University of Liverpool - Department of Computer Science

COMP284 Scripting Languages (2017-18) -- Assignment 3 Re-sit: 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:

Perl	PHP
The syntax for conditional statements is `if (condition) { } elsif (condition) { } else { }`	The syntax for conditional statements is `if (condition) { } elseif (condition) { } else { }`

   - 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>/numbers.html

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 6 x 6 cells. The game involves *red pieces* controlled by the user, *black pieces* controlled by your programs, and *blocks* that are placed on the grid and do not move. There are up to 12 red pieces and up to 12 black pieces. Each piece has a number between 1 and 6. There are at most 3 red pieces with the number 1, at most 2 red pieces each with the number 2, 3, 4, and 5, and at most 1 red piece with the number 6, The same for the black pieces.

The game always starts in the *setup* stage and the setup stage consists of three *rounds*. During that stage the user is shown the grid and a button that allows the user to proceed to the next round of the setup stage.

After three rounds have been completed, clicking on that button takes the game to the *play* stage.
During the setup stage the user places objects on the grid, red pieces, black pieces, and blocks. No grid cell can contain more than one object and once an object has been placed on a cell it cannot be changed.
In the first round of the setup stage, the user places an arbitrary number of blocks on the grid. The user does so by clicking on a cell and typing the letter "b".

- If the user types a character different to the letter "b", an error message should be shown and nothing is placed on the grid.
- If the user tries to change an object already on the grid, then an error message should be shown and nothing changes on the grid.

In the second round of the setup stage, the user places up to 12 red pieces on the grid. The user does so by clicking on a cell and typing one of the numbers 1 to 6.

- If the user types a character that is not among the numbers 1 to 6, an error message should be shown and nothing is placed on the grid.
- If the user tries to place more pieces with a particular number than is allowed (e.g., the user tries to place three pieces with the number 5), an error message should be shown and nothing is placed on the grid.
- If the user tries to place more than 12 red pieces, then an error message should be shown and nothing is placed on the grid.
- If the user tries to proceed to the next round without placing any red piece on the grid, an error message should be shown and the game remains in the current round and stage.
- If the user tries to change an object already on the grid, then an error message should be shown and nothing changes on the grid.

In the third round of the setup stage, the user places up to 12 black pieces on the grid. The user does so by clicking on a cell and typing one of the numbers 1 to 6. The same error conditions apply as for the second round, only with 'red' replaced by 'black'.

If in the third round of the setup stage, the user clicks on the button for the next round, and there is at least one red piece and one black piece on the grid, then the game proceeds to 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 number of the *round* currently played, the number of *red pieces* left on the grid, and the number of *black pieces* left on the grid. In addition, there must be the possibility for the user to end the *play* stage at any time, for example, via a button. For each piece it must be clearly visible what number it has.

While in the *play* stage, the game proceeds in rounds, each round starts with the *user's turn* followed by the *computer's turn*. At the start of a round, the number of the *round* currently played is increased by one (the first round has the number 1), and the *status information* shown to the user is updated.
During each turn, the user and the computer try to move one of their pieces. It might be the case that either the user or the computer do not have a piece left that they can move. This should be checked at the start of each turn and the game should proceed to the end stage if this case occurs.
During his/her turn, the user attempts to move one of the red pieces horizontally or vertically on the grid by clicking on the piece (the piece becomes *selected*) and then typing one of four letters:

- "a" attempts to move the selected piece one grid cell to the left,
- "d" attempts to move the selected piece one grid cell to the right,
- "w" attempts to move the selected piece one grid cell up,
- "s" attempts to move the selected piece 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 selected piece ending up outside the grid, or on a cell occupied by a block, or on a cell occupied by a red piece, then the selected piece does not move, an error message should be shown and the user has the possibility to move a different piece (i.e., the user's turn does not end). Otherwise, the attempted move is successful and the selected piece changes cells:

- If the cell to which the selected piece has moved was previously empty, then nothing special happens and the user's turn is over.
- If the selected piece ends up on a grid cell that contains a black piece, then one piece is eliminated as follows:
  if either the number of the red piece is greater than the number of the black piece or the number of the red piece is 1 and the number of the black piece is 6 (i.e., the numbers 1 to 6 form a ring in

which 1 > 6), then the black piece is eliminated, else the red piece is eliminated. The number of red and black pieces is adjusted accordingly and the status information shown to the user is updated.

Once the user has successfully move exactly one of the red pieces, the user's turn ends and the computer's turn starts.

During the computer's turn your program attempts to move exactly **one** black piece. Its aim is to eliminate all red pieces from the grid.

- Black pieces move in the same way as red pieces, that is, they can move horizontally and vertically but **not** diagonally.
- Black pieces can also not `leave the grid' (e.g., a black piece in the left-most column of the grid **cannot** move left and thereby end up in the right-most column of the grid.
- Black pieces **cannot** move onto a cell containing a block and **cannot** move onto a cell containing another black piece.
- Black pieces can move to empty cells and to cells containing a red piece. In the latter case, one piece is then eliminated as follows:
  if either the number of the black piece is greater than the number of the red piece or the number of the black piece is 1 and the number of the red piece is 6 (i.e., the numbers 1 to 6 form a ring in which 1 > 6), then the red piece is eliminated, else the black piece is eliminated. The number of red and black pieces is adjusted accordingly and the status information shown to the user is updated.
- If there is a black piece and a red piece such that the black piece could move to the cell containing the red piece, then one black piece which satisfies this condition must move to a cell containing a red piece. However, if several pairs of black and red pieces satisfy this condition then the computer can choose which black piece moves to which red piece.
- Exactly one black piece must move if there is any black piece that can move.

Once exactly one black piece has moved, the computer's turn and the current round ends.

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);
- there are no red pieces left that could move (this includes the case that there are no red pieces at all left);
- there are no black pieces left that could move (this includes the case that there are no black pieces at all left);

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 black pieces left on the grid; the outcome is a *win* for the computer if there are no red pieces left on the grid; 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:

- You should carefully analyse in which situations none of the red pieces or none of the black pieces can move in order to correctly end the *play* stage in such situations.
- Ideally your program would move the black pieces in a way that increases the computer's chances of eliminating all red pieces. Techniques that you have learned on the AI module might help you with that.
- 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, like jQuery, should not be used.
- Your code should follow the COMP284 Coding Standard. This includes pointing out which parts of your code have been developed with the help of on-line sources or textbooks and references for these sources.

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 (COMP284-31: 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

- Friday, 10 August 2018, 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 Friday, 10 August 2018, 17:01 is considered to be a day late.

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 programming/scripting languages will be marked separately.

Comprehensiveness and quality of the report will determine 25% (10/40) of the mark for this assignment.

The following penalties apply to the report:

- If the report is not submitted as a PDF files, then 10 marks will be subtracted from the mark for the report.
- For every additional page that a report exceeds the specified page limit, 10 marks will be subtracted from the mark for the report.

Penalties will not take the mark for the report below 0.

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, works without producing script errors, all required files were submitted, the files accessible via the web are identical to those that were submitted: 8
- Quality of the interface design: 7
- Correctness and quality of the implementation of the *setup* stage: 23
- Correctness and quality of the implementation of the *play* stage: 35
- Correctness and quality of detecting the end of the game and of the *end* stage: 15
- Formatting, commenting, and quality of code: 12

Marks are given according to the extent to which the system behaves in the expected way, the game is visually appealing, and the computer behaves `intelligently', and, to a lesser extent, how well the code is written. Code that has no observable effect will receive no marks.

The mark for a JavaScript program that uses a JavScript library/framework, like jQuery, will be capped at 7.

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/. You should follow the COMP284 Lab Rules to ensure that you do not breach that policy.

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