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. To access the system a user has to perform authentication so that only students and teachers are able to work in the system.

To program a robot a student uses an environment designed for writing scripts for robots. The program can be loaded from hard drive or saved to hard drive. After writing a program, the student can release the programmed robot to a level and launch it. After that, the student can learn how their robot performed on the level by reading a simulation log or by watching a playback. The log can be saved to hard drive if the student needs it.

Using built-in tools, a teacher can create or load from hard drive levels with certain tasks and limitations or load a level from hard drive to change or observe. Once created, a level can be saved to hard drive. The teacher also has access to students’ performance, e.g. their code that was sent to server or number of levels completed by each student. The teacher can manage students accounts: create, edit and delete them, block and unblock certain students.

Levels have goals: e.g. 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, students can program robots for multi-user levels, where their decisions will compete with each other under a teacher's supervision.

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 programs 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 the environment and send the information to user’s programming code.

Simulation – the process of interaction between the code written by the student and the level, during which the level 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 the 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 level, 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.

Log viewer – a user-interface screen that shows the simulation log to the student.

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.

Available levels – levels, stored on the server, chosen by the teacher to be available for individual students or groups.

Main menu – a user-interface screen that appears right after authentication.

Lobby – a user-interface screen related to one particular simulation. If accessed by multiple users, it represents a simulation of a multi-user level.

Available lobbies – the lobbies that are active and can be entered.

Results of a simulation – a set of data about a completed simulation, which includes its outcome (e.g. success or failure), states of the simulation at different simulation cycles, and actions performed by student’s robot.

1. Actors

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 the ability to edit the existing levels or create new ones as well as supervise the students’ performance.

Unauthenticated user - a user that has not authenticated in the system yet.

1. Functional requirements
   1. Use-cases for Student
      1. *Use-case “Simulate robot on the level”*

**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 ‘Connect to server’ button
2. List of available lobbies and levels loaded from the server appears
3. The student clicks a lobby in the list or clicks ‘Create lobby’
4. If student created lobby, they choose a level
5. The code editor appears
6. Student edits the code for the robot
7. Student clicks the ‘Submit’ button
8. The code is sent to the server
9. The code is checked for syntax errors by the system
10. The server conducts the simulation using the code sent by the student
11. Results of the simulation are sent back to the client
12. The outcome (success or failure, statistics) is shown to the student

**Alternative scenario <Code has syntax errors>:**

1. When check 9 failed (the code contains syntactic mistakes), show the errors and suggest to correct the code
2. Student returns to step 5

**Alternative scenario <Cannot get levels due to a network error>:**

1. When unable to get levels in step 2 due to a network error, show student an error message
2. Student returns to the main menu

**Alternative scenario <Cannot send code due to a network error>:**

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 <Simulation error>:**

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”*

**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”*

**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 “View simulation log”*

**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”*

**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”*

**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
4. The student can change the rate of the playback (making it run faster or slower), pause the playback, jump to a certain simulation cycle in the playback or go to the next or previous cycle of the playback. These actions are accessed by clicking corresponding buttons
5. Once finished watching, the student clicks ‘Close’ button
6. The playback is closed and the previously shown screen is shown again
   1. Use-cases for Teacher
      1. *Use-case “Create a new level”*

**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 an environment
4. The Teacher clicks the ‘Release’ button
5. The code is checked for syntactic errors by the system
6. The code and level are sent to the server

**Alternative scenario <Syntactic errors found>:**

1. When check 5 failed (the code contains syntactic mistakes), show the errors and suggest to correct the code
   * 1. *Use-case “Load level to level editor”*

**Actors:** Teacher

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

**Precondition:** Level editor is opened

**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 <Unable to load level>:**

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

**Actors:** Teacher

**Goals:** save level data to the hard drive

**Precondition:** Level editor is opened

**Main success scenario:**

1. The Teacher clicks the ‘Save’ button
2. File explorer window appears
3. The Teacher chooses where to save the file and clicks ‘OK’
4. The explorer window is closed and the source data file is saved
   * 1. *Use-case “See the students rating”*

**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”*

**Actors:** Teacher

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

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

**Main success scenario:**

1. Teacher clicks the ‘Students’ button
2. A list of students is shown
3. Teacher clicks on student’s name
4. A list of levels for which the student created scripts appears, providing a list of all the scripts sent to the server
5. Teacher clicks a script from the list
6. Code editor appears where the student’s script is shown
   * 1. *Use-case “Create student’s account”*

**Actors:** Teacher

**Goals:** Create a new account for student

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

**Main success scenario:**

1. Teacher clicks the ‘Students’ button
2. A list of students is shown as well as a ‘Add student’ button
3. Teacher clicks the ‘Add student’ button
4. An account creation dialog appears
5. Teacher enters new student’s login, password and other optional info
6. Teacher clicks the ‘Submit’ button
7. The data is sent to the server, where a new account is created
8. The result (success or failure) of the account creation is sent back to the client
9. The outcome (success or failure) is shown to the teacher

**Alternative scenario <Unable to create a new account>:**

1. When check 7 failed (invalid data for account), show the errors and suggest to correct the data.
   * 1. *Use-case “Edit student’s account”*

**Actors:** Teacher

**Goals:** Edit student’s account info

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

**Main success scenario:**

1. Teacher clicks the ‘Students’ button
2. A list of students is shown
3. Teacher clicks the ‘Edit’ button near the student’s name
4. An account edition dialog appears
5. Teacher edits student’s account info
6. Teacher clicks the ‘Submit’ button
7. The data is sent to the server, where an account data is modified
8. The result (success or failure) of the account edition is sent back to the client
9. The outcome (success or failure) is shown to the teacher

**Alternative scenario <Unable to edit an account info>:**

1. When check 7 failed (invalid data for account), show the errors and suggest to correct the data.
   * 1. *Use-case “Delete student’s account”*

**Actors:** Teacher

**Goals:** Delete all the data associated with student’s account

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

**Main success scenario:**

1. Teacher clicks the ‘Students’ button
2. A list of students is shown
3. Teacher clicks the ‘Delete’ button near the student’s name
4. A confirmation dialog appears
5. Teacher clicks the ‘I understand that this action cannot be undone and all the account data will be lost completely’ checkbox
6. The ‘Confirm’ button becomes active
7. Teacher clicks the ‘Confirm’ button
8. The request is sent to the server, where student’s account is deleted
9. The result (success or failure) of the account deletion is sent back to the client
10. The outcome (success or failure) is shown to the teacher
    * 1. *Use-case “Block student’s account”*

**Actors:** Teacher

**Goals:** Forbid the student to log in

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

**Main success scenario:**

1. Teacher clicks the ‘Students’ button
2. A list of students is shown
3. Teacher clicks the ‘Block’ button near the student’s name
4. A confirmation dialog appears
5. Teacher clicks the ‘Confirm’ button
6. The request is sent to the server, where student’s account is blocked
7. The result (success or failure) of the account blocking is sent back to the client
8. The outcome (success or failure) is shown to the teacher
   * 1. *Use-case “Unblock student’s account”*

**Actors:** Teacher

**Goals:** Allow the student to log in again

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

**Main success scenario:**

1. Teacher clicks the ‘Students’ button
2. A list of students is shown
3. Teacher clicks the ‘Unblock’ button near the student’s name
4. A confirmation dialog appears
5. Teacher clicks the ‘Confirm’ button
6. The request is sent to the server, where student’s account is unblocked
7. The result (success or failure) of the account unblocking is sent back to the client
8. The outcome (success or failure) is shown to the teacher
   * 1. *Other use-cases are similar to actor 'Student'*
   1. Use-cases for Unauthenticated user
      1. *Use-case “Authentication”*

**Actors:** Unauthenticated user

**Goals:** The unauthenticated user becomes either a teacher or a student

**Precondition:** The user has launched the client application

**Main success scenario:**

1. The user enters login and password
2. The client connects to the server and checks if the entered login and password are valid
3. The user becomes a student or a teacher depending on whose the entered login is
4. Main menu is shown to the user

**Alternative scenario <Incorrect login/password>:**

1. When check 2 failed (the entered login/password is invalid), the user is asked to enter them again

**Alternative scenario <Cannot connect to the server>:**

1. When check 2 failed (can’t reach the server due to a network error), a notification about that is shown to the user
2. System-wide functional requirements

* 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 level 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 levels calculated at the same time can be increased by adding extra computers to the system.