Description of the Program

This program is a mips clone of the number tower game found at <http://www.childrenspuzzles.net/number-tower.php>.

It includes an optional graphical display using a modified version of mars. It also contains sound effects and royalty-free music from bensound.com. It includes a separate java utility to generate the initial puzzle bank.

All subroutines follow standard register preservation convention.

Challenges Faced by Individual and Team and how they were overcome

Saran: I had to figure out everything related to the bonus credits such as Graphics and Sound. This was extremely difficult since there was no documentation for MARS anywhere online so I spent a week really just figuring out how to decompile the source code and recompile it in and IDE and then spent days browsing over the source code trying to reverse engineer how the MARS developer simulated MIPS. Eventually patience and trial and error won out and I got graphics and music working with custom annotations placed in the mips code.

What I have learned

Saran: I learned how to take apart a Jar and put it back together as well as how complicated it is to simulate an architecture just by looking through the hundreds of source files in java required to run MIPS.

Discussion of Algorithms and Techniques used in the Program

Input:

Input into the program is all text based. Letter coordinates are parsed via a custom function. Numbers are translated to integers via a function based on C’s atoi. This method of using only string input syscalls allows the program to be super stable, as the integer input syscalls would otherwise kill the program when letters are entered.

Entering letters when the program expects numbers, or numbers when letters are expected, will not crash the program, and instead only prompt for a new input.

Puzzle Creation:

The game essentially stores hundreds of puzzles generated in a high level language, in this case Java. Every puzzle gets a randomized bottom row which is then solved all the way up to create the pyramid. We store a copy of the final puzzle and then a copy of the puzzle where we randomly select pieces to be removed in such a way that it’s still solvable. Both solved and partially solved version of the pyramids are stored directly into a “source.txt” file, from which the program will read.

Validation:

The program stores two copies of every puzzle. One copy is the working copy, and the other is the ‘knowns.’ Once input numbers are parsed, they are compared against the knowns. Failed matches are rejected. This allows for a simple and effective method of validation.

Win Detection:

Win detection is done by comparing the working copy and known copy of the puzzle. This allows for extremely robust and simple win detection.

Ascii Display:

The ascii text display is not generated for each run. Instead, a premade piece of ascii art is stored. As numbers are entered, that location’s position in the ascii art is looked up from a table, and that part of the art is overwritten using a custom int2ascii function.

BONUS EXTRA CREDIT

GUI Display/Mars modification:

To get a custom extension working in Mars first I had to decompile the Mars JAR file to get the source code. This process took a few days to get it in a format that was acceptable by an IDE such as IntelliJ, so I could run and debug the code. After I got Mars running from IntelliJ I quickly realized it was not packaged the same way as if created from the .bat file that was found in the code. So to fix the packaging issues I had to write a shell script to call the .bat file to compile, create, and run the new MIPS Jar that had any modified code such as the extension we made for graphics. Getting an extension to appear in the menu itself wasn’t too hard, but getting it to interact with the register took a few days of understanding how Mars simulated MIPS code and just honestly reading all the source code since there was no documentation available online. After days of reverse engineering Mars I got the values properly loaded at the right time intervals from MIPS to Java which I then used to interact with my GUI Pyramid. I used the annotation principle so to interact with the graphics API you just have to use any random MIPS command and place # @DRAW at the end. For example you can push the pyramid value like 220 to $a0 then the index location from 0-27 going from top of pyramid to bottom left to right would be pushed to $a1. So it would look like:

move $a0, $s1 # s1 contains the pyramid value like 220

move $a1, $s2 # s2 contains the pyramid block position 0 being top 27 being bottom right

li $s1, 0 # @DRAW this can be any random mips command, tells extension to update GUI.

Sound:

There are two portions to the sound design, assembly based sounds for short sound effects and audio cues, and java based sounds for playing background music over the duration of the game.

1. Assembly based - The MARS simulator provides syscalls 31 and 33 to play a single note. Syscall 31 returns immediately after the beginning of the sound whereas syscall 33 effectively stalls the program for the duration of the sound. A simple subroutine was designed to implement 33, which requires $a0 - $a3 to all be given values. The subroutine expects the first two, pitch and duration, to be provided. The second two, instrument and volume, are set to specific values within the subroutine, as customization of those two characteristics was not necessary. The subroutine can be, and is, called back to back to produce the effects desired.
2. Java based - The background game music was implemented with a custom Sound.java class that allows the .wav game music file obtained copyright free from bensound.com to play in the background as the game runs. It starts playing with the annotation # @MUSIC-START attached to any mips command and stops right before the game ends with # @MUSIC-STOP attached to any mips command.

File loading:

A prerequisite for this program is that the “source.txt” file generated by our pyramid generator (written in Java, explained above) must be located in the same directory as the Mars.jar file being used.

Initially, the file must be opened. Syscall 13 allows for the opening of a file (in read mode, for this situation), which returns a file descriptor that identifies the file. If the file fails to open, the program notifies the user and terminates immediately.

Assuming the file opens properly, the program will generate a random number (syscall 42) that determines how many pyramid puzzles to skip within the file. Every puzzle occupies exactly 170 characters, including new line characters. In a loop, we read 170 characters (syscall 14) into an arbitrary buffer for the number of times specified by the random number.

Once the specified random number of puzzles are skipped, the working and known copies of the puzzle are loaded from file. This is performed by loading three characters at a time, and converting them to integers via the atoi function. Unknowns in the working copy (Aka: the blanks to be filled by the player), are encoded as zeros, so no special cases are required.ß