Game Lab 01: Creating a Simple Game Environment

V. Markos
Mediterranean College

February 2, 2026

Abstract

What is this? Probably an ongoing lab for this course, where you will create simple game in C#. Do not expect spectacular graphics and all that stuff, since this course is not about that, but expect to encounter and attack several data structures and algorithm–related problems as we go into more sophisticate designs for our game. Stay tuned for more…

1 Introduction (Or Some First Comments)

Before we actually start with this lab series, some rough comments on what to expect (and what not to) from this lab:

- This is **not an introduction to games design,** but this module is such an introduction[1].
- This is, and should be treated as such, **an introduction to C# programming** and, especially, getting our hands dirty with some actual code, other than converting temperatures, checking primality and all that stuff we see in lab exercises.
- I do not have a full idea of the final version of the game, yet. And, probably, I will not have until Week 12. Many of the features will be decided on the spot, by you and me, as we discuss in class.
- However, I do have a plan on what this game will assess, in terms of data structures and relevant algorithms. Consequently, there are certain features that I have already decided to include.
- However (vol. 2), I also have a certain rough plan for the abstract game outline. There will be a simple game grid, initially covered by a Fog of War, and you will have a player exploring the grid while having to avoid computer-controlled enemies and find some relics which will be collectible and stuff like that — we will further specify features together throughout the next weeks.
- This would also have been implemented in C# last semester, but I had this damn surgery to undergo…

[1] This might be a bit mind-boggling, but you will get what is going on soon.

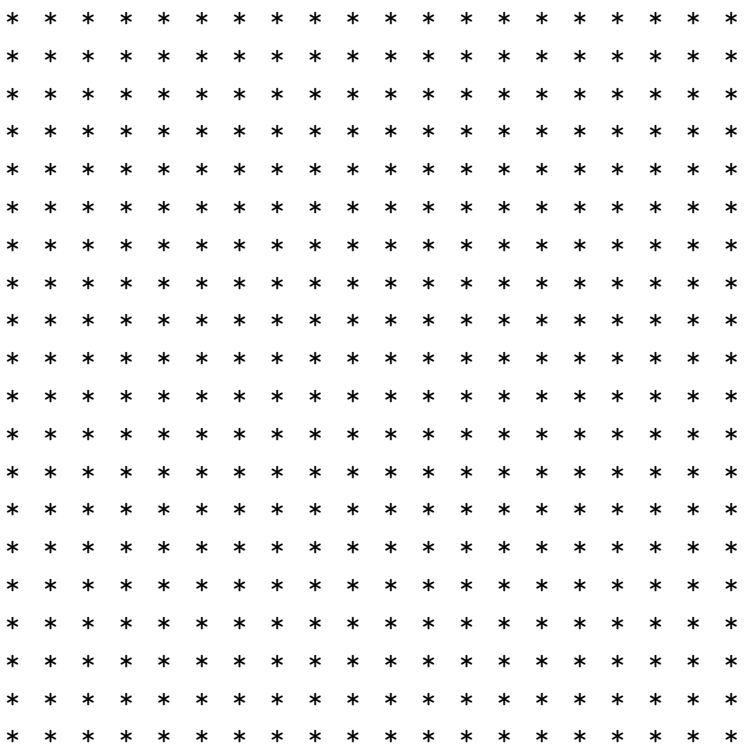

Figure 1: Our initial 20×20 game grid, covered by the Fog of War…

2 Designing A Game Grid

As a first step, we shall start by designing a game grid. So, since the grid is initially covered by the Fog of War, we will just print a 20×20 grid full of asterisks, as the one shown in Figure 1. So, here comes your first task:

Task 1: Printing the Game Grid

Write a C# function that prints a 20×20 grid as the one shown in Figure 1.

Now, what if we want in the future to allow players choose the size of the game grid? For instance, some players might want to play a quick round of our (nameless) game while some others might want to spend some time on it. It would be nice to have some predetermined grid sizes to allow the players choose from. Let's say we offer three options:

- A small, 15×15, grid;
- A medium sized, 20×20, grid, and;
- A large, 30×30, grid.

Here comes your next task:

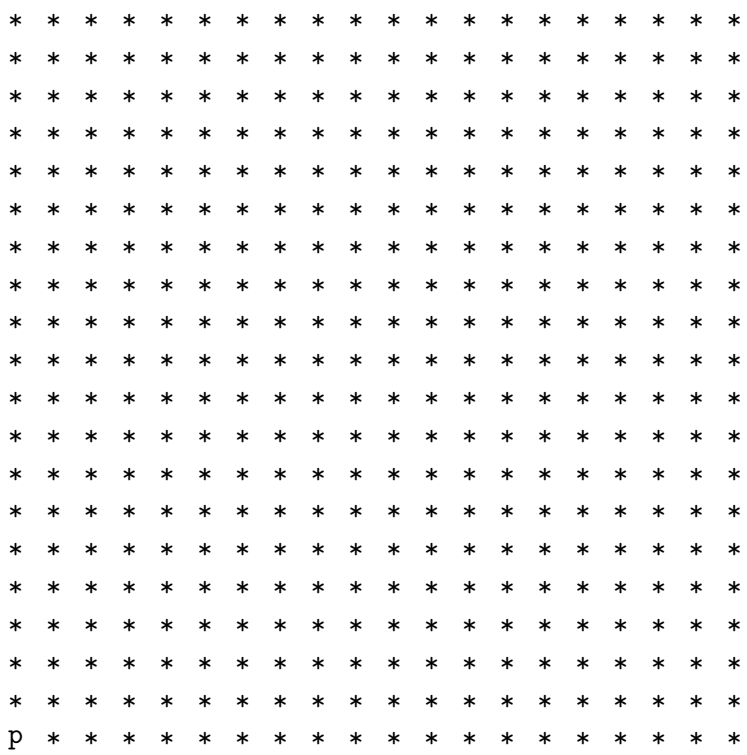

Figure 2: A 20×20 grid with our game's main character printed at the bottom left position of the grid (denoted by a p).

Task 2: Customisable Game Grids

Improve your previous version of the grid printing function to accommodate the above three sizes. You should ask the user for their preference (making sure the user provides a valid input), and draw the corresponding grid on screen. You can also allow for a default option, if you want to (to this end, C#'s default values for function arguments might come in handy).

3 Creating a Player

So far, we have implemented a simple grid printing function and provided the user with the option to choose level size. But, no player has appeared on the grid so far. So, to begin with, we should make a player show up on the grid, as shown in Figure 2.

Task 3: Printing the Main Character

Print the game's main character (which will be controlled by the player themselves) on the game grid. The character's starting position should be at the bottom left corner of the grid, as shown in Figure 2. The player position should be denoted by the lowercase letter p.

We have a grid, we have a (visible) main character, what is still missing? **Movement!** We should allow our player to move the main character around so long as they stay within the game's grids. We will allow four types of movement:

- North, i.e., upwards, which will be implemented whenever the player enters

the character `n`.

- South, i.e., downwards, which will be implemented whenever the player enters the character `s`.
- West, i.e., leftwards, which will be implemented whenever the player enters the character `w`.
- East, i.e., rightwards, which will be implemented whenever the player enters the character `e`.

In all cases, the main character should move by **one cell**. These are the only possible moves that are allowed, so here comes your next task:

Task 4: Moving Around

Implement character movement as described above. You should take into account the following:

- No other form of movement, e.g., diagonally or skipping cells is allowed.
- You should properly check for and prohibit movement outside the grid's boundaries. In such cases, the user should be informed by an appropriate message and a new instruction should be asked for.
- You are free to choose whether you will accept both lowercase (`nswe`) and / or uppercase (`NSWE`) movement instructions.

In the above, you should take into account how you will properly store the main character's location, i.e., which C# data structure is the most appropriate for that purpose? Also, how are you going to implement the game loop?

4 Further Considerations

So far, we have implemented some simple yet fully functional (hopefully) game mechanics. What follows below is not necessary to be implemented for the next labs, but serves more as some further thoughts that could help you work on some interesting problems. Here are two simple tasks to get you started:

Task 5: Circular Movement (Grid Wrapping)

In Task 4 we prohibited movement beyond the grid boundaries. However, this is not the only way to handle boundaries in our game. Another option could be to allow the player move from one side of the grid to the other, i.e., west from the westmost end of the grid would make the main character appear on the same row, just at its eastmost part, moving south from the southmost part of the grid would result to the main character appearing on the same column, just on the southmost part and so on.
Bearing in mind the above, implement the above circular movement across the game board. Which one do you prefer as the default way? If you want to, you can allow players to choose how their character will move around the grid, i.e., whether movement will be restricted, as implemented in Task 4 or unrestricted as here.

Playing around and testing our game so far, you might have observed that re–printing the grid every time is a bit cumbersome and also results to a really stacked output on the terminal. You next task requires you to devise ways to avoid printing the grid all the time.

Task 6: Overwriting the Game Grid

Using special ASCII / Unicode characters, and especially the carriage return character, `\r`, improve the grid printing function so that each time the main character moves, the grid is not printed below the previous version of the grid but on top of it, essentially saving up space on the terminal's feed.