Software requirements specification for project “Robotics”

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

This is a comprehensive system for people with little experience in programming and robotics (for example middle and high school students) to practice their skills and acquire knowledge. The student is supposed to write a program for the robot, which determines its further actions in the virtual world based on the readings of the sensors of this robot. After writing a program, student can release the programmed robot to a level and launch it. Using built-in tools, a teacher can create models and levels with certain tasks and limitations or load a level from hard drive to change or observe.

Levels have goals: for example, a robot vacuum cleaner can be controlled on the “room” level, and its task will be to clean all the dust in it. By default there are different levels with different tasks. Also, players can program robots for multi-user levels, where their decisions will compete with each other.

To program the robot, the player is provided with an environment where they can write scripts. In this environment player can save code on hard drive or load it. Additionally, the player can debug the program, observing its behavior at any given sensor readings.

* 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 available via getter functions to the code written by the student
* The student code is run, it can access the sensors readings by using the above-mentioned functions and declares which actions should be performed by the robot
* 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.
* If the state of simulation on a given simulation cycle can already be calculated a certain number of cycles before it, the individual calculation of these cycles can be skipped to reduce the amount of computation.

Simulation log – a human-readable text which describes the sequence of events that happened during the simulation (state of the level, sensor readings and robot actions on each simulation cycle). It is generated each time after the simulation is run.

Simulation playback – a real-time representation of a completed simulation. At each moment of time it shows the student the state of simulation at a certain cycle, so it appears to the student as an animation with changing data about the level and its graphical representation.

* 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 leveland find out if the robot is successful

**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
* The server conducts the simulation using the code sent by the student
* The results of the simulation are sent back to the client
* The outcome (success or failure) is shown to the student

**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

**Alternative scenario <scenario-name4>:**

* An error/exception occurred on the server during the simulation
* A report about what happened is sent to the client and is shown to the student
* *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 “Load 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 “View simulation log”*

*[Full UC description]*

**Actors:** Student

**Goals:** read the logs of the simulation to figure out what happened during the simulation

**Precondition:** Student is on the level and it has been calculated

**Main success scenario:**

* The student clicks the ‘Log’ button
* The log generated from data sent by the server is shown
* The student reads the log
* The student clicks ‘Close’ button
* The log viewer is closed and the previously shown screen is shown again
* *Use-case “Save simulation log”*

*[Full UC description]*

**Actors:** Student

**Goals:** save the log to the hard drive

**Precondition:** Student is in the log viewer

**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 log file is saved
* *Use-case “Watch the level playback”*

*[Full UC description]*

**Actors:** Student

**Goals:** watch what happened on the level during the simulation

**Precondition:** Student is on the level and it has been calculated

**Main success scenario:**

* The student clicks the ‘Playback’ button
* The sequence of events that happened on the level during the simulation is taken from the data sent by the server and is shown to the student at a constant rate starting with the first simulation cycle
* The sensors readings and actions performed by the robot are also shown
* *Use-case “Jump to a certain cycle in the simulation”*

*[Full UC description]*

**Actors:** Student

**Goals:** see what happened at a specific moment of the simulation

**Precondition:** Student is watching the level playback

**Main success scenario:**

* The student clicks the ‘Jump’ button
* A window with a simulation cycle selection tool appears
* The student enters the desired cycle number (or corresponding time) and clicks ‘OK’
* The playback jumps to the cycle specified by the student

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

* The cycle number entered by the student is invalid (is negative, not a number or bigger than the simulation length)
* A notification is shown and no action is performed
* The user can try to enter the cycle number again
* *Use-case “Change the rate of simulation playback”*

*[Full UC description]*

**Actors:** Student

**Goals:** watch the playback on a different speed (slower of faster)

**Precondition:** Student is watching the level playback

**Main success scenario:**

* The student clicks the ‘Increase playback rate’ or ‘Decrease playback rate’ button
* The playback continues running with a new rate according to the change chosen by the student

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

* The playback is already running with the highest/lowest possible rate
* The corresponding button is inactive
* If the student clicks it anyway, no action is performed – the playback keeps running at the same rate as before
* 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]*

The application can be used on any device that has a browser with JavaScript support and that is connected to the internet.

A student can write scripts for the robot on Groovy programming language.

A teacher can create a new model describing it on Java/Groovy programming language.

* Performance

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

The server’s response time in standard situation should be under 2 seconds. Only exceptionally may take up to 10 seconds.

The server’s calculation time depends on the complexity of the level and student’s script and also on the server’s load.

The server can take up to 100 responses at the same time.

The server should work during a week without restart.

* 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]*