MSO Lab Assignment 2:

Programming Learning App – Iteration 1

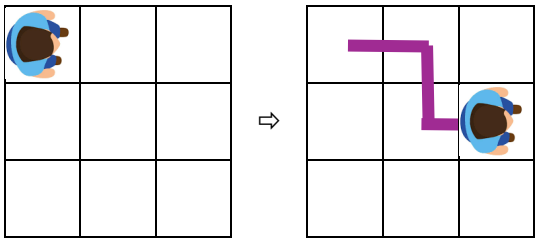
Version 1-2024

Introduction & description

This assignment is the first part of designing, implementing, and testing a larger educational application. The goal of the application is to introduce people to the basics of programming. These could be kids, or anyone else who is interested to learn programming. Our application will be a simplified version of block-based programming environments such as Scratch.[1] Users can build and run simple programs with a few buildings blocks, with the goal of moving a character around on a grid. This character could be anything, such as a cat, a car, or a fantasy figure.

This grid is a two-dimensional coordinate system. The character starts at the (0,0) coordinate, facing east, and can move to the other cells in the grid, drawing a path along the way. The character can only move horizontally and vertically, not on the diagonal.

In the example below the character moves forward one step, turns to the right, moves forward one step again, turns left, and moves forwards one final step, ending up in (2,1).



For the first version of the system, the following **commands** should be supported. In upcoming versions, we will expand the set of commands.

`Turn left/right`	The character's direction changes (but its position remains the same).
`Move [x]`	The character moves x steps forward.
`Repeat [x] times [command list]`	A list of commands is repeated x times. This can be any command: a turn, a move, or another repeat.

A program (which has a name) consists of a list of commands. Below are a few examples of what a program in text-format could look like:

[1] https://scratch.mit.edu/

Example 1 “Rectangle1”: Moving in a rectangle	Example 2 “Rectangle2”: Moving in a rectangle, but more efficiently	Example 3 “Random”: Some random movements, just to show what a valid program looks like

`Move 10`
`Turn right`
`Move 10`
`Turn right`
`Move 10`
`Turn right`
`Move 10`
`Turn right`
	`Repeat 4 times`
	`    Move 10`
	`    Turn right`
		`Move 5`
		`Turn left`
		`Turn left`
		`Move 3`
		`Turn right`
		`Repeat 3 times`
		`    Move 1`
		`    Turn right`
		`    Repeat 5 times`
		`        Move 2`
		`Turn left`

In the future, we will need a graphical interface to build programs using these blocks. In this assignment, we first set up the core elements we need to construct programs. We also need the following features:

We should be able to **execute** a program. For now, this will result in a textual trace and a final state of the character (position and direction). As an example, executing example 1 and 2 will both give:

```
Move 10, Turn right, Move 10, Turn right, Move 10, Turn right, Move 10, Turn right.
End state (0,0) facing east.
```

We should be able to **import** a program from a text file. The contents of such a file are the textual representation as seen above, but we might expect other import formats in the future.

We want to have a few **hard-coded examples** that create a few basic (just moves and turns), advanced (using a repeat statement), and expert programs (with nested repeats). We should be able to call a method to get these programs directly.

We want to calculate some simple **metrics** for a program: number of commands, the maximum nesting level, and the number of repeat-commands.

For now, there should be a simple command-line interface, that offers the following functions:

- Step 1: Select a random example program (basic, advanced, or expert) or import a program from a text file.
- Step 2: Either execute the loaded program, or calculate all metrics for the program.

We focus this assignment on the basic functionality. In the follow-up assignment a user interface will be added, and several new features will be requested. Your design must therefore be easily expandable!

Part 1 Domain model and clarifications

Start by sketching a domain model for the application. Clearly show how the entities in your model are related, using descriptive names and cardinalities. Include attributes for the entities, and methods only when it really clarifies something.

If you have made any assumptions about how something should work, give these in a list. If you asked a TA to clarify something that is not in the description, add that to the list as well (mentioning that you got the answer from a TA).

Part 2 Software design and patterns

Create a software design for the basic system as described above, in the form of an UML class diagram. You do not need to worry about the graphical user interface at this point. Instead focus on identifying the classes and their relations for the core of the application.

For each class in the diagram, specify attributes (with their types) and methods. For classes that need an additional explanation, provide a description in a separate text section with the specifics.

If you use any design patterns, indicate which ones, which problem it solves, and how it is implemented (by describing how the pattern elements map to your design).

Part 3 Evaluation

Evaluate your design. Explain the different kinds of variation that you have encountered in this domain, and how your design handles this variation. Discuss alternatives you considered and explain why you did not choose those alternatives.

Also explain how it is capable of handling changing requirements. Identify at least three likely future changes to the specification. How will your design handle such changes? What kind of changes would be difficult to incorporate? Finally, give an analysis of your design regarding cohesion and coupling. How does your design exhibit high cohesion and low coupling?

Part 4 Implementation and testing

Set up a Git-repository for your solution. If needed, do a tutorial on how to work with version control.[2] Note that in Lab 3 you will be asked to add your grader (TA) to the repository. Start implementing the base classes in C#. Implement unit tests to test functionality.

The follow-up assignment will be published in week 8: make sure that you have as much of the basis of the application as possible. In terms of grading, this part does not carry much weight (yet). After submitting this assignment, you will also have the opportunity to get feedback on your first design from the TAs, and in the next iteration you will have the opportunity to make adjustments to your design and code.

Part 5 Work distribution & retrospective

Give a detailed overview of the task distribution: who has worked on which part?

Conduct a *retrospective* in which you reflect on your collaboration. What went well, and what didn't? What should change in upcoming assignments?

[2] https://missing.csail.mit.edu/2020/version-control/

General considerations

- *Language*. Both English and Dutch are allowed. The writing and layout of the document is a knockout criteria for a passing grade!
- *Task distribution*. The work should be distributed equally among the two team members, and *both team members contribute to all components*: design, programming, testing, and reflection. If this is clearly not the case, this will be reflected in the grading.
- *Grading*: check the attached Excel-sheet to see how you will be graded (note this sheet is subject to minor changes).
- *Authenticity:* The assignments should be you and your teammate's own work; you may not use work from other students/websites. The design of the classes, attributes, methods, tests, and patterns must be your own work. If in doubt, ask first if you can use something!
- *Using existing code:* If you use code that is not your own, you must cite your source in your report and in the code. This is only allowed for tools such as libraries to read files, UI elements, etc. You may use AI chat tools such as ChatGPT only for finding bugs in your code and generating code *within 1 method*. You should switch off Copilot and other AI-autocompletion for this project. If you generated some code, you must include the full chat transcript and indicate in the code which pieces were generated.

Tip

Set up a Kanban-board for you and your partner with lanes such as 'backlog', 'implementing, 'testing', and 'done', and add cards for user stories and tasks, to organize and keep track of our progress. You can use software such as Trello or Microsoft Planner for this.

How to submit

Create a zip or rar file of the complete project, including files to build the project (i.e. the project and solution files if you use Visual Studio). Clean the project before you compress it, so that the zip file does not become unnecessarily large due to all kinds of intermediate files and executables. This means removing the .git folder and bin folders, as well as IDE specific files and folders, such as .vs, .vscode, etc.

If necessary, create a readme file with instructions on how to run the application. The TA's should be able to easily run your application!

Attach a PDF file with parts 1-3 and 5.