Software requirements specification for project “Robotics”

* Authors
* V.V. Chernikov
* O.S. Markelov
* S.O. Morozov
* I.V. Shatalov
* Introduction

*[General description of the project and its functionality.]*

* Glossary

Level – a virtual world simulated on the server with unique environment, rules and goals.

Multi-User Level – level where several robots manageable by different programming code may compete with each other.

Robot – an object in virtual world manageable by user’s programming code. Every robot has the set of sensors and actions available to a user.

Sensor – a module of a robot whose purpose is to detect changes in environment and send the information to user’s programming code.

Simulation – the process of interaction between the code written by the student and the model, during which the model state is constantly updated according to the actions specified by the code.

Simulation cycle – a unit of time used in the system. The level simulation is divided into cycles. A cycle goes as follows:

* The current sensors readings are assigned to variables that are available to the code written by the student
* The student code is run, it can access the sensors readings by using the above-mentioned variables and declares which actions should be performed by the robot by changing another set of designated variables
* The actions are read by the model, and according to them a new state of the level is calculated
* The code can declare that a certain set of actions should be performed on each simulation cycle during a certain number of cycles. In this case, for cycles on which these actions are performed, the student code is not run and on each cycle nothing happens except updating the state of the level. Once these cycles are over, the following cycles run as usual.

Simulation rate – the number of simulation cycles executed per unit of real time (in cycles per second).

* Actors

*[All the actors come here. For each actor it should be definition for its role and the general description of its goals and responsibilities within the given system.]*

Student – a user that writes a programming code for the robot, sends it to the server for simulation and waits for report (textual or/and visual).

Teacher – a user that inherits all the rights from a student in addition to possibility to edit the existing levels or create the new ones.

* Functional requirements
* Strategic Use-cases

*[Optional. White-level use-cases. This section is useful when there are too many blue-level use-cases and they should be grouped somehow.]*

* *Use-case <UC-S-1>*
* *Use-case <UC-S-2>*
* Use-cases for Student

*[In case the white-level use-cases are defined, here could be one additional level that groups blue-level use-cases by white-level ones, in addition to grouping by actors.]*

* *Use-case “Start a level”*

*[Full UC description]*

**Actors:** Student

**Goals:** release the programmed robot to a level

**Precondition:** Student is in the main menu of the client app

**Main success scenario:**

* The student clicks the ‘Choose a level’ button
* List of available levels loaded from the server appears
* The student clicks a level in the list
* The code editor appears
* Student enters the code for the robot
* Student clicks the ‘Run’ button
* The code is checked for syntactic errors by the system
* The code is sent to the server and the level starts

**Alternative scenario <scenario-name1>:**

* When check 7 failed (the code contains syntactic mistakes), show the errors and suggest to correct the code

**Alternative scenario <scenario-name2>:**

* When unable to get levels in step 2 due to a network error, show student an error message
* The main menu is shown again

**Alternative scenario <scenario-name3>:**

* When unable to send the code to the server due to a network error, show student an error message
* The code editor is shown again
* *Use-case “Save program code typed in code editor”*

*[Full UC description]*

**Actors:** Student

**Goals:** save program code to the hard drive

**Precondition:** Code editor is opened

**Main success scenario:**

* The Student clicks the ‘Save’ button
* File explorer window appears
* The Student chooses where to save the file and clicks ‘OK’
* The explorer window is closed and the source code file is saved
* *Use-case “program code to the code editor”*

*[Full UC description]*

**Actors:** Student

**Goals:** load program code from the hard drive

**Precondition:** Code editor is opened

**Main success scenario:**

* The student clicks the ‘Load’ button
* If there are unsaved changes, the student is asked if they should be saved first
* File explorer window appears
* The student chooses a source code file
* The code appears in code editor
* *Use-case “leave the level”*

*[Full UC description]*

**Actors:** Student

**Goals:** return from the level to the code editor

**Precondition:** Student is on a level

**Main success scenario:**

* The student clicks the ‘Quit’ button
* A window asking confirmation appears
* The student confirms the intention to leave the level
* The code editor appears with the same code as before entering the level
* *Use-case “leave the code editor”*

*[Full UC description]*

**Actors:** Student

**Goals:** close the code editor and return to main menu

**Precondition:** Student is in the code editor

**Main success scenario:**

* The student clicks the ‘Quit’ button
* If there are unsaved changes to the code, the student is asked if they should be saved
* The code editor is closed and main menu appears
* *Use-case “pause the simulation”*

*[Full UC description]*

**Actors:** Student

**Goals:** halt the simulation temporarily without leaving the level

**Precondition:** Student is on the level and simulation is running

**Main success scenario:**

* Student clicks the “Pause” button
* The simulation runs until the current simulation cycle is finished
* The simulation halts
* *Use-case “resume the simulation”*

*[Full UC description]*

**Actors:** Student

**Goals:** continue the simulation from the point where it was paused

**Precondition:** Student is on the level and simulation is paused by the student

**Main success scenario:**

* Student clicks the “Resume” button
* The simulation starts the simulation cycle that follows the last executed cycle on the same rate as before the level was paused
* *Use-case “change simulation rate”*

*[Full UC description]*

**Actors:** Student

**Goals:** make the simulation run faster or slower

**Precondition:** Student is on the level and simulation is running

**Main success scenario:**

* Student clicks the “Increase simulation rate” or “Decrease simulation rate”
* The simulation runs until the current simulation cycle is finished
* Starting from the following simulation cycle, the simulation runs with a new rate according to which button was clicked

**Alternative scenario <scenario-name1>:**

* Unable to change simulation rate because the highest/lowest possible rate has already been achieved
* No action is performed
* *Use-case “fast-forward to a specific simulation cycle”*

*[Full UC description]*

**Actors:** Student

**Goals:** advance to a given simulation cycle which has not yet been executed

**Precondition:** Student is on the level

**Main success scenario:**

* Student clicks the “fast-forward” button
* A window with a simulation cycle choosing instrument (e.g. text field) appears
* Student enters desired simulation cycle number
* The simulation of cycles between the current cycle and the target one is performed in background
* The simulation continues as before from the cycle which it fast-forwarded to

**Alternative scenario <scenario-name1>:**

* The target simulation cycle is the current one or is already finished, or its number is invalid (e.g. negative)
* A message notifies the user that the fast-forward is impossible

**Alternative scenario <scenario-name2>:**

* During the fast-forward (check 4) the level goal was reached or the mission was failed (e.g. the robot was destroyed)
* The simulation stops, the student is shown the state of the simulation on the cycle during which the event occurred
* The student is notified about the reached outcome
* *Use-case “get the result of the level”*

*[Full UC description]*

**Actors:** Student

**Goals:** find out if the program written by the student succeeded or failed

**Precondition:** Student is on the level

**Trigger condition:** the goal of the level or failing condition (including time limit) is achieved

**Main success scenario:**

* The simulation stops
* The result (“success” or “failure”) is shown to the student
* The “pause/resume” and “change simulation rate” buttons are disabled
* Use-cases for <actor2>
* *Use-case “Create a new level”*

*[Full UC description]*

**Actors:** Teacher

**Goals:** create a new level for task

**Precondition:** Teacher is in the main menu of the client app

**Main success scenario:**

1. The Teacher clicks the ‘Create a level’ button
2. Level editor appears
3. List of tools appears
4. Code editor appears
5. Teacher uses different tools, write code and create level
6. The code is checked for syntactic errors by the system
7. The teacher clics the 'Save' button
8. File explorer window appears
9. The Teacher chooses where to save the file and clics 'Ok'
10. The explorer window is closed and the source code file is saved

**Alternative scenario <scenario-name1>:**

* When check 6 failed (the code contains syntactic mistakes), show the errors and suggest to correct the code
* *Use-case “Load level to level editor”*

*[Full UC description]*

**Actors:** Teacher

**Goals:** load level from the hard drive

**Precondition:** Teacher is in the level editor

**Main success scenario:**

1. The Teacher clicks the ‘Load’ button
2. If there are unsaved changes, the Teacher is asked if they should be saved first
3. File explorer window appears
4. The Teacher chooses a level file
5. The level appears in level editor

**Alternative scenario <scenario-name1>:**

* When check 4 failed (program can't load level in editor), show the errors.
* *Other use-cases are similar to actor 'Student'*
* System-wide functional requirements

*[Optional. System-wide functional requirements that weave with multiple use-cases. Examples: authorization, audit]*

* Non-functional requirements

*[All the subsections are optional.]*

* Environment

*[Environment requirements are limitations for hardware and software usage including supported hardware platforms, networking infrastructure and protocols, programming languages, libraries and external services]*

* Performance

*[The performance characteristics of the system should be outlined in this section. Examples are response time, throughput, capacity and startup or shutdown times.]*

* Reliability

*[Reliability includes the product and/or system's ability to keep running under stress and adverse conditions. Specify requirements for reliability acceptance levels, and how they will be measured and evaluated. Suggested topics are availability, frequency of severity of failures and recoverability.]*

* Extensibility

*[This section indicates requirements that will enhance the extensibility including extension points, compatibility, scalability, configurability]*