University of Liverpool - Department of Computer Science

COMP284 Scripting Languages (2016-17) -- Assignment 3: JavaScript

Your task for the third and final COMP284 assignment consists of the following parts:

1. Write a report comparing Perl, PHP, JavaScript and Java:
   - Identify three language constructs on which Perl, PHP, and JavaScript pairwise differ (excluding the example given below) and present these in tabular form. If you identify more than three language constructs in your report, then the marks awarded for the excess language constructs will be discarded (starting with the lowest mark).

     Example:

	PHP

The syn
 	
`if (condition) { } elsif (conditio`
`}`
	syntax for conditional statements is
	
	`ition) { } elseif (condition)`

   - Describe what differentiates scripting languages like Perl, PHP and JavaScript from programming languages like C, C++ and Java.

   Your answers must take up at most two A4 pages using a 12pt font. The report should have a cover sheet with your name, student id, and departmental user name. Sources, including the lecture notes, must be referenced. The list of references does not count towards the length of your answers. Make sure that you use your own words. The report must be submitted in PDF.
2. Develop a JavaScript program that provides the functionality stated in the Requirements section below.
3. Make the JavaScript program that you have created accessible and usable via the URL

   http://cgi.csc.liv.ac.uk/~<your user name>/game.html

   taking care that the access rights for the file are neither too restrictive nor too permissive.

Part 1 is worth 25% (10/40) of the overall mark for this assignment. Parts 2 and 3 are together worth 75% (30/40) of the overall mark for this assignment.

Requirements

The JavaScript program implements a simple game that consists of three stages, *setup*, *play* and *end*. During the *play* stage the game proceeds in *rounds*. The game is played on a grid with 10 x 10 cells. It involves a submarine that is controlled by the user and a couple of robotic killer submarines that are controlled by the computer (that is, your program). The user and your program are the two *players* of the game. All submarines can move around on the grid, but while the robotic killer submarines are fueled by nuclear power that never runs out, the user's submarine only has a limited amount of fuel and needs to collect fuel cells to keep going. The robotic killer submarines hunt the user's submarine; the user's submarine tries to avoid getting caught while also collecting all the fuel cells on the grid.

The game always starts in the *setup* stage. During that stage the user is shown the grid and can place three different types of objects on the cells of the grid:

- by clicking on a cell and typing a number between 1 and 9, a *fuel cell* is placed on a grid cell, the number indicates the *amount of fuel in the fuel cell*;
- by clicking on a cell and typing the letter "o", an *obstacle* is placed on a cell;
- by clicking on a cell and typing the letter "u", the *user's submarine* is placed on a cell.
- by clicking on a cell and typing the letter "k", a *robotic killer submarine* is placed on a cell.

There is no limit on the number of fuel cells, obstacles and robotic killer submarines, but there is obviously only one user's submarine. No grid cell can contain more than one object and once an object has been placed on a cell it cannot be changed. If the user tries to change the object placed on a grid cell, then an error message should be shown. If the user types a character that is not among 1 to 9, "o", "u" and "k", an error message should be shown.

In addition to the grid, the user must have a means to end the *setup* stage of the game, for example, via a button. If the user tries to end the *setup* stage of the game without placing his own submarine, then an error message should be shown and the user remains in the *setup* stage. Otherwise the game continues with the *play* stage.

At the start and during the *play* stage, the user is again shown the grid, initially with all the objects that have been placed on the grid during the setup stage, plus additional *status information*: The *rounds* played so far, the *number of units of fuel* available to the user's submarine, the *user's score*, the *computer's score*. Initially, zero rounds have been played, the user's submarine has 10 units of fuel, the user's and the computer's score are both zero. In addition, there must be the possibility for the user to end the *play* stage at any time, for example, via a button.

While in the *play* stage, the game proceeds in rounds, each round starting with the user's turn followed by the computer's turn. At the start of a round, the number of rounds played is increased by one, and the information shown to the user is updated.

During his/her turn, if the number of units of fuel of the user's submarine is zero at the start of the turn, then the user's submari ting that the submarine is out of fuel and the user's turn

If the number of units t the start of the turn, the user can attempt to move his/her submarine horizontally or vertically on the grid by typing one of four letters:

- "a" attempts to move the user's submarine one
- "d" attempts to move the user's submarine one
- "w" attempts to move the user's submarine one
- "x" attempts to move the user's submarine one grid cell down.

If the user types any other character, then an error message should be shown and the user has the possibility to type another character. If the attempted move would result in the user's submarine ending up outside the grid or on a cell occupied by an obstacle, then an error message should be shown, the attempt to move fails, the user's submarine does not move, the number of units of fuel available to the user's submarine does not change, and the user's turn is over. Otherwise, the attempted move is successful, the user's submarine changes cells, and the number of units of fuel available to the user's submarine reduces by one. If the user's submarine ends up on a grid cell that contains a fuel cell, then that fuel cell is removed from the grid, the value V of the fuel cell is added both to the user's score and to the number of units of fuel available to the user's submarine, and the status information is updated. If the user's submarine ends up on a cell occupied by a robotic killer submarine, then the user's submarine is destroyed, and the game proceeds to the *end* stage.

During the computer's turn your program attempts to move each of the robotic killer submarines in an order that allows each to move if at all possible. Unlike the user's submarine, the robotic killer submarines are not only able to move horizontally and vertically but also diagonally. Just like the user's submarine, each robotic killer submarine only moves at most one cell in a turn. If the user's submarine is in a grid cell immediately surrounding a robotic killer submarine, then that robotic killer submarine must move to the cell occupied by the user's submarine, the user's submarine is destroyed, the computer's score increases by 100, the status information is updated, and the game proceeds to the *end* stage. If the user's submarine is not in a grid cell immediately surrounding a robotic killer submarine, but one or more of those grid cells contains a fuel cell, then the robotic killer submarine must move to one of those fuel cells, the fuel cell is removed from the grid (thereby the computer deprives the user of fuel), the value V of the fuel cell is added to the computer's score, and the status information is updated. If none of the surrounding grid cells

contains the user's submarine nor a fuel cell, then a robotic killer submarine can move to an arbitrary surrounding cell provided that this move does not take it to a grid cell that is outside the grid or occupied by an obstacle or by another robotic killer submarine. A robotic killer submarine is not allowed to stand still if it can move. However, if a robotic killer submarine cannot move at all, then the computer should simply proceed to the next robotic killer submarine. Once an attempt has been made to move each of the robotic killer submarines, the computer's turn and the current round ends, and the status information is updated.

The *play* stage ends if one of the following conditions becomes true

- the user ends the *play* stage (by pressing the button provided for that);
- the user's submarine is destroyed;
- there are no fuel cells left on the grid;
- neither the user's submarine nor any of the robotic killer submarines is able to move.

Once the *play* stage has ended, the game is in the *end* stage. In the *end* stage the program determines the outcome of the game. The outcome is a *win* for the user if there are no robotic killer submarines left on the grid or the user's score is higher than the computer's score; the outcome is a *win* for the computer if the user's submarine has been destroyed or the computer's score is higher than the user's score; otherwise, the outcome is a *draw*. The program should display a message indicating the outcome of the game and then stop. During the *end* stage the program should not react to any user input or actions.

Additional requirements and comments:

- It is possible that during the *setup* stage the user does not place any fuel cells on the grid. On entering the *play* stage your program should recognise that, immediately proceed to the *end* stage, and declare the outcome of the game.
- It is also possible that during the *setup* stage the user does not place any robotic killer submarines on the grid. On entering the *play* stage your program should recognise that, immediately proceed to the *end* stage, and declare the outcome of the game.
- You should carefully analyse in which situations the user's submarine and the robotic killer submarines mi *play* stage in such situations.
- Ideally your pr such a way that they increase their chances o e by each robotic killer submarine trying to decrease the distance to the user's submarine with each move. But you could also implement a strategy by which the robotic `encircle' the user's submarine in orders to increase their chances. T he specific fuel cell, knowing that the user must eventually try to co
- Ideally your program would allow the size of the grid to be changed easily (by the maintainer of the system), independently for each dimension.
- JavaScript engines differ from browser to browser. You should make sure that your system works in all commonly used browsers (e.g., Google Chrome, Mozilla Firefox, Microsoft Internet Explorer 9 or higher) and on all commonly used platforms (e.g., Linux derivatives and Microsoft Windows).
- Your JavaScript program should only depend on your own code. JavaScript libraries/frameworks should not be used.
- Your code should be properly commented. This includes pointing out which parts of your code have been developed with the help of on-line sources or textbooks by including references for these sources at the appropriate points.

A script that deals satisfactorily with these additional requirements and comments, in addition to providing the basic functionality required, will receive higher marks.

Submission

Submit the following files (as separate, individual files; not as part of an archive file) via the departmental submission system at https://sam.csc.liv.ac.uk/COMP/Submissions.pl (COMP284-3: Assignment 3N (JavaScript)):

- a PDF file of your report on the comparison of Perl, PHP, JavaScript and Java;
- the HTML/CSS file or files, any auxiliary JavaScript files, image files for the JavaScript program.

Deadline

The deadline for this assignment is

- Tuesday, 2 May 2017, 17:00 -

Earlier submission is possible, but any submission after the deadline attracts the standard lateness penalties. Please remember that a strict interpretation of `lateness' is applied by the Department, that is, a submission on Tuesday, 2 May 2017, 17:01 is considered to be a day late. Note also that the University's definition of `lateness' for on-line submissions has changed and now counts Saturday and Sunday as `working days'.

Assessment

This assignment will address the following learning outcomes of the module:

- compare and contrast languages such as JavaScript, Perl and PHP with other programming languages
- rapidly develop simple applications, both computer and web-based, using an appropriate scripting language.
- document and comment applications written using a scripting language.

This assignment will contribute **40%** to the overall mark of COMP284. Failure on this assignment may be compensated by higher marks on other assignments for this module.

The report on the comparison of Perl, PHP, JavaScript and Java will be marked separately. Comprehensiveness and quality of the report on the comparison of Perl, PHP, JavaScript and Java will determine 25% (10/40) of the mark for this assignment. The remaining 75% (30/40) will be determined by the remaining tasks of this assignment and marks will be awarded according to the following scheme:

- The JavaScript program is accessible via the required URL and works without producing error messages on a range of browsers and operating systems: 8
- Correctness and quality of the solution to the JavaScript programming task (92 in total):
    - Quality of the interface design: 7
    - Correctness and quality of the implementation of the *setup* stage: 14
    - Correctness and quality of the implementation of the *play* stage: 44
    - Correctness and quality of detecting the end of the game and of the *end* stage: 15
    - Formatting, commenting, and quality of code: 12

The following penaltie

- If the report is ubtracted from the mark for the assignment.
- For every additional page that a report exceeds , 5 marks will be subtracted from the mark for the assignment.

No work that achieves the pass mark of 40 or more bef ese two penalties will be reduced below the pass mark once the two penalties ar ork that only achieves a mark below 40 before the application of these two penalties will see no reduction.

As stated above, the University policy on late submissions applies to this assignment as does the University policy on academic integrity, which can be found at http://www.liv.ac.uk/student-administration/student-administration-centre/policies-procedures/academic-integrity/.

Department of Computer Science, University of Liverpool
Ashton Building, Ashton Street, Liverpool L69 3BX, United Kingdom
+44 (0)151 795 4275
Maintained by Ullrich Hustadt, u.hustadt@liverpool.ac.uk