

***Proj\_258: Sokoban***

# **USER GUIDE**

Author: Donghee (Danny) Han

CSC258H5, Fall 2024  
University of Toronto

# TABLE OF CONTENTS

What is Sokoban? ..... 1

Game Setup Instructions ..... 2

How To Play *Proj\_258: Sokoban* ..... 5

Implementation Specs ..... 9

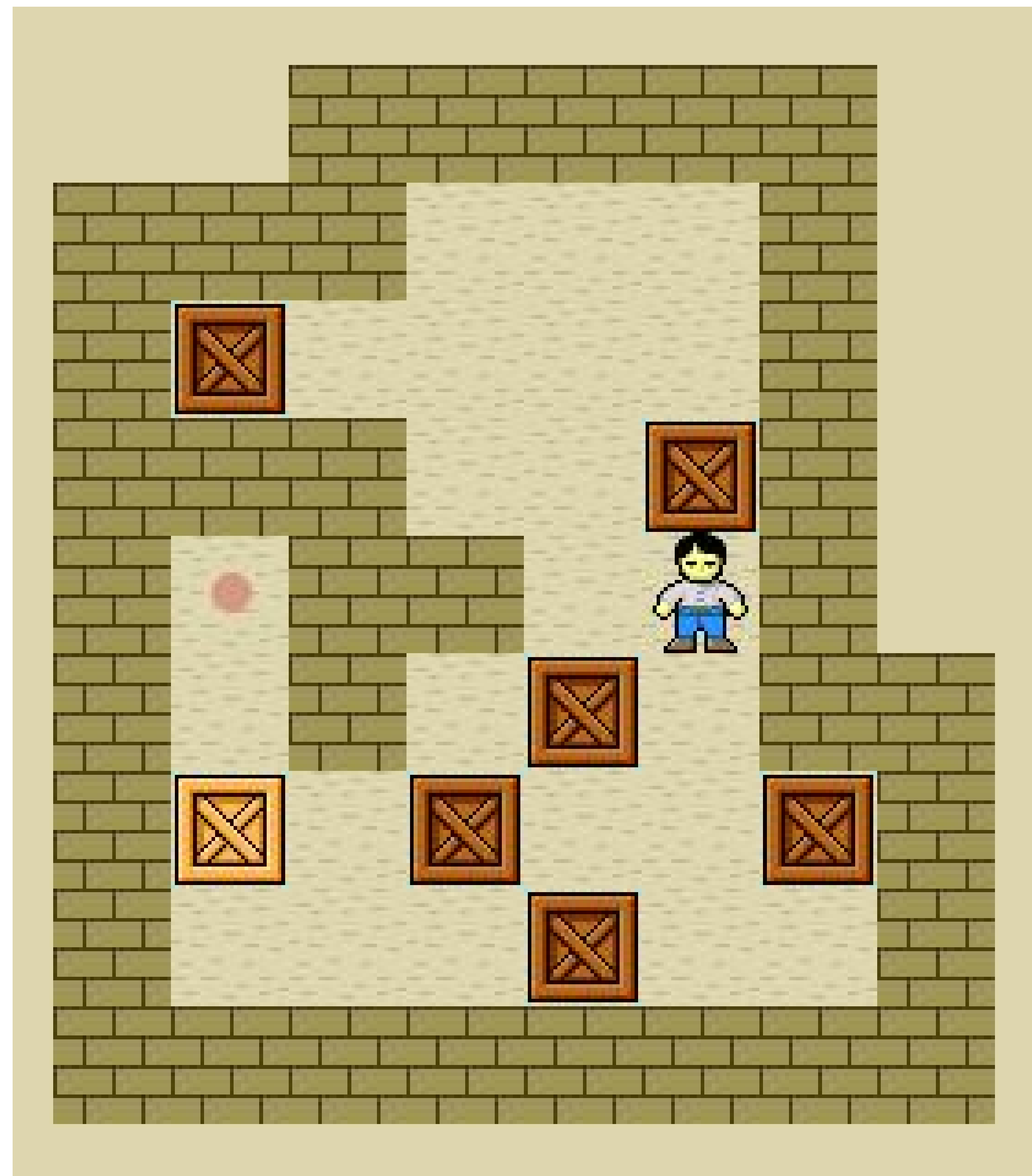
# What is Sokoban?

Sokoban (倉庫番) is a Japanese puzzle game developed in 1981 by Hiroyuki Imabayashi.

Within a randomly generated warehouse, the player character must organize boxes by moving each box to a target position.

The player successfully solves a level by moving every box to a target position on the map.

However, be careful: When any single box becomes stuck in a position from which it cannot be pushed (i.e., re-positioned) by the player character, thereby rendering the level impossible to beat, the user must reset the level and try again from the start.



The next section of this user guide details the technologies needed to play this implementation of Sokoban, Proj\_258: Sokoban.

```
= = = = = = = = = = = = = = = = = = = Boxes Left: 3 / 17
= = = = = = = = = = = = = = = = = = =
= = = = = = 0 0 = = = = = = = = KEYMAP:
= = 0 0 = = = = = = = = [W] Up [R] Reset Current Level
= = 0 0 0 = = = = = = = = [S] Down [G] Generate New Level
= = 0 0 A * * = = [A] Left [Q] Exit Proj_258: Sokoban
= = 0 * = = [D] Right
= = 0 * = =
= = 0 % % = =
= = 0 % = =
= = 0 = =
= = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = =
```

# Game Setup Instructions

## Technologies Needed:

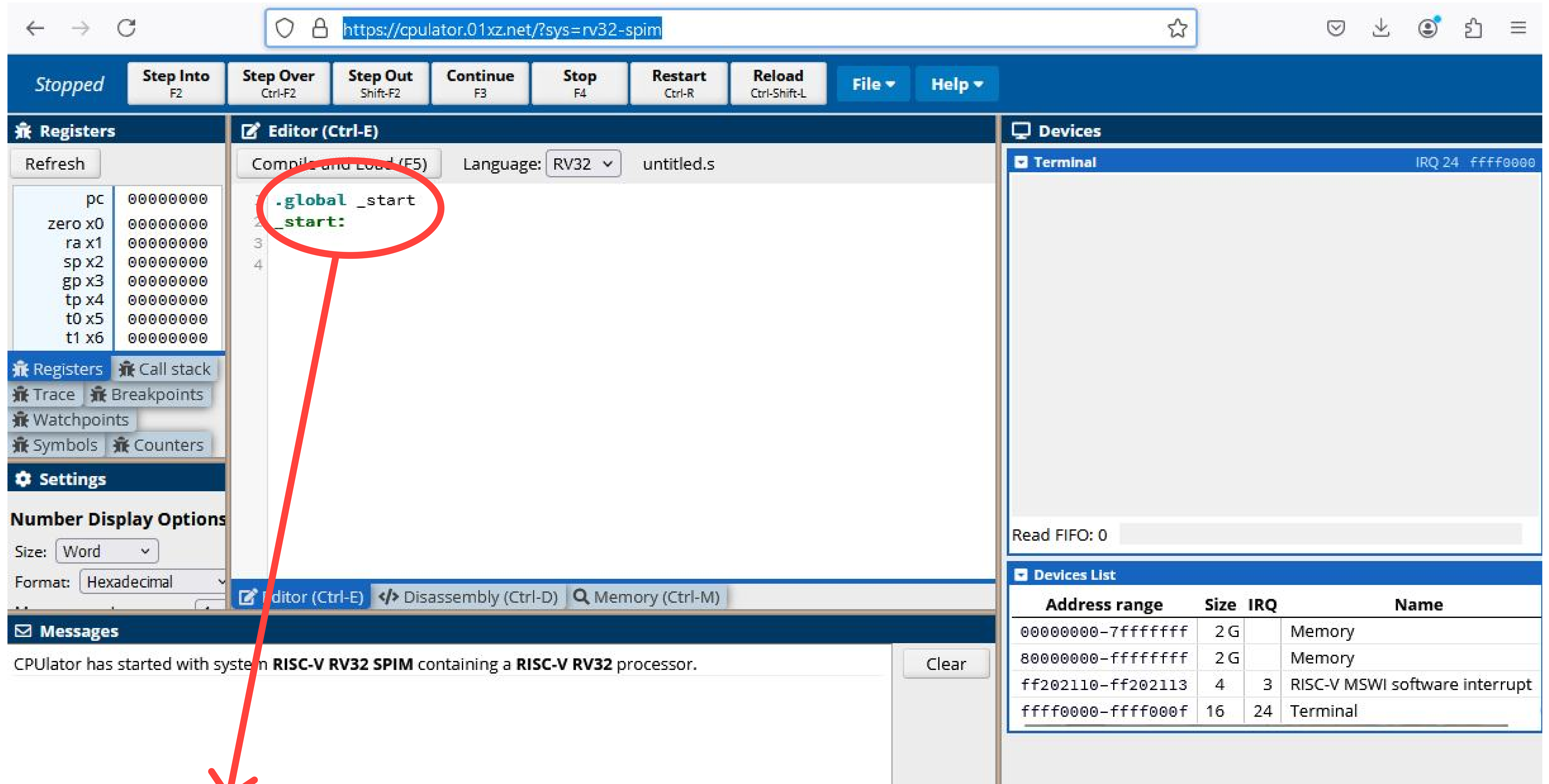
- Internet connection
- Web browser (e.g., Chrome, Firefox, Edge, etc.)
- Assembly file accompanied with this document. This **.s** file is named *CSC258H5\_Proj\_Sokoban\_Danny\_Han\_FINAL\_SUBMISSION-1.s*

## Setup Instructions:

In order to load up the game, open a web browser of your choice. Afterwards, copy and paste the following link into your browser:

<https://cpulator.01xz.net/?sys=rv32-spim>

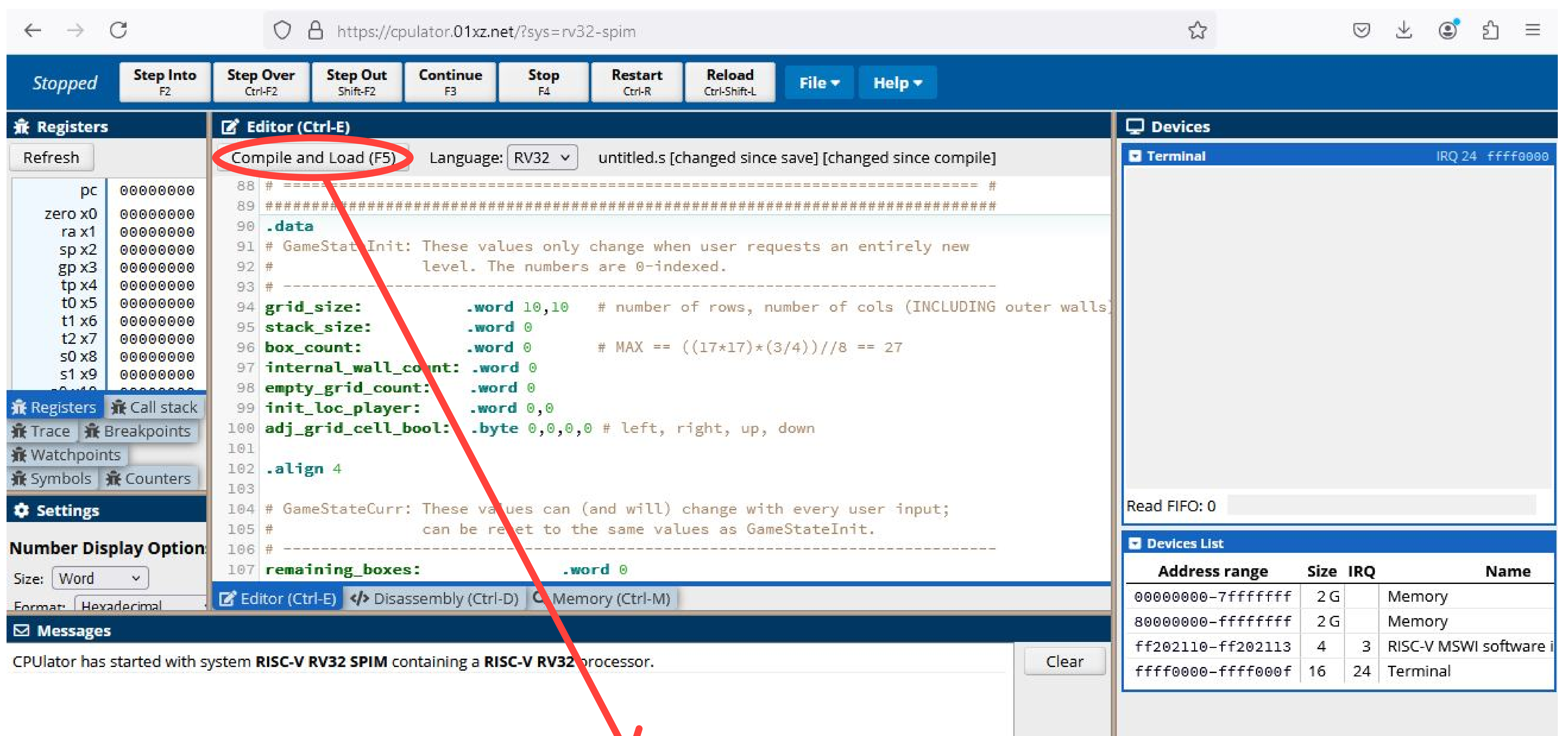
*CPUlator* is the website that will be used to load and play *Proj\_258: Sokoban*. You will be directed to the web page below:



Delete these two lines of code.

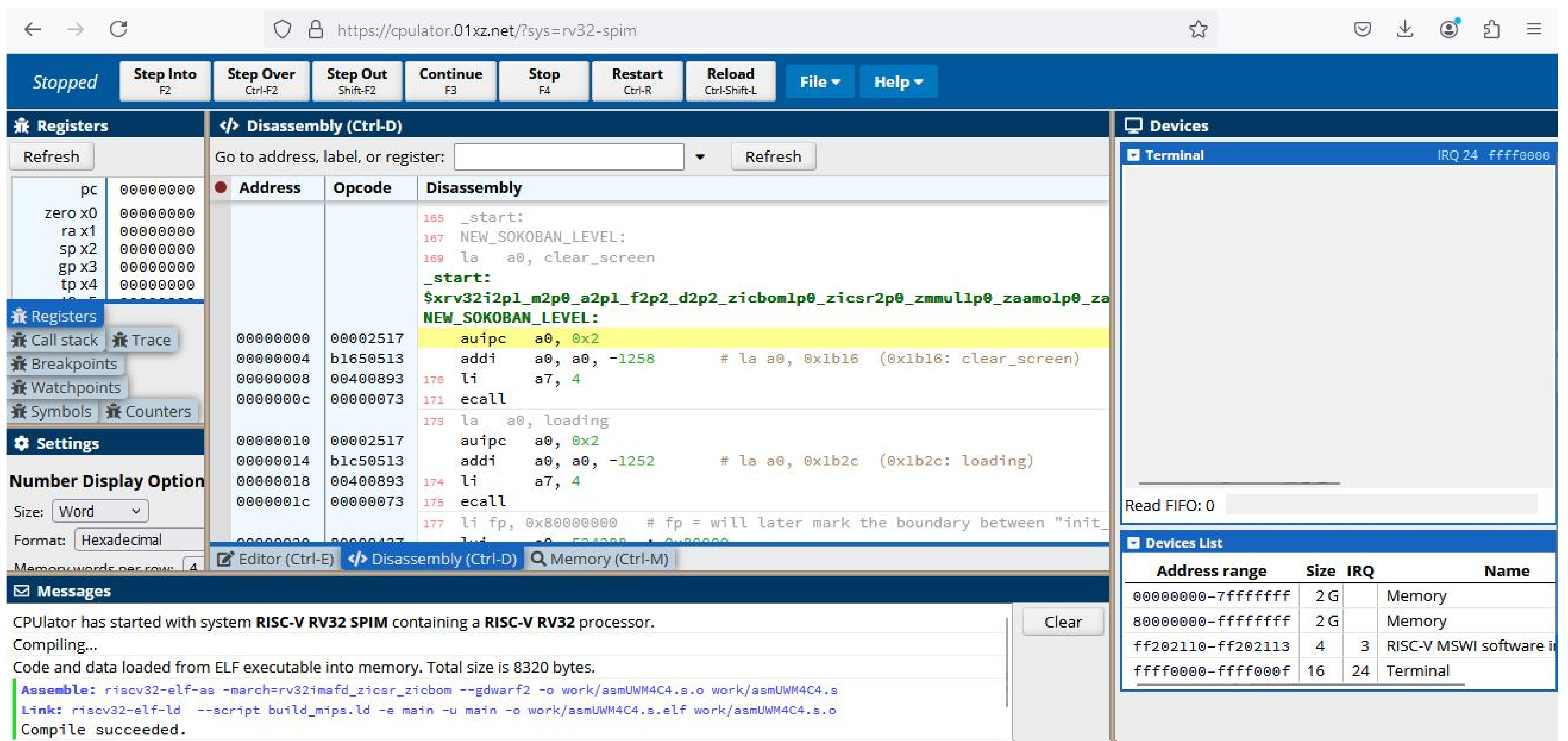


Next, copy and paste all of the content in the assembly file into the “Editor” section of *CPUlator*, as shown below.



The screenshot shows the CPUlator web interface. The top navigation bar includes buttons for 'Step Into', 'Step Over', 'Step Out', 'Continue', 'Stop', 'Restart', and 'Reload'. The left sidebar contains sections for 'Registers', 'Call stack', 'Trace', 'Breakpoints', 'Watchpoints', 'Symbols', 'Counters', 'Settings', 'Number Display Option', and 'Messages'. The main area is the 'Editor (Ctrl-E)' section, which contains assembly code. A red circle highlights the 'Compile and Load (F5)' button, and a red arrow points to it. The assembly code includes comments about GameStateInit and GameStateCurr, and defines variables like grid\_size, stack\_size, box\_count, etc. The right sidebar contains sections for 'Devices' and 'Devices List'.

Finally, click on the **Compile and Load (F5)** button circled above to load the game. Upon success, you will be met with the interface below, and the game is ready for playing!



The screenshot shows the CPUlator web interface after compilation. The top navigation bar is the same. The left sidebar is the same. The main area is the 'Disassembly (Ctrl-D)' section, which displays the assembly code in a table. The table has columns for Address, Opcode, and Disassembly. The code includes instructions like auipc, addi, li, and ecall. The right sidebar contains sections for 'Devices' and 'Devices List'.

Address range	Size	IRQ	Name
00000000-7fffffff	2 G		Memory
80000000-ffffffff	2 G		Memory
ff202110-ff202113	4	3	RISC-V MSWI software i
ffff0000-ffff000f	16	24	Terminal

The screenshot shows the CPUTator web interface at <https://cputator.01xz.net/?sys=rv32-spim>. The interface includes a top navigation bar with buttons like 'Stopped', 'Step Into', 'Step Over', 'Step Out', 'Continue', 'Stop', 'Restart', and 'Reload'. The main area is divided into several panels: 'Registers' on the left, 'Disassembly (Ctrl-D)' in the center, and 'Devices' on the right. The 'Disassembly' panel shows assembly code with addresses, opcodes, and comments. The 'Devices' panel includes a 'Terminal' section and a 'Devices List' table.

Address range	Size	IRQ	Name
00000000-7fffffff	2 G		Memory
80000000-ffffffff	2 G		Memory
ff202110-ff202113	4	3	RISC-V MSWI software interrupt
ffff0000-ffff000f	16	24	Terminal

**NOTE:** This area of the *CPUTator* interface, labelled “Terminal”, will be where the game is displayed *and* played

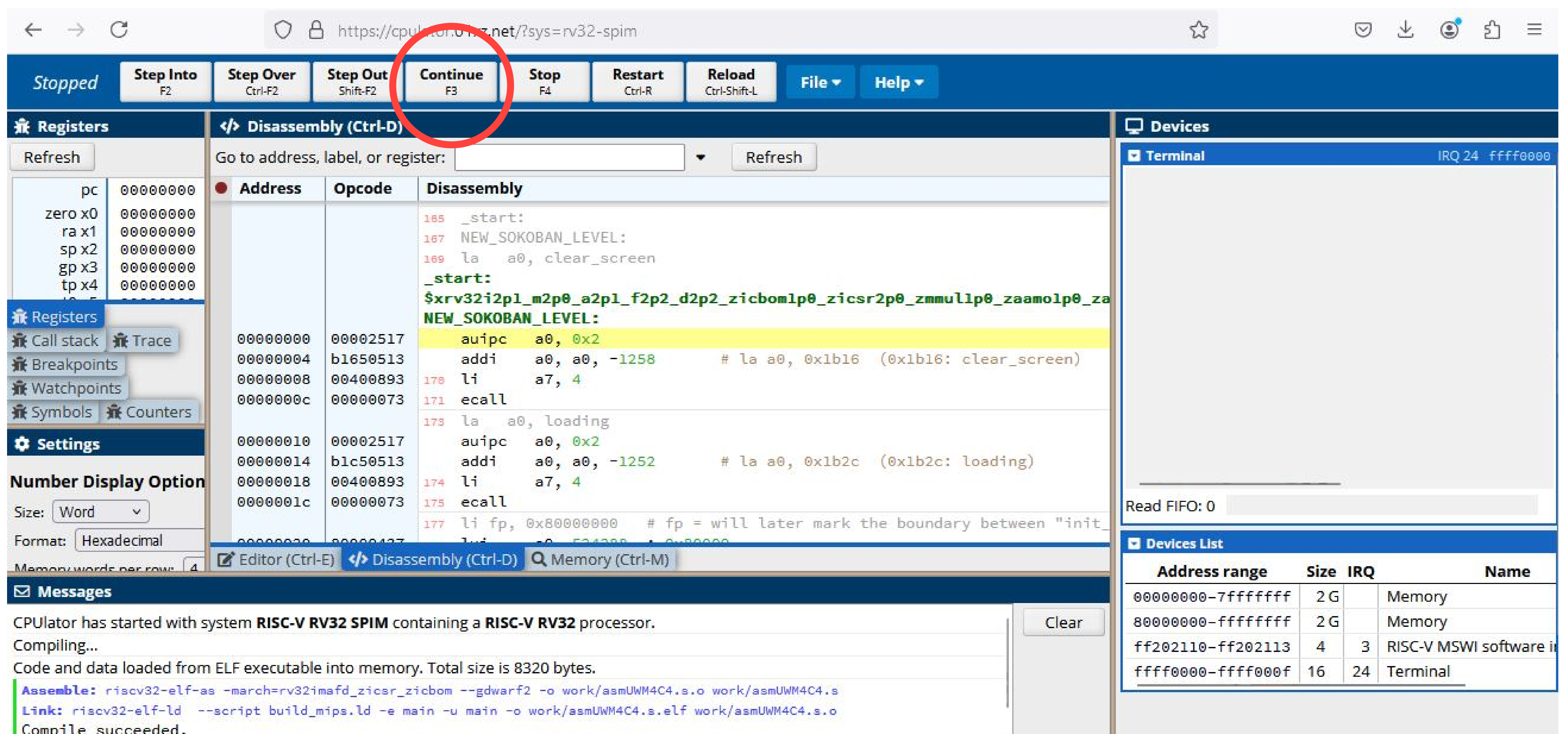
The next section of this user guide details how to (finally) play the game.



# How To Play *Proj\_258: Sokoban*

After following the steps detailed in the previous section of this user guide, press the “Continue” button circled below. This will start the game.

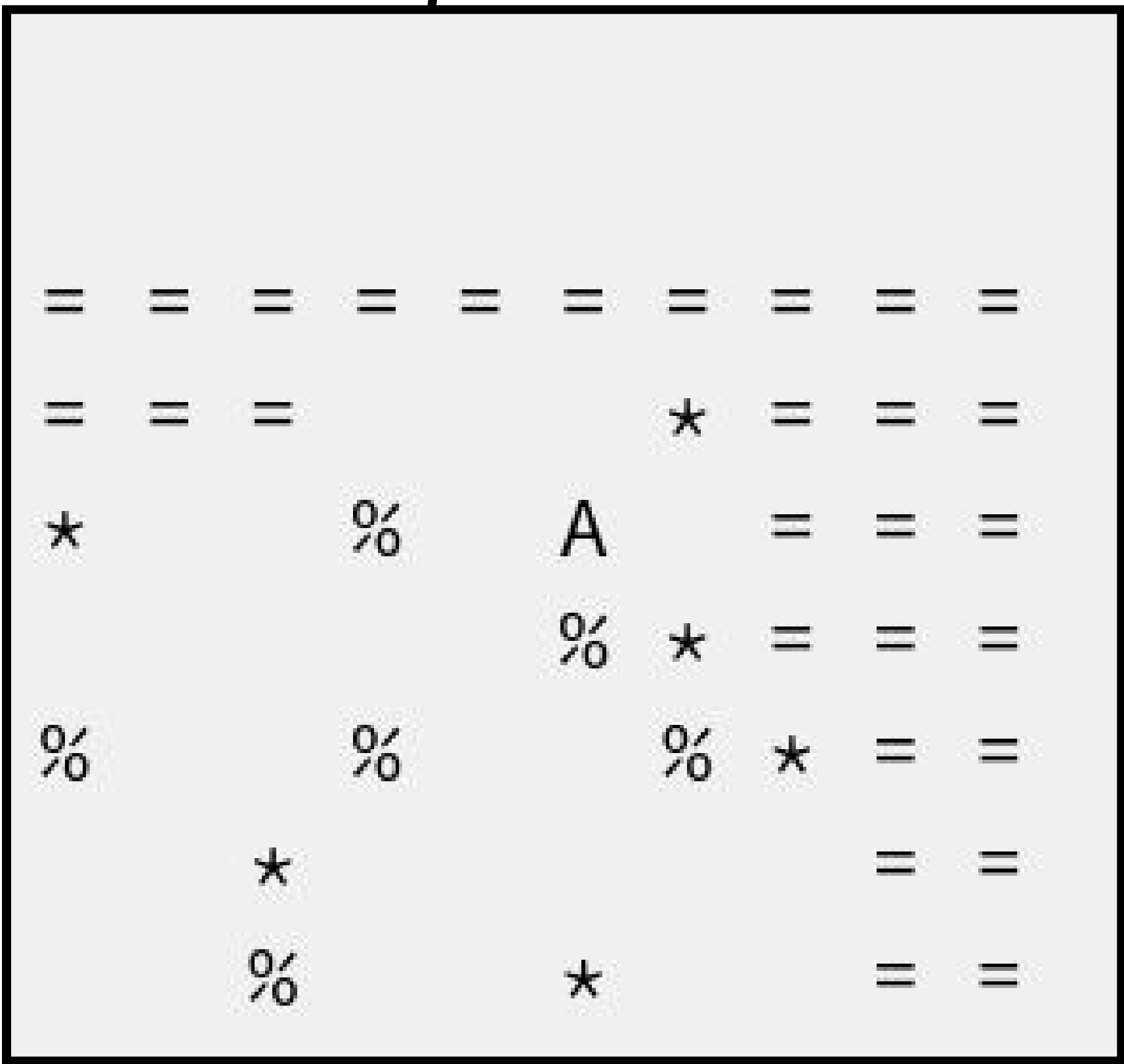
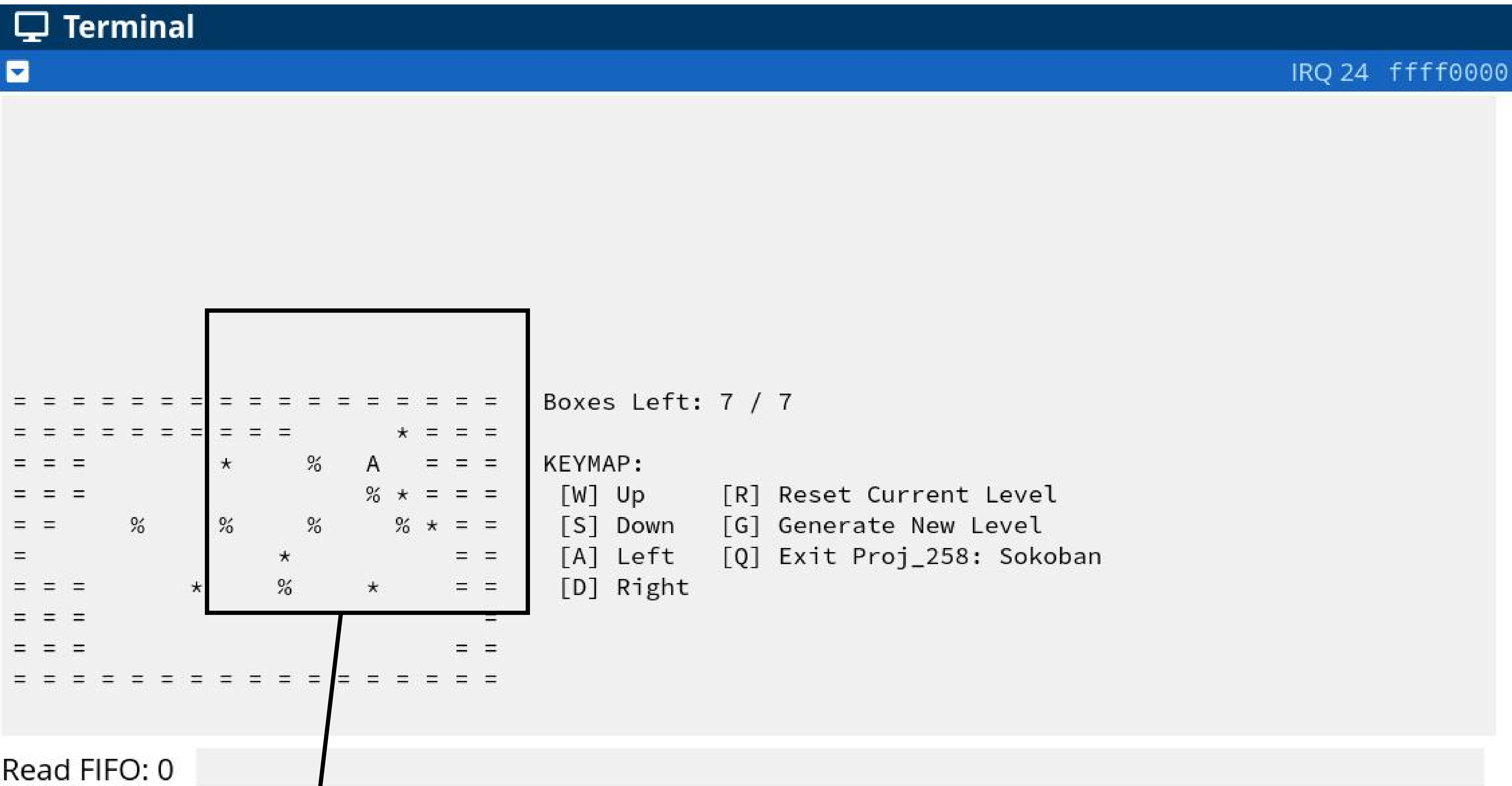
**IMPORTANT:** After starting the game, **be sure to click anywhere within the “Terminal” section of the *CPUlator* interface.** Otherwise, the game cannot take user input.



Once the game starts, it will take anywhere between 5 to 10 seconds to load as shown below.



Below is an example of a randomly generated *Proj\_258: Sokoban* level.



**Map Legend:**

- A : player character
- = : wall
- % : box
- \* : target spot to be filled
- O : target spot that has been filled



Use the direction keys shown on-screen to navigate the player character around the grid to push boxes into target spots. The level is solved when the “*Boxes Left*” count on-screen reads 0.

```

Terminal
IRQ 24 ffff0000

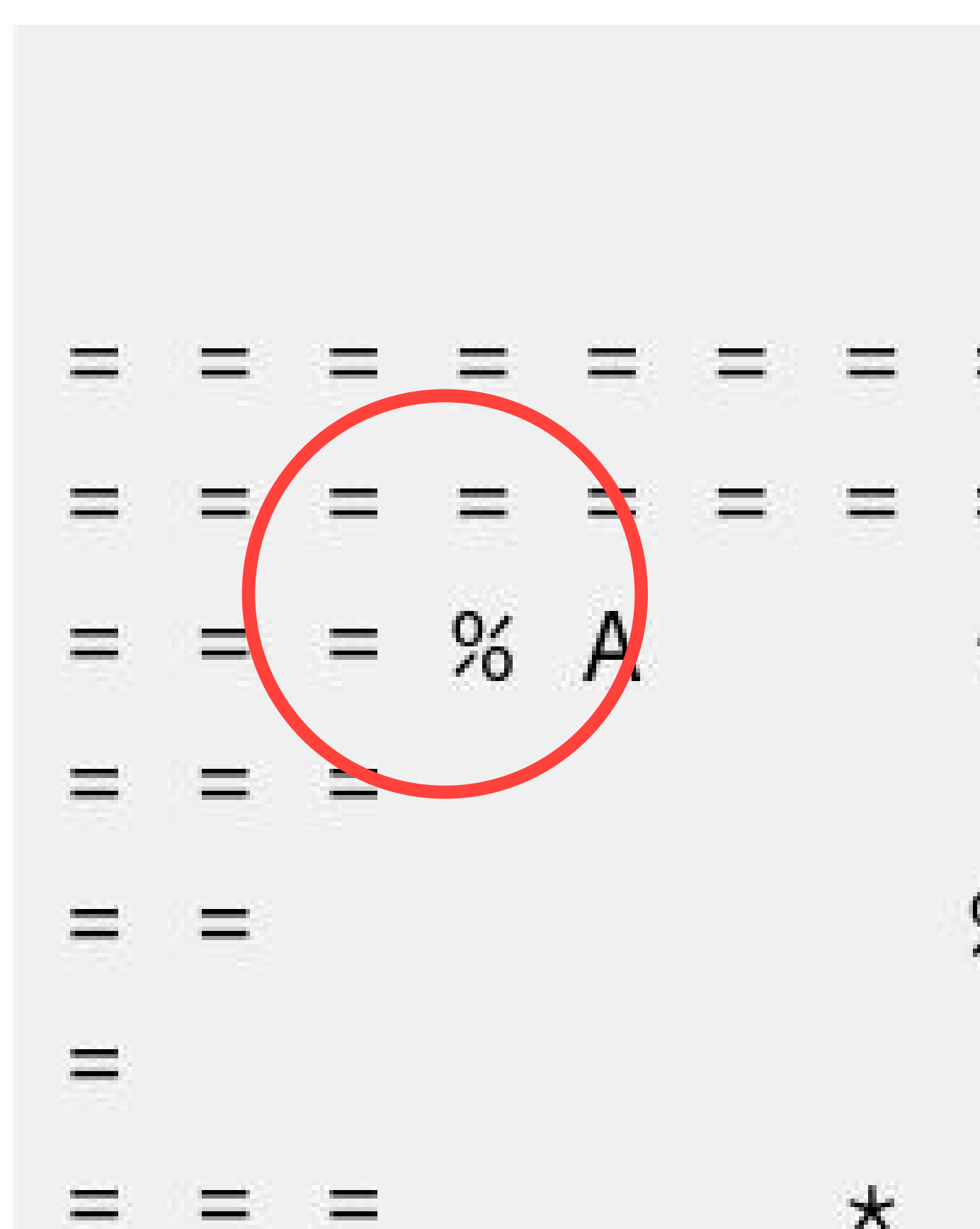
Boxes Left: 3 / 7

  = = = = = = = = = = = = = = = =
  = = = = = = = = = = 0 = = = =
  = = =      *      = = =      KEYMAP:
  = = =      0 = = =      [W] Up      [R] Reset Current Level
  = =      %      %      0 = =      [S] Down    [G] Generate New Level
  =      *      *      = =      [A] Left    [Q] Exit Proj_258: Sokoban
  = = =      *      %      0 = =      [D] Right
  = = =
  = = =      A      = = =
  = = = = = = = = = = = = = = = =

Read FIFO: 0

```

Whoops—accidentally pushed a box to a corner? The game is now unsolvable, but maybe a second run will do the trick.



Just press [R] on the keyboard to reset the same level to its original, starting state.

Alternatively, to generate a completely different map of random size and box count, press [G] on the keyboard.

To exit *Proj\_258: Sokoban*, press [Q] on the keyboard.

Upon successful completion of a level, the following congratulatory message will be displayed on screen. Refer to the image below.

Follow the on-screen instruction to either retry the same level, regenerate an entirely new level, or exit *the game*.

```

Terminal
IRQ 24 ffff0000

Boxes Left: 0 / 7

=====
===== 0 =====
===== 0 =====
===== 0 =====
===== A 0 =====
===== 0 =====
===== 0 0 =====
=====
=====
=====
=====
***** LEVEL SOLVED! :) Choose [R] or [G] or [Q]:
```

To play *Proj\_258: Sokoban* again at a later time, simply follow the same instructions thus far in this user guide. Be sure to save the assembly file that accompany this user guide so that its contents can be copied and pasted into *CPUlator* again.

# Implementation Specs

1. The user is NOT given the ability to choose the board size. The possible dimensions of the playable grid are randomly generated from 8 to 17. The number of rows and columns are generated independently from one another. I.e., the grid will not always be a perfect square.
2. Just like the original Sokoban game, *Proj\_258: Sokoban* will NOT warn the user when the game no longer becomes solvable. It is up to the user to recognize such a mishap, and manually reset the level to its initial state.
3. The user is NOT warned about “invalid inputs”. Rather, inputs that do not result in a change in game state will simply be ignored. The program will only listen for a particular set of keyboard inputs.
4. ENHANCEMENT 1: A custom LFSR algorithm (Linear Feedback Shift Register) is used to generate internal walls that cover exactly 25% of the empty space within the outer walls of the game board. The number of internal walls are calculated by dividing the number of empty grid cells by 4. The internal walls are placed such that the game is solvable when other game objects later spawn inside the grid.
5. ENHANCEMENT 2: The number of boxes (and, thus, the number of targets) are randomly generated. The minimum possible box count in every case is 1. Depending on the randomly generated size of the game board, the box count can increase up to a maximum of 19.