Cryptogram Part 4 – It Lives!

VS

WebServer

Introduction

In this assignment (the last one!) we are going to make our Crytogram game run INSIDE our web server! We will turn it into a web based game instead of a command line based game! We will reuse a LOT of the logic, really only tweaking a few things and re-writing displayWorld().

There are a few things that you need to know about web pages. They use HTML – a markup language based on tags. The tags must (except for specific exceptions) be in matched pairs. The page starts with <html> and ends with </html>. There are two parts, the header and the body. We don't need a header, so our page will look like this:

```
<html><body>Encrypted: (Our encrypted puzzle here)
 <P> Decrypted: (The current state of decryption here)
<form submit="crypt"><input type="text" name=move autofocus maxlength=2>
</input></form></html>
```

The <P> is one of the exceptions – it doesn't require a match. The form tag indicates to the browser that this is a web form – to submit to the server any data in the "input" fields. Note the input field – it has a name "move". If we were to enter "AB" into this text field and hit enter (submit), we would get this URL: 127.0.0.1/crypt?move=AB

We will be using that distinction – if the URL starts with "crypt", we will not just serve files, we will call our cryptogram code. If the URL has a ?move= in it, we will add this move to the current game and redisplay the board. If not, we will start a new game.

Detail

Copy the following items from cryptogram into webserver:

Your globals, the quote structure, getNewNode(), loadPuzzles(), getPuzzle(), shuffle(), initialization(), teardown(). We will not need main or gameloop and we will rewrite handleInput() and displayWorld(). You might want some of the logic from displayWorld(), so feel free to reuse whatever you want from it.

Make a new function, isGameOver() – it does the same work we did in displayWorld() before to detect end of game, but only generates a Boolean, it doesn't output a string.

Add a call to loadPuzzles() in the main of the web server before you start accepting connections.

In the web server, before looking for files, look to see if the URL starts with crypto. If so, instead of loading files, call a new function: handleGame().

In handleGame, there are two possibilities – this is a new game request OR this is a game request in progress. How can we tell the difference? Check for a "?" as the next part of the URL.

If the URL is just "crypt" then make a new puzzle (call getPuzzle()) then display it.

If the "?move=" is there, get the 2 character move and enter it in the decryption (just like handleInput used to). Then call isGameOver().

Now there are two more possibilities – the game is over or it is not.

If the game is not over, we display the game state (displayWorld()) and wait for another request.

If the player decrypted everything (the game is over), output a nice HTML page (mine was simple) along with a link to start over. Here is mine:

```
<html><body>Congratulations! You solved it! <a
href=\"crypto\">Another?</a></body></html>
```

You will notice (if you run this) that the web browser shows the HTML instead of rendering it to a pretty page. That's because we must tell the browser that we are sending HTML. If we don't, it assumes just text. Fortunately, that's easy. Here is my header string:

```
char *msg200 = "HTTP/1.1 200 OK\r\ncontent-type: text/html;
charset=UTF-8 \r\n\r\n";
```

You will also need this in our new version of displayWorld() – instead of outputting text to the screen, we will be sending data to the web browser. HTML is pretty easy to use. You can start with plain text, then look up some tags. One I do want you to use is the text box. Here is mine:

```
<form submit="crypt"><input type="text" name=move autofocus
maxlength=2></input></form>
```

This is telling the browser that we are making a form – something to submit. The URL is "crypt". Inside the form make a text box named move. Autofocus makes this textbox "on" by default, so you can type in it without clicking on it. The maxlength limits the user from typing more than 2 characters.

My total program ended up at 310 lines of code. My display was pretty bare bones, though. The 310 lines can get to be very large if you start adding a lot of "pretty". That's OK, though not required.

Rubric	Poor	OK	Good	Great
Comments	None/Excessive (0)	"What" not "Why", few (5)	Some "what" comments or missing some (7)	Anything not obvious has reasoning (10)
Variable/Function naming	Single letters everywhere (0)	Lots of abbreviations (5)	Full words most of the time (8)	Full words, descriptive (10)
Structure	None of: indentation matches braces {}, helper functions (0)		One of: indentation matches braces {}, helper functions (5)	Both of: indentation matches braces {}, helper functions (10)

isGameOver	Missing(0)			Detects when game is over (10)
handleGame – new game	Missing (0)			Gets a new puzzle and calls display world (5)
handleGame – existing game	Missing (0)			Updates decryption, checks for game over (5)
handleGame – Winner!	Missing (0)			Displays a reasonable indication of competing the decryption (5)
handleGame – non-winner	Missing (0)			Calls displayWorld(5)
displayWorld	Missing(0)			Prints a reasonable game board (10)
Playable	Not playable(0)			Game can be played beginning to end and a new game can be started (20)
Serving files still works	Not working (0)			Works as previous (10)