COMP1206 PROGRAMMING II LECTURE NOTES
SEMESTER TWO, 2020

Home

comp1206 programming ii - coursework

(Submission Deadline: 2pm, Monday 27th April)

Please ask any question about the coursework on the COMP1206 Team or email Sebastian Stein (ss2@ecs.soton.ac.uk).

assignment instructions: mathdoku

In this coursework, you will implement a JavaFX application to play a logic puzzle game called MathDoku.

Rules of the Game:

A player needs to fill the cells in an NxN square grid with the numbers 1 to N (one number per cell), while adhering to the following constraints:

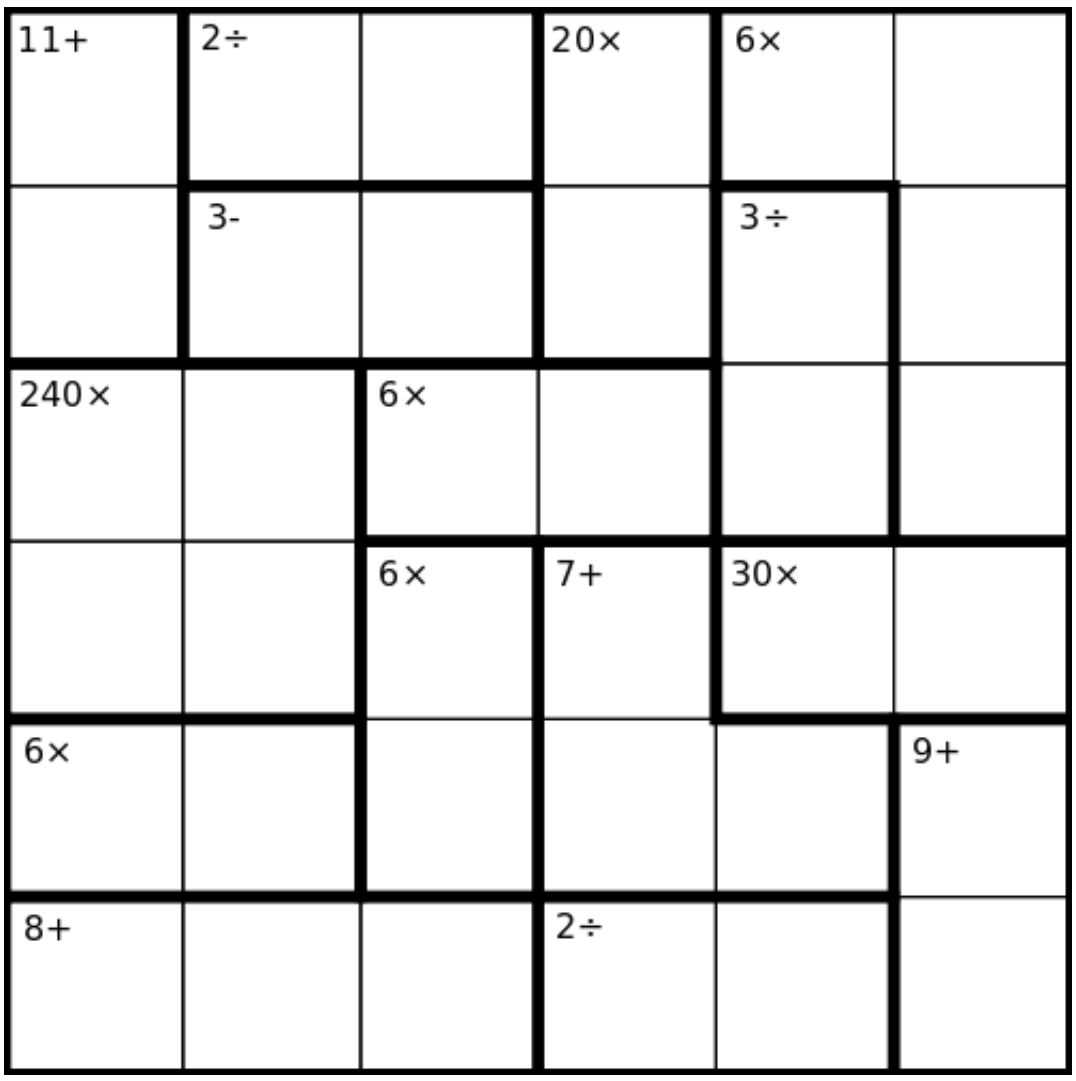
- Each number must appear **exactly once in each row.**
- Each number must appear **exactly once in each column.**

Furthermore, there are groups of adjacent cells called **cages**, which are highlighted on the grid by thicker boundaries. Within each cage is a label showing a **target** number followed by an arithmetic operator (+, −, x, ÷). There is an additional constraint associated with these cages:

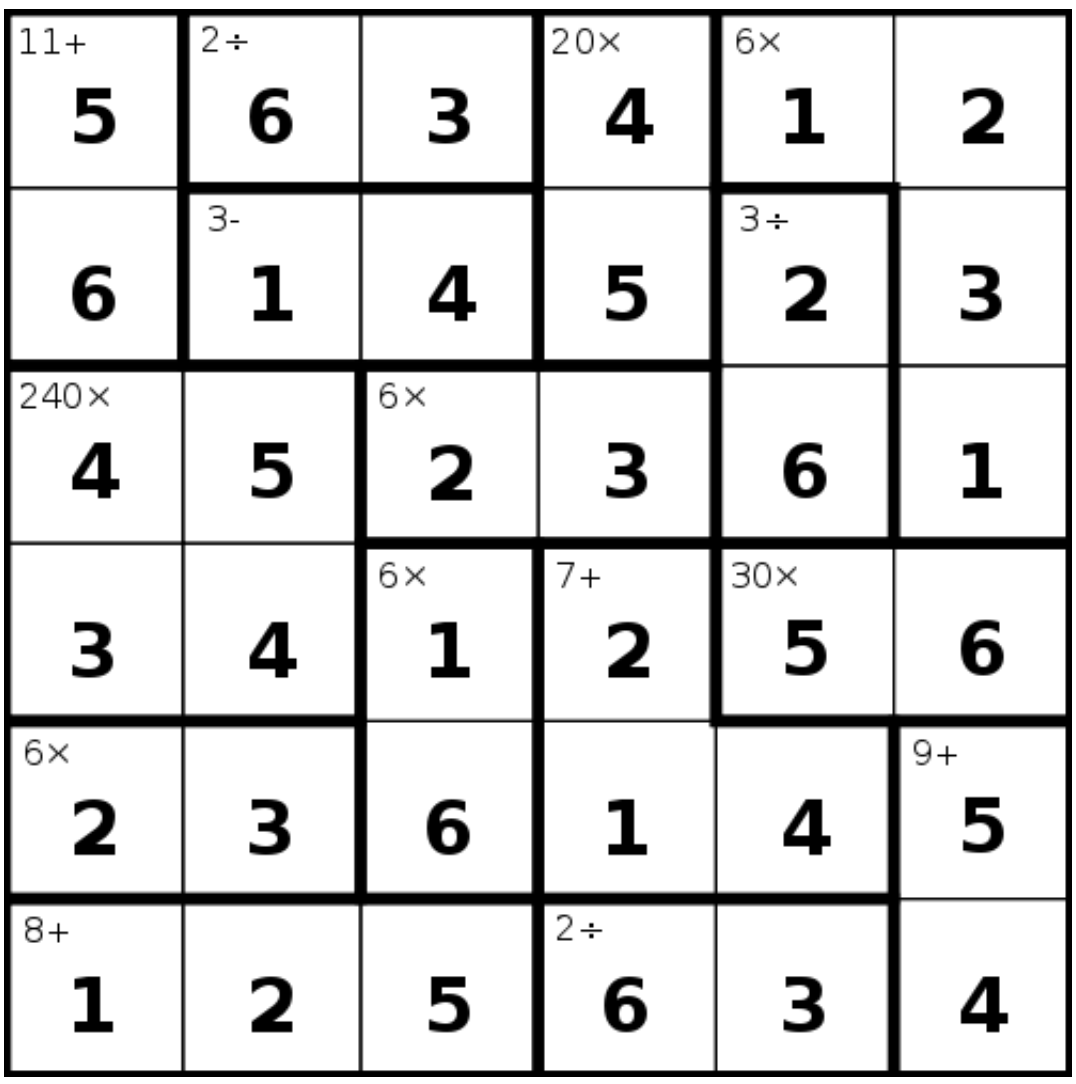
- It must be possible to **obtain the target by applying the arithmetic operator** to the numbers in that cage. For − and ÷, this can be done in any order.

Note: If a cage consists of a single cell, then no arithmetic operator is shown. The label simply shows the number that must be in that cell.

Here's an image of what a MathDoku game looks like (courtesy of Wikipedia):



And here's what the solved puzzle looks like:



instructions

You will build your application in parts of increasing difficulty (marks will be awarded for achieving each part, but you only need to hand in your final application). There are 40 marks available in total.

Part One: A Skeleton GUI

First, build a dumb GUI without functionality. This GUI should have controls for the following user actions:

- Undo / redo actions.
- Clear the board.
- Load a game from file.
- Load a game from text input.
- Option to show mistakes on grid.

There should also be space in your GUI to show the grid, which you will implement in Part Two.

Note: Try to make your GUI resizable, so that it adjusts to different window sizes.

[4 marks]

Part Two: The Grid

Display a MathDoku grid in your application. You could start by displaying the example grid in the image above.

Note: think carefully about how to keep your design general, so that it can cope with different layouts and grid dimensions (you can assume the smallest grid is a 2x2 grid and the largest is a 8x8 grid).

[4 marks]

Part Three: Cell Completion

Now it's time to add some functionality. Write appropriate event handling code to allow users to select cells, enter numbers into them and clear individual cells again. You should provide functionality to allow number entering and clearing via the keyboard (e.g., by pressing the number keys or the backspace key) and via the mouse (e.g., by showing appropriate number buttons in your GUI).

[4 marks]

Part Four: Win and Mistake Detection

Add functionality to detect when the grid has been filled correctly and solved (adhering to all the constraints in the rules of the game above). Show an

appropriate message to the user when this happens.

If the user has selected the option to show mistakes on the grid, immediately highlight rows, columns or cages that do not meet the constraints as numbers are being entered (e.g., if a row contains two 2s, that entire row should be highlighted).

Note: It is tricky to handle cages with the − and ÷ operators that consist of more than two cells. If you wish, you can ignore these, but for full marks, make sure that you can handle them too.

[5 marks]

Part Five: Clearing, Undo, Redo

Allow the user to clear the board using the control defined in Part One. A dialog box should be displayed to the user to confirm this action.

Also add functionality to undo the last cell modification(s) and redo previously undone modifications. Disable the buttons as appropriate when undoing or redoing is not possible.

[4 marks]

Part Six: Loading Files

Allow the user to load pre-defined MathDoku puzzles. The user should be able to do this in two ways: by choosing a specific file on their computer or by entering the puzzle through an appropriate text input control.

A pre-defined puzzle must be given in the following text format:

- Each line defines one cage of the puzzle.
- The line starts with the target followed immediately by the arithmetic operator (or none if it's a single cell) for that cage.
- This is followed by a space and a sequence of cell IDs that belong to the cage (consecutive cell IDs are separated by a comma). Here cells are numbered from 1 to (NxN), where 1 to N are the cells in the top row (left to right), N+1 to 2N are the cells in the second row from the top, and so on.

As an example, the MathDoku puzzle in the images above can be defined as shown in this file: example.txt.

You can find more examples of different sizes in this zip file: examples.zip (4x4_divdiff.txt is the only one that contains − and ÷ cages with more than 2 cells).

Note: The same puzzle could be specified in multiple ways. Specifically, the order of the cages and of the cells within a cage do not matter. You should do simple error checking on the input (e.g., whether cells within a cage are adjacent and whether each cell is part of exactly one cage) and notify the user if a mistake was detected. You do not need to check whether the puzzle can be solved.

[7 marks]

> If you have made it this far and everything is working as expected, well done! You should be able to obtain a mark in the region 60-70%. If you are brave, carry on with the extensions below. Warning: the last two are quite challenging - only attempt them if you have some spare time.

Part Seven: Font Sizes

Add an option to change the font size for your grid display (affecting the values in cells and the cage labels). Add at least three options (e.g., small, medium and large) and make sure that the text remains aligned within cells.

[2 marks]

Part Eight: Winning Animation

Add some appropriate animations when a win is detected. Be creative: you could animate the grid, have a colourful display or even add some fireworks over the grid.

[2 marks]

Part Nine: Solver

Add the option to automatically solve any puzzle (including those loaded from a file or text input). Use this functionality to also add a "Hint" option, which briefly reveals the correct value of one cell to the user.

[4 marks]

Part Ten: Random Game Generator

Include functionality to generate random games. This should give some appropriate options to the user (e.g., board size or difficulty level).

Note: Try to ensure that a randomly generated game has exactly one unique solution.

[4 marks]

mark scheme

In total there are 40 marks available, which will contribute 40% towards your overall course mark. The breakdown of available marks is as follows:

- Part One : 4 Marks
- Part Two : 4 Marks
- Part Three : 4 Marks
- Part Four : 5 Marks
- Part Five : 4 Marks
- Part Six : 7 Marks
- Part Seven : 2 Marks
- Part Eight : 2 Marks
- Part Nine : 4 Marks
- Part Ten : 4 Marks

You can find a detailed marking scheme here.

additional information

Submission Information:

Please place all of your source files into a single directory and make a zip archive of it. Please do not use RAR archiver and **do not include any compiled bytecode** (i.e. .class files). Include a README.txt file that explains how to compile and run your code from the command line. Also add a short user guide (instructions.docx or instructions.pdf) that explains how to carry out the main functionality of the application (complete this template here). You should submit your zipped directory.

You should use the automated ECS hand-in facilities found at:

https://handin.ecs.soton.ac.uk

Originality of work:

Please be aware of and adhere to the University regulations on collusion and plagiarism.

Regulations Governing Academic Integrity

ECS also has extensive information about the topic at https://secure.ecs.soton.ac.uk/notes/ai/. Please note that we run automatic plagiarism detection tools on all submissions.

frequently asked questions

Do we need to consider possible bracketing when evaluating cages with the division and subtraction operators? E.g., 2-(1-6) = 7

No, you can assume there are no brackets.

Is there a maximum size for a cage?

No, there is no maximum. But you only need to consider grids of sizes 2x2 to 8x8. Thus, the largest possible cage will have 64 cells.

Can a target be negative or a fraction?

No, only positive (or zero) integers are allowed. The operation on the cells in a cage must result in exactly the target (i.e., no further rounding is allowed).

Can we use GUI builders or FXML?

No, please construct the GUI programmatically as we covered in the lectures.

Will our comments be assessed?

No, this won't be part of the mark scheme.

Can I be creative in designing the GUI, e.g., add less frequently used functionality to tabs, use menus or show pop ups to enter numbers via mouse?

Yes, this is all fine and I encourage you to think about good GUI design. Please be sensible and use common sense though.

Should the random game generator produce a different game each time?

Yes. If you want, you can ask the user to enter a seed, which then produces the same game for a given seed (but entering other seeds should be possible).

Should I highlight mistakes in cages before they are completely filled?

No, you only need to highlight mistakes in cages that are completed filled (and where the numbers cannot be combined to reach the target).

Regarding the "Application can identify puzzles in the wrong input format and notify the user" requirement, what are the restrictions I should impose?

Some examples are given at the end of Part Six, but try to think about other things that could go wrong in the input file.

Is it ok to create a corresponding puzzle file when a user uses the text field to input the puzzle?

This is not the intended behaviour and would not be an elegant way of implementing the functionality. However, you won't be penalised if you implement it in this way.