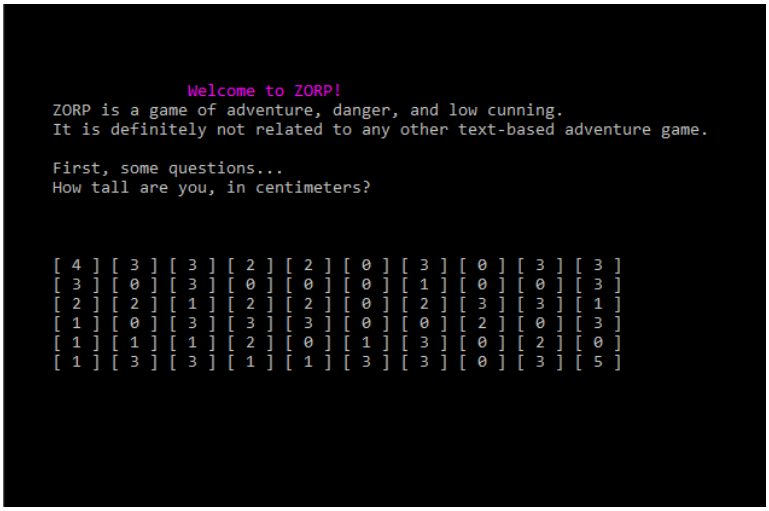
Tutorial – Arrays: Zorp part 3



Introduction:

In this tutorial, we will create and display the world map that we will use for our game.

The map will be stored in a two-dimensional array, initialized with data when the game starts, and then drawn to the console as text.

Requirements:

You will need to have completed the following previous tutorials:

- Zorp Part 1: Variables. Available under the topic *Variables and Constants*
- Zorp Part 2: Constants. Available under the topic *Variables and Constants*

Creating a Map:

For this game, we are going to create a game world. The game world will consist of a chessboard-like arrangement of rooms, and for each room we'll store some data that describes what is in that room.

In a future tutorial we'll add the ability for the player to navigate and interact with these rooms, but for now we're only concerned with simply creating the room map.

We'll use a 2D array to store our collection of rooms. The result will be a grid of rooms, with each room being connected to the rooms on either side (to simply our game we won't block out areas of the map).

Our 2D array will integers, and we'll use the value of these integers to indicate what type of room is represented by the data at a specific array index.

We'll define the following room types:

- An empty room,
- A room containing an enemy,
- A room containing treasure,
- A room containing food,
- A room that is the entrance to the maze, and
- A room that is the exit to the maze.

Apart from the entrance and exit rooms, each room in the array will have a randomly generated type.

Now that we've defined what we want, let's put that in code by first creating a series of constant definitions to define the different room types:

```
#include <iostream>
#include <windows.h>

const char* ESC = "\x1b";
const char* CSI = "\x1b[";

const char* TITLE = "\x1b[5;20H";
const char* INDENT = "\x1b[5C";
const char* YELLOW = "\x1b[93m";
const char* MAGENTA = "\x1b[95m";
const char* RESET_COLOR = "\x1b[0m";
const char* SAVE_CURSOR_POS = "\x1b[s";
const char* RESTORE_CURSOR_POS = "\x1b[u";

void main() {
	const int EMPTY = 0;
	const int ENEMY = 1;
	const int TREASURE = 2;
	const int FOOD = 3;
	const int ENTRANCE = 4;
	const int EXIT = 5;
	const int MAX_RANDOM_TYPE = FOOD+1;

	const int MAZE_WIDTH = 10;
	const int MAZE_HEIGHT = 6;

...
```

You can see here that we have a *const* variable for each of the six types of rooms. We've also defined *MAX_RANDOM_TYPE*. This is because the entrance and exit rooms will not be random, so we want to limit which values will be randomly assigned to each room.

Finally, I've also defined the maze with and height as *const* variables. This will make it easy for us to change the maze dimensions in the future (if we ever need to). But perhaps more importantly it makes our code more readable by assigning a name to what would otherwise be a hard-coded value.

Next, we need to create the array that will store our map. Add the following line of code to your game:

```
// create a 2D array
int rooms[MAZE_HEIGHT][MAZE_WIDTH];
```

You can see how we've used the constant variables to specify the size of the rooms array.

As you would know from the lecture, you can only use variables in your array declaration like this if they are constant. To use non-constant variables like this, we'd need to create a dynamic array (which would be stored on the heap as opposed to the stack). Dynamic arrays are covered in a future session (under the *Pointers and Memory* topic), but for now simply understand that we need to know the exact size of our array at compile time.

Initializing a Map:

We've defined a 2D array (grid) that we'll use for our map, but we haven't filled it with data yet. That's our next step.

We want the first and last rooms to be the entrance and exit respectively, but the type of every other room should be randomly allocated. To achieve this we'll use the *rand()* function (short for random).

The *rand()* function uses a pseudo-random number generator to output a seemingly random number. I say 'seemingly random' because creating truly random numbers in a computer is incredibly difficult.

The *rand()* function works by using a formula to return the next number in a series of numbers. If this sounds like it would return the same series of numbers every time, then you'd be right. But because *rand()* uses the previous number to calculate the next number in the series, if you start with a different initial number you get a whole new series of random numbers.

The way we usually use this function is by telling it the first number to start with. If we set this number to the current system time then we're essentially guaranteed to get a different series of random numbers every time or program runs – which is good enough for what we want.

We will loop through our 2D array and, for each index in the array, call *rand()* to get a random number. If we use the modulus operator (modulus gives the remainder of integer division) we can then limit this value between 0 and 3 inclusively (3 being our max room type).

Here's the code that achieves this:

```
#include <iostream>
#include <windows.h>
#include <random>
#include <time.h>

const char* ESC = "\x1b";
const char* CSI = "\x1b[";

const char* TITLE = "\x1b[5;20H";
const char* INDENT = "\x1b[5C";
const char* YELLOW = "\x1b[93m";
const char* MAGENTA = "\x1b[95m";
const char* RESET_COLOR = "\x1b[0m";
const char* SAVE_CURSOR_POS = "\x1b[s";
const char* RESTORE_CURSOR_POS = "\x1b[u";

void main() {
	const int EMPTY = 0;
	const int ENEMY = 1;
	const int TREASURE = 2;
	const int FOOD = 3;
	const int ENTRANCE = 4;
	const int EXIT = 5;
	const int MAX_RANDOM_TYPE = FOOD+1;

	const int MAZE_WIDTH = 10;
	const int MAZE_HEIGHT = 6;

	// Set output mode to handle virtual terminal sequences
	DWORD dwMode = 0;
	HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
	GetConsoleMode(hOut, &dwMode);
	dwMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
	SetConsoleMode(hOut, dwMode);

	int height = 0;
	char firstLetterOfName = 0;
	int avatarHP = 0;

	// create a 2D array
	int rooms[MAZE_HEIGHT][MAZE_WIDTH];

	srand(time(nullptr));

	// fill the arrays with random room types
	for (int y = 0; y < MAZE_HEIGHT; y++)
	{
		for (int x = 0; x < MAZE_WIDTH; x++) {
			rooms[y][x] = rand() % MAX_RANDOM_TYPE;
		}
	}
	// set the entrance and exit of the maze
	rooms[0][0] = ENTRANCE;
	rooms[MAZE_HEIGHT-1][MAZE_WIDTH-1] = EXIT;

...
```

srand() is the random seed function. This function is what tells the random function which starting value to use. Here we've set it to the current system time by passing *null* to the *time()* function.

We then have a nested *for* loop which will step through each value in our array, assigning a random value at each index.

Notice the way we've used the modulus operator *%* to make sure the value returned by *rand()* is between 0 and 3 inclusively.

Finally, we set the first and last value in our maze to the entrance and exit respectively.

Drawing the Map:

Our final task is to print the map to the screen.

Because we're using *virtual terminal sequences* now, we have the ability to jump around the console window adding, removing, and updating text as we need.

This actually simplifies our program a little because as long as our map doesn't change, we don't need to redraw it. (Imagine if we cleared the whole screen using *system("cls")*. We'd need redraw the map after each clear!)

To draw the map we'll use the following code:

```
	for (int y = 0; y < MAZE_HEIGHT; y++)
	{
		std::cout << INDENT;
		for (int x = 0; x < MAZE_WIDTH; x++) {
			std::cout << "[ " << rooms[y][x] << " ] ";
		}
		std::cout << std::endl;
	}
```

At the moment, we're simply writing the integral value of the room type (so empty rooms appear as '0', etc.). We'll look at making this a bit fancier in a future tutorial.

We've now got all our pieces in place. The last task is updating the rest of our program to accommodate our new changes.

Putting it all Together:

Below is the full program. I've highlighted all of the changes between this program and the program we ended with at the end of the previous tutorial.

As you read through this code, notice how we reset the cursor and delete and insert lines so that only text that is actually changing is removed and redrawn.

You'll also notice that we now clear the *std::cin* buffer before we read any input. This should be standard practice – you can never tell what the user might have typed in at any point in time, so you should always assume the input buffer might contain junk and clean it before you attempt to read any kind of input from the command line.

```
#include <iostream>
#include <windows.h>
#include <random>
#include <time.h>

const char* ESC = "\x1b";
const char* CSI = "\x1b[";

const char* TITLE = "\x1b[5;20H";
const char* INDENT = "\x1b[5C";
const char* YELLOW = "\x1b[93m";
const char* MAGENTA = "\x1b[95m";
const char* RESET_COLOR = "\x1b[0m";
const char* SAVE_CURSOR_POS = "\x1b[s";
const char* RESTORE_CURSOR_POS = "\x1b[u";

void main() {
	const int EMPTY = 0;
	const int ENEMY = 1;
	const int TREASURE = 2;
	const int FOOD = 3;
	const int ENTRANCE = 4;
	const int EXIT = 5;
	const int MAX_RANDOM_TYPE = FOOD+1;

	const int MAZE_WIDTH = 10;
	const int MAZE_HEIGHT = 6;

	// Set output mode to handle virtual terminal sequences
	DWORD dwMode = 0;
	HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
	GetConsoleMode(hOut, &dwMode);
	dwMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
	SetConsoleMode(hOut, dwMode);

	int height = 0;
	char firstLetterOfName = 0;
	int avatarHP = 0;

	// create a 2D array
	int rooms[MAZE_HEIGHT][MAZE_WIDTH];

	srand(time(nullptr));
```

```
	// fill the arrays with random room types
	for (int y = 0; y < MAZE_HEIGHT; y++)
	{
		for (int x = 0; x < MAZE_WIDTH; x++) {
			rooms[y][x] = rand() % MAX_RANDOM_TYPE;
		}
	}
	// set the entrance and exit of the maze
	rooms[0][0] = ENTRANCE;
	rooms[MAZE_HEIGHT-1][MAZE_WIDTH-1] = EXIT;

	std::cout << TITLE << MAGENTA << "Welcome to ZORP!" << RESET_COLOR <<
std::endl;
	std::cout << INDENT << "ZORP is a game of adventure, danger, and low
cunning." << std::endl;
	std::cout << INDENT << "It is definitely not related to any other text-based
adventure game." << std::endl << std::endl;

	std::cout << INDENT << "First, some questions..." << std::endl;

	// save cursor position
	std::cout << SAVE_CURSOR_POS;

	// output the map
	std::cout << std::endl;
	std::cout << std::endl;
	std::cout << std::endl;
	std::cout << std::endl;
	for (int y = 0; y < MAZE_HEIGHT; y++)
	{
		std::cout << INDENT;
		for (int x = 0; x < MAZE_WIDTH; x++) {
			std::cout << "[ " << rooms[y][x] << " ] ";
		}
		std::cout << std::endl;
	}

	// move the cursor back to the top of the map
	std::cout << RESTORE_CURSOR_POS;
	std::cout << INDENT << "How tall are you, in centimeters? " << INDENT
			<< YELLOW;

	std::cin >> height;
	std::cout << RESET_COLOR << std::endl;

	if (std::cin.fail()) {
		std::cout << INDENT << "You have failed the first challenge and are
			eaten by a grue.";
	}
	else {
		std::cout << INDENT << "You entered " << height;
	}

	// clear input buffer
	std::cin.clear();
	std::cin.ignore(std::cin.rdbuf()->in_avail());
	std::cin.get();
```

```
	// move the cursor to the start of the 1st question
	std::cout << RESTORE_CURSOR_POS;
	// delete the next 3 lines of text
	std::cout << CSI << "3M";
	// insert 3 lines (so map stays in the same place)
	std::cout << CSI << "3L";
	std::cout << INDENT << "What is the first letter of your name? "
		  << INDENT << YELLOW;

	std::cin.clear();
	std::cin.ignore(std::cin.rdbuf()->in_avail());
	std::cin >> firstLetterOfName;
	std::cout << RESET_COLOR << std::endl;

	if (std::cin.fail() || isalpha(firstLetterOfName) == false) {
		std::cout << INDENT << "You have failed the second challenge and are
			eaten by a grue.";
	}
	else {
		std::cout << INDENT << "You entered " << firstLetterOfName;
	}
	std::cin.clear();
	std::cin.ignore(std::cin.rdbuf()->in_avail());
	std::cin.get();


	// move the cursor to the start of the 1st question, then up 1,
	// then delete and insert 4 lines
	std::cout << RESTORE_CURSOR_POS << CSI << "A" << CSI << "4M" << CSI << "4L";

	if (firstLetterOfName != 0) {
		avatarHP = (float)height / (firstLetterOfName * 0.02f);
	}
	else {
		avatarHP = 0;
	}

	std::cout << INDENT << "Using a complex deterministic algorithm, it has been
		calculated that you have " << avatarHP << " hit point(s)." <<
		std::endl;

	std::cout << std::endl << INDENT << "Press 'Enter' to exit the program.";

	std::cin.clear();
	std::cin.ignore(std::cin.rdbuf()->in_avail());
	std::cin.get();
}
```