Blockr: A Visual Programming Game

Software-ontwerp
2019-2020

Iteration 3

Contents

1 Introduction **2**

2 General Information **2**
2.1 Team Work 2
2.2 Iterations 3
2.3 The Software 3
2.4 Testing 3
2.5 UML Tools 4
2.6 What You Should Hand In 4
2.6.1 Late Submission Policy 5
2.6.2 When Toledo Fails 5
2.7 Evaluation 6
2.7.1 Presentation Of The Current Iteration 6
2.8 Peer/Self-assessment 7
2.9 Deadlines 7

3 Blockr **7**

4 Use Cases **9**

5 Implementation **10**

1 Introduction

For the course *Software-ontwerp*, you will design and develop *Blockr*, a visual programming game for non-programmers. The main challenge will be the user interface layer, which you will design from scratch. In Section 2, we explain how the project is organized, discuss the quality requirements for the software you will design and develop, and describe how we evaluate the solutions. In Section 3, we explain the problem domain of the application. The use cases are discussed in Section 4. Finally, we specify some implementation constraints in Section 5.

2 General Information

In this section, we explain how the project is organized, what is expected of the software you will develop and the deliverables you will hand in.

2.1 Team Work

For this project, you will work in groups of four. Each group is assigned an advisor from the educational staff. If you have any questions regarding the project, you can contact your advisor and schedule a meeting. When you come to the meeting, you are expected to prepare specific questions and have sufficient design documentation available. *If the design documentation is not of sufficient quality, the corresponding question will not be answered.* It is your own responsibility to organize meetings with your advisor and we advise to do this regularly. Experience from previous years shows that groups that regularly meet with their advisors produce a higher quality design. If there are problems within the group, you should immediately notify your advisor. Do not wait until right before the deadline or the exam!

To ensure that every team member practices all topics of the course, a number of roles are assigned by the team itself to the different members at the start of each iteration (or shortly thereafter in case of the first iteration). A team member that is assigned a certain role will give the presentation or demo corresponding to that role at the end of the iteration. That team member is *not supposed to do all of the work concerning his task*! He must, however, take a coordinating role in that activity (dividing the work, sending reminders about tasks to be done, make sure everything comes together, etc.), and be able to answer most questions on that topic during the evaluation. The following roles will be assigned round-robin:

Design Coordinator The design coordinator coordinates making the design of your software.

Testing Coordinator The testing coordinator coordinates the planning, designing, and writing of the tests for the software.

Domain Coordinator The domain coordinator coordinates the maintenance of the domain model.

As already mentioned, the goal of these roles is to make every team member participate in all aspects of the development of your system. **During each presentation or demo, every team member must be able to explain the**

used domain model, the design of the system, and the functioning of your test suite.

2.2 Iterations

The project is divided into 3 iterations. In the first iteration, you will implement the base functionality of the software. In subsequent iterations, new functionality will be added and/or existing functionality will be changed.

2.3 The Software

The focus of this course is on the *quality* (maintainability, extensibility, stability, readability,...) of the software you write. We expect you to use the development process and the techniques that are taught in this course. One of the most important concepts are the General Responsibility Assignment Software Principles (GRASP). These allow you to talk and reason about an object oriented design. **You should be able to explain all your design decisions in terms of GRASP**.

You are required to provide class and method documentation as taught in previous courses (e.g. the OGP course). When designing and implementing your system, you should use a *defensive programming style*. This means that the *client* of the public interface of a class cannot bring the objects of that class, or objects of connected classes, into an inconsistent state.

Unless explicitly stated in the assignment, you do not have to take into account persistent storage, security, multi-threading, and networking. If you have doubts about other non-functional concerns, please ask your advisor.

2.4 Testing

All functionality of the software should be tested. **For each use case, there should be a dedicated scenario test class.** For each use case flow, there should be at least one test method that tests the flow. Make sure you group your test code per step in the use case flow, indicating the step in comments (e.g. `// Step 4b`). Scenario tests should not only cover success scenarios, but also negative scenarios, i.e., whether illegal input is handled defensively and exceptions are thrown as documented. You determine to which extent you use unit testing. The testing coordinator briefly motivates the choice during the evaluation of the iteration.

Tests should have good coverage, i.e. a testing strategy that leaves large portions of a software system untested is of low value. Several tools exist to give a rough estimate of how much code is tested. One such tool is Eclemma[1]. If this tool reports that only 60% of your code is covered by tests, this indicates there may be a serious problem with (the execution of) your testing strategy. However, be careful when drawing conclusions from both reported high coverage and reported low coverage (and understand why you should be careful). The testing coordinator is expected to use a coverage tool and briefly report the results during the evaluation of the iteration.

[1] http://www.eclemma.org

2.5 UML Tools

There are many tools available to create UML diagrams depicting your design. You are free to use any of these as long as it produces correct UML. One of these UML tools is Visual Paradigm. Instructions to run Visual Paradigm in the computer labs is described in the following file:

`/localhost/packages/visual_paradigm/README.CS.KULEUVEN.BE`

This file also contains the location of the license key that you can use on your own computer.

2.6 What You Should Hand In

Exactly *one* person of the team hands in a ZIP-archive via Toledo. The archive contains the items below and follows the structure defined below. **Make sure that you use the prescribed directory names.**

- directory `groupXX` (where XX is your group number (e.g. 01, 12, ...))
  - `domain.pdf`: the domain model diagram
  - `usecases`: the use case diagram and use case texts
  - `gameworldapispec`: the Game World API Specification. This consists of:
    - a `game_world_api_spec.md` Markdown document which clearly defines the notions of "valid Game World API implementation" and "valid Game World API client" such that any valid implementation can be composed with any valid client to obtain a correctly functioning system.
    - a `javadoc` subdirectory containing the HTML documents (and auxiliary files) generated from the Javadoc comments in the Game World API's source code using the `javadoc` tool. The `game_world_api_spec.md` document may refer to these HTML documents for details.
  - `doc`: a folder containing the Javadoc documentation of your entire system
  - `diagrams`: a folder containing UML diagrams that describe your design (at least one structural overview of your entire design, and sufficient detailed structural and behavioural diagrams to illustrate every use case)
  - `gameworldapi/src`, `robotgame/src`, `mygame/src`, `blockr/src`, `simplegameapp/src`: for each module, a folder containing the module's source code
  - `gameworldapi.jar`, `robotgame.jar`, `mygame.jar`, `blockr.jar`, `simplegameapp.jar`: for each module, a JAR file containing its compiled code. For any valid Game World API implementation `impl.jar` (as defined by the Game World API Specification) with root class `impl.root.package.ImplRootClass` and any valid Game World API

client `client.jar` (again, as defined by the Game World API Specification) with main class `client.main.package.ClientMainClass`, running[2]

```
java -classpath gameworldapi.jar;impl.jar;client.jar
    client.main.package.ClientMainClass
    impl.root.package.ImplRootClass
```

(on one line) shall produce the correct behavior expected of the composition of the given implementation and the given client. (If a different command line is required to launch the various combinations of game world implementations and clients, this shall be described and motivated clearly in a `RUNNING.md` file.)

`robotgame.jar` and `mygame.jar` shall be valid Game World API implementations; `blockr.jar` and `simplegameapp.jar` shall be valid Game World API clients.

– `design.pdf` (optional): a document that clarifies your design and the main decisions; this document can be a prose text or a slide deck, or any other form that you deem appropriate.

When including your source code into the archive, make sure to *not include files from your version control system.* Make sure you choose relevant file names for your analysis and design diagrams (e.g. `SSDsomeOperation.png`). You do **not** have to include the project file of your UML tool, only the exported diagrams. We should be able to start your system by executing the JAR file with the following command: `java -jar system.jar`.

Needless to say, the general rule that anything submitted by a student or group of students must have been authored exclusively by that student or group of students, and that accepting help from third parties constitutes exam fraud, applies here.

2.6.1 Late Submission Policy

If the zip file is submitted N minutes late, with $0 \leq N \leq 240$, the score for all team members is scaled by $(240 - N)/240$ for that iteration. For example, if your solution is submitted 30 minutes late, the score is scaled by 87.5%. So the maximum score for an iteration for which you can earn 4 points is reduced to 3.5. If the zip file is submitted more than 4 hours late, the score for all team members is 0 for that iteration.

2.6.2 When Toledo Fails

If the Toledo website is down – and *only* if Toledo is down – at the time of the deadline, submit your solution by e-mailing the ZIP-archive to your advisor. The timestamp of the departmental e-mail server counts as your submission time.

[2] On macOS and Linux, use colons instead of semicolons to separate classpath entries.

2.7 Evaluation

After iteration 1, and again after iteration 2, there will be an intermediate evaluation of your solution. An intermediate evaluation lasts 35 minutes and consists of: a presentation about the design and the testing approach, accompanied by a demo of the system and the test suite.

The intermediate evaluation of an iteration will cover only the part of the software that was developed during that iteration. Before the final exam, the *entire* project will be evaluated. It is your own responsibility to process the feedback, and discuss the results with your advisor.

The evaluation of an iteration is planned in the week after that iteration. Immediately after the evaluation is done, you mail the PDF file of your presentation to Prof. Bart Jacobs & Tom Holvoet and to your advisor.

2.7.1 Presentation Of The Current Iteration

The main part of the presentation should cover the design. The motivation of your design decisions *must* be explained in terms of GRASP principles. Use the appropriate design diagrams to illustrate how the most important parts of your software work. Your presentation should cover the following elements. Note that these are not necessarily all separate sections in the presentation.

1. An updated version of the domain model that includes the added concepts and associations.

2. A discussion of the high level design of the software (use GRASP patterns). Give a rationale for all the important design decisions your team has made.

3. A more detailed discussion of the parts that you think are the most interesting in terms of design (use GRASP patterns). Again we expect a rationale here for the important design decisions.

4. A discussion of the testing approach used in the current iteration.

5. An overview of the project management. Give an approximation of how many hours each team member worked. Use the following categories: group work, individual work, and study (excluding the classes and exercise sessions). In addition, insert a slide that describes the roles of the team members of the current iteration, and the roles for the next iteration. Note that these slides do not have to be presented, but we need the information.

Your presentation should not consist of slides filled with text, but of slides with clear design diagrams and keywords or a few short sentences. The goal of giving a presentation is to communicate a message, not to write a novel. All design diagrams should be *clearly readable* and use the correct UML notation. It is therefore typically a bad idea to create a single class diagram with all information. Instead, you can for example use an overview class diagram with only the most important classes, and use more detailed class diagrams to document specific parts of the system. Similarly, use appropriate interaction diagrams to illustrate the working of the most important (or complex) parts of the system.

2.8 Peer/Self-assessment

In order for you to critically reflect upon the contribution of each team member, you are asked to perform a peer/self-assessment within your team. For each team member (including yourself) and for each of the criteria below, you must give a score on the following scale: *poor/lacking/adequate/good/excellent*. The criteria to be used are:

- Design skills (use of GRASP and DESIGN patterns, ...)
- Coding skills (correctness, defensive programming, documentation,...)
- Testing skills (approach, test suite, coverage, ...)
- Collaboration (teamwork, communication, commitment)

In addition to the scores themselves, we expect you to briefly explain for each of the criteria why you have given these particular scores to each of the team members. The total length of your evaluation should not exceed 1 page.

Please be fair and to the point. Your team members will not have access to your evaluation report. If the reports reveal significant problems, the project advisor may discuss these issues with you and/or your team. Please note that your score for this course will be based on the quality of the work that has been delivered, and not on how you are rated by your other team members.

Submit your peer/self-assessment by e-mail to both Prof. Bart Jacobs & Tom Holvoet and your project advisor, using the following subject: **[SWOP] peer-/self-assessment of group $groupnumber$ by $firstname$ $lastname$**.

2.9 Deadlines

- The deadline for handing in the domain model diagram, the use case diagram and the use case texts is **26 May, 2020, 3:30pm** (see Subsection 4).
- The deadline for handing in the ZIP-archive on Toledo is **26 May, 2020, 3:30pm**.
- The deadline for submitting your peer/self-assessment is **28 May, 2020, 3:30pm**, by e-mail to both your project advisor and Prof. Bart Jacobs & Tom Holvoet.

3 Blockr

The goal of the project is to develop a Blockly[3]-like visual programming game.

In this third iteration, you shall extend Blockr to offer two additional types of blocks: function definition blocks and function call blocks. Blockr functions have no parameters and no return value; they only have a body, defined by the contents of the function definition block's cavity.

The client area of the Blockr application window is divided into three parts: on the left-hand side is the Palette; in the center is the Program Area; and on the right-hand side is the Game World.

[3] `https://blockly.games/`

The Game World shows a two-dimensional grid. Some of the grid cells contain walls. One cell contains a robot. At any point in time, the robot is oriented either to the right, to the left, up, or down. This is shown visually. One cell is a goal cell.

The point of the game is for the user to build a program that causes the robot to reach the goal cell. Programs are built by dragging blocks from the Palette to the Program Area and connecting them by dragging a block until one of its sockets is near a compatible plug of another block. In this iteration, the application supports the following types of blocks:

- Move Forward, Turn Left, Turn Right, and function call blocks have one socket at the top and one plug at the bottom of their outsides.
- While and If blocks have one socket at the top and one plug at the bottom of their outsides; additionally, they have one socket to their right side where a condition can be connected. Furthermore, they have a cavity to their right side, and they have a plug at the top and a socket at the bottom of the cavity. The cavity initially has zero height, and the plug plugs into the socket. However, when some other block is dragged until its top socket is near the plug of the cavity and then dropped, the cavity expands to accommodate the block.
- Wall In Front blocks have one plug to their left.
- Not blocks have one plug to their left and one socket to their right.
- Function definition blocks have no plugs or sockets on their outside. They have a cavity to their right side, and they have a plug at the top and a socket at the bottom of the cavity. The cavity initially has zero height, and the plug plugs into the socket. However, when some other block is dragged until its top socket is near the plug of the cavity and then dropped, the cavity expands to accommodate the block.

The user can press F5 to step through an execution of the program. This will have an effect only if the Program Area contains exactly one group of connected blocks plus zero or more function definitions, where all condition sockets are connected to condition blocks. If so, the application performs one step of execution and updates the Game World to show the new position and orientation of the robot. During execution, the block to be executed next is highlighted. Pressing the Escape key or making any change to the program resets program execution and the state of the game world.

A game is defined by a grid configuration and a maximum total number of blocks available. The Palette initially contains one block of each type, except that it contains no function call blocks. When the user drags a block from the Palette to the Program Area, then if the maximum number of blocks available has not yet been reached, a new block of the same type appears in the Palette. Otherwise, all types of blocks disappear from the Palette. They reappear after the user drags a block back from the Program Area to the Palette.

When a function definition block is dragged from the Palette to the Program Area, a unique number is assigned to the function definition. For each function

definition in the Program Area, a function call block labelled with the corresponding definition's number is shown in the Palette. For any function definition in the Program Area, any number of calls of that function may be dragged into the Program Area. When a function definition is removed from the Program Area, all calls of that function are also removed from the Program Area.

You shall implement Undo and Redo functionality in the Blockr application: the user shall be able to undo any modification of the program in the Program Area and any change to the execution state of the program (together with the corresponding change in the game world) by pressing Ctrl+Z. As in most applications, the user shall be able to undo multiple changes by pressing Ctrl+Z repeatedly; the user shall also be able to undo the most recent Undo operation (this is known as a Redo) by pressing Shift+Ctrl+Z; and the user shall be able to Redo all of the modifications that were undone, except for the ones that happened before the most recent new original modification, by pressing Shift+Ctrl+Z repeatedly.

Part of the assignment for this iteration is to define a domain model that describes this problem domain in terms of entities, associations, and attributes, and to graphically represent it in the form of a UML class diagram. We recommend that you use the UMLet UML drawing tool to create this diagram.

4 Use Cases

Part of the assignment for this iteration is to create a requirements model for this assignment in the form of a set of use cases. We recommend writing the use cases document in Markdown as follows:

```
# Use Cases

## Use Case 1: Add Program Block

### Main Success Scenario

1. The user moves the mouse cursor over a block in the Palette,
   then presses the left mouse key, then moves the mouse cursor
   to the Program Area, and then releases the left mouse key.
2. The system adds a new block of the same type to the Program
   Area.

### Extensions

1a. When the user releases the mouse key, one of the block's
    connectors is near a compatible opposite connector of another
    block.

  1. The system adds a new block of the same type to the Program
     Area; the new block is inserted into an existing group of
     connected blocks at the matching connection point.
```

Note: if you use Markdown to author your use cases document, do not hand in (a PDF listing of) the raw Markdown code. Hand in a PDF of the rendered view, as obtained using a Markdown renderer such as the Markdown Viewer[4] browser extension.

Also create an overview of the use cases in the form of an UML use case diagram. We recommend using UMLet for this.

The domain model diagram, the use cases document, and the use case diagram must be handed in in PDF form via Toledo along with the rest of your submission.

Notes:

- The above example use case should be further elaborated to clarify the application's behavior in certain specific cases.
- A system's requirements can be specified at various levels of abstraction with respect to the nature of the interface between the system and its environment (e.g. a human user). Often the specification abstracts over the nature of the interface to focus on the aspects that are most relevant to the usefulness of the system. However, in this assignment, since the design of the code that implements the user interface is a core challenge, it is appropriate to specify the interaction with the user in unusually specific detail.
- While the tool you develop should be functional, the user interface need not be of the level of "finish" that would be expected of a commercial product. For example:
  - You need not support scrolling the window if the information does not fit into the window.
  - You need not (in this iteration) provide a menu, Save/Open functionality, printing functionality, etc.

5 Implementation

You must implement your system in Java.

Since the main intended challenge of this assignment is that you design your user interface layer from scratch, for this assignment you are not allowed to use an existing GUI toolkit, such as Java's Swing/AWT or SWT. Instead, we require that you use only the CanvasWindow Java class that we provide to implement the user interface. (You can also use the AWT elements that are necessary to use this class, but you cannot use the AWT or Swing component hierarchies.)

You shall modularize Blockr. Specifically, you shall factor the block programming logic and the robot-wall-grid logic out into separate components, separated by a Game World API. The Game World API shall not involve any block programming concepts. It shall be possible to implement this API with different kinds of game worlds, and to consume this API from different kinds of applications. To prove this, you shall develop, in addition to the Blockr application, another simple Game World API client application that simply allows the user

[4] `https://github.com/simov/markdown-viewer`

to play the game directly using the mouse; and you shall develop, in addition to the robot-wall-grid game world, another Game World API implementation that implements a completely different (but still very simple) kind of game that you come up with yourself.

Correspondingly, instead of delivering a single monolithic `system.jar` executable, you shall design and deliver the following:

- a Game World API, probably consisting mostly of interfaces and perhaps enums, compiled, with no dependencies, into `gameworldapi.jar`.
- a Robot Game World component that implements the Game World API and depends only on `gameworldapi.jar`, and does not depend in any way on block programming concepts or artifacts. Compiled into `robotgame.jar` with only `gameworldapi.jar` in the classpath.
- another game world component of your own invention that also implements the Game World API and also depends only `gameworldapi.jar`. Keep it simple—this component's only purpose it to show that indeed your application can work with game worlds other than the Robot Game World. Compiled into `mygame.jar`.
- the Blockr application. Its only compile-time dependency is the Game World API. It takes the class name of the root class of a Game World API implementation (which implements an appropriate Game World API interface) as a command-line argument. It uses `Class.forName` and `Constructor.newInstance` to create an instance and call the Game World API interface methods on it. It does not depend in any way on robot, wall, or grid concepts or artifacts. Compiled into `blockr.jar` with only `gameworldapi.jar` in the classpath.
- another application, also built using the provided `CanvasWindow` class, that takes a game world implementation class name as a command-line argument. It simply allows the user invoke the game world's actions directly using the mouse. (Each action is exposed as a button that the user can click.) Compiled into `simplegameapp.jar` with only `gameworldapi.jar` in the classpath.

Develop each component in its own Eclipse/IntelliJ project. Make sure each project declares only permitted dependencies on other projects in its Build Path. Specifically: `gameworldapi` shall have no dependencies, and the other components shall have only `gameworldapi` as a dependency.

(Note: whereas `robotgame` and `mygame` shall not be in `blockr` or `simplegameapp`'s compile-time classpath (or vice versa), they do need to be in their run-time classpath during testing. Unfortunately, Eclipse does not have separate compile-time and run-time classpaths. However, you can simulate this by adding `robotgame` and `mygame` as dependencies in the Projects tab of the Eclipse Build Path page, and then, still in the Projects tab of the Build Path page, expanding the corresponding nodes, selecting the "Access Rules" node, clicking Edit, and adding a rule `Forbidden: **`. This causes Eclipse to generate an error if you use any type from the dependency in your project.)

The Game World API defines a GameWorldType interface, that offers methods to:

- retrieve the list of Actions supported by the GameWorldType. For Robot, this is: Turn Left, Turn Right, Move Forward.
- retrieve the list of Predicates supported by the GameWorldType. For Robot, this is: WallInFront.
- create a new game world instance, which implements interface GameWorld.

The GameWorld interface offers methods to:

- perform one of the supported Actions. It returns a result indicating successful execution, failure to execute because the action is illegal in the current state, or end of the game due to reaching the goal state.
- evaluate one of the supported Predicates.
- create an (opaque, i.e. non-inspectable) snapshot of the game world state and to restore the game world state to a given snapshot.
- paint the current state of the game world, given a graphics object (either java.awt.Graphics or another graphics API of your own design).

An important deliverable of your modularization effort is a Game World API Specification, that clearly specifies the requirements to be satisfied by Game World API implementations and clients so that they can be composed successfully to obtain a correctly functioning system. Good luck!

The SWOP Team members