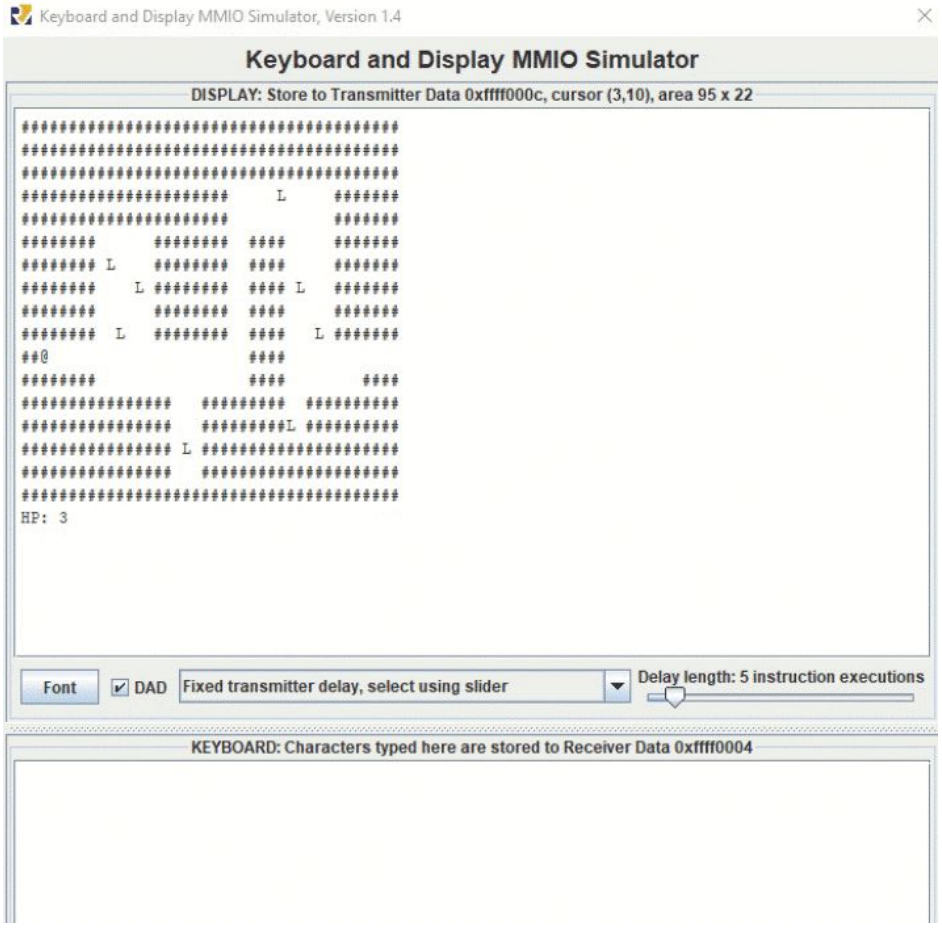
Lab 4: Dungeon Crawler

CMPUT 229

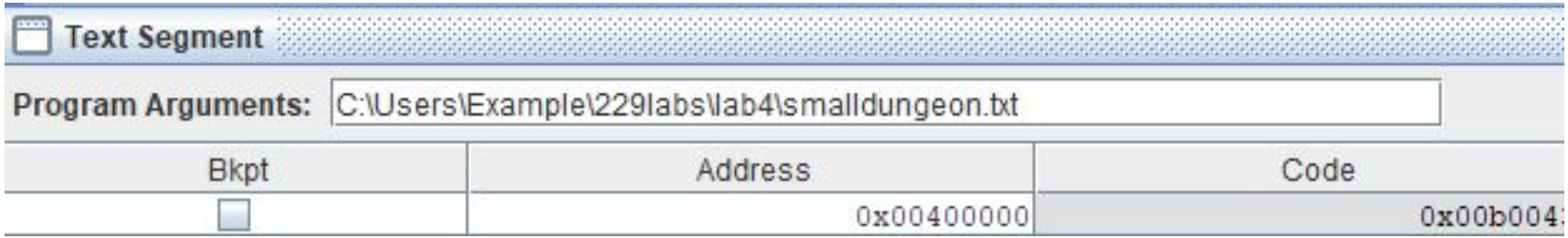
Game Overview



Legend

#: Dungeon Wall
L: Loot
E: Enemy
@: Agent
Space: Path

Program Arguments



This is one of the input files provided to you:

smalldungeon.txt:

```
# Player start position:
1,1

# Player finish position:
16,8

# Max dungeon coordinates:
16,10

# Number of path vectors:
11

# Number of loot:
4

# Number of Enemies:
5
```

```
# Path vectors:
1,1 6,1
1,2 6,2
5,7 11,7
5,8 16,8
5,9 11,9
5,3 5,6
6,3 6,6
11,3 11,6
12,3 12,7
13,3 13,5
14,3 14,5
```

```
# Loot positions:
3,1
6,9
12,3
13,4

# Enemy Positions:
3,2
5,4
6,4
7,7
11,4
```

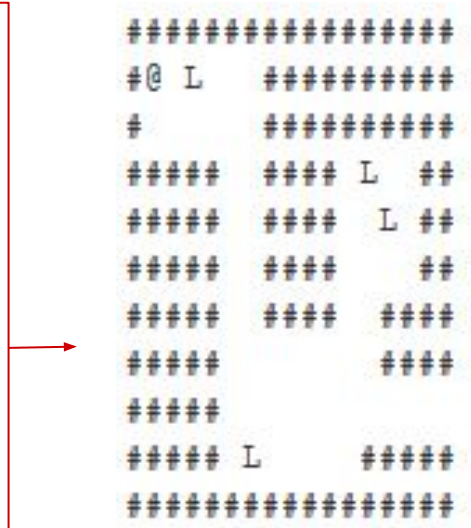
The input file defines a unique dungeon configuration

smalldungeon.txt:

```
# Player start position:
1,1

# Player finish position:
16,8

# Max dungeon coordinates:
16,10

# Number of path vectors:
11

# Number of loot:
4

# Number of Enemies:
5
```

```
# Path vectors:
1,1 6,1
1,2 6,2
5,7 11,7
5,8 16,8
5,9 11,9
5,3 5,6
6,3 6,6
11,3 11,6
12,3 12,7
13,3 13,5
14,3 14,5
```

```
# Loot positions:
3,1
6,9
12,3
13,4

# Enemy Positions:
3,2
5,4
6,4
7,7
11,4
```



Building the Dungeon

The provided common.s script parses the input file provided as a program argument into three designated arrays and some global variables:

Arrays:

1. Paths Array
2. Loot Array
3. Enemies Array

Global Variables:

1. PLAYER_X
2. PLAYER_Y
3. MAX_X
4. MAX_Y
5. FINISH_X
6. FINISH_Y

Paths Array

The paths array is an array of path “structs”. Each path struct contains four 32-bit integers that represent the start and end coordinates of a path: start x, start y, end x, end y.

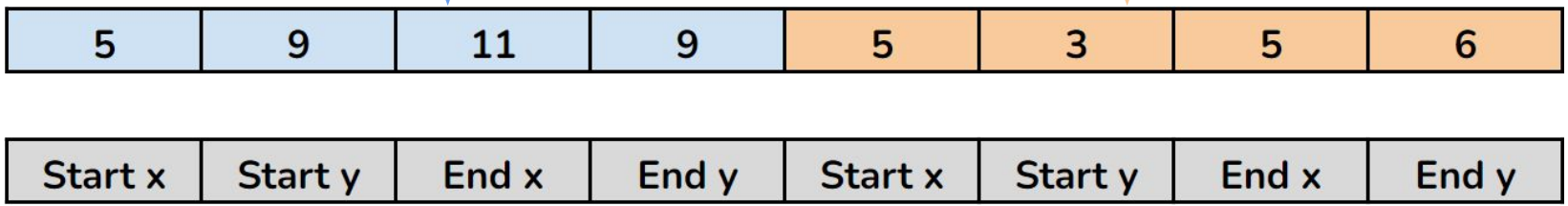
```
struct Path {
    int start_x;
    int start_y;
    int end_x;
    int end_y;
};
```

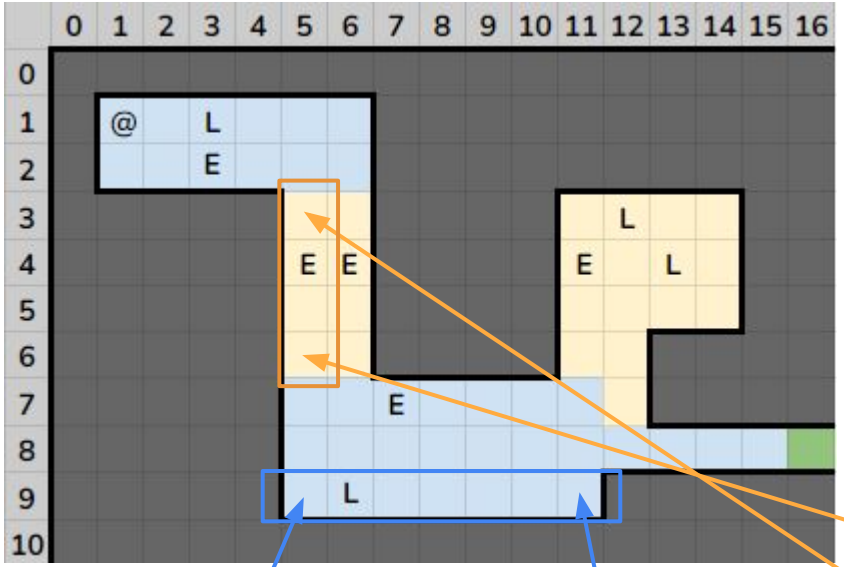
A path can be imagined as this C struct

Paths Array

smalldungeon.txt:

```
# Path vectors:
1,1 6,1
1,2 6,2
5,7 11,7
5,8 16,8
5,9 11,9
5,3 5,6
6,3 6,6
11,3 11,6
12,3 12,7
13,3 13,5
14,3 14,5
```





Blue: Horizontal paths
Yellow: Vertical paths
Green: Finish point

5	9	11	9	5	3	5	6

Start x	Start y	End x	End y	Start x	Start y	End x	End y

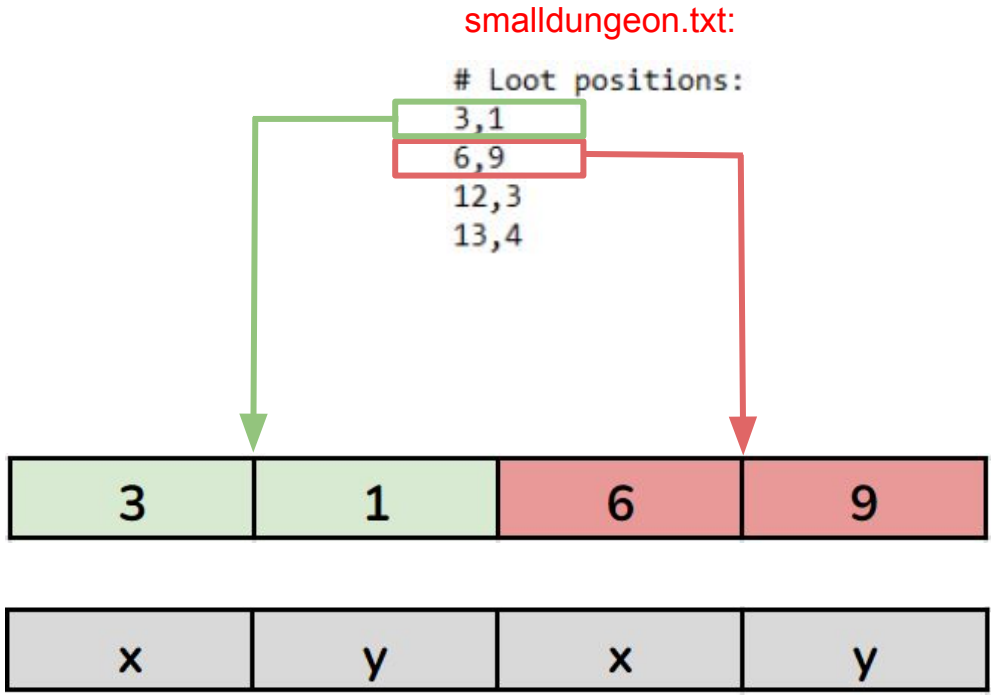
Loot Array

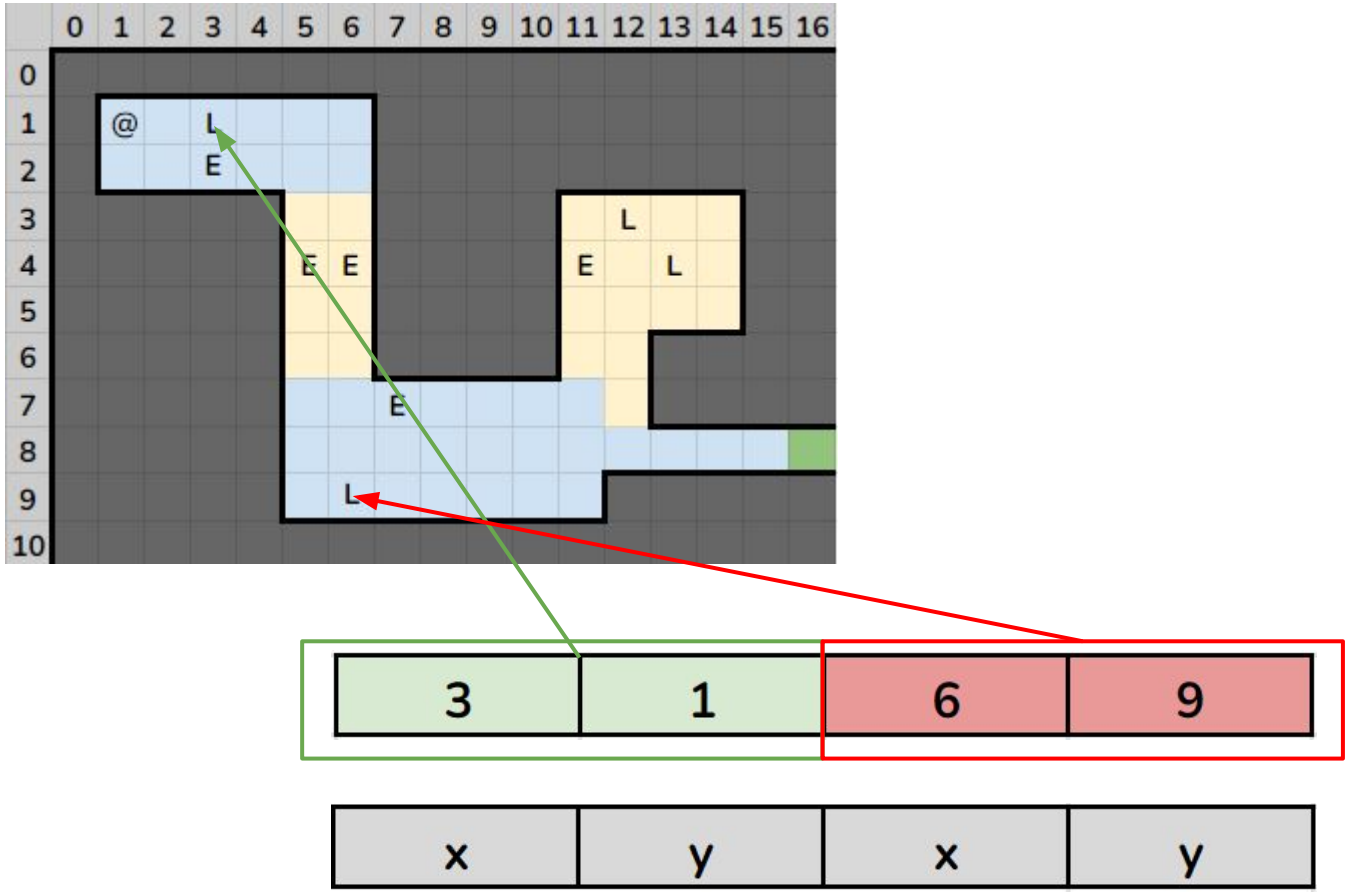
The loot array is an array of loot “structs”. Each loot struct contains two 32-bit integers that represent the coordinates of a loot item.

```
struct Loot {
    int pos_x;
    int pos_y;
};
```

Loot can be imagined as this C struct.

Loot Array




0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0
1
2
3
4
5
6
7
8
9
10
@
L
E
E E
L
E L
E
L
3
1
6
9
x
y
x
y

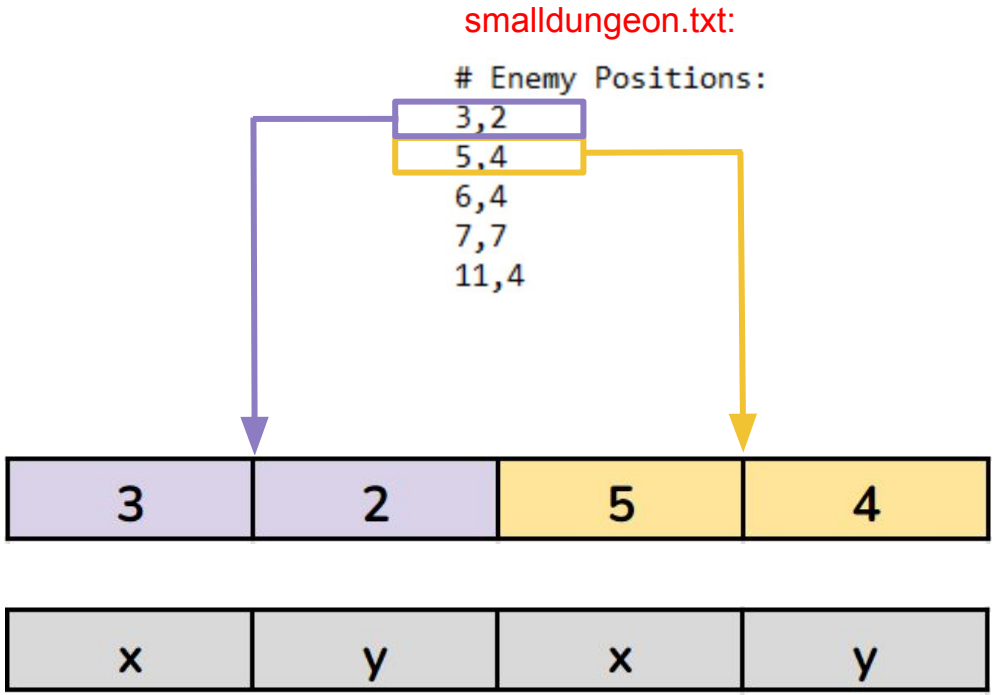
Enemies Array

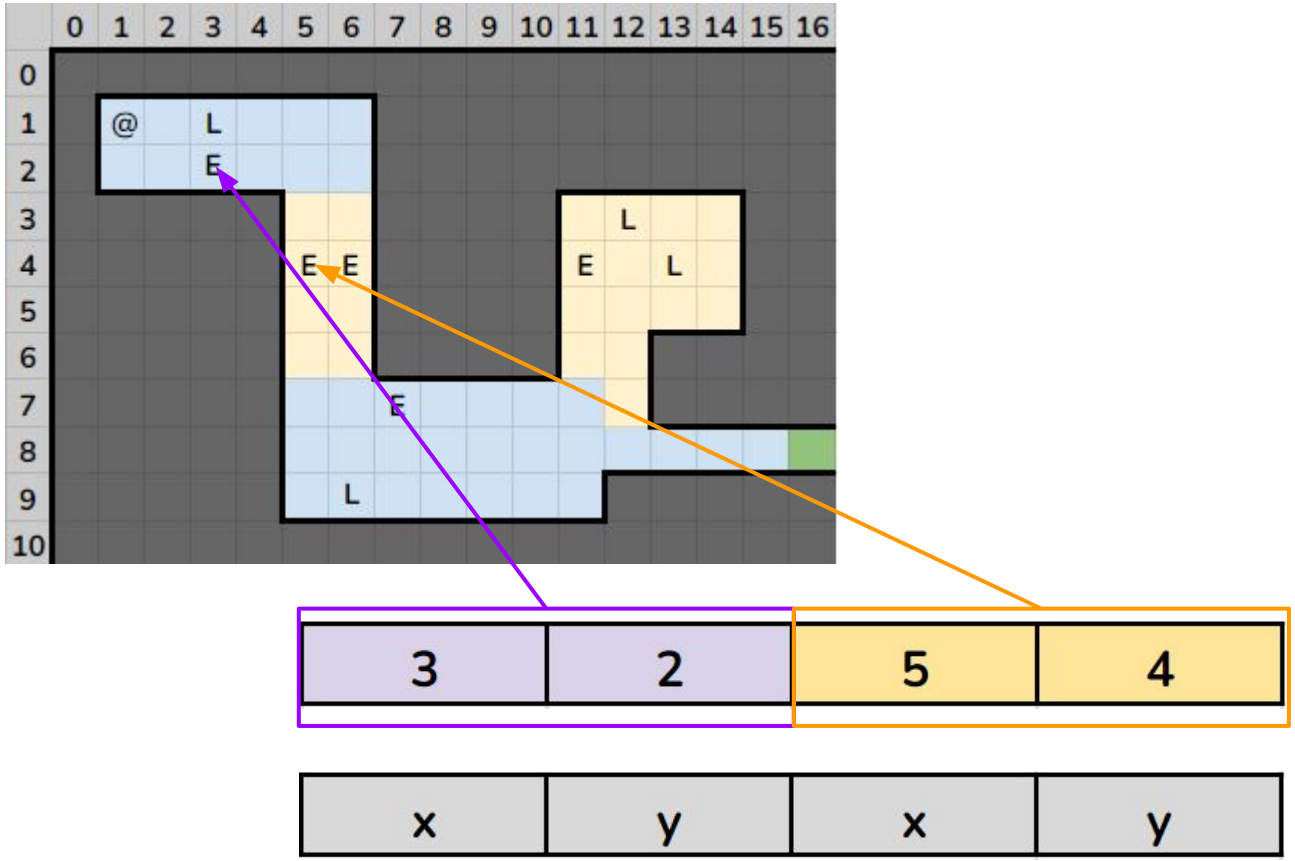
The enemies array is an array of enemy “structs”. Each enemy struct contains two 32-bit integers that represent the coordinates of an enemy.

```
struct Enemy {
    int pos_x;
    int pos_y;
};
```

An enemy can be imagined as this C struct.

Enemies Array




0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0
1
2
3
4
5
6
7
8
9
10
@
L
E
E E
L
E L
E
L
3 2 5 4
x y x y

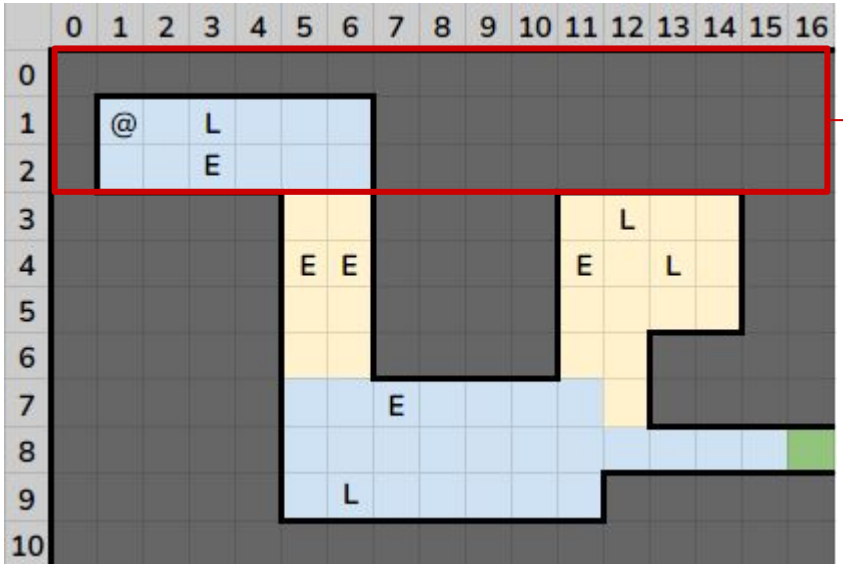
Representing the Dungeon Map as a 2D Array

Your program must use the data in the paths, loot, and enemy arrays to construct a 2D representation of the dungeon map.

Four pointers will be provided to your program's primary function:

1. Pointer to the paths array
2. Pointer to the loot array
3. Pointer to the enemies array
4. Pointer to an empty array used to store the 2D representation of the dungeon

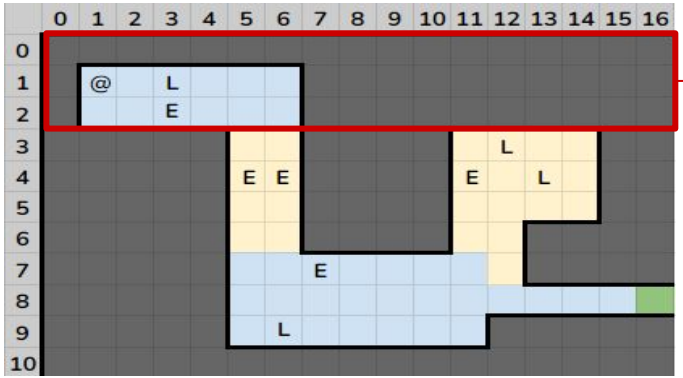
Representing the Dungeon Map as a 2D Array



[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 …]



[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 ...]

Each element of the 2D array is a 32-bit integer:

- 0 = Dungeon wall
- 1 = Path
- 2 = Loot
- 3 = Hidden enemy
- 4 = Shown enemy

Provided Global Variables

PLAYER_X: Current x coordinate of the agent

PLAYER_Y: Current y coordinate of the agent

MAX_X: Maximum x coordinate of the map

MAX_Y: Maximum y coordinate of the map

FINISH_X: x coordinate of the exit point of the dungeon

FINISH_Y: y coordinate of the exit point of the dungeon

Gameplay Details

Timer

The game starts with 5 seconds on the timer. The timer decreases by 1 each second, implemented using timer interrupts.

If the timer reaches 0, the game will stop and you will lose.

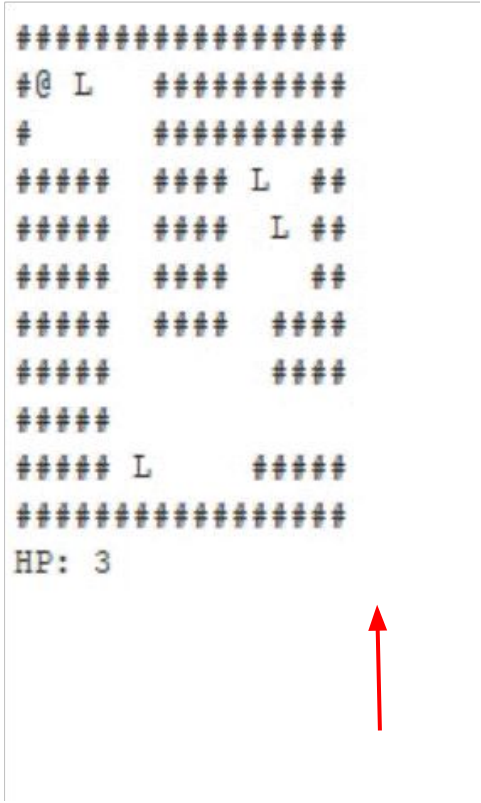
Movement

The w,a,s,d keys are used to move the agent around the dungeon, implemented using keyboard interrupts.

The agent is only able to move along paths.

Encountering Loot

When your agent moves to a position that contains loot, 5 seconds are added to the timer and the loot is removed from the map.



Health Points

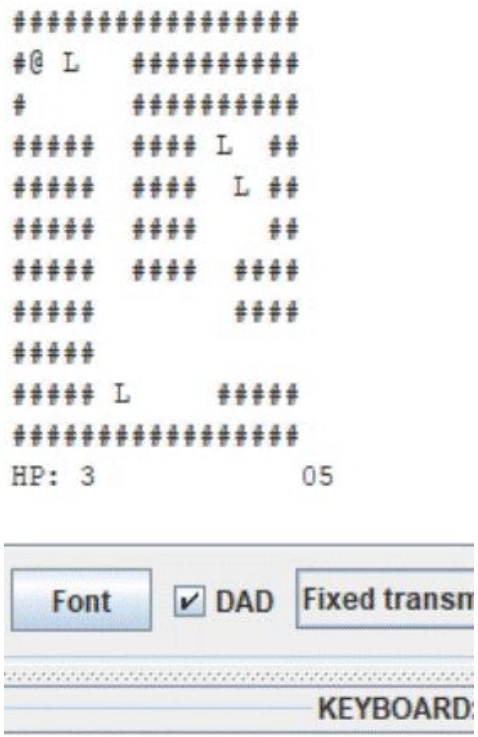
Your agent starts the game with 3 health points. Encountering enemies causes your agent to lose health points.

If your agent’s health points reach 0, the game stops and you lose.

Encountering an enemy

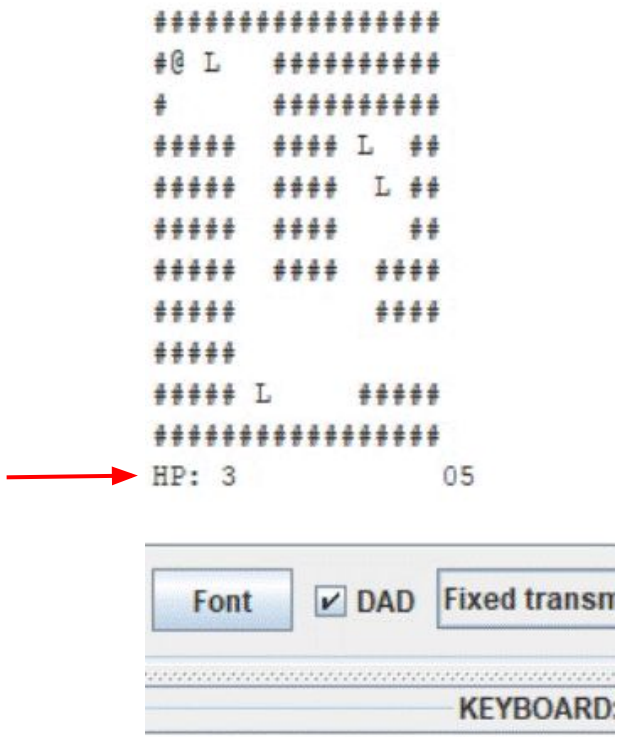
Enemies are hidden until the agent encounters them.

When your agent moves to a position that contains an enemy, it will be unable to move until you press the spacebar to attack.



Encountering an enemy

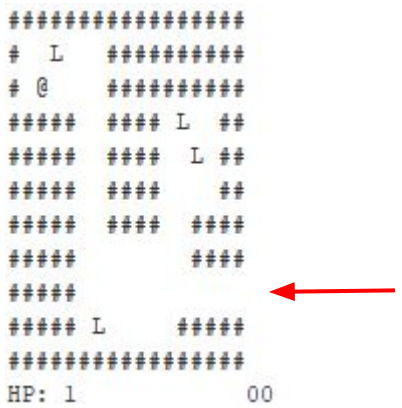
Your agent's health points decrease by 1 for each second that your agent is next to an enemy.



Win Conditions

To win the game, the following conditions must be true:

1. Your agent must reach the dungeon exit
2. There must be time remaining on the timer
3. The agent must have at least 1 health point remaining
4. All loot must be collected



Exit represented by a hole in the dungeon wall

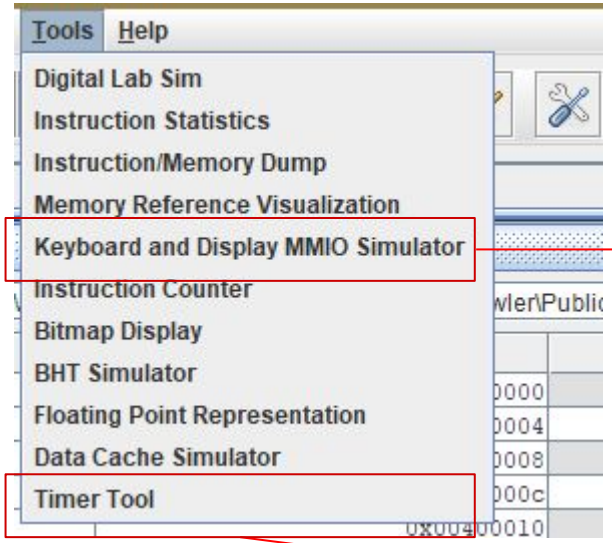
Interrupts

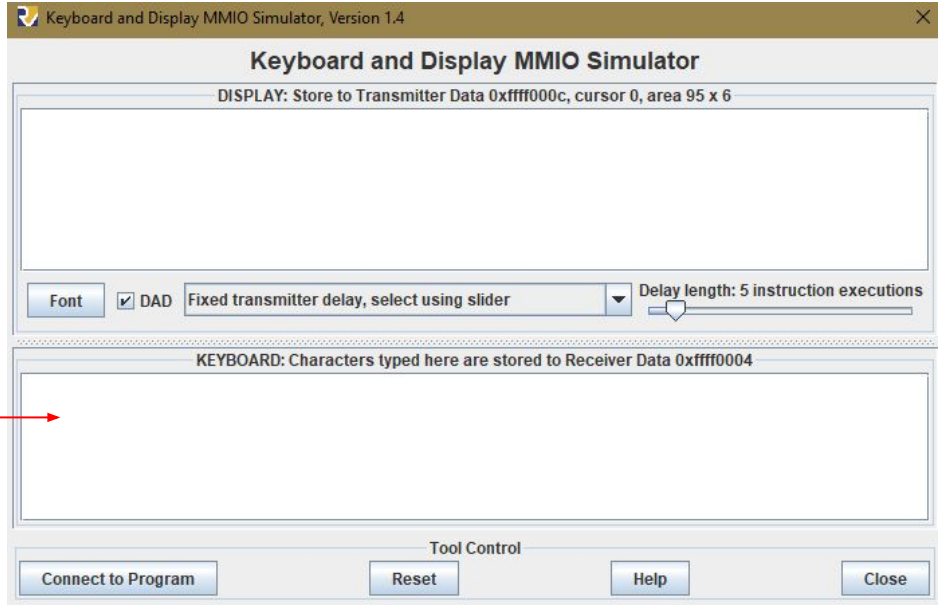
Interrupts

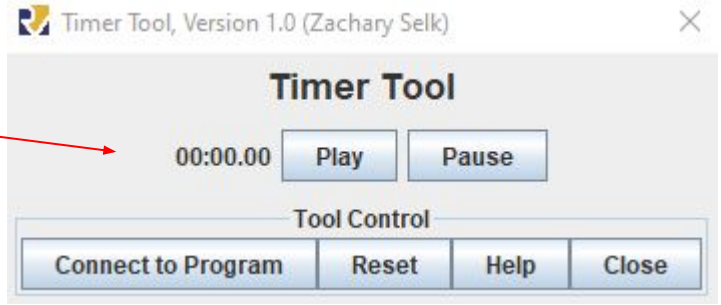
The timer, player movement, and attack elements use external interrupts from hardware.

- The rars timer tool is used to simulate RISC-V timing functionality, and is necessary for timer interrupts in this lab.
- The rars keyboard and display MMIO simulator simulates printing to an external display device, and is necessary for keyboard interrupts in this lab.

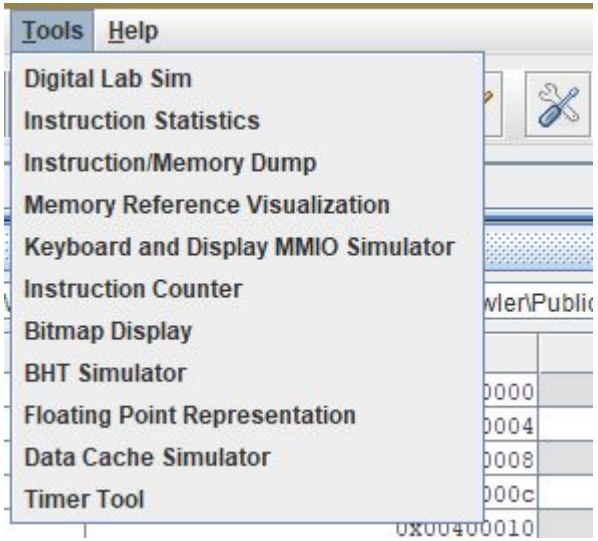
Required RARS Tools

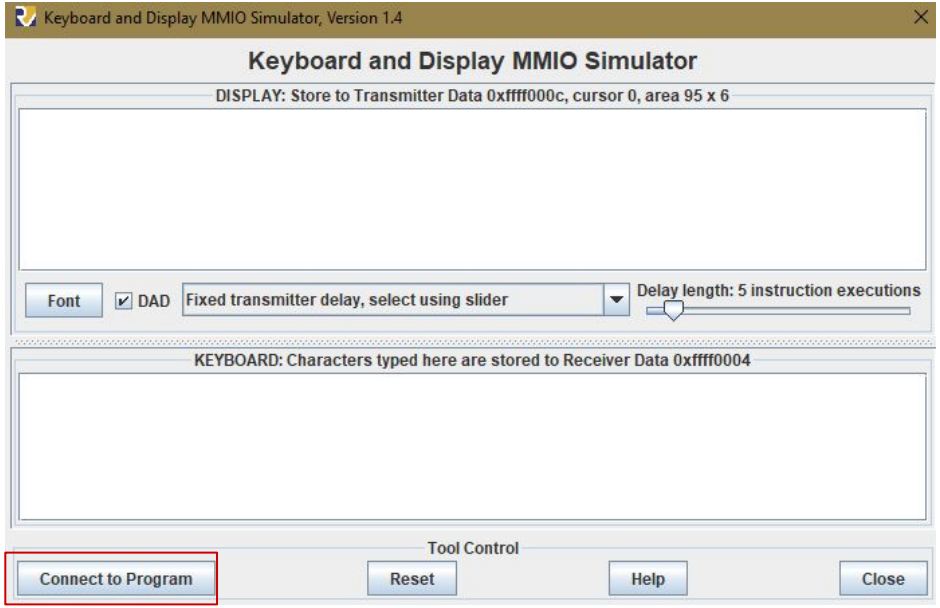


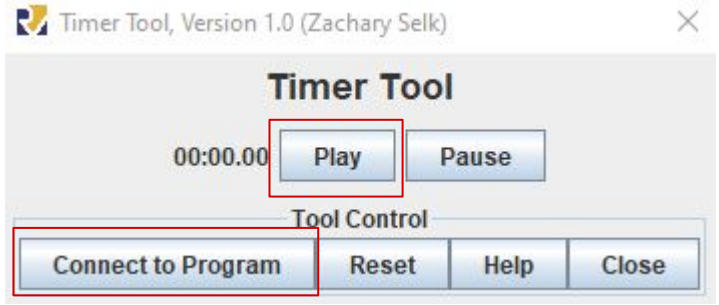




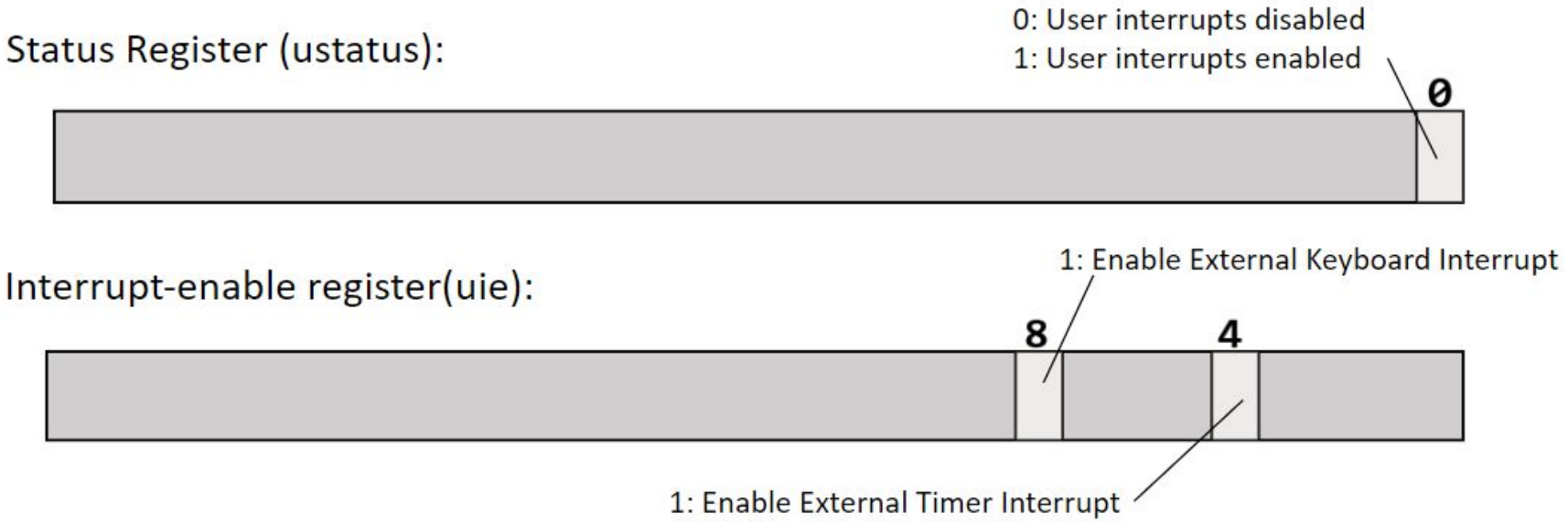
Required RARS Tools







Enabling Interrupts using Control and Status Registers



Keyboard and timer interrupts are both user interrupts. The bits above need to be set to **enable** keyboard and timer interrupts.

Interrupts

When an enabled interrupt is raised (such as the player pressing the “w” key), the program is paused and execution is transferred to the interrupt handler.

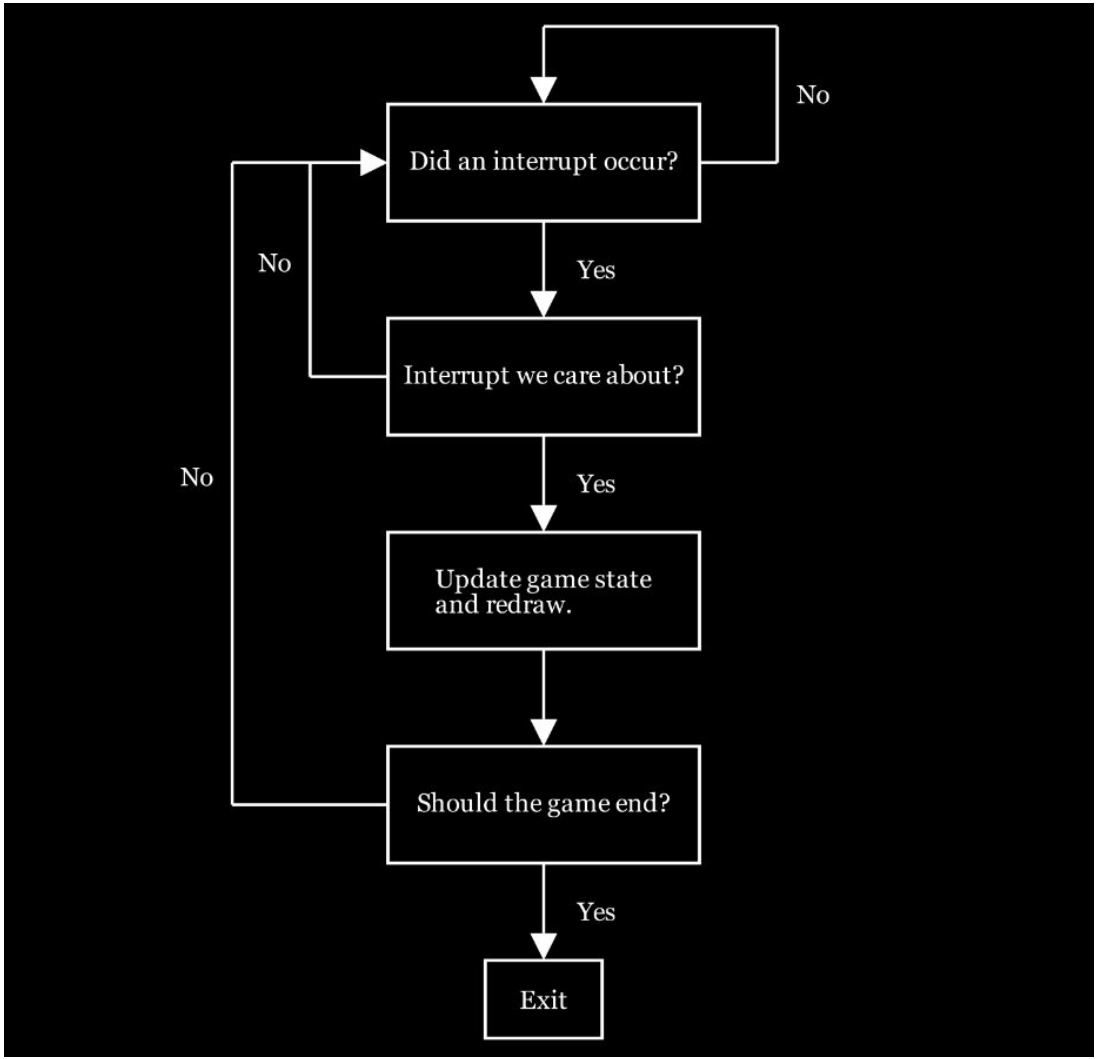
You will write a custom interrupt handler to handle keyboard and timer interrupts.

To use your custom handler, the address of your handler must be stored in the utvec CSR (CSR #5).

Global flags

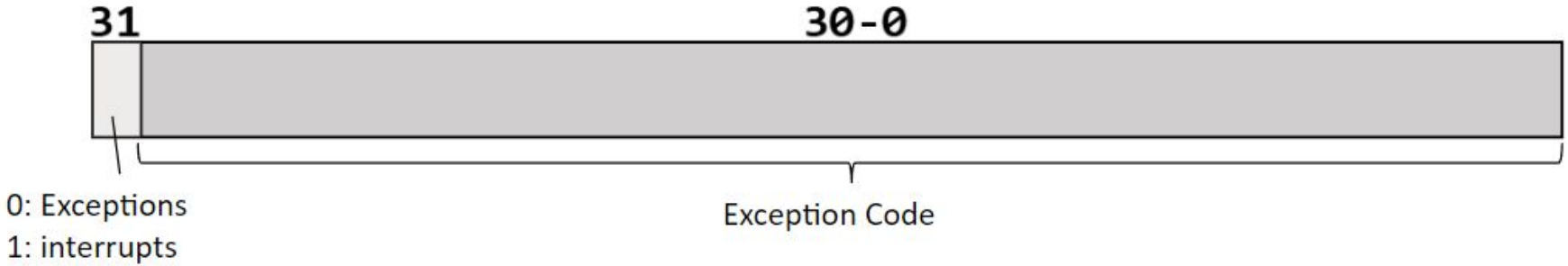
Setting global variable flags within your handler can signal to your main game loop what interrupt occurred. The main game loop can then update the game state accordingly after your program exits the handler.

Example Gameflow



Ucause

- The **ucause register** contains the current exception/interrupt that is raised. Note the 31st bit in the ucause register indicates if it was an exception or an interrupt.



Memory-Mapped IO

Memory-Mapped IO

Memory-mapped IO allows interaction with external devices through an interface pretending to be system memory. This mapping allows the processor to communicate with these devices using the load-word and store-word instructions.

In this lab, keyboard, time, and display I/O registers are important.

Keyboard Interrupts and MMIO Registers

Register Name	Memory Address	Description
Keyboard control	0xFFFF0000	For keyboard interrupts to be enabled, bit 1 of this register must be set to 1; after the keyboard interrupt occurs, this bit is automatically reset to 0.
Keyboard data	0xFFFF0004	The ASCII value of the last key pressed is stored here.

Time Interrupts and MMIO Registers

Register Name	Memory Address	Description
Time	0xFFFF0018	This is a read-only register that holds the time since the program has started in milliseconds.
Timecmp	0xFFFF0020	User-specified value. When less than or equal to the value in the **Time** register an interrupt is generated. Writing to this register is required to set up a timer.

Printing to MMIO Display using MMIO Registers

Register Name	Memory Address	Description
Display control	0xFFFF0008	Bit 0 of this register indicates whether the processor can write to the display. While this bit is 0 the processor cannot write to the display. Thus, the program must wait until this bit is 1.
Display data	0xFFFF000C	When a character is placed into this register, given that the display control ready bit (bit 0) is 1, that character is drawn onto the display.

Note that direct communication to the display via the Display Data register has been implemented for you in the provided printChar and printStr functions.

Register Name	Memory Address	Description
Display control	0xFFFF0008	Bit 0 of this register indicates whether the processor can write to the display. While this bit is 0 the processor cannot write to the display. Thus, the program must wait until this bit is 1.
Display data	0xFFFF000C	When a character is placed into this register, given that the display control ready bit (bit 0) is 1, that character is drawn onto the display.

Functions to implement in dungeon.s

dungeon:

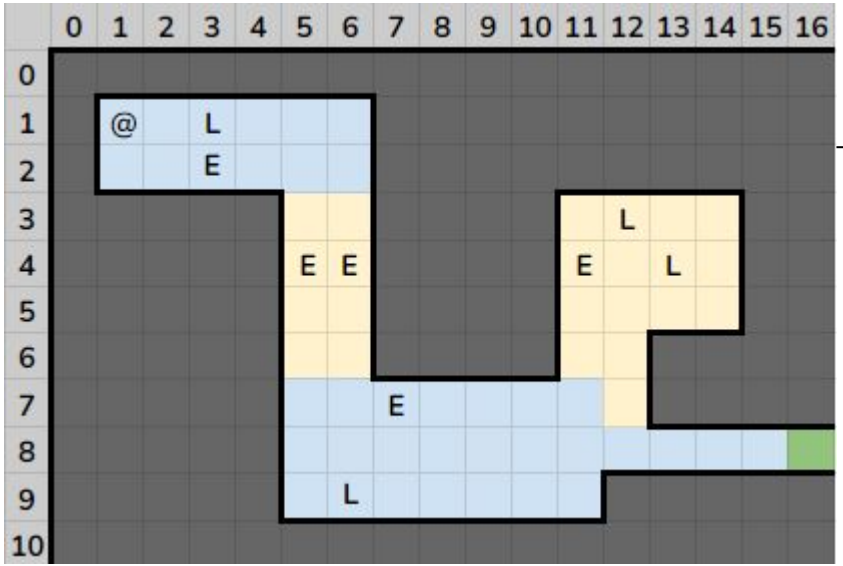
This function is the entry point of the game and it executes the main gameplay loop.

handler:

This handler will catch and handle keyboard and timer interrupts.

buildPaths

This function adds the in-memory representation of the path positions in a 2D array of 32-bit integers.



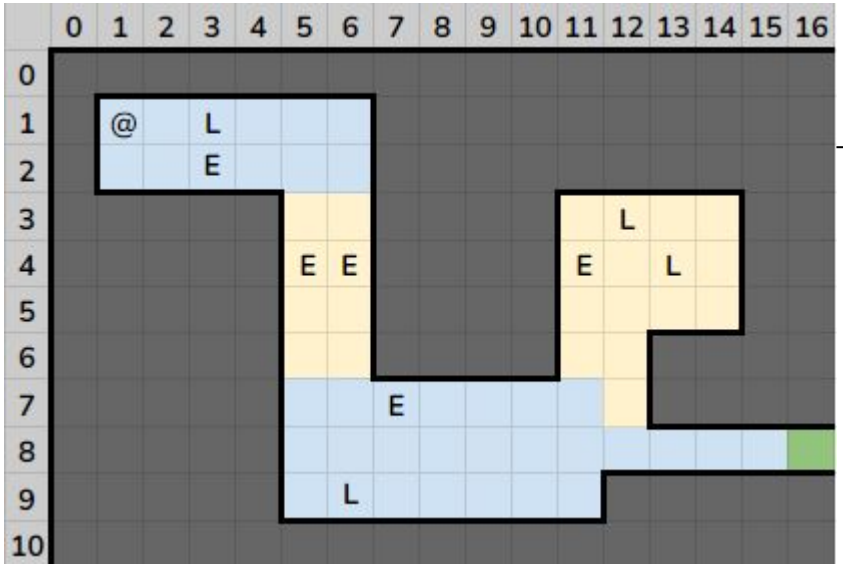
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 …]

buildLootOrEnemies

This function adds the loot or hidden enemies (depending on the input arguments) to the 2D representation of the dungeon array.



[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

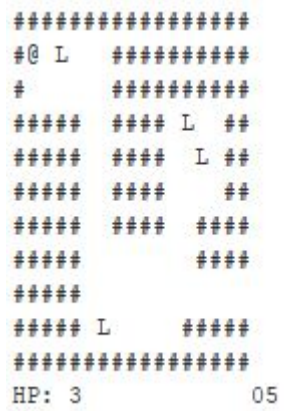
0 1 1 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 …]

displayDungeon

This function handles the logic to print the map to the MMIO display.

2D dungeon array:

```
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 … ]
```



getDestination

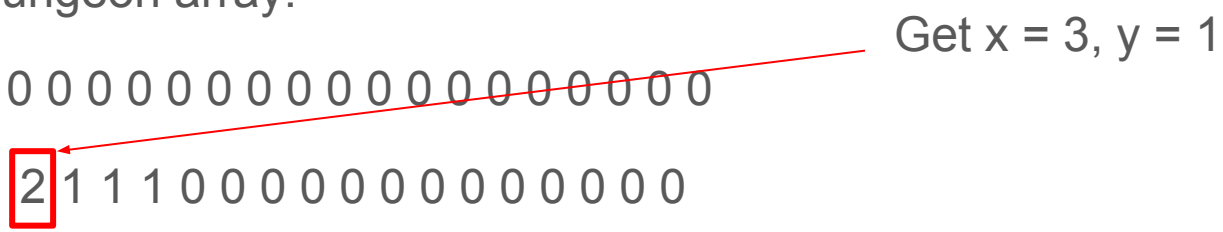
This function returns what type of element is located at a given x,y point in the 2D representation of the map array.

2D dungeon array:

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 …]



replacePoint

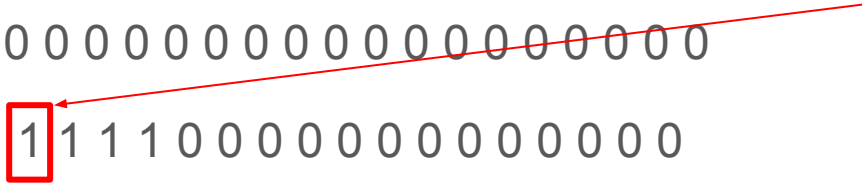
This function replaces the value at a given x,y point in the 2D representation of the map array with a new value.

2D dungeon array:

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 …]



Functions provided to you in dungeon.s

printStr

Prints a string to the Keyboard and Display MMIO Simulator terminal at the x,y coordinates provided as arguments.

printChar:

Prints a single character to the Keyboard and Display MMIO Simulator terminal at the x,y coordinates provided as arguments.

intToStr:

Converts at most a two digit integer into its ascii equivalent.

Testing your solution

Test dungeons

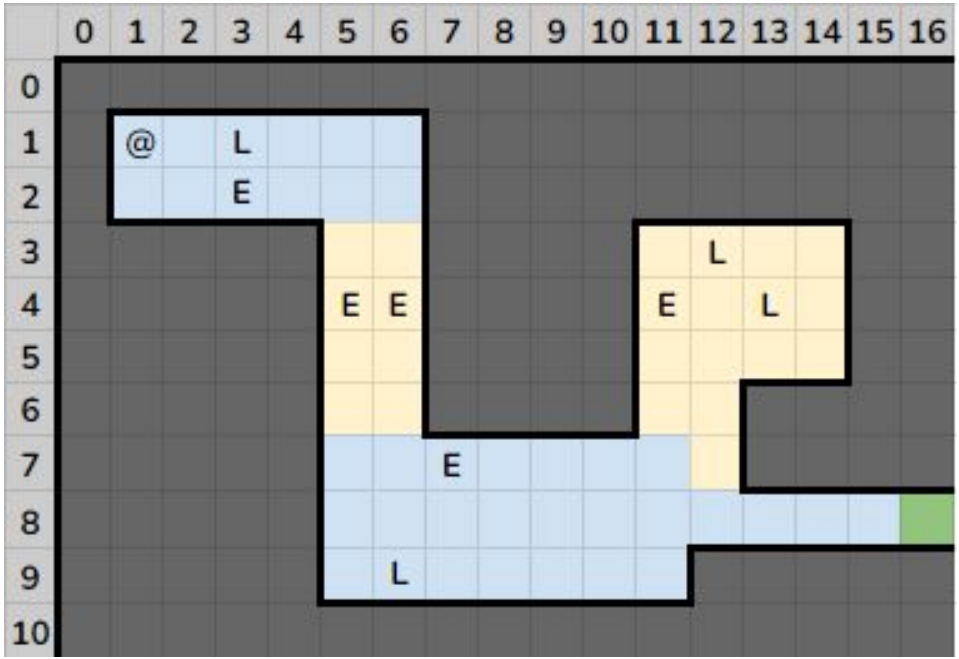
Three test dungeon inputs are provided to you:

smalldungeon.txt

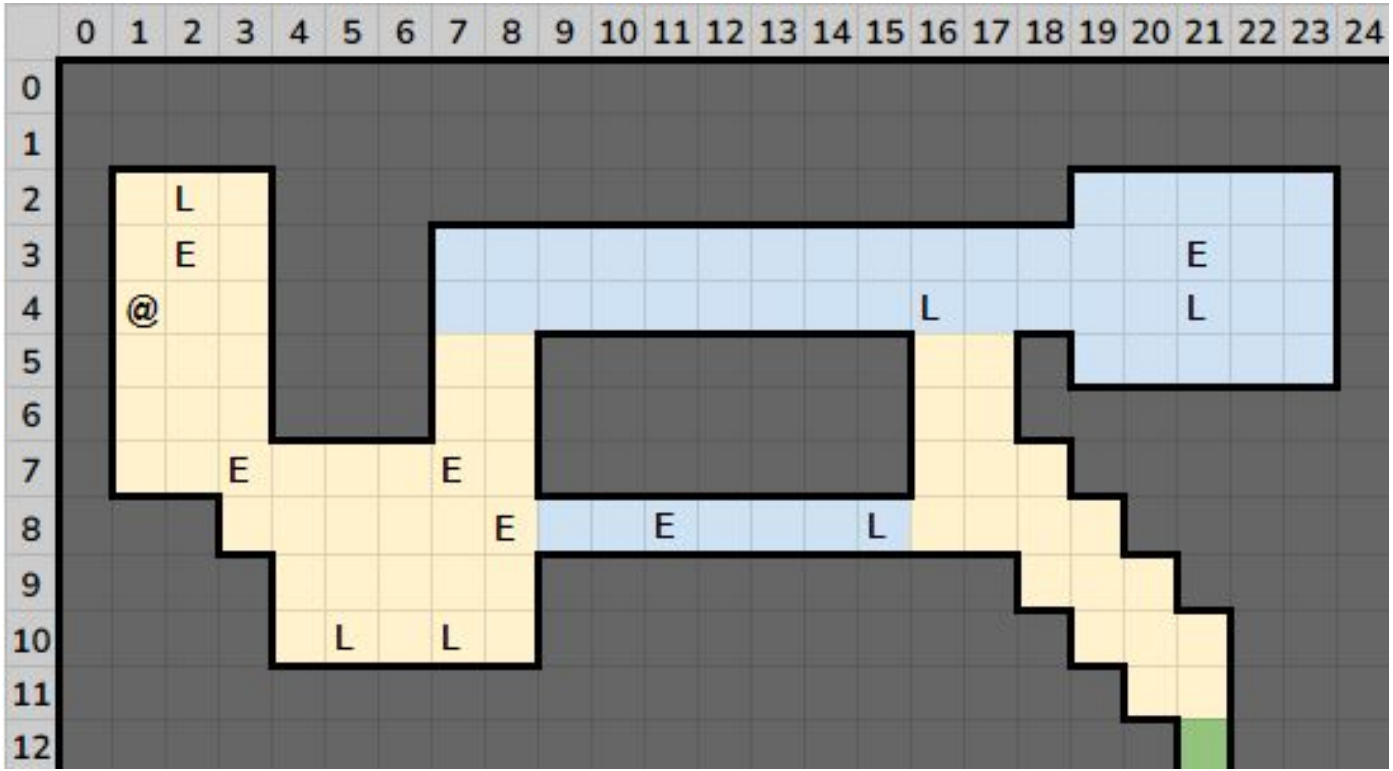
mediumdungeon.txt

largedungeon.txt

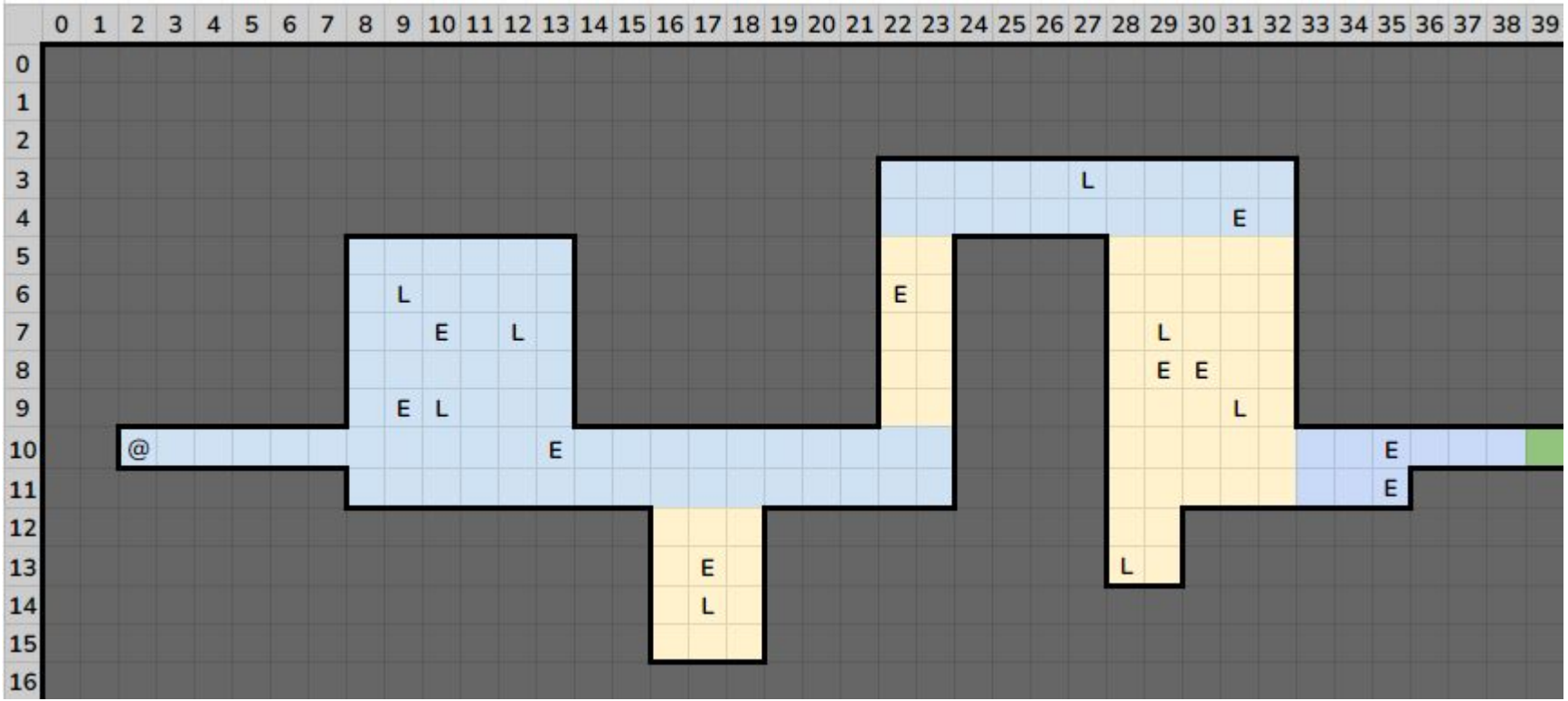
smalldungeon.txt



mediumdungeon.txt



largedungeon.txt



Hints

- Implementing the handler with timer and keyboard interrupts that work correctly will likely take longer than you would expect. Debugging can be tricky. It is highly recommended to start this lab early!