Programming Lab 02: Collecting Relics

V. Markos
Mediterranean College

February 16, 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 Last Time Recap

Last time we implemented a game with the following features (at least):

- A customisable game grid, initially covered by a fog of war (i.e., `*`), which could have any of the following sizes:
  - 15×15;
  - 20×20;
  - 30×30.
- A player printed on screen, initially positioned at the bottom left corner of the grid.
- Player movement using `nswe` or `wasd` keys to move around the grid.
- Relevant movement restrictions, i.e., the player cannot cross grid borders (or, movement wraps around the grid, whichever you prefer).
- Optionally, you might have also your grid printing over its previous version to avoid overloading the terminal with unnecessary past versions of the grid.

2 Hiding Relics

So far, our main character can move around the grid exploring the grid, but last time we did not implement something really crucial: lifting the fog of war. Evidently, once our player has passed through a certain grid cell, we should denote this cell as explored, letting the player know that this cell has already been visited. So, here comes your first task for this lab:

```
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
* * * * * * * * * * * * * * *
. . * * * * * * * * * * * * *
. . * * * * * * * * * * * * *
. . * * * * * * * * * * * * *
. . . . . * * * * * * * * * *
. . . . p * * * * * * * * * *
. . . . . * * * * * * * * * *
. . . . . * * * * * * * * * *
```

Figure 1: A small, 15 × 15 grid where the player has explored some of its parts. Explored parts are indicated by a "`.`".

Task 1: Lifting the FoW

You should keep track of player movement around the grid and print any already visited cells using a `.`(dot). You should choose an appropriate C# data structure for the visited cells in your grid (take into consideration which operations you will need to perform most often on them). So, each cell might now be in either of the following states:

- Unexplored, covered by the FoW, indicated by a `*`.
- Explored, indicated by a dot (`.`).
- Containing the player, indicated by a `p`.

For instance, a small game grid where the player has explored some of its bits is shown in Figure 1.

Leaving Task 1 behind us, we move to a more juicy part of our game: hiding relics. To this end, let us first sketch a rough idea on how things should work:

- Our main character wanders around the grid, exploring the world.
- At random places throughout the grid, collectible relics are hidden.
- Whenever the player encounters a relic, the relic is considered as collected and a point is awarded to the player (we might make use of this point system in subsequent labs somehow).

Bearing the above in mind, here comes your next task:

```
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . + . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . + . . . . .
. . . . . . . . + . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . .
. . . . . . . . . . . . . . +
. . . . + . . . . . . . . . .
. . . . . . + . . . . . . . .
. . . . . . . . . . . . . . .
```

Figure 2: Debugging relics positioning, just with fewer relics than expected. In the above, relics are denoted by a "plus" (+).

Task 2: Placing Relics

Implement any relevant functionality that will allow you to place some relics on the game grid. As a rule of thumb, place as many relics as the number of rows / columns for each size of the grid. So, in a 15×15 grid you should place 15 relics, on a 20×20 grid place 20 relics, etc. The relics should be placed **at random** so starting a new game from scratch results to different relic positioning (to avoid players reverse engineering our game).
Some things to take into account:

- You can use C#'s `random` module and all relevant functions it offers for randomly generating numbers.
- There are several pitfalls you will need to **avoid**, the most common probably being **generating less relics** than you might think (if you do not utilise `random` utilities appropriately).
- Choose an appropriate representation for your relics in terms of C# data structures! Bear in mind that in subsequent tasks you will need to frequently check whether a relic is in a certain cell or not!

In order to facilitate debugging, you can create a function that prints a grid containing only the relics, e.g., as shown in Figure 2.

3 Collecting Relics

Having placed our relics on the map, we should now implement any infrastructure related to actually collecting relics.

Task 3: Collecting Relics

Implement any required functionality regarding collecting relics. Your implementation should address, at least, the following:

- Whenever the main character moves onto a relic, it should be collected, keeping track of how many relics have been collected so far.
- The player should be informed about such events by an appropriate message on screen.
- The total number of collected relics should be displayed at some reasonable place on the game's screen — this will vary depending on whether you print every version of the game grid or overwrite previous ones.

What if our player collects all available relics? For now this will be the "Game Over" scenario of our game. So, here comes your last (small) task for today:

Task 4: Game Over

Implement any game over related functionality that informs the player that they have collected all collectible relics, while you should also offer them the choice to start a new game or quit the game at all. Consider creating a `GameUI` class which includes and offers all utilities related to how one interacts with the game, to separate game logic from the UI itself.

4 Further Considerations

One possible twist regarding relic collection could be the introduction of relics of various values. For instance, a simple relic could equal 1 point while a super–relic could offer 3 points.

Task 5: Super Relics

Implement a differentiated scoring regarding relics. You should offer at least two different relic types (feel free to improvise regarding their names). This might also affect the way a game ends. For instance, in a small, 15×15, grid, the primary objective might be to collect 10 points, be it by collecting 10 simple relics or 5 super relics or a combination of those, and, possibly, even more complex relic types. Explain your design choices appropriately!

What if a player does not want to collect a relic? For instance, some players might prefer to first explore the game map and then collect any relics they encounter following an optimal path for doing so. So, here comes another interesting task:

Task 6: Dropping Relics

Allow players to drop relics, in the sense of ignoring relics whenever they encounter them. This can be implemented as a short in–game dialogue that is displayed below the game grid, where the player provides some sort of Y/N (yes/no) response to a simple prompt. You can also keep track of rejected relics and any relevant statistics.