Software requirements specification for project “Robotics”

1. Authors

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

1. Introduction

This is a comprehensive system for people with little experience in programming and robotics (e.g. 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.

1. 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.

Virtual time – number, representing time in simulation. It is independent of time in real world. Most actions performed by the robot takes some virtual time to complete. Current virtual time is usually sum of time taken to perform robot’s actions, e.g. current virtual time is 0 seconds, robot is performing “move left for 3 seconds”, after he has finished, virtual time equals 3 seconds.

Simulation cycle – atomic part of the simulation. 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 for certain virtual time. Student’s code will not be executed until level state at specified virtual time is calculated.

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 replay of a completed simulation represented as a sequence of states at each simulation cycle. 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. It can be paused so that the student watches it frame by frame.

Code editor – a built-in tool that allows user to create and edit source code.

Level editor – code editor with access to premade templates, classes and levels, used to create or change existing levels.

1. 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 as well as supervise the students’ performance.

1. Functional requirements
   1. Use-cases for Student
      1. *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:**

1. The student clicks the ‘Choose a level’ button
2. List of available levels loaded from the server appears
3. The student clicks a level in the list
4. The code editor appears
5. Student enters the code for the robot
6. Student clicks the ‘Run’ button
7. The code is sent to the server
8. The code is checked for syntactic errors by the system
9. The server conducts the simulation using the code sent by the student
10. The results of the simulation are sent back to the client
11. The outcome (success or failure) is shown to the student

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

1. When check 8 failed (the code contains syntactic mistakes), show the errors and suggest to correct the code

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

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

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

1. When unable to send the code to the server due to a network error, show student an error message
2. The code editor is shown again

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

1. An error/exception occurred on the server during the simulation
2. A report about what happened is sent to the client and is shown to the student
   * 1. *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:**

1. The Student clicks the ‘Save’ button
2. File explorer window appears
3. The Student chooses where to save the file and clicks ‘OK’
4. The explorer window is closed and the source code file is saved
   * 1. *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:**

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

1. The student clicks the ‘Quit’ button
2. If there are unsaved changes to the code, the student is asked if they should be saved
3. The code editor is closed and main menu appears
   * 1. *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 has sent the code to the server and received the result of the simulation

**Main success scenario:**

1. The student clicks the ‘Log’ button
2. The log generated from data sent by the server is shown
3. The student reads the log
4. The student clicks ‘Close’ button
5. The log viewer is closed and the previously shown screen is shown again
   * 1. *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:**

1. The student clicks the ‘Save’ button
2. File explorer window appears
3. The Student chooses where to save the file and clicks ‘OK’
4. The explorer window is closed and the log file is saved
   * 1. *Use-case “Watch the level playback”*

*[Full UC description]*

**Actors:** Student

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

**Precondition:** Student has sent the code to the server and received the result of the simulation

**Main success scenario:**

1. The student clicks the ‘Playback’ button
2. 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
3. The sensors readings and actions performed by the robot are also shown
   * 1. *Use-case “Go to the next/previous simulation cycle”*

*[Full UC description]*

**Actors:** Student

**Goals:** watch how simulation went cycle by cycle

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

**Main success scenario:**

1. The student clicks the ‘Next cycle’ or ‘Previous cycle’ button
2. The playback jumps to the corresponding simulation cycle

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

1. The next/previous cycle does not exist (the student was already observing the first/last cycle)
2. The corresponding button is inactive
3. If the student clicks it anyway, no action is performed
   * 1. *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:**

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

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

1. The cycle number entered by the student is invalid (is negative, not a number or bigger than the simulation length)
2. A notification is shown and no action is performed
3. The user can try to enter the cycle number again
   * 1. *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:**

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

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

1. The playback is already running with the highest/lowest possible rate
2. The corresponding button is inactive
3. If the student clicks it anyway, no action is performed – the playback keeps running at the same rate as before
   1. Use-cases for Teacher
      1. *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. Teacher writes code and creates level
4. The code is checked for syntactic errors by the system
5. The teacher clicks the 'Save' button
6. File explorer window appears
7. The Teacher chooses where to save the file and clicks 'Ok'
8. The explorer window is closed and the source code file is saved

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

1. When check 4 failed (the code contains syntactic mistakes), show the errors and suggest to correct the code
   * 1. *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>:**

1. When check 4 failed (program can't load level in editor), show the errors.
   * 1. *Use-case “See the students rating”*

*[Full UC description]*

**Actors:** Teacher

**Goals:** learn how successful the students are at doing assignments in the system

**Precondition:** Teacher is in the main menu

**Main success scenario:**

1. Teacher clicks the ‘Students’ button
2. A list of students is shown, providing the list of successfully completed levels, success rate and other stats
   * 1. *Use-case “See the students code”*

*[Full UC description]*

**Actors:** Teacher

**Goals:** learn the student’s script sent to the server

**Precondition:** Teacher is in the main menu

**Main success scenario:**

Teacher clicks the ‘Students’ button

A list of students is shown

Teacher clicks on student’s name

A list of models for which the student created scripts appears, providing the list of all the scripts sent to the server

Teacher clicks specific version of scripts

Code editor appears where the student’s script is shown

* + 1. *Other use-cases are similar to actor 'Student'*

1. System-wide functional requirements

* When client is connecting to server for the first time, it should provide credentials.
* Users should not be able to access server without authorization.
* Students should not be able to perform denial of service attack using their solutions (robot programs).

1. Non-functional requirements
   1. Environment

* Machines in system should have at least 4GB RAM and a quad-core processor.
* The application can be used on any device that has a JVM (Java Virtual Machine) installed and that is connected to the internet.
* A student can write scripts for the robot in Groovy programming language.
* A teacher can create a new model describing it in Java/Groovy programming language.
  1. Performance
* Calculation time in standard situation should be under 2 seconds. Only exceptionally may take up to 10 seconds.
* One server should be enough for at least 60 students.
  1. Reliability
* In case part of machines is offline, system should redirect users to online ones.
* The server should work during a week without restart.
  1. Extensibility
* The client application not dependent on JVM can be created using the same server API.
* The amount of models calculated at the same time can be increased by adding extra computers to the system.