are to be next to each other on screen, make sure one of them is black or white, and the other has a reasonably high contrast to the first colour. E.g. Blue text on white background is fine, as is black text on light green background.

Colour switches

make.rb has the following switches to control colours:

`-dm:0`

Disables darkmode. (-dm or -dm:1 can be used to enable it, but it's already enabled by default unless the game is Beyond Zork)

`-fgcol:<colourname>`

Foreground colour: This picks the colour to use as default foreground colour. (Games in z5+ format can change this colour at will)

`-bgcol:<colourname>`

Background colour: This picks the colour to use as default background colour. (Games in z5+ format can change this colour at will)

`-bordercol:<colourname>`

Border colour. This picks the colour to use as border colour. Special colournames: bg = same as background colour (default), fg = same as foreground colour. If the game itself changes the screen colours, as games in z5+ format may do, values bg and fg mean the border changes too.

`-statuscol:<colourname>`

Statusline colour: This picks the colour to use as statusline colour. This is only possible with version 1, 2 and 3 story files (z1/z2/z3).

`-inputcol:<colourname>`

Input colour: This picks the colour to use for player input text. This is only possible with version 1, 2, 3 and 4 story files (z1/z2/z3/z4).

`-cursorcol:<colourname>`

Cursor colour: This picks the colour for the cursor shown when waiting for player input. fg = same as foreground colour (default). If the game itself changes the foreground colour, as games in z5+ format may do, value fg mean the cursor changes too.

```
-dmfgcol: (same as -fgcol but for darkmode)

-dmbgcol: (same as -bgcol but for darkmode)

-dmbordercol: (same as -bordercol but for darkmode)

-dmstatuscol: (same as -statuscol but for darkmode)

-dminputcol: (same as -inputcol but for darkmode)

-dmcursorcol: (same as -cursorcol but for darkmode)
```

Palette

Z-code normally has a palette of eight colours, numbered 2-9:

```
2 = black        (blk, black)
3 = red          (red)
4 = green        (grn, green)
5 = yellow       (yel, yellow)
6 = blue         (blu, blue)
7 = magenta      (magenta, pur, purple)
8 = cyan         (cyn, cyan)
```

```
9 = white        (wht, white)
```

Additionally, Ozmoo provides eight more colours in the palette:

```
16 = orange      (orng, orange)
17 = brown       (brn, brown)
18 = light red   (lred, lightred)
19 = dark grey   (dgry, darkgrey, dgrey, darkgray, dgray)
20 = medium grey (mgry, mediumgrey, mgrey, grey, mediumgray, mgray, gray)
21 = light green (lightgreen, lgreen)
22 = light blue  (lblu, lightblue, lblue)
23 = light grey  (lgry, lightgrey, lgrey, lightgray, lgray)
```

The names in parenthesis are some of the synonyms you can use to refer to the colours on the command line. The short forms of the colours (e.g. blk and mgry) are the names printed on the key caps of the C64 and MEGA65.

These sixteen colours are the colours that are provided by the C64. On other platforms, the same sixteen colours or approximations of these colours are used.

An example of setting colours

Use cyan text on black background, have the border be the same colour as the text, and make the statusbar light grey (Please note that specifying the colour of the statusbar only works for z2/z2/z3 games!):

```
make.rb -fgcol:cyan -bgcol:black -bordercol:fg -statuscol:lightgrey game.z3
```

Cursor switches

The shape and the blinking of the cursor can also be customized:

`-cb:(delay)`

Cursor blinking frequency. delay is 1 to 99, where 1 is fastest.

`-cs:(Cursor shape)`

Cursor shape: either of b,u or l; where b=block (default) shape, u=underscore shape and l=line shape.

Fonts

When building a game with make.rb, you can choose to embed a font (character set) with the game using the -f option. This will use up 2 KB of memory which would otherwise have been available for game data. The font file should be exactly 2048 bytes long and just hold the raw data for the font, without load address or other extra information.

The font files are organized into subfolders under the "font" folder, with one subfolder per language:

da: Danish en: English de: German es: Spanish fr: French it: Italian sv: Swedish

Included with the Ozmoo distribution are these custom fonts:

- Clairsys, by Paul van der Laan.
- Clairsys Bold, by Paul van der Laan.
- PXLfont-rf, by Retrofan.
- System, the standard C64 system font, with accented characters added by the Ozmoo team.

You are free to use one of these fonts in a game you make and distribute, regardless of whether you make any money off of the game. You must however include credits for the font, stating the name of the font and the creator of the font. We strongly suggest you include these credits both in the docs / game distribution and somewhere within the game (Some games print "Type ABOUT for information about the game." or something to that effect as the game starts).

To see all the licensing details for each font, read the corresponding license file in the "fonts" folder. The full information in the license file must also be included with the game distribution if you embed a font with a game.

Sound

While several Infocom games had high and low-pitched beeps, a few games had extended sound support using sample playback. Ozmoo supports the basic sound effects (beeps) on all platforms, and extended sounds on the MEGA65, using the sound_effect opcode.

Sample format

The extended sound support uses sample files stored in the AIFF or WAV formats, with WAV being the recommended format for new games. The WAV files need to be 8 bit, mono. Audacity can be used to export wav files in the correct format:

- Select "File/Export/Export as WAV" from the main menu
- Select "Other compressed files" as the file type
- Select "Unsigned 8-bit PCM" as the encoding
- Save the file

Switches

`-asw path`

Add Sounds: Enable extended sound support and add all .wav files in path

`-asa path`

Add Sounds: Enable extended sound support and add all .aiff files in path

If extended sound is to be used, then make.rb should be called with the `-asw path` (or `-asa path`) switch. If set, then all .wav (or .aiff files) in `path` will be added to the .d81 floppy created for the MEGA65, and the SOUND assembly flag will be set when building Ozmoo.

Since sound effect 1 and 2 are reserved for beeps, the sample based sound effects start from position 3, and the files should be named 003.wav, 004.wav and so on. All files found using this pattern are added, and skips are allowed. For example,

if there is no sound effect 5, then no 005.wav needs to be added, and instead 006.wav is added next, if available. The highest sound effect number that can be used is 255.

Legacy support for Sherlock and The Lurking Horror

Ozmoo includes support for Sherlock and The Lurking Horror from Infocom, with sound effects. While they can't be built with sound support on Ozmoo Online, there is a link there to download them for the MEGA65. Go to https://microheaven.com/ozmooonline/ and search for "sherlock" on the page.

If you have Ozmoo installed on your own computer, you can download a Blorb archive of AIFF versions of the sound files from: https://ifarchive.org/indexes/ifarchive/infocom/media/blorb/

The individual sound files must be extracted from the archive. For blorb files there are various tools, such as rezrov, available: https://ifarchive.org/indexes/ifarchiveXprogrammingXblorb.html

The AIFF files should be moved to a folder that is later included with the -asa switch when using the make.rb script to build the game. Make sure that the filenames follow the pattern described above (starting with 003.aiff).

Example: assuming that the sound files are stored in a folder called "lurking_sounds", this command will build and start the Lurking Horror in the xemu-xmega65 emulator

```
ruby make.rb -s -ch -t:mega65 -asw lurking_sounds lurkinghorror-r221-s870918.z3
```

Loader image

When building for the Commodore 64, the Plus/4, or the MEGA65, it is possible to add a loader which shows an image while the game is loading, using -i (show image) or -if (show image with a flicker effect in the border). The image file must be:

- For C64: a Koala Paint multicolour image (10003 bytes in size)
- For Plus/4: a Multi Botticelli multicolour image (10050 bytes in size)
- For MEGA65: an IFF image in 320x200 resolution, with 1-8 bitplanes. One way to convert an image to this format is to use megascr.sh.

Border flicker is only supported for the C64. Example commands:

```
make.rb -if mountain.kla game.z5
make.rb -i spaceship.mbo -t:plus4 game.z5
make.rb -i city.iff -t:mega65 game.z5
```

Command line history

There is an optional command line history feature that can be activated by -ch. If activated, it uses the wasted space between the interpreter and the virtual memory buffers to store command lines, that can later be retrieved using the cursor up and down keys. The maximum space allowed for the history is 255 bytes, but the stored lines are saved compactly so if only short commands like directions, “i” and “open door” etc are used it will fit quite a lot.

Since memory is limited on old computers this feature is disabled by default, except for MEGA65. To enable it use -ch or -ch:1. This will allocate a history buffer large enough to be useful. It is also possible to manually define the minimal size of the history buffer with -ch:n, where n is 20-255 (bytes). Use -ch:0 to disable it.

Scrollback buffer

Allows the player to press F5 to enter scrollback mode, where they can scroll up and down through the text that has scrolled off the screen. This feature uses an REU if available on C64 or C128, and AtticRAM on MEGA65. Optionally, it can use a smaller portion of RAM on Plus/4 as well as C64 or C128 without REU. Scrollback buffer is enabled by default for MEGA65 only. Enable it with -sb or -sb:1. Disable it with -sb:0. Add a RAM buffer on Plus/4, C64 or C128 with -sb:6|8|10|12.

Undo

This feature allows the game state to be saved to memory each turn, so the player can undo the last turn. It is enabled by default on the MEGA65. It's also available for Commander X16, and C64 and C128 computers that use a RAM Expansion Unit. For C128 only, there is also an option to use undo without requiring an REU, by allocating some of the RAM as an undo buffer, for games with a moderately large dynamic memory.

In addition to the standard undo support for z5+ games (enabling the UNDO command which many games have), Ozmoo adds a keyboard shortcut (Ctrl-U) to enable undo for z1-z4 games.

To build a game with undo support, use option -u or -u:1. To disabled undo support, use -u:0 (for MEGA65, where it's otherwise enabled by default). Use -u:r to enable undo using REU *and* allocate a RAM buffer to use if no REU is detected (C128 only).

Smooth scrolling

This feature adds smooth scrolling support. When active, text is scrolled up one pixel (raster line) per frame rather than an entire character (text row) at a time, providing a "smooth" visual experience.

The build option -smooth can be used to include smooth scrolling support on supported targets (currently only available on the C64 and C128 (and doesn't work on C128 in 80 column mode)).

Smooth scrolling is automatically activated at program startup if the support was included. While playing the game, the user can disable smooth scrolling with Ctrl-0 .. Ctrl-8, and re-enable smooth scrolling with Ctrl-9.

Miscellaneous options

Option -df:n

-df:n provides additional control to the ZIP build mode. ZIP mode creates a folder, adding the game binary and the zcode story file to it, and then compressing the folder into the final zip archive. -df controls what will happen with this temporary folder. The accepted values are 0 (create zip archive and keep the folder), 1 (delete the folder if zip archive created successfully) and f ('force'/always delete the folder). The default is 0.

Option -sp:n

-sp:n is used to set the size of the Z-machine stack, in pages (1 page = 256 bytes). The default value is 4. Many games, especially ones from Infocom, can be run with just two pages of stack. The main reason for reducing this to two pages would be to squeeze in a slightly bigger game in build mode P, or to build a game where dynamic memory is slightly too big with the standard settings.

To run an Inform 7 game (which may be feasible on the MEGA65), you want to set the number of stack pages to its highest setting: 64.

Option -cm:[xx]

Ozmoo has some support for using accented characters in games. This is documented in detail in the tech report, but assuming that suitable fonts and character maps have been prepared, the -cm option is used to enable this character map. By default, Ozmoo has support for these character maps: sv, da, de, it, es and fr, for Swedish, Danish, German, Italian, Spanish and French, respectivily.

Option -um:[:0|1]

This enables or disables the default unicode table mapping. I.e. if a game tries to print one of the 69 characters that are in the default Z-code unicode table, and the character has not been remapped to a special character using -cm, Ozmoo maps it to the closest single-character approximation, and prints that instead. E.g. "ä" is printed as "a". This feature is enabled by default. Disabling it saves 83 bytes.

Option -in:[n]

-in sets the interpreter number.

The interpreter numbers, originally defined by Infocom, are as follows:

```
1 = DECSystem-20
2 = Apple IIe
3 = Macintosh
4 = Amiga
5 = Atari ST
6 = IBM PC
7 = Commodore 128
8 = Commodore 64
9 = Apple IIc
10 = Apple IIgs
11 = Tandy Color
```

The interpreter number is used by a few games to modify the screen output format. In Ozmoo we set 2 for Beyond Zork, 7 for C128 builds, and 8 for other games by default, but -in allows you to try other interpreter numbers.

Option -rb[:0|1]

-rb or -rb:1 enables REU Boost, while -rb:0 disables it. REU Boost adds ~160 bytes to the interpreter size.

Option -re[:0|1]

-re or -re:1 enables extended runtime error checks, while -re:0 disables them. They are enabled by default on MEGA65 only.

Option -sl[:0|1]

-sl or -sl:1 enables slow mode, while -sl:0 disables it. This has an effect on builds for C64 only, and not in -P build mode. Slow mode removes some optimizations

for speed, making the interpreter slightly smaller.

Option -x[:0|1]

Auto-replace X with EXAMINE. Default is to enable this for Infocom games that need it only.