# Tutorial – Structures: Zorp part 8

## Introduction:

In this tutorial will add the *Point2D* structure discussed in the lecture as a convenient way of passing coordinates to the various functions in our game.

## 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.*
- Zorp Part 3: Arrays. Available under the topic *Arrays.*
- Zorp Part 4: Conditional Statements. Available under the topic *Conditional Statements*.
- Zorp Part 5: Loops. Available under the topic *Loops*.
- Zorp Part 6: Functions. Available under the topic *Functions*.
- Zorp Part 7: Character Arrays. Available under the topic *Arrays*.

## The Point2D Structure:

The structure itself is relatively simple, and taken straight from the explanation in the lecture for this session:

```
struct Point2D {
    int x;
    int y;
};
```

You should place the structure at the top of your *main.cpp* file, outside of any functions (so it has global scope). I prefer to place mine after the constant variable declarations.

We want to use this *Point2D* structure to store the player's location, instead of using the individual *playerX* and *playerY* variables.

The player position variables are defined in the *main()* function, so update that function as follows:

```
void main() {
    // create a 2D array
    int rooms[MAZE_HEIGHT][MAZE_WIDTH];

    bool gameOver = false;
    int playerX = 0;
    int playerY = 0;

    Point2D player = { 0, 0 };

    ...
```

In order to pass the new *player* variable around to our various functions, we'll need to change the following function declarations:

```
void drawRoom(int map[MAZE_HEIGHT][MAZE_WIDTH], Point2D position)

void drawPlayer(Point2D position)

void drawValidDirections(Point2D position)
```

If you're comfortable with doing the code refactoring yourself, you should attempt that now. After changing the above function declarations to use the new *Point2D* structure, the errors you'll receive in Visual Studio should lead you to the changes you'll need to make in the rest of your code.

If you need further guidance with making these changes, or would like to double-check your own work, continue reading.

## Passing the Point2D Structure:

We need to modify four function so that we can correctly use (x,y) coordinate pairs throughout our program and treat them as a single variable.

In addition to the three functions listed above, we'll also need to update the body of the *drawMap()* function (as it calls the *drawRoom()* function during its execution).

The following code extract lists the updates to these four functions. Copy the changes to these functions in your game:

```
void drawRoom(int map[MAZE_HEIGHT][MAZE_WIDTH], Point2D position) {
    // find the console output position
    int outX = INDENT_X + (6 * position.x) + 1;
    int outY = MAP_Y + position.y;

    // jump to the correct location
    std::cout << CSI << outY << ";" << outX << "H";
    // draw the room
    switch (map[position.y][position.x]) {
    case EMPTY:
        std::cout << "[ " << GREEN << "\xb0" << RESET_COLOR << " ] ";
        break;
    case ENEMY:
        std::cout << "[ " << RED << "\x94" << RESET_COLOR << " ] ";
        break;
    case TREASURE:
        std::cout << "[ " << YELLOW << "$" << RESET_COLOR << " ] ";
        break;
    case FOOD:
        std::cout << "[ " << WHITE << "\xcf" << RESET_COLOR << " ] ";
        break;
    case ENTRANCE:
        std::cout << "[ " << WHITE << "\x9d" << RESET_COLOR << " ] ";
        break;
    case EXIT:
        std::cout << "[ " << WHITE << "\xFE" << RESET_COLOR << " ] ";
        break;
    }
}

void drawMap(int map[MAZE_HEIGHT][MAZE_WIDTH]) {
    Point2D position = { 0, 0 };

    // reset draw colors
    std::cout << RESET_COLOR;
    for (position.y = 0; position.y < MAZE_HEIGHT; position.y++) {
        std::cout << INDENT;
        for (position.x = 0; position.x < MAZE_WIDTH; position.x++) {
            drawRoom(map, position);
        }
        std::cout << std::endl;
    }
}

void drawPlayer(Point2D position) {
    Point2D outPos = {
        INDENT_X + (6 * position.x) + 3,
        MAP_Y + position.y };

    // draw the player's position on the map
    // move cursor to map pos and delete character at current position
    std::cout << CSI << outPos.y << ";" << outPos.x << "H";
    std::cout << MAGENTA << "\x81" << RESET_COLOR;
}

void drawValidDirections(Point2D position) {
    // reset draw colors
    std::cout << RESET_COLOR;
    // jump to the correct location
    std::cout << CSI << MOVEMENT_DESC_Y + 1 << ";" << 0 << "H";
    std::cout << INDENT << "You can see paths leading to the " <<
        ((position.x > 0) ? "west, " : "") <<
        ((position.x < MAZE_WIDTH - 1) ? "east, " : "") <<
        ((position.y > 0) ? "north, " : "") <<
        ((position.y < MAZE_HEIGHT - 1) ? "south, " : "") << std::endl;
}
```

The last thing to do is to update the code that calls each function, changing the function arguments from *playerX, playerY* to simply *player*.

Again, if you feel confident in making these changes by yourself, go ahead and do that now. If you feel you need guidance or want to check your own code against the tutorial, continue reading.

We only need to update the *main()* function, as all these *draw()* functions are only called from within the main game loop.

Update your *main()* function as follows:

```
void main() {
    // create a 2D array
    int rooms[MAZE_HEIGHT][MAZE_WIDTH];
    bool gameOver = false;
    Point2D player = { 0, 0 };

    if (enableVirtualTerminal() == false) {
        std::cout << "The virtual terminal processing mode could not be activated."
            << std::endl;
        std::cout << "Press 'Enter' to exit." << std::endl;
        std::cin.get();
        return;
    }

    initialize(rooms);
    drawWelcomeMessage();
    // output the map
    drawMap(rooms);
    // game loop
    while (!gameOver) {
        drawRoomDescription(rooms[player.y][player.x]);

        drawPlayer(player);

        if (rooms[player.y][player.x] == EXIT) {
            gameOver = true;
            continue;
        }

        // list the directions the player can take
        drawValidDirections(player);

        int command = getCommand();
        // before updating the player position, redraw the old room character over
        // the old position
        drawRoom(rooms, player);

        // update the player's position using the input movement data
        switch (command) {
        case EAST:
            if (player.x < MAZE_WIDTH - 1)
                player.x++;
            break;
```

(continued on next page)

```
        case WEST:
            if (player.x > 0)
                player.x--;
            break;
        case NORTH:
            if (player.y > 0)
                player.y--;
            break;
        case SOUTH:
            if (player.y < MAZE_HEIGHT - 1)
                player.y++;
            break;
        case LOOK:
            drawPlayer(player);
            std::cout << EXTRA_OUTPUT_POS << RESET_COLOR << "You look around, but
                see nothing worth mentioning" << std::endl;
            std::cout << INDENT << "Press 'Enter' to continue.";
            std::cin.clear();
            std::cin.ignore(std::cin.rdbuf()->in_avail());
            std::cin.get();
            break;
        case FIGHT:
            drawPlayer(player);
            std::cout << EXTRA_OUTPUT_POS << RESET_COLOR << "You could try to
                fight, but you don't have a weapon" << std::endl;
            std::cout << INDENT << "Press 'Enter' to continue.";
            std::cin.clear();
            std::cin.ignore(std::cin.rdbuf()->in_avail());
            std::cin.get();
            break;
        default:
            // the direction was not valid,
            // do nothing, go back to the top of the loop and ask again
            drawPlayer(player);
            std::cout << EXTRA_OUTPUT_POS << RESET_COLOR << "You try, but you just
                can't do it." << std::endl;
            std::cout << INDENT << "Press 'Enter' to continue.";
            std::cin.clear();
            std::cin.ignore(std::cin.rdbuf()->in_avail());
            std::cin.get();
            break;
        }
    } // end game loop

        // jump to the correct location
    std::cout << CSI << PLAYER_INPUT_Y << ";" << 0 << "H";
    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();
}
```

That's it! Compile and execute your code to confirm no errors were introduced. If should look and perform exactly as it did before we added the *Point2D* structure.

While the functionality of the game remains the same, we've just added a change that is going to make our code much easier to extend in future tutorials.