```
You are in a dungeon. You must find your way out.
You can go South. (s)
s
There is a spider web here.
You can go North or East. (n/e)
e
There is a small pool of water here.
You can go North, East, or West. (n/e/w)
n
You are standing in front of a wooden door.
You can go south or open the door. (s/open door)
```

Emily Davis and Kaisa Taipale

with Cynthia Qualey and Doug Thorpe

Coder's Quest

Create your *own* adventure

Emily Davis and Kaisa Taipale

Contents

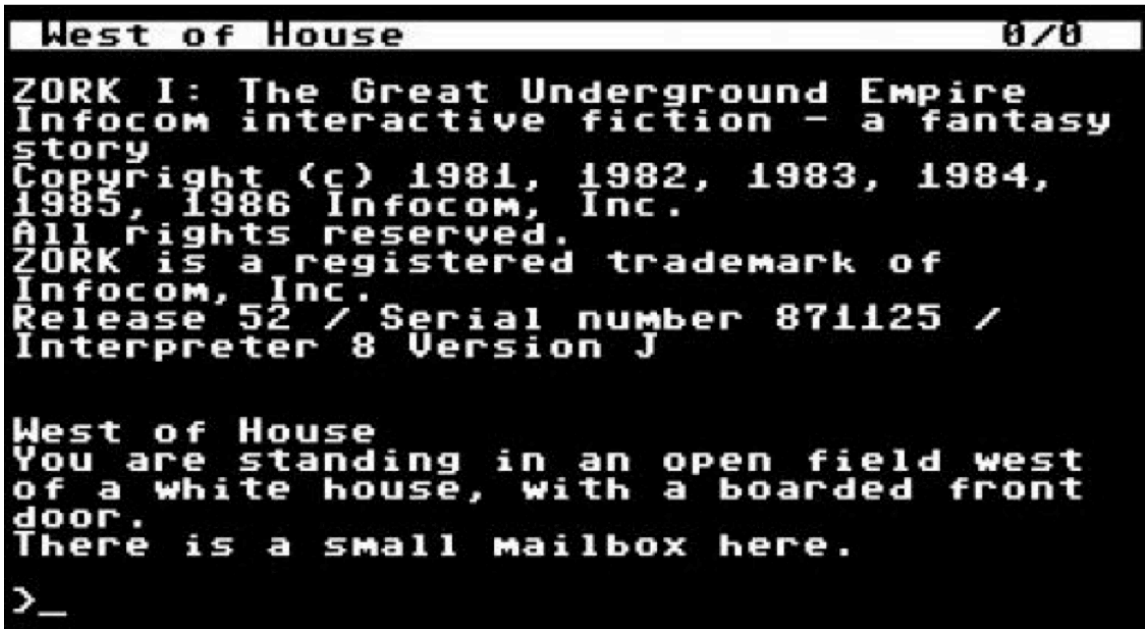
Preface

Emily started teaching about coding through text adventures at That Conference, where she gave a presentation to an audience of kids, parents, and other adults. At Startup Weekend Twin Cities 7, she pitched the idea of taking this public, and here's the result! Emily's team included Cynthia Qualey on business, Doug Thorpe on computers, and Kaisa Taipale writing and coding. Joey Vazquez did our fabulous logo (check him out at at his site). Learn to code through writing your own adventure, and have fun!

Chapter 1

What is a text adventure?

You want to build your own text adventure... but what is that? A *text adventure* game is a game that takes you through an imaginary world, step by step. You can only navigate by typing your commands in as text. Here's an example of what the beginning of the most classic text adventure ever:



This game is called Zork and it's more than 35 years old – but you can still play it today online. It's fun to explore other peoples' words, but even better to create your own.

Our starting point is a basic text adventure written by Emily Davis.

Here we will show you a process for creating your own world and your own text adventure using C#. You'll

- play with Emily's code
- make a simple map for your first world
- learn about how to write the code to move around your simple world
- put together your game
- and then level up!

Good luck on your Coder's Quest!

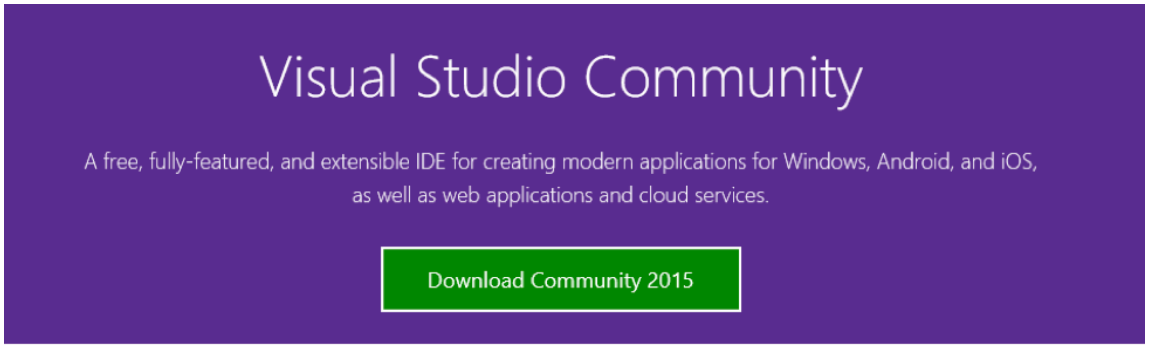
Chapter 2

Level 0: Setting up

If you want to do fun stuff with text adventures you need to set up your environment! Emily works with C# in Microsoft Visual Studio, Community Edition. Downloading will be in section 2.1 and setup will be in section 2.2.

2.1 Download Visual Studio

You will need to be using a Windows computer to code in C#. Download Visual Studio Community Edition from this site.

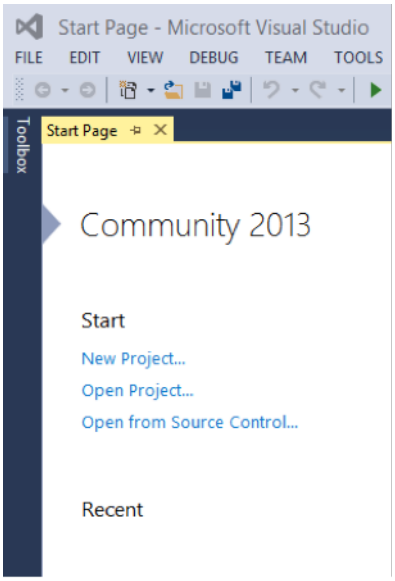


When you click on "Visual Studio Community," you'll download an installer program file called `vs_community.exe`. After it is downloaded, run vs_community.exe and follow the prompts to install Visual Studio on your Windows computer. After it is installed, start the Visual Studio program by clicking

on the Visual Studio icon in the Start Menu (or search for Visual Studio under All apps). You may wish to pin it to the start menu or the task bar for ease of access.

2.2 Set up your first file

When you open Visual Studio Community, you'll get something that looks like this:



Download Emily's game here. Use the "Download ZIP" link on the right-hand side of the page. Open it using Visual Studio by clicking the `CodersQuestChp1.sln` file that appears.

2.3 Play Emily's game

Give the game a try. Run the program by hitting the green "play" near the top of the screen.

See what commands you can use and which ones don't work: you enter commands by typing your instruction and then pressing enter. (Type commands

like this: "n" for north, or "open door" to open a door, but without the quotation marks.)

How do *you* want to change the game?

2.4 Play with Emily's code

Go back to the code Emily wrote and think about what you wanted to change. Do you want to change the screen output, or the map? How are you going to do this?

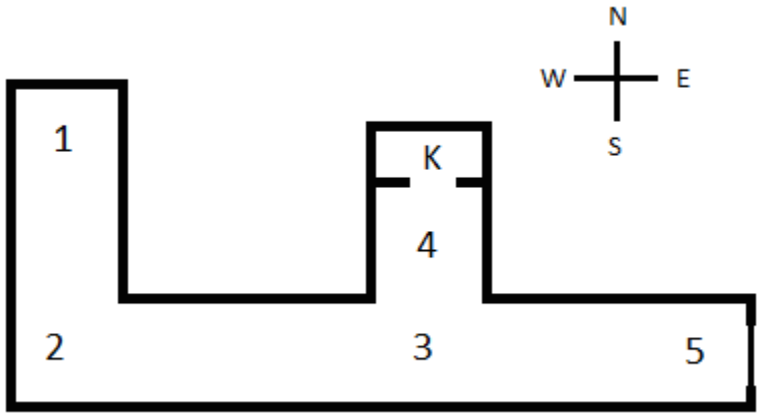
It's easy to change the landmarks along the way by just carefully changing the text in the file. Try changing the "dirty window" to a "brick wall" and then save. Run the program again and see if your change works!

Next step: start writing your own.

Chapter 3

Level 1: Map your world: planning a game

The first step for your first game is planning your world. Let's start simple this time: take a look at Emily's map for the game you ran in the previous chapter.

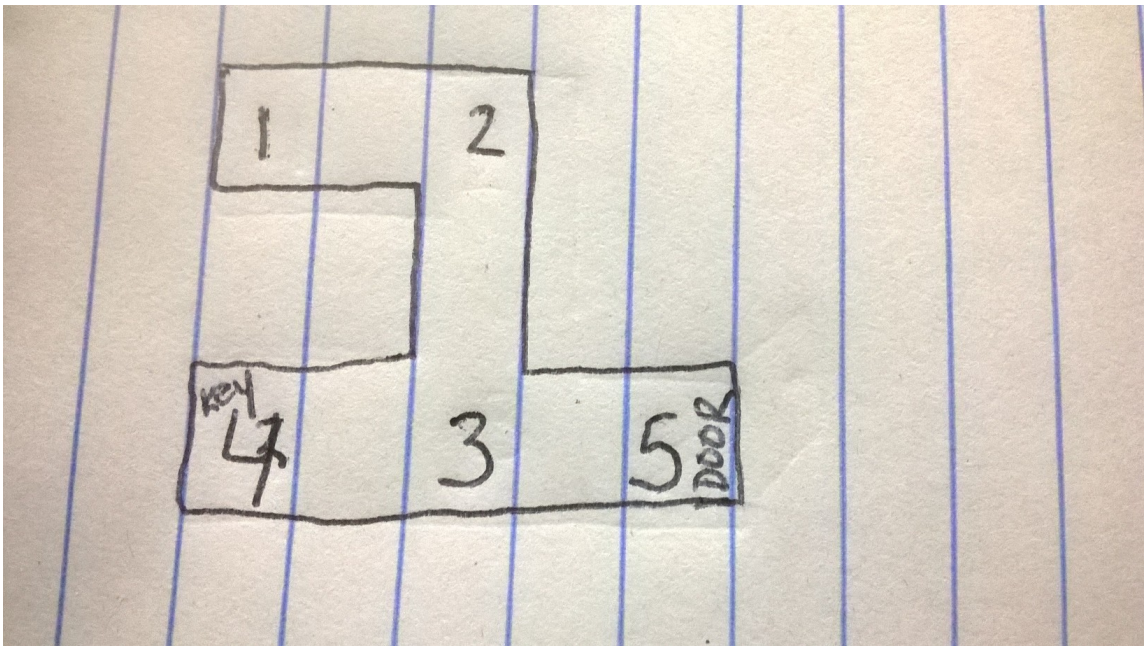


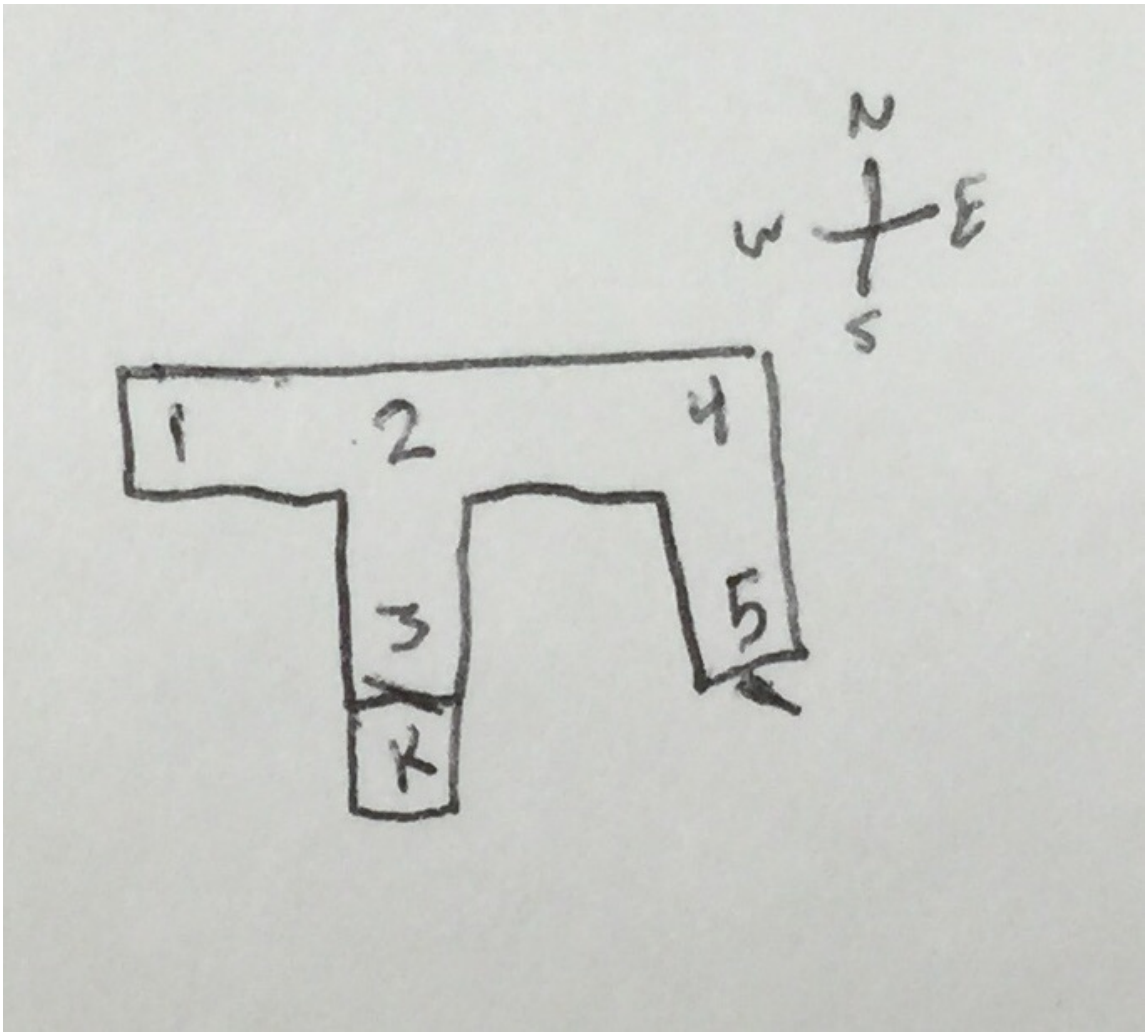
When you played the game, every time you got to a corner or an intersection on the map you arrived at a numbered position. We'll keep track of these positions in a *variable* in the next chapter, so that when you play the computer can tell you what options you have for movement and actions. Of course, these options are up to you when you code your game!

Draw a map for yourself with five numbered positions. Remember, you need a position number at every intersection, every corner, and every place you want to have an action. Also decide where you want to pick up the key and where you want to open the door. Don't worry: as you level up your skills you can make monsters and more elaborate games, but we want you to get the hang of the basics first.

Another nice thing to include on your map is a drawing of the compass directions. It will help you keep track when you are coding. (Quick question: if you go a step east and then a step west, where do you end up? Where you started!)

In the end your map should look something like one of these:




N
W
E
S
1
2
4
3
5
K

Chapter 4

Level 2: Variables and text

4.1 Three variables to start with

A variable is like a suitcase: you store values in it to keep track of them. You can change the value of your variable by putting a new value in the suitcase.

Variables store different types of things. For our text adventure we want to keep track of our position (the positions you wrote on your map) and whether we are alive. These are different types of variables, because the positions are integers (whole numbers) and whether you are still alive is a *boolean* – it is either True or False. There is one last kind of variable we need: a *string*. We need to be able to store words and sentences, so that when you type "open door" the computer can deal with your choice.

Try to find these in Emily's code:

- Integers for position look like

```
int pos = 1;  //tells us where we will be in the maze.
```

- Booleans for whether you're still alive or not look like

```
bool KeepGoing = true;  //Can we keep playing the game?
bool haskey = false;  //Do we have the key for the locked door?
```

- Strings are sneaky – we start with an empty one at the beginning by writing

```
string input = null;  //What has the user typed?
```

These are the settings we start the game with: you start at position one, you start being alive (you can KeepGoing) with no key (haskey is false), and you start with no commands yet entered for movement. When you are playing the game and type your commands to go north or east or open a door, that's when the string changes: you change the value in the suitcase named "input". Can you see where this happens in Emily's code?

4.2 Console output

When you played the game, each step gave you instructions. They are printed in C# using the Console.WriteLine command:

```
Console.WriteLine("You are trapped in a Maze.");
Console.WriteLine("You must find your way out without getting eaten by the monster.");
Console.WriteLine("You can go East. (e)");
```

You can change anything between (" and)"; to change the text that the player gets.

4.3 Taking user input

To use the variable **input**, we need to use this line of code:

```
input = Console.Readline();
```

Console.Readline is waiting for the user to hit the enter key. In our program,the line **input = Console.Readline();** takes whatever the user types into the **input** suitcase/variable.

4.4 Equals: the ultimate battle!

One thing that Emily really struggled with when she started was `=` versus `==`. When you have one equals sign, like `pos = 2`, this is saying, "pos, you now have a value of 2!" But `==` is saying, "Hey pos! Do you have a value of 2?" Take a look at Emily's game code to see where she says `=` and where she asks `==`.

Chapter 5

Level 3: If you go east...

In this section you'll learn about how to change the game to match your map.

5.1 If statements

At every intersection and in every battle, your player makes choices within your world. "If I turn left..." "If I pick up the key..."

Take a look at Emily's code for the first position on her map:

```
if (pos == 1)
                {
                    Console.WriteLine("You are in a dungeon. You must find your way out.");
                    Console.WriteLine("You can go South. (s)");
                    input = Console.ReadLine();
                    if (input == "s") //You can only go south. The computer will ignore all oth
                    {
                        pos = 2;
                    }
```

The console output from **`Console.WriteLine`** tell you that you only have one choice for movement at the beginning (and Emily nicely mentions that in the comments!). Since there is only one option, there is only one **`if`** statement, and it moves you to position 2 if you go south.

```
    if (input == "s")
    {
        pos = 2;
    }
```

At position 3, though, you've got some choices. Look at position 2. You can go north, east, or west:

```
Console.WriteLine("There is a small pool of water here.");
                    Console.WriteLine("You can go North, East, or West. (n/e/w)");
                    input = Console.ReadLine();
                    if (input == "n")
                    {
                        pos = 4;
                    }
                    else if (input == "e")
                    {
                        pos = 5;
                    }
                    else if (input == "w")
                    {
                        pos = 2;
                    }
```

The program follows Emily's map: if you go north from position 3, you arrive at position 4; else if you go east from position 2, you arrive at position 5, and so on. Since there are three options, there are three blocks of instructions. If you have multiple options, the first check will be `if` and the rest will be `else if`.

5.2 Pesky semicolons

Notice all the semicolons ; floating around. What are they for and where do they have to go?

Semicolons end a statement – they act like a period at the end of a code sentence. Every time you have an `if` or `else if` statement you need a semicolon, and you have to put the semicolon at the end of a statement like `pos = 2` rather than outside your curly brackets `}`. If your program isn't running

correctly check to see if you lost any semicolons. Visual Studio will show you a wiggly red underline, as if you made a spelling error in Word, if you lost a semicolon.

Chapter 6

Level 4: While loops

While you are alive, you can play the game... until you win or die!

6.1 One big while

Take a look at Emily's game code again. After all the lines that are automatically included by Visual Studio, it starts with a **while** loop. The while loop will end when we exit. Otherwise we will play the game forever. The loop will keep going until **keepGoing = false**, which in this game happens when you find the door and open it with the key.

```
namespace CS101Level1
{
    class Program
    {
        static void Main(string[] args)
        {
            int pos = 1;                //tells us where we will be in the maze.
            bool keepGoing = true;      //Can we keep playing the game?
            bool haskey = false;        //Do we have the key for the locked door?
            string input = null;        //What has the user typed?

            while (keepGoing == true)
            {
              // all the game instructions in here!
            }
        }
```

```
    }
}
```

This is part of the game you probably don't want to change right now. Once you finish this game you could make a more complex one with more while loops, but figure this one out first!

6.2 Troubleshooting

Inside your while statement, you might run into trouble. Check to make sure you:

- you have an if statement for every direction
- dont repeat a position (`pos 2` twice)
- and be careful with the last step: you are asking the program if the user has the key, and then asking the user if they want to use it.

Chapter 7

ASCII art

7.1 More coming soon. . . .

Chapter 8

Arrays: carrying more game items

8.1 More coming soon....

You want to be able to carry more game items. Coming soon....

Chapter 9

DRYing up your code

You might have noticed a lot of repeating yourself, a lot of copy and paste. DRY stands for "don't repeat yourself": it's about ways to make your coding simpler and faster.

9.1 More coming soon...