Assembly Project - Simon Game

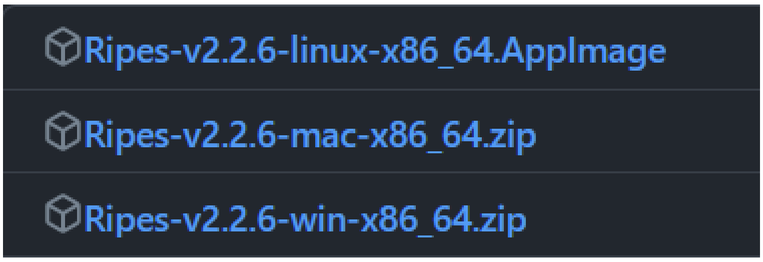
March 1st, 2023

1. Introduction

Simon is a short-term memory game developed in the 1970s.

2. Prerequisites

The user must have Ripes downloaded from this link:
https://github.com/mortbopet/Ripes/releases
The user must download the version that is compatible with their system.

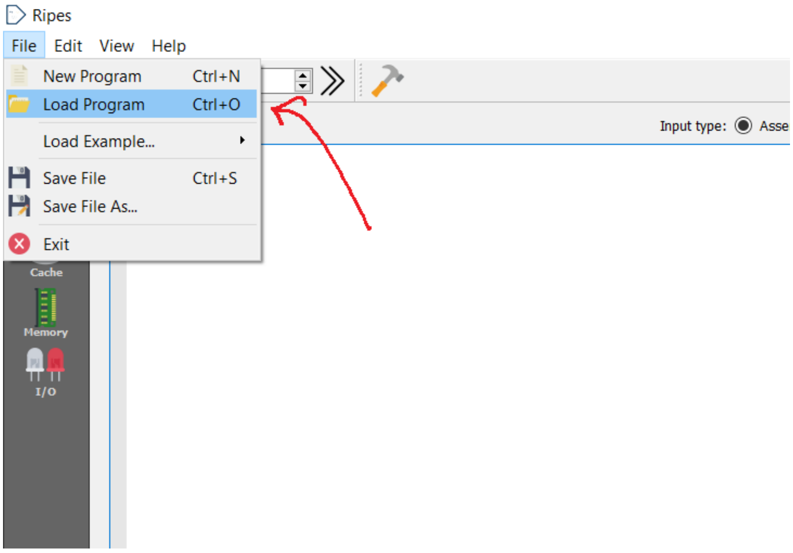


Ripes-v2.2.6-linux-x86_64.AppImage
Ripes-v2.2.6-mac-x86_64.zip
Ripes-v2.2.6-win-x86_64.zip

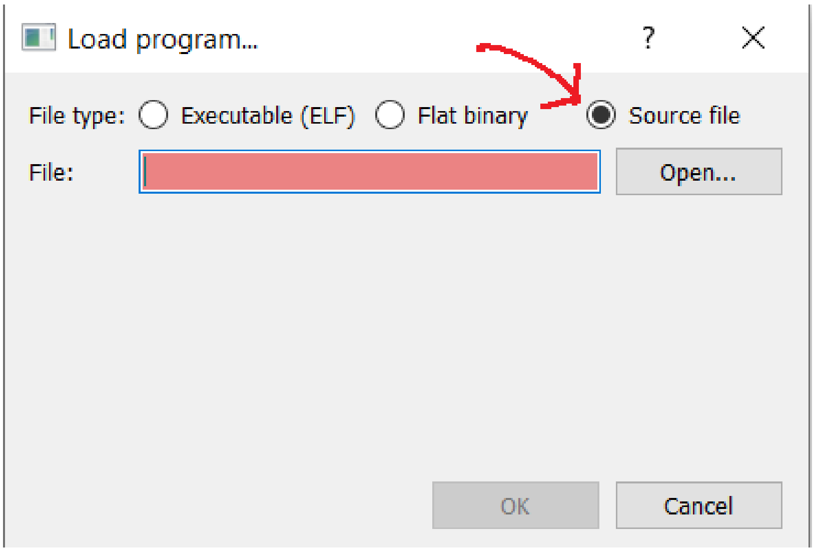
A linux user will download the first file, a mac user will download the second file, and a windows user will download the third file.
The user must also have starter.s downloaded.

3. Setting up the game

Start up Ripes. Navigate to the top left corner of the application and select File -> Load Program.



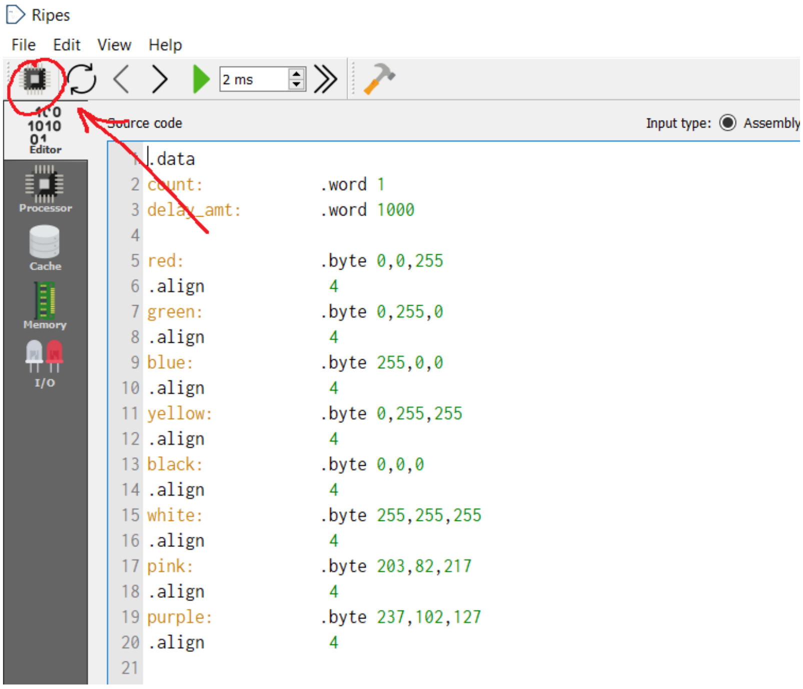
Select the “Source File” option and open starter.s



File type: ○ Executable (ELF) ○ Flat binary ● Source file

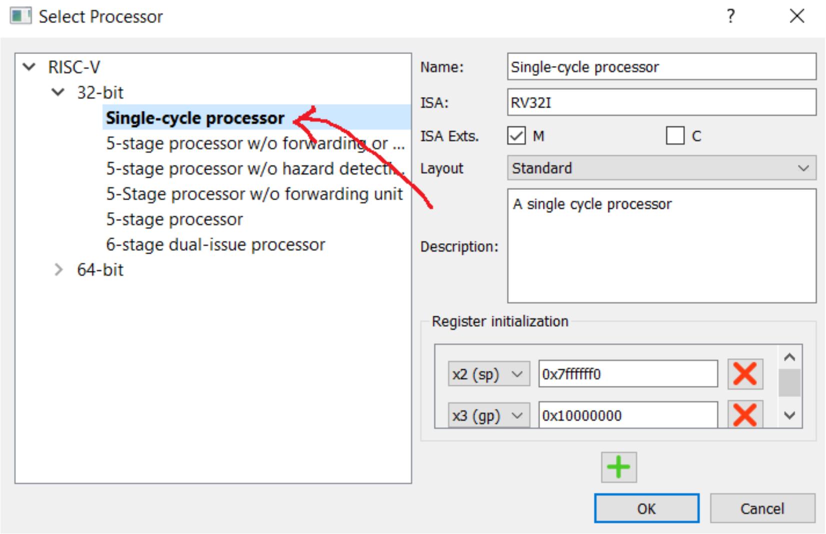
File:

Navigate to the top left corner of the application and select “Select processor”. The single cycle processor is required to play the game.



Input type: ● Assembly

```
.data
count:          .word 1
delay_amt:      .word 1000

red:            .byte 0,0,255
.align           4
green:          .byte 0,255,0
.align           4
blue:           .byte 255,0,0
.align           4
yellow:         .byte 0,255,255
.align           4
black:          .byte 0,0,0
.align           4
white:          .byte 255,255,255
.align           4
pink:           .byte 203,82,217
.align           4
purple:         .byte 237,102,127
.align           4

```



- RISC-V
  - 32-bit
    - **Single-cycle processor**
    - 5-stage processor w/o forwarding or ...
    - 5-stage processor w/o hazard detecti...
    - 5-Stage processor w/o forwarding unit
    - 5-stage processor
    - 6-stage dual-issue processor
  - 64-bit

Name: Single-cycle processor

ISA: RV32I

ISA Exts. ☑ M ☐ C

Layout: Standard

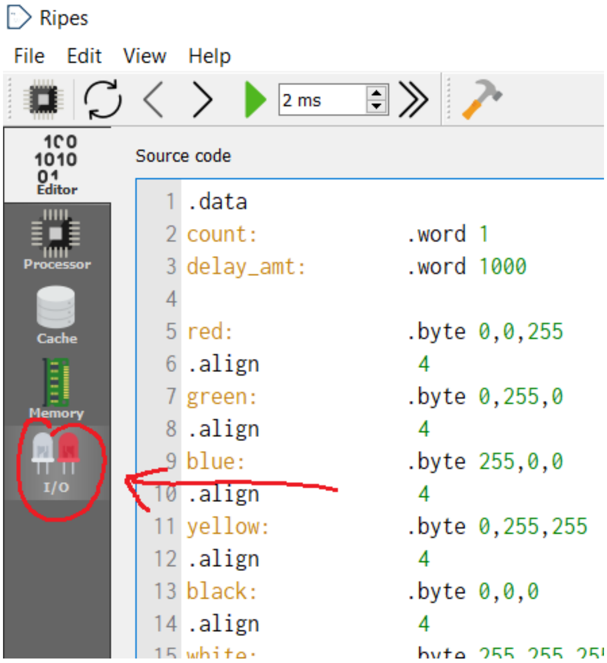
Description: A single cycle processor

Register initialization

x2 (sp) 0x7ffffff0

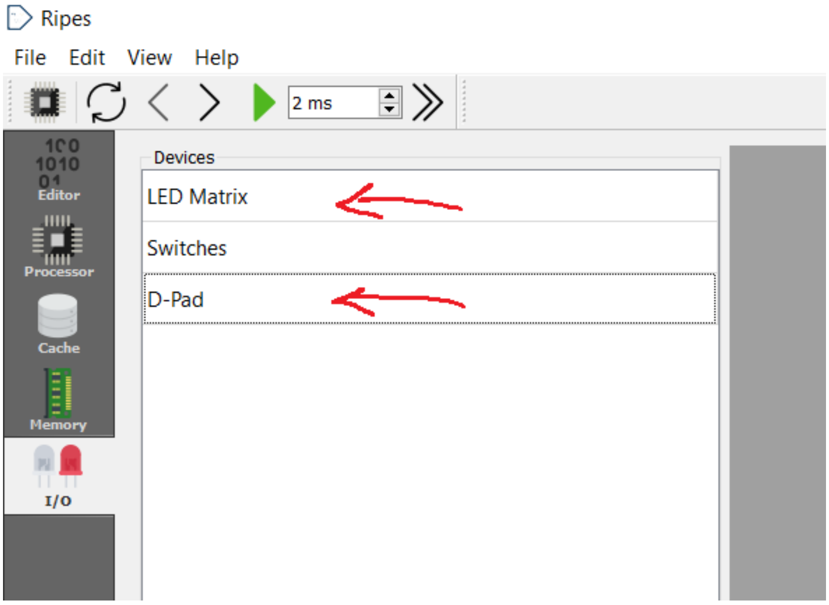
x3 (gp) 0x10000000

Navigate to I/O.

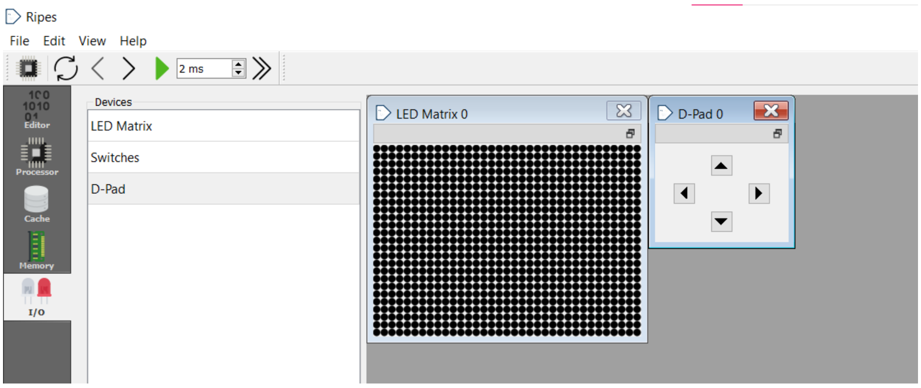


```
.data
count:          .word 1
delay_amt:      .word 1000

red:            .byte 0,0,255
.align           4
green:          .byte 0,255,0
.align           4
blue:           .byte 255,0,0
.align           4
yellow:         .byte 0,255,255
.align           4
black:          .byte 0,0,0
.align           4
white:          .byte 255,255,25
```

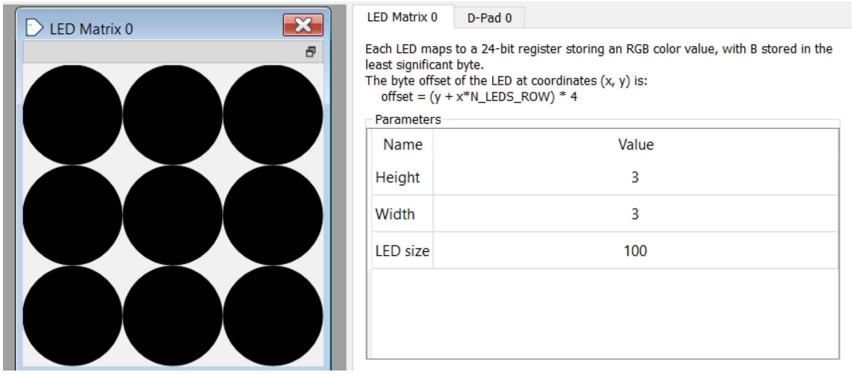
Within the I/O tab, double click on LED Matrix and D-Pad.



Upon successful initialization, these 2 will be displayed.



Click on the LED Matrix 0 to focus on it. Change the height, width, and LED size to 3, 3, and 100, respectively.

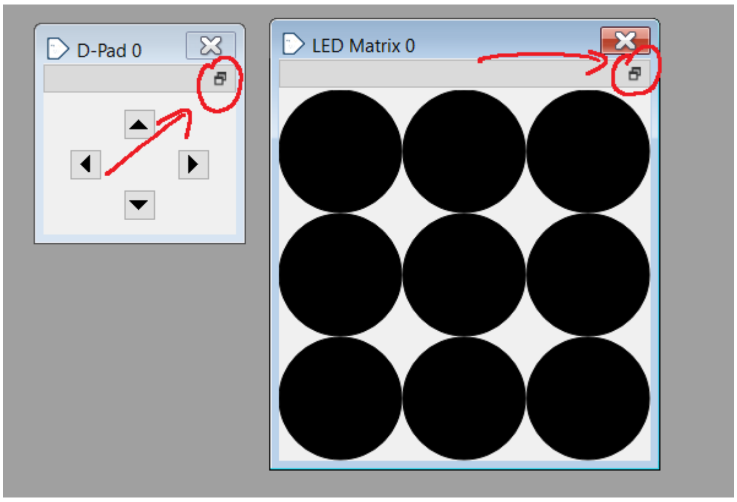


Each LED maps to a 24-bit register storing an RGB color value, with B stored in the least significant byte.
The byte offset of the LED at coordinates (x, y) is:
    offset = (y + x*N_LEDS_ROW) * 4

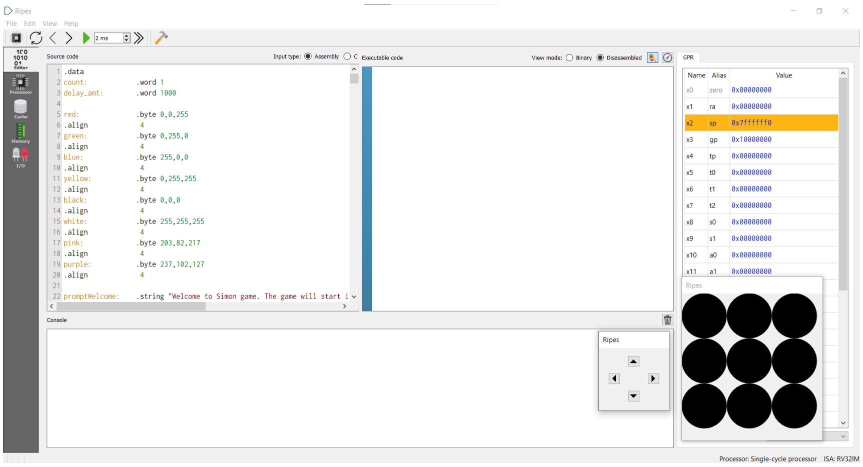
Parameters

Name	Value
Height	3
Width	3
LED size	100

Select the button right below the close button in the D-Pad and the Matrix to pop the display out.



We are now ready to navigate back to the Editor tab along with the D-Pad and the LED Matrix.



Input type: ● Assembly ○ C

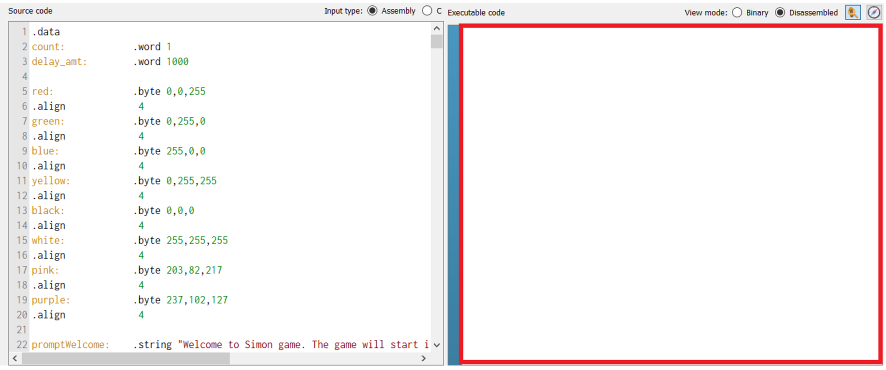
View mode: ○ Binary ● Disassembled

```
.data
count:          .word 1
delay_amt:      .word 1000

red:            .byte 0,0,255
.align           4
green:          .byte 0,255,0
.align           4
blue:           .byte 255,0,0
.align           4
yellow:         .byte 0,255,255
.align           4
black:          .byte 0,0,0
.align           4
white:          .byte 255,255,255
.align           4
pink:           .byte 203,82,217
.align           4
purple:         .byte 237,102,127
.align           4

promptWelcome:    .string "Welcome to Simon game. The game will start i
```

Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7ffffff0
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000000
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x00000000
x9	s1	0x00000000
x10	a0	0x00000000
x11	a1	0x00000000

Processor: Single-cycle processor ISA: RV32IM

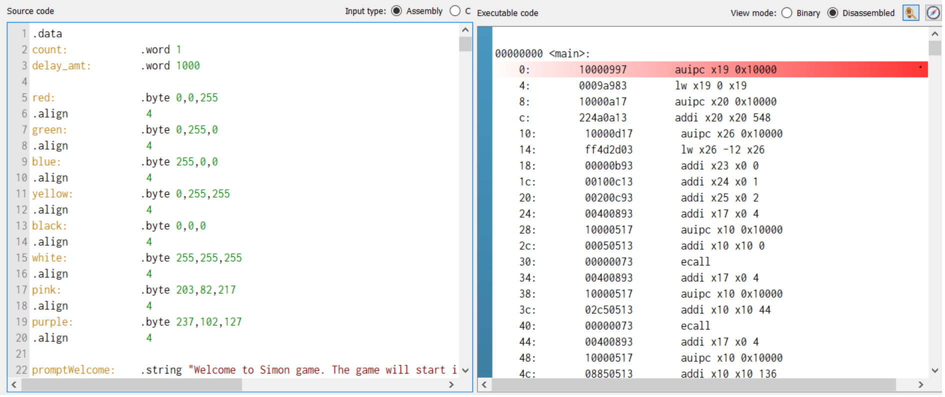
We now need to load the program into memory, specifically, we need to initialize the screen on the right.



Source code
Input type: ● Assembly ○ C
Executable code
View mode: ○ Binary ● Disassembled

```
.data
count:              .word 1
delay_amt:          .word 1000

red:                .byte 0,0,255
.align               4
green:              .byte 0,255,0
.align               4
blue:               .byte 255,0,0
.align               4
yellow:             .byte 0,255,255
.align               4
black:              .byte 0,0,0
.align               4
white:              .byte 255,255,255
.align               4
pink:               .byte 203,82,217
.align               4
purple:             .byte 237,102,127
.align               4

promptWelcome:      .string "Welcome to Simon game. The game will start i
```

To do so, edit the source code screen on the left by adding redundant contents, such as adding a letter, which can be done by moving the cursor to any place in the code, press any alphabetical key on the keyboard, and then delete the newly added content. As a result, the screen on the right will now be updated. The game is ready to be started.

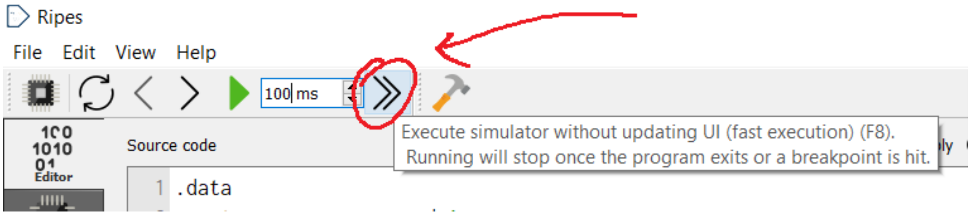


Source code
Input type: ● Assembly ○ C
Executable code
View mode: ○ Binary ● Disassembled

```
.data
count:              .word 1
delay_amt:          .word 1000

red:                .byte 0,0,255
.align               4
green:              .byte 0,255,0
.align               4
blue:               .byte 255,0,0
.align               4
yellow:             .byte 0,255,255
.align               4
black:              .byte 0,0,0
.align               4
white:              .byte 255,255,255
.align               4
pink:               .byte 203,82,217
.align               4
purple:             .byte 237,102,127
.align               4

promptWelcome:      .string "Welcome to Simon game. The game will start i
```

```
00000000 <main>:
    0:        10000997        auipc x19 0x10000
    4:        0009a983        lw x19 0 x19
    8:        10000a17        auipc x20 0x10000
    c:        224a0a13        addi x20 x20 548
    10:        10000d17        auipc x26 0x10000
    14:        ff4d2d03        lw x26 -12 x26
    18:        00000b93        addi x23 x0 0
    1c:        00100c13        addi x24 x0 1
    20:        00200c93        addi x25 x0 2
    24:        00400893        addi x17 x0 4
    28:        10000517        auipc x10 0x10000
    2c:        00050513        addi x10 x10 0
    30:        00000073        ecall
    34:        00400893        addi x17 x0 4
    38:        10000517        auipc x10 0x10000
    3c:        02c50513        addi x10 x10 44
    40:        00000073        ecall
    44:        00400893        addi x17 x0 4
    48:        10000517        auipc x10 0x10000
    4c:        08850513        addi x10 x10 136
```

Select fast execution to start playing.

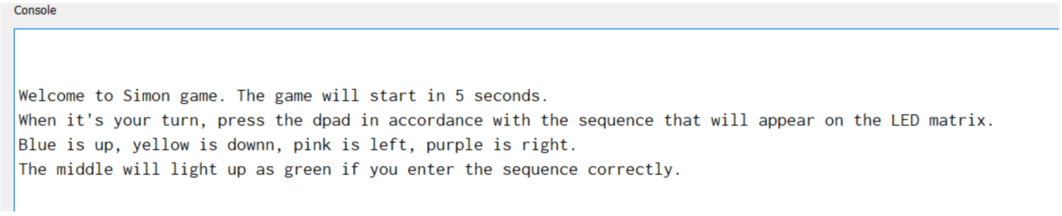


Execute simulator without updating UI (fast execution) (F8).
Running will stop once the program exits or a breakpoint is hit.

```
.data
```

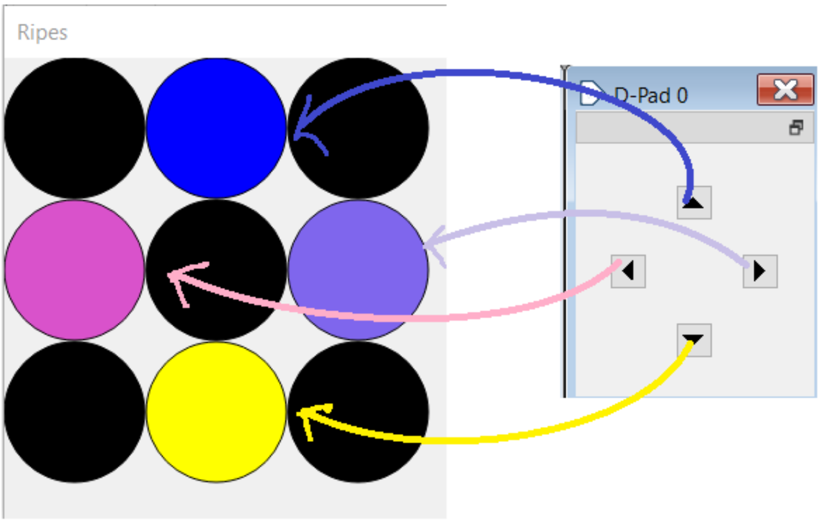
4. Gameplay

Upon starting the game, the console will display the game instruction to the player.



```
Welcome to Simon game. The game will start in 5 seconds.
When it's your turn, press the dpad in accordance with the sequence that will appear on the LED matrix.
Blue is up, yellow is downn, pink is left, purple is right.
The middle will light up as green if you enter the sequence correctly.
```

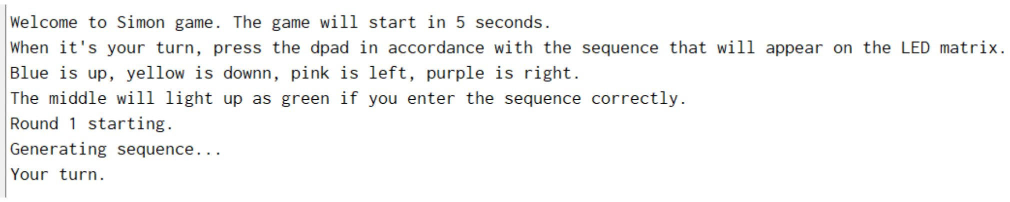
The D-Pad button corresponding colors are as follows.



The game will allow the player to have time and look at the displayed sequence. When it's the player's turn, the console will say so.

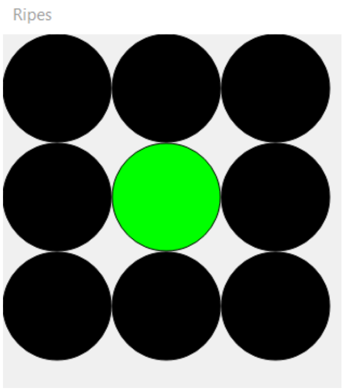
As an example, suppose that before the player's turn, the D-Pad lights up in the following order: Blue -> Purple -> Yellow -> Pink, the player should press the following D-Pad keys: UP -> RIGHT -> DOWN -> LEFT.

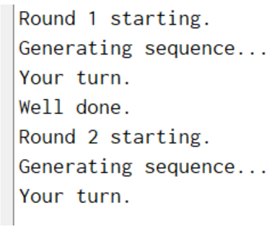
The corresponding LED will light up as the player press the buttons on the D-Pad.



```
Welcome to Simon game. The game will start in 5 seconds.
When it's your turn, press the dpad in accordance with the sequence that will appear on the LED matrix.
Blue is up, yellow is downn, pink is left, purple is right.
The middle will light up as green if you enter the sequence correctly.
Round 1 starting.
Generating sequence...
Your turn.
```

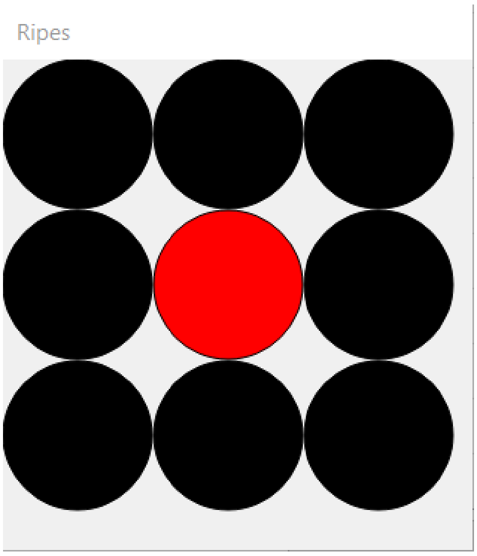
If the entered sequence is correct, the middle LED will blink green twice. A "Well done" message will also be displayed to the console to motivate the player.

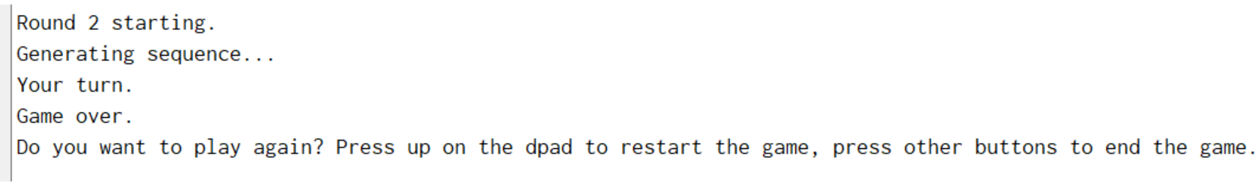




```
Round 1 starting.
Generating sequence...
Your turn.
Well done.
Round 2 starting.
Generating sequence...
Your turn.
```

If the entered is wrong, the middle LED will blink red twice, the game is over. The console will prompt the user if they want to play again or not.





```
Round 2 starting.
Generating sequence...
Your turn.
Game over.
Do you want to play again? Press up on the dpad to restart the game, press other buttons to end the game.
```

If the player presses up on their keyboard, the game starts anew. Otherwise, the game ends.

Have fun.