5 Seaside: Getting Started

Exercise 1 Make sure you are using a pharo-web image with Seaside loaded. If you have a class named `WATask` you are ready to go. If not, grab the latest pharo-web image from http://www.pharo-project.org/pharo-download. Make sure your Seaside server is up and running by accessing the example application at http://localhost:8080/seaside/examples/counter.

Exercise 2 Load the Monticello package `Tutorial.mcz` that you can download from the lecture website by dragging and dropping it over the image. The package contains some class skeletons that will assist you to solving these and the following exercises. Save your image.

5.1 Development Tools

Exercise 3 Use your web browser to navigate to the counter example application. Toggle on the halos to see the border of the component this application is built of. Experiment and interact with the application in render- and source-mode.

Exercise 4 Change the behavior of the increase and decrease buttons: edit the methods `#increase` and `#decrease` from within the web browser to increase by `2` and decrease by `3`.

Exercise 5 Inspect the living component from within the web browser. There are two instance variables visible, whereas `count` is representing the state of the component. The other instance variable is defined in a super-class of `WACounter` and will be discussed later on.

Exercise 6 Change the background color of the web application by using the style editor from within your web browser. Try using something like `body { background-color: yellow; }`.

Exercise 7 Introduce an error to the method `#increase` using your web browser. Play with your application so that the error occurs. Click on the *debug* link which opens a debugger within your image. Fix the bug and proceed the evaluation.

5.2 Control Flow

We will implement a simpler guess-a-number game. Some skeletons are provided, so you don't need to implement all by yourself.

5.2.1 User Guesses a Number

Exercise 8 Have a look at the source code of `STUserNumberGuesser` in the category *Tutorial-Tasks* and play the game several times to make sure it works

as expected.

Exercise 9 Modify the method `#go` in `STUserNumberGuesser` to count the number of guesses. Show the total number of guesses the user required to get the right number in the end of the game.

Question 10 Try using the back button while playing the game. How does the application handle this?

Question 11 What happens if you open multiple windows in the same session and play within the different windows independently?

Question 12 Is it possible to cheat the counter by using the back button or by opening new windows within the same session? Does this behavior change if you use an instance variable instead of a temporary one for counting?

5.2.2 Computer Guesses a Number

Exercise 13 Write a new web application that allows the computer to guess a number the user is thinking of. In case you run into troubles, you can always have a look at the implementation of `STUserNumberGuesser`.

1. Create a subclass of `WATask` called `STComputerNumberGuesser`.
2. Create an initialization method on the class side of the newly created class, registering the component as a new web application with the path segment `cng`. Make sure to evaluate the newly created method.
3. Implement the method `#go` following the rules of the game. Use `#inform:` to tell the user what he should do and `#confirm:` to ask the user if the guess of the computer is too big.
4. Play the game several times to make sure it works as expected.

Exercise 14★ Implement yet another task asking the user if he wants to guess or not. Depending on the answer either call `STUserNumberGuesser` or `STComputer-NumberGuesser`. Modify those two classes to answer the numbers of steps required and call them from within your new task. Don't forget to register your new application with a class initialization method.

5.2.3 TicTacToe Game

There are three prepared classes for this game in the category *Tutorial-TicTacToe* following the *MVC-Pattern*:

Model `STTicTacToeController` is a simple model of a game holding the current board configuration. It includes methods to access and modify its configuration (`#boardAt:` and `#boardAt:put:`), a method to look for the best possible move of a given player (`#find:`) and several tester to query the state of the game (`#isEmpty`, `#isFinished`, `#isWinner:`).

View `STTicTacToeView` is a simple Seaside view onto the game model. You will learn later on how to create views with Seaside.

Controller `STTicTacToeController` is a subclass of `WATask` and this is the place that needs your work now. It already implements a few convenience methods like `#newModel`, `#computerMove` and `#userMove`.

Exercise 15 Register `STTicTacToeController` as a new web-application, but this time don't use a class initialization method but the configuration interface. Make sure that you have a method `#canBeRoot` on the class-side so that Seaside recognizes this class as a possible root of a web application. Browse to http://localhost:8080/seaside/config, enter your password, add a new entry point with the name `ttt` and select `STTicTacToeController` as the root component.

Exercise 16 Implement the game in the method `#go` using the provided convenience methods. You will also need some testing methods of the model to check if the game is finished (`#isFinished`) and who was the winner (`#winner`). Don't put all your code into one single method, split it among different ones to ensure readability. Ask the user in the beginning of the game if he prefers to start playing or not.

Please save the Monticello package `Tutorial` and send it by mail to st-staff@iam.unibe.ch. Attach your written solutions that are not part of the source-code to the mail or hand them in as hardcopy at the beginning of the next exercise session. Your mail and solutions should be clearly marked with names and matrikel numbers of the solution authors.