# Learning Modeling Languages Using Strategies from Gaming

Øyvin Richardsen

Master of Science in Communication Technology
Submission date: June 2014
Supervisor: Frank Alexander Krämer, ITEM

Norwegian University of Science and Technology
Department of Telematics

# Problem Description

When learning to use a software modeling language, people are usually directed towards the language's documentation, or "language tutorials" in the form of a set of instructions required to get started with the language. Most of the information about the language is generally delivered early in the process, requiring the developer to either understand (and remember) a lot of concepts at once, or to go back and search for the information when it's needed.

On the contrary, many computer games come with no instructions at all, but rather start by guiding the player through a set of training levels. Each level introduces a new element, concept or strategy, and sometimes let the player experiment with the new concepts to solve more advanced problems.

In this thesis, it will be examined to which degree this approach can be used for introducing and learning a new modeling language. Methods for creating more immersive tutorial experiences with required information delivered in-context will be explored and tested, using UML Activities and the Reactive Blocks tool as an example. The thesis will also explore whether the context of the tutorial matters, and if there are possible advantages gained by designing the tutorial experience itself as a game.

**Assignment given:** *Learning Modeling Languages Using Strategies From Gaming*
**Supervisor:** Frank Alexander Kraemer

# Abstract

Learning about software modeling languages from documentation can be a difficult and confusing process, and many of the currently existing modeling language tutorials are only marginally better. At the same time, players of video games spend hours upon hours learning to play games that require mastering complex strategies and concepts, without losing motivation or interest. This success for motivating learning effort seen in many games, is in turn supporting an emerging trend of educational games, designed to teach a wide range of subject to people of all ages. This thesis presents an exploration of the principles and strategies used by video games to teach players their mechanics, and an attempt to use these principles to teach software modeling in an engaging way.

Focusing on modeling with Unified Modeling Language (UML) activities in the context of Reactive Blocks, two different approaches for teaching the concepts of this topic are presented. The first approach is simply an "improved" tutorial, utilizing principles such as *interactiveness* and *context-sensitivity of information and instruction* to engage learners. The second approach is an educational game, adding *immersion* and *visualization* to the learning experience.

The design and prototype implementation of both the interactive tutorial and the educational game, and the principles they are based on, are described in detail. Both prototypes are also evaluated with respect to these principles, focusing on their usability and teaching potential, with the support of data from user tests.

# Abstract (Norsk)

Å lære et språk for software-modellering fra dens dokumentasjon kan være både vanskelig og forvirrende, og mange eksisterende *tutorials* for slike språk er bare marginalt bedre. Samtidig bruker mennesker som spiller videospill timesvis på å lære spill som krever mestring av komplekse strategier og konsepter, uten å miste hverken interessen eller motivasjonen. Disse spillenes evne til å gi motivasjon for læringsinnsats støtter en voksende trend for læringsspill, en type spill ment for å lære bort ulike emner til mennesker i alle aldre. Denne avhandlingen utforsker prinsippene og strategiene som gjør videospill til gode læringsplattformer, og gjør et forsøk på å bruke disse prinsippene til å lære bort software-modellering på en engasjerende måte.

Avhandlingen fokuserer på modellering med *UML activities* i kontekst av Reactive Blocks, og presenterer to forskjellige tilnærminger for å lære bort de ulike konseptene det innebærer. Den første tilnærmingen er en forbedret tutorial, som bruker prinsipper om interaktivitet og å gi informasjon og instruksjon i riktig kontekst for å engasjere studenter. Den andre tilnærmingen er et læringsspill, som i tillegg lar studentene visualisere konseptene og fordype seg i læringsomgivelsene.

Avhandlingen beskriver i detalj design og implementasjon av prototyper for både den interaktive tutorialen og læringsspillet, og prinsippene de er basert på. Begge prototypene evalueres også med hensyn til disse prinsippene, med fokus på brukervennlighet og potensiale som læringsverktøy, og med støtte i data fra brukertester.

# Preface

This thesis is submitted as the concluding part of my Master of Science degree in Communication Technology at the Norwegian University of Science and Technology (NTNU). The supervisor for this thesis has been Associate Professor Frank A. Kraemer, and I would like to thank him for his help and guidance. I would also like to thank the people who helped me with the testing of my work.

Trondheim, June 2014
Øyvin Richardsen

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**API** Application Programming Interface.

**ESM** External State Machine.

**HCI** Human-Computer Interaction.

**HUD** Head-Up Display.

**MDA** Model-Driven Architecture.

**MDD** Model-Driven Development.

**MIT** Massachusetts Institute of Technology.

**NTNU** Norwegian University of Science and Technology.

**OMG** Object Management Group.

**SDL** Specification and Description Language.

**UI** User Interface.

**UML** Unified Modeling Language.

# Chapter 1 Introduction

Video games are a popular form of entertainment, played by children and adults alike [ESA14]. In addition to being a source of entertainment, video games provide environments that facilitate learning of many different skills [Gee07]. The popularity of video games, combined with their potential as good learning environments, has resulted in a boom of *educational games* and *game-like teaching methods*, covering a vast amount of different subjects in all levels of education [EFG13, MOJ11, Pat81, RMMH$^{+}$09]. Some even consider educational games to be more suitable than many traditional teaching methods for today's younger generations [Ass14, SG05].

Software modeling is the process of expressing the architecture, structure, or semantics of a system through visual diagrams and abstractions, as opposed to lower-level representations like source code. There can be many reasons for using models to describe software, including making its structure easier to understand, and having a specification that is independent of any implementation [Bræ04, Sel03].

When wanting to learn about a modeling language, the resources available are often limited to the language's documentation, or "tutorials" that simply provide short summaries of the modeling language.[1] Both options are relatively cheap and easy by creators to provide, and generally describe the given language thoroughly and accurately, but neither are particularly good as primary *learning* resources.

The purpose of this thesis is to explore the use of learning principles and strategies from video games to teach modeling languages. The goal is that by using these principles, learners will hopefully be as motivated and interested to learn about a modeling language as when they are playing and learning a video game. More specifically this includes providing an interactive learning experience, teaching advanced concepts one step at a time, in an environment that is intuitive to use and understand.

[1]For an example, see http://www.tutorialspoint.com/uml/uml_activity_diagram.htm

## 1.1 Scope and Methodology

This thesis focuses on using strategies and principles from games to teach concepts within UML, more specifically UML activities in the context of Reactive Blocks. Various teaching principles used in video games and their tutorials, as well as some related research on the usability of tutorials, is analyzed in order to provide a theoretical foundation for a better learning environment for UML activities.

Based on the theoretical foundation, two different learning environments are designed and implemented: an improved and interactive tutorial, and an educational game for UML activities. The implementations are then user tested and qualitatively evaluated with respect to usability and teaching potential.

The work of this thesis provides a starting point for exploring this particular field of study. We establish a set of principles that should be considered when creating a good learning environment for modeling languages, outline the various challenges associated with this, and suggest some concrete solutions in the form of tutorial and game prototypes. Because of the proprietary nature of Reactive Blocks, not all of the established principles are possible to implement in the prototypes within the scope of this thesis, but their absence is accounted for in the evaluations.

## 1.2 Organization of the Thesis

The work of this thesis has been an incremental process, and is presented as such. First, the world of tutorials and educational games is explored in order to establish a set of principles to base the thesis on. These principles are then used to design, implement, and test an improved tutorial for UML activities in Reactive Blocks. In turn, experiences and results from evaluation of the tutorial serve as the basis for an educational game, in an attempt to improve the learning experience even further. Finally, the educational game is evaluated with respect to the principles established in the first part of the thesis.

Chapters 2 and 3 form the background part, offering mostly introductions to the primary topics touched by this thesis, and brief summaries of related work. A brief analysis of the background material is also provided, highlighting the parts that are most important for the work described in the following chapters.

Chapter 4 covers the design, implementation, testing, and evaluation of the UML activity/Reactive Blocks tutorial. First, parameters are established, such as target audience and design goals. A design for the tutorial is then proposed and partially implemented, including a teaching order for concepts, short concept introductions, an exercise for each tutorial step, and design for a “tutorial mode” in Reactive Blocks.

The tutorial is then tested on a small number of subjects, and evaluated with respect to the design goals, general usability, and teaching potential.

Chapter 5 contains the second part of the work of this thesis, namely the process of designing, implementing, testing and evaluating a prototype of the Reactive Blocks game. This process is similar to that of creating the tutorial, starting with some design goals. An overall game concept is then established, with designs of levels and a tutorial part. Based on the design, a prototype is implemented and tested with focus on usability and uncovering problems. Finally, the game is evaluated with respect to usability and the design goals.

As the concluding part of this thesis, Ch. 6 provides a summary of all the work done, with some additional reflections and suggestions for future work.

Some of the thesis material is inconvenient or inappropriate to include in this document, and is provided either as supplementary files or links to online resources. More specifically, this includes source code for the tutorial and the Reactive Blocks game, slide shows for the introductions in the game, and video recordings of each level in the game.

# Chapter 2 Background

This chapter introduces some background material that serves as the basis for this thesis, covering in part the various topics it touches. The chapter is meant only to introduce these topics and provide brief summarizations, providing a "quick reference" for further reading of this thesis. Some additional background material, more specifically *related work*, is covered in Ch. 3.

## 2.1 Software Modeling and Modeling Languages

In the world of software development, potential problems and challenges that may arise when developing a product are plentiful. In anything but trivial systems, bugs are almost inevitable, and may be caused by anything from plain programming errors to intermittent problems as a result of resource contention. Systems with concurrent behavior are particularly difficult to implement sensibly, as there are several additional complexities and pitfalls associated with these. Potential problems are also not only related to bugs, but possibly also performance and correctness.

In addition to many types of software being nontrivial to implement, keeping the code *readable* is an undertaking of its own. When new developers are added to a team that is already deep into the development process, it can take significant time and effort to become familiar with the system. This can be a result of several factors, such as different code styles, massive spread of code (over a vast amount of files or classes), or simply bad code. In many cases, we may also want non-programmers, such as customers or salespeople, to understand what the system does.

Software modeling serves as a solution for smoothing out these processes. Instead of the system specification existing only as code, or a list of functionality and bugs, it is possible to create a visual and formal model of the system. Such models are more readily understood by non-programmers, and also help other developers become familiar with the overall system architecture more quickly.

### 2.1.1 The Purpose of Software Modeling

The real world is often complex, and correctly expressing this complex world in terms a computer can understand is not a trivial task. Software modeling in languages such as the UML or the Specification and Description Language (SDL) is a useful tool, helping smooth out this process in several ways.

**Conceptual Abstractions** The most important purpose of software models is perhaps to provide *conceptual abstractions*, by describing system functionality and collaborations on a higher level, removing less relevant detail [Bræ04]. This allows developers and other interested parties to get a clear overview of the system architecture, without having to dig through thousands of lines of code.

**Understandability** In addition to providing abstractions that hide details, software models are able to present architecture and functional concepts in ways that better appeal to our intuition [Sel03]. This further reduces the learning effort required, and may help us to a better understanding of the workings of a system.

**Separation of Concerns** Another purpose of software modeling is *separation of concerns*, which involves reducing the perceived complexity of the system by modeling parts that are fairly independent as separate, but collaborating, entities [Bræ04]. This may also allow the system to be modularized, potentially simplifying both implementation and maintenance of the system.

**Formal Analysis** With the use of formal and well-established modeling methods, we also open up the system design to formal analysis. Depending on the method, we may be able to mathematically or logically analyze the system architecture in order to uncover inconsistencies or errors, or calculate additional system requirements such as hardware capabilities or bandwidth. This also adds *predictiveness* to the system, providing indications of how the final implementation will behave under various conditions [Sel03].

### 2.1.2 UML Activities

The UML activity diagram is a modeling concept suitable for expressing processes and workflows, particularly where concurrent behavior is observed. UML activities use *Petri-net*-like semantics, and can be mathematically or logically verified in various ways [EW02, KBH09, Stö05].

In general, UML activity diagrams may consist of a number of different types of elements. These include, among others, logical elements such as *forks* and *joins*, allowing concurrent execution and synchronization of these. Other examples are *decisions*, implementing alternate branches, and *wait nodes*, allowing the application to wait for an event before continuing execution. The activity is executed as a series of *activity steps*, where each step involves performing the logic of one or more elements. An example of a UML activity diagram containing many of these elements is displayed in Fig. 2.1. The various elements are described in more detail within the context of Reactive Blocks, in Sect. 2.1.4.

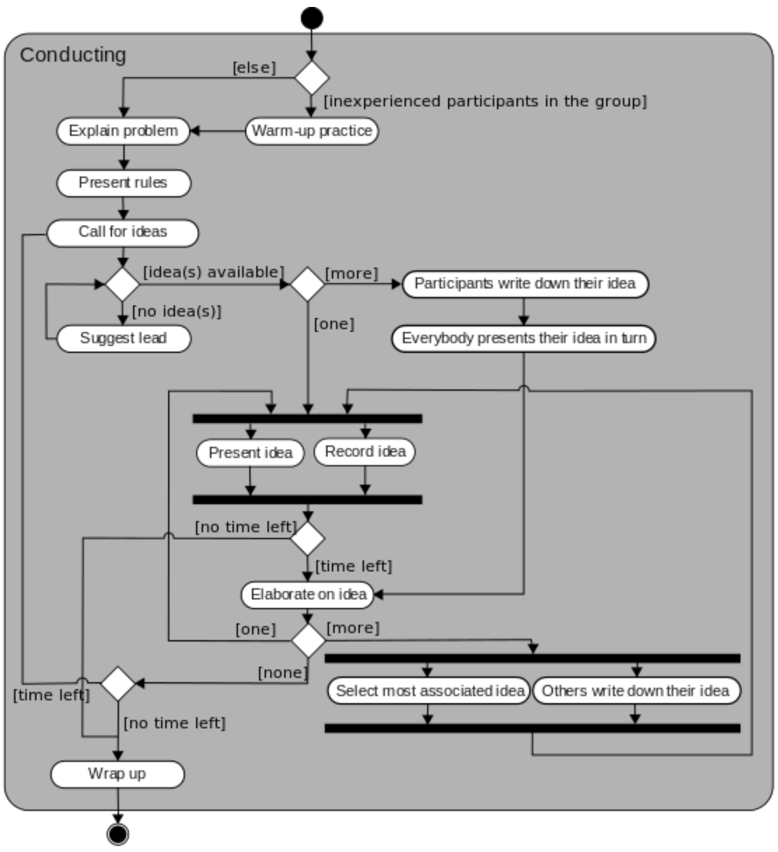


**Figure 2.1:** An UML activity diagram for a brainstorming process. The process consists of various *actions*, such as "Explain problem" or "Present idea", that are separated primarily by conditional logic through *decisions*. We also see some examples of concurrent behavior with synchronization, more specifically before and after the "Present idea" and "Record idea" actions. Both actions are performed simultaneously, and execution does not continue until both actions have finished. We also see the start and end points of activity execution, the *initial node* and the *final node*, as the two black circles above and below the gray area. *Image source: http://en.wikipedia.org/wiki/Activity_diagram*

### 2.1.3 Model-Driven Development

While software modeling simply can be used to provide better documentation of systems and visualize their architecture, it is also possible to go one step further and employ the *Model-Driven Development (MDD)* approach (closely related to the concept of *Model-Driven Architecture (MDA)*[1]). With the MDD approach, software models are at the center of the development process, rather than playing a supplementing or secondary role. Software models have the advantage that they can be very independent from the implementation platforms, allowing the software structure to be mapped, or *deployed*, to more than one platform [Bræ04, Sel03].

Digging deeper into the MDD approach, it has been argued that its full potential can only be realized if the models are used to automatically generate complete and fully executable programs [Sel03]. Given a robust and correct deployment, automatic generation of code can help improve correctness, maintainability, and number of software bugs.

There are various commercial and non-commercial projects working on solutions for MDA and MDD, many of them listed on the Object Management Group (OMG) website.[2] Reactive Blocks, currently in development by Bitreactive AS,[3] is an example of a solution that generates fully executable applications based on UML activity-like models.

### 2.1.4 Reactive Blocks

Reactive Blocks is the result of various research efforts in the field of compositional service engineering and MDD at NTNU[Kra07]. The software is released as a plugin for the Eclipse IDE, providing a modeling environment for UML activities. Using the Reactive Blocks tool, these models can be formally analyzed, and further used to automatically generate fully executable applications.

Models Reactive Blocks are based on and very closely resemble UML activity diagrams. Figure 2.2 shows an example of a model in Reactive Blocks, containing many of the UML activity elements mentioned in Sect. 2.1.2.

Activities in Reactive Blocks are executed like regular UML activities, as a series of *activity steps*. These steps are implemented in the form of *tokens* moving through sequences of elements (*flows*), activating them along the way. The step finally ends when the token reaches a *stable position*, where it will either stay until the start of another step or be removed. Reactive Blocks models can be built from a number of modeling elements and nodes:

[1]OMG: MDA - The Architecture Of Choice For A Changing World (link)
[2]OMG: MDA - Committed Companies And Their Products (link)
[3]Bitreactive website (link)

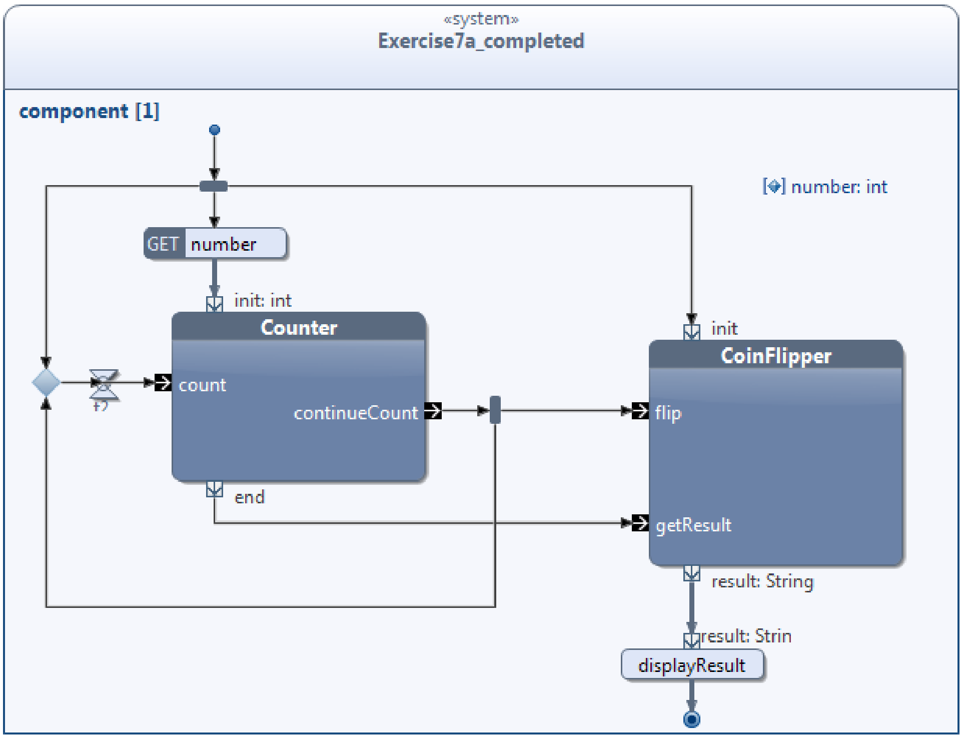


**Figure 2.2:** An example of a model in Reactive Blocks, resembling the UML activity diagram in Fig. 2.1. The *Counter* and *CoinFlipper* elements are building blocks, containing modularized logic exposed through an API.

– **Edge:** *Edges* connect other elements in Reactive Blocks, carrying tokens and implementing the flow between them. All edges are directed, and may additionally carry data objects with tokens. With data, the edge is called an *object flow*, and without data, simply a *control flow*.

– **Initial Node:** The *Initial Node* is where the activity execution starts. When the application is launched, a token is sent out from all initial nodes.

– **Operation/Java method:** Corresponding to the *actions* displayed in Fig. 2.1, *operations* are elements that work on data or APIs, implemented in Reactive Blocks as Java methods. Operations may take one or more parameters, and in the case of more, they also work like *join* nodes.

– **Variables** *Variables* are elements that store data, and can be accessed by regular operations, or directly within the activity diagram via special *set* and *get* operations.

- **Flow Final:** The *Flow Final* node terminates a flow, removing any incoming token.
- **Activity Final:** The *Activity Final* node works like the flow final, except it also terminates the application.
- **Timer:** The *Timer* node realizes simple delays by putting any incoming tokens in a stable position until the timer has expired, upon when a new activity step is started.
- **Decision:** *Decision* nodes are used to implement alternate branches. Incoming tokens must carry data, and the outgoing edge is chosen by checking the carried data against the *guards* set on these edges. Guards may be either boolean, integer or String values, or *null*.
- **Fork:** *Fork* nodes are used to implement concurrent branches and parallel behavior, duplicating any incoming tokens to all outgoing edges.
- **Join:** The complementary part to forks, the *Join* node synchronizes several flows, concurrent or not. When a token has been received on *all* incoming edges, a *single* token is sent on the outgoing edge.
- **Merge:** *Merge* nodes are similar to join nodes, but instead of waiting for a token to arrive on each incoming edge, *all* tokens from incoming edges are simply forwarded on the outgoing edge.
- **Event Reception:** *Event Receptions* are stable positions that wait for an internal signal, or an *event*, before sending a token on the outgoing edge. These events are generally sent from operations somewhere else in the system.
- **Building Blocks:** *Building Blocks* are independent sub-modules that implement some functionality. These modules are themselves constructed like activity diagrams, but do not have initial or activity final nodes. Instead, they have input/output pins that accept tokens, and External State Machines (ESMs) that determine which pins can accept tokens in a given state.

Because it is a tool for UML activity modeling, with model checking and code generation capabilities, Reactive Blocks will serve as the basis for the work in this thesis.

## 2.2 Learning Programming and Software Modeling

In order to teach potential users to best utilize topics such as a programming language, programming paradigm, protocol or modeling language, various tools or sources of information may be provided. Generally, some form of formal documentation or specification of the language or protocol is considered mandatory, and serves as the *definitive* source of information.

While a topic's documentation often serves as its primary source of information, such formal documents are not necessarily the best source of information when *learning* about the given topic. Often it is not intended as a learning resource, but rather as a formal reference for users already familiar with the topic. In the documentation, the topic is generally presented in a structured way with respect to its various aspects, so that it is easy for someone already familiar with the topic to find the required information.

A good example of this is the UML specification.[4] While the specification offers detailed information about every aspect of UML in a way that may be suitable for an experienced user, it is likely confusing and not particularly helpful for a user with no previous experience or knowledge about UML. Using this specification as a starting point for learning UML modeling is likely to require a lot of effort from the user. In the worst case, the user may not even learn all the concepts properly, despite the provided information being very specific and accurate.

More importantly, the user may not learn how to properly apply the learned concepts to a given situation. There might be subtleties that are difficult to discover without proper guidance, and possibly even potential for misconceptions. This can leave many users with insufficient knowledge and skills within the topic, decreasing the effectiveness and quality of their work, and in the worst case, even preventing them from completing more advanced task.

While documentation for a topic undoubtedly is a valuable resource, we can safely conclude that it is not always the best learning tool. It has some advantages, such as generally providing very complete and precise information, but this may simply be too much for novice users to comprehend. Instead, novice users more often turn to *tutorials* when wanting to learn about a new topic. For many topics, various tutorials and exercises are often created in order to provide a better introduction to these, decreasing the threshold for learning.

[4]OMG: Documents Associated With Unified Modeling Language (UML), v2.4.1 (link)

## 2.3 Tutorials

Tutorials are often used to teach and introduce new topics to users who previously have little or no experience with it. Most commonly, tutorials cover topics and concepts that are difficult to understand intuitively, or to highlight aspects of a concept that are nontrivial and less obvious. Tutorials may be designed for learning a vast range of topics, such as:

- Programming, or using specific programming or modeling languages[5,6]
- Spoken languages[7]
- Software products[8]
- Video games (these tutorials are usually presented inside the game itself)
- Real-life skills, like photography[9]
- Human sciences[10]

The above list is in no way exhaustive, but meant to offer some examples of the numerous and diverse topics that may be introduced with the help of a tutorial. In the following chapters, we are primarily concerned with tutorials for programming, software products and video games.

Additionally, tutorials come in many forms. The most common form of tutorial is likely the text-based tutorial, often supplemented by illustrations and pictures, but tutorials also come in the form of videos, animations, audio, or in the case of many video games, an interactive experience combining many or all of these.

The term “tutorial” also defines a concept in the context of British (and other) academia, which is a small class tutored by a lecturer. This type of tutorial is not relevant to this thesis and will not be considered further.

### 2.3.1 The Structure of a Tutorial

Tutorials for different types of topics are generally structured in a way the author believes will provide a good introduction for the given topic, starting with the necessary basic information, and then building on this to learn more advanced

[5]The Java Tutorials (link)
[6]Tutorialspoint UML Tutorial (link)
[7]IELanguages: Free Language Tutorials (link)
[8]Photoshop Tutorials (link)
[9]Tuts+ Photography Tutorials (link)
[10]Anthropology Tutorials (link)

concepts. Depending on both the topic and the author, this may result in very different structures.

Looking at some of the examples from Sect. 2.3, we see that while the Java tutorials provide stepwise instructions for reaching a specific goal, the spoken language tutorials serve as more of a lookup reference for the most basic concepts within the language. The spoken language tutorials are actually in some ways similar to the separate Java API documentation.[11]

While there are differences in how tutorials for various topics are made, we can identify some general patterns and elements that are present in different types of tutorials:

- A list of prerequisites, such as knowledge, equipment or both.
- Basic and/or advanced information about the topic, depending on the scope of the tutorial. Often presented in a stepwise manner, starting from the most basic and moving on to the more advanced.
- Examples on how to use the information provided in specific cases or contexts.
- Exercises where the reader must try to use the concepts introduced in a specific context. Often the whole tutorial is designed as an exercise, presented as a series of steps the user must complete.
- Illustrations, figures, or animations, providing the reader with additional examples, information about concepts, or desired results from exercises.
- Some kind of motivation for learning about the topic, often as part the tutorial introduction.

In various combinations, these elements aim to make the introduction to a topic more interesting and intuitive for new users.

It is also worth noting that most topics are rarely introduced by a single tutorial. A tutorial is more often designed to introduce only a specific concept within or a part of a topic, with additional tutorials covering other parts. This allows the user to first become familiar with basic concepts, before moving on to tutorials covering the more advanced parts.

[11]Java Platform, Standard Edition 8 API Specification (link)

### 2.3.2 What Makes a Tutorial Good?

Tutorials are meant to improve the learning process for a given topic, but a bad tutorial is unlikely to offer much improvement over other learning resources. When creating a tutorial for something, it is important to think about how the tutorial can be created and organized in the best way possible. Some good practices for making tutorials will seem obvious to most people, such as starting with the basics. Other practices may be less obvious, and possibly even contested among professionals.

There are numerous informal sources offering guidance on how to make good tutorials.[12] Many cover specific types of tutorials, such as video tutorials or tutorials for games, while others provide more general guidance. Some of the guidelines these informal guides provide are listed below, in no particular order.

- Structure the tutorial in logical, meaningful steps, and give the user an overview of these steps
- Focus on a target audience, and tailor the tutorial to this audience
- Provide clear examples, and possibly even working demonstrations.
- Make sure the basics are covered sufficiently
- Identify the problem the tutorial aims to solve
- Suggest one or more paths the user can take to continue learning
- Identify the role of the tutorial and concepts it teaches in the larger picture
- Keep the tutorial as short as possible, and communicate information effectively
- Provide sensible visualizations where appropriate

These guidelines should not be interpreted as dogmas, but seem fairly sensible, and should likely be taken into consideration when creating a tutorial for something.

In addition to these informal guidelines, Sect. 3.1 covers some more formal research efforts on various practices and ideas for making good tutorials.

[12]See for example *SpyreStudios* (link), *Udemy* (link), [Ada11], *Men with Pens* (link), and *Chris Pirillo* (link)

### 2.3.3 Tutorials in Video Games

Tutorials for video games are generally different from other types of tutorials. These tutorials usually provide an *immersive* learning experience within the game itself, by letting players experiment with concepts and offering direct visualization of the consequences. Because of this property, some aspects and characteristics of video game tutorials require extra attention and consideration.

The game developer website *Gamasutra*, through contributer and game developer Ernest Adams, offers insight into some *bad* practices for video game tutorials, by describing some of the pitfalls tutorial creators may want to avoid [Ada11]. While the article does not offer much concrete advice on how these can be avoided in a good way, it illustrates some of the significant differences between video game tutorials and other tutorials when compared with other guides. For example, Adams argues that players should not be forced to do the tutorial at all, but have the option to skip parts of or the whole tutorial. Additionally, required reading effort should be kept to a minimum, and the player should not be punished for performing the wrong actions because of lack of experience.

Andersen et.al [AOL$^+$12] identify and experiment with four video game tutorial characteristics that may affect player engagement and retention:

**Presence** The first characteristic the researches considered was whether the game offered a tutorial, or not. They discovered that the presence of a tutorial only made a significant positive difference in the most complex games, where players were less able to discover the workings of the game intuitively though exploration.

**Context-sensitivity** The second characteristic explored was the context-sensitivity of the information and guidance provided in the tutorial. Some tutorials provide guidance at the point in the game where it is needed, while others simply offer manuals and documentation outside the context of the game, forcing players to remember information and instructions. The results found were similar to those related to the presence characteristic: providing information and instructions in-context only mattered for complex games, where they positively influenced player engagement.

**Degree of freedom** Third on the list was the degree of freedom offered to players in the tutorial. A low degree of freedom means players are forced to perform the actions the tutorial teaches, guiding their exploration, while a high degree of freedom allows players to explore the mechanics more freely. It has been argued that restricting users' actions improves their performance in the tutorial [KP05], but too much restriction can also have a negative influence [BSG$^+$09]. The results from the research team's experiments however, showed no significant differences in player engagement between the approaches restricting player actions, and allowing complete freedom. It is therefore unclear whether restricting players' actions improve a tutorial or not, and it likely also depends on various other factors.

**Help-on-demand** Finally, the availability of help-on-demand resources was considered. Making additional help resources available to players and allowing them to use these when needed, may decrease player frustration when obstacles are encountered, and improve retention. Like with the degree of freedom, it was difficult to consistently determine whether adding help-on-demand improved player engagement and retention, since results varied. The implications of adding help-on-demand also likely depend on other factors in the tutorial.

While designing tutorials for video games requires extra attention to some particular details, many of the things that apply other types of tutorials are also important for video game tutorials. The purpose is still to try and teach various concepts to a specific audience in the best way possible.

## 2.4 Game-Based Learning

The concept of using games and game-like approaches to teach is becoming more and more widespread. Educational institutions, ranging from elementary schools to universities, as well as independent educational organizations, let their students play games in order to learn everything from mathematics and languages to history and social sciences. This is in part because technological advances have made it possible to explore these approaches to teaching, but also because educators are realizing that in the 21st century, students need to master a different set of skills than before in order to be competitive [Ass14]. These skills include among others, communication, critical thinking and collaboration, skills that are paramount for success in many games.

### 2.4.1 Serious and Epistemic Games

Games that are designed for a purpose that is not pure entertainment are often called *Serious Games*, a term likely popularized by *The Serious Games Initiative*[13] in 2002 [DAJR11]. This genre encompasses many different types of games, but a prime example of a serious game is the *flight simulator*, a realistic type of game that exists in various forms, and is extensively used to teach pilots how to fly aircraft.

The idea behind serious games is to improve student motivation and engagement by providing immersive learning experiences, similar to how many professions are taught. I can read a lot about woodworking, but in order to become a good carpenter, I

[13]The Serious Games Initiative website (link)

must practice. Seeing actual results of my knowledge and skills in woodworking is also likely to be more enjoyable than simply imagining how it might look. Games can provide a similar kind of (simulated) hands-on experience for topics that are difficult or inconvenient to practice immersively, especially more abstract topics like math or computer programming, and this has also proved to be a more efficient learning method in many cases. Children especially learn and are better motivated by the kind of problem-solving we find in games, as opposed to traditional textbook learning [dFL11].

Several organizations working with serious games exist. Some examples are the *Serious Games Institute*,[14] the *Games Learning Society*,[15] the *Learning Games Network*,[16] and *Serious Games Interactive*.[17] These represent both commercial, societal, and academic interest in serious games.

Researchers at the University of Wisconsin coined the term *Epistemic Games* for the subset of serious games that aim to teach specific professions or skills [SG05]. They argue that games can help teach students to apply their knowledge, instead of simply remembering, in addition to facilitate for *innovative* learning. With this as a basis, *The Games and Professional Simulations Research Consortium*[18] has been formed in order to solve educational challenges through games and simulations.

### 2.4.2 Examples of Game-Based Learning

There are numerous examples of games that are designed to teach various concepts. Especially for children there are many sources of different games teaching subjects like math, language, history, or other primary education topics. The target audience for these types of games is however not limited to children, but exist within all levels of education. Some examples of these types of games follow in this section, with some additional examples related to learning programming mentioned in Sect. 3.2.

Many of the educational games for children are found on the web,[19] as the result of volunteer work or the efforts of educational organizations. These are mostly simple games, where children get to practice their skills in the respective subjects in fun ways.

Some educational game efforts are present not only on the web and by initiative of individual teachers, but have been widely adopted by educational institutions. One

[14]Serious Games Institute website (link)
[15]Games Learning Society website (link)
[16]Learning Games Network website (link)
[17]Serious Games Interactive website (link)
[18]The Games and Professional Simulations Research Consortium website (link)
[19]See for example Learning Games for Kids (link), Mr. Nussbaum (link), or Games to Learn English (link)

such game is *Enki*,[20] a relatively new Norwegian web-based game that teaches elementary and middle school students subjects like English and mathematics. Another example is *Scratch*, a game-like environment for learning programming-like skills developed at the Massachusetts Institute of Technology (MIT) [RMMH$^+$09]. *Scratch* is available in more than 40 languages, and used for education in more than 150 countries.

Some educational organizations provide game-like learning in a way that is slightly different from the simple-practice-game approach, and *KhanAcademy* is one such organization. The *KhanAcademy* website is mostly about lectures, but it also provides exercises students can work with to better understand the subjects. These exercises are like regular text book exercises, but award *badges* and *points* for each skill “mastered”, and additionally if answers are provided quickly, or without any errors. This approach is in line with the current trend of *Gamification* that exists in fields like Human-Computer Interaction (HCI)[DSN$^+$11]. *KhanAcademy* has lectures and exercises for a several different subjects, and for concepts in the whole range of educational institutions, from elementary school to university level.

Game-like learning can also be a part of classroom teaching, with the purpose of engaging students more in the subjects being taught. *Kahoot*,[21] a classroom response system recently developed at NTNU, is an example of this. *Kahoot* allows both students and teachers to ask and answer questions and quizzes, providing a more interactive and engaging learning experience.

The examples mentioned in this section are only the tip of the iceberg for educational games and game-like learning, and many more exist.

### 2.4.3 Learning from Non-Educational Games

While there is little doubt about the teaching potential of educational games, there are also lessons to be learned from games that are designed purely for entertainment. James P. Gee, a famous advocate for game-based learning, argues that many entertainment games are designed in ways that force players to learn complex concepts, and through this process gain knowledge and skills that are useful also outside the context of the game [Gee05]. They even make the players *want* to spend time learning these concepts, by providing good learning environments and keeping up motivation and engagement in various ways. His main points are summarized below.

- **Empowered Learners** Players feel like active agents while playing, and not just passive recipients of information. Games are interactive, which leads to

[20] Aftenposten 2013-10-29 (link, Norwegian)
[21] Kahoot website (link)

perceived ownership and engaged participation.

- **Customization** Players are in many cases allowed to make choices about how to play, such as adjusting difficulty or playing style. People are different, and learn in different ways.
- **Identity** Players often take on new identities within the game, in which they become heavily invested. This leads to a level of commitment that facilitates deep learning.
- **Manipulation and Distributed Knowledge** When players are able to control and manipulate a character or an object in the game environment, they feel expanded and empowered. Often, part of the knowledge required for manipulation is stored in the game itself (automated), so that the player can focus on the parts that are important for their task (and "level of abstraction").
- **Well-Ordered Problems** Players are exposed to problems in a well-ordered manner, so that they can form hypotheses that not only work in the moment, but prepare them for more difficult challenges later in the game.
- **Pleasant Frustration** Players are exposed to problems that are neither too easy or too hard, but at the edge of the players' competence, and at their own pace.
- **Cycles of Expertise** Players are allowed to repeat and practice skills until they become nearly automatic. Then, as the game progresses, they might have to adapt their skills to new conditions, and repeat the cycle.
- **Context-sensitivity of Information** Players are often presented with the information they need *when* they need it, instead of having to memorize it in advance.
- **Fish Tanks** In many cases, games serve as simplified versions of real-world systems, and illustrate some important concepts while hiding complexities that might be too difficult to handle for novices. Sometimes, such fish tanks are also created within the game itself, in the form of tutorials. This allows players to exercise their skills without having to worry about *all* the details.
- **Sandboxes** Games also provide a safe environment for exercising skills, where the cost of failure generally is low compared to the real world.
- **Skills as Strategies** Instead of practicing for the sole purpose of becoming good, players see the skills they learn as a strategy towards accomplishing goals within the game. This provides better motivation by allowing "in-context" practice.

- **System Thinking** Many games consist of smaller elements, where players must understand how all the elements interact fit into the overall system of the game.
- **Meaning as Action Image** Instead of just providing definitions and descriptions, games present concepts through visualizations and experiences, which is closer to how people actually think.

These are clearly principles that should be considered also educational games, not only those made for entertainment. As Gee points out, *"When we think of games, we think of fun. When we think of learning, we think of work"* [Gee05]. If done right, it is likely possible to merge the tedious process of learning with the fun of games, providing equal or even better results.

Player engagement for learning in entertainment games can also go beyond what the game itself teaches. One example of this is the concept of *theorycrafting*, which is a term used to describe the process of applying mathematical analysis and simulation in order to optimize playing styles in games like *Starcraft* and *World of Warcraft* [Pau11]. This is not only an example of players bringing complex concepts into a game, but also a desire to learn more about a game than is actually required to play it.

# Chapter 3
# Analysis of Related Work

This chapter outlines previous work by other researchers that is relevant to this thesis, supplementing the background information provided in Ch. 2. A lot of work and research has been and is being done in the field of learning with games and game-like approaches, but this chapter only summarizes what I've found to be most relevant. This primarily includes work on tutorial design and teaching computer-related tasks through games.

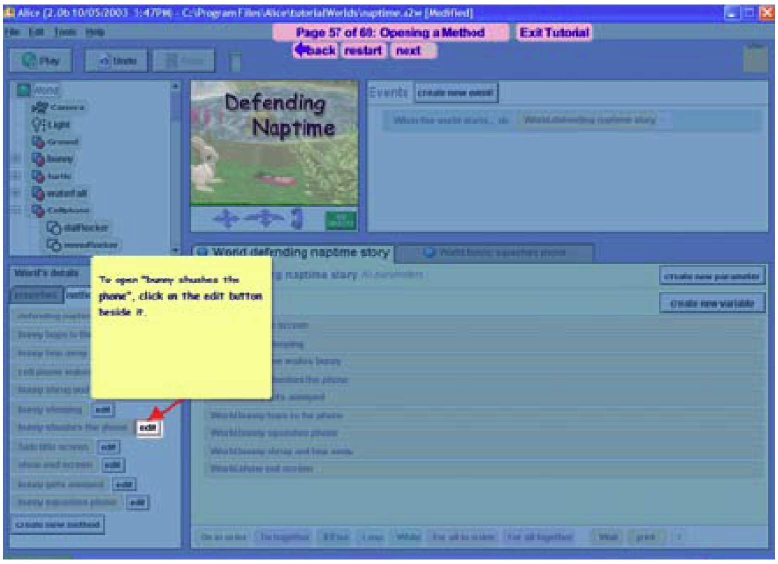
## 3.1 Tutorial Design

Tutorials, as described in Sect. 2.3, are widely used to teach users how to learn various concepts, such as working with software products, among other things. With the tutorial, the creators aim to teach users how to perform various tasks with their tool, preferably in a **quick**, **intuitive**, **memorable**, and **error-free** way. Unfortunately, not all tutorial designs succeed in fulfilling all of these goals, but many efforts have been made to explore different approaches and improve on the standard tutorial design. Some of these are briefly described in the following sections.

### 3.1.1 Stencils-Based Tutorials

The authors of the paper *Stencils-Based Tutorials: Design and Evaluation* [KP05] identify some problems with traditional tutorials: users may miss steps or perform actions that were not intended, and it is often difficult to find the components described in the tutorial.

In their work, the authors introduce *Stencils*, which is an alternative way of presenting a tutorial by adding a translucent colored interface layer on top of the original User Interface (UI), with holes highlighting the relevant elements, as seen in Fig. 3.1. Additionally, tutorial information is displayed on this layer in the form of sticky notes.



To open "bunny slushes the phone", click on the edit button beside it.

**Figure 3.1:** An example of highlighting relevant UI elements with *Stencils* through holes in the colored layer. Tutorial information is added with the yellow sticky note. *Image source: [KP05]*

With the results of a user study, the authors show that with a *Stencils*-based tutorial, users were able to complete tutorials faster, with fewer errors and less human assistance. They also note that their tutorial approach can likely be improved by decreasing the level of assistance depending on the users familiarity with the system. A need for tutorial tasks that are directly relevant to the users, as opposed to "artificial" tutorial exercises, is also mentioned.

### 3.1.2 DocWizards

The authors of the paper *DocWizards: A System for Authoring Follow-me Documentation Wizards* [BCLO05] identify that there are some problems with teaching people to use software (*computer-based procedures*) through documentation alone. Users must find UI elements based on documentation descriptions on their own, understand and handle conditional branches, and at the same time keep track of where they are in the process.

In their work, they propose the use of a tutorial-like documentation process called *follow-me documentation wizards*, an approach that combines the advantages of conventional wizards and documentation. With their approach, processes are automatically captured from demonstrations made by expert users, and made available to new users in the form of highlighting both text from the documentation (see Fig. 3.2) as well as UI elements for each step.

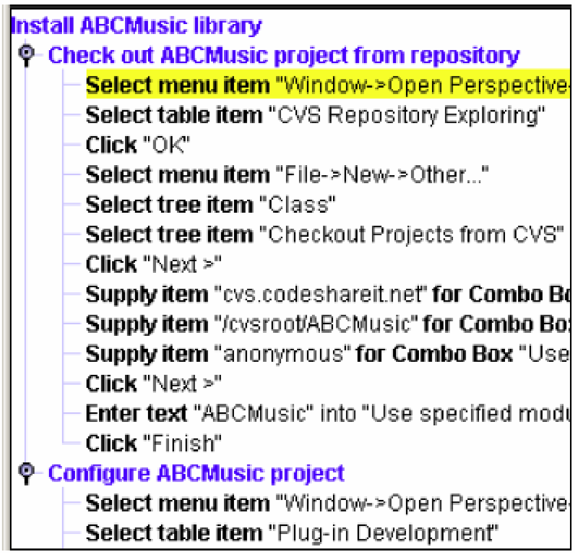

Install ABCMusic library
Check out ABCMusic project from repository
Select menu item "Window->Open Perspective
Select table item "CVS Repository Exploring"
Click "OK"
Select menu item "File->New->Other..."
Select tree item "Class"
Select tree item "Checkout Projects from CVS"
Click "Next >"
Supply item "cvs.codeshareit.net" for Combo B
Supply item "/cvsroot/ABCMusic" for Combo Bo
Supply item "anonymous" for Combo Box "Use
Click "Next >"
Enter text "ABCMusic" into "Use specified mod
Click "Finish"
Configure ABCMusic project
Select menu item "Window->Open Perspective
Select table item "Plug-in Development"

**Figure 3.2:** An example of stepwise instructions in *DocWizards*. The current step is highlighted in yellow. *Image source: [BCLO05]*

The authors also conducted a user study in order to verify the usefulness of their work. Several issues were identified, but overall the study yielded a lot of positive results. The usefulness of the *DocWizards* approach has also been verified in a separate study [GBCB07].

### 3.1.3 Graphstract

The authors of the paper *Graphstract: Minimal Graphical Help for Computers* [HT07] identify problems with the commonly used approach of providing only a textual explanations in software tutorials and help. Users are unlikely to read these explanations carefully enough, if at all. Simply adding screenshots of the UI is not an adequate solution, since these usually add far more information than necessary, and thus increase the perceived size and complexity of the explanation. Problems with animation and video are also identified, such as making it difficult for the user to move at their own pace.

Instead of relying on text-only descriptions or simple screenshots, the authors propose the use of graphical help in the form of partial screenshots, combined to show a complete sequence of actions required to perform a task (see Fig. 3.3). This approach provides graphical help directly mapped to the UI, without adding a lot of extra information. Additionally, the whole sequence is presented in a small space, making it easy to get an overview.

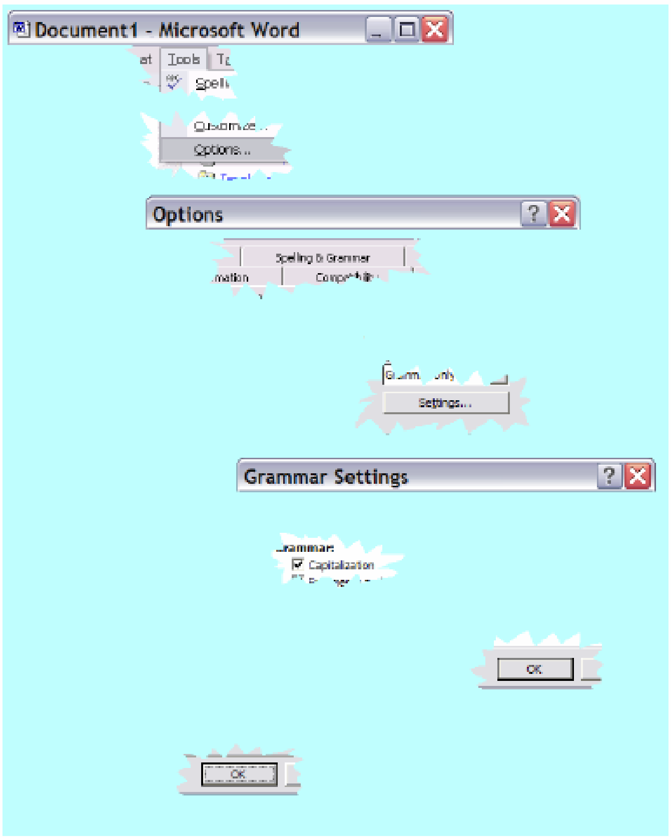


**Figure 3.3:** An example of stepwise instructions in *Graphstract* in the form of screenshot snippets, showing the steps required to toggle auto-capitalization in Microsoft Word. *Image source: [HT07]*

Three iterations of user studies on a prototype is conducted by the authors, showing that *Graphstract* performs better than conventional approaches overall, if not in all cases. They also conclude that adding text to the images is useful in many situations, despite their approach relying on graphical help only.

### 3.1.4 Photo Tutorials

The authors of the paper *Generating Photo Manipulation Tutorials by Demonstration* [GAL$^+$09] argue the use of static visual tutorials (stepwise text-based tutorials accompanied by graphics) over video-tutorials for image processing software.

In their work, the authors design a system for auto-generating static visual tutorials for a specific software product. These tutorials provide stepwise instructions with screenshots for completing a particular task, and additionally highlight the parts of the screenshots that are relevant to this task, as seen in Fig 3.4. This is combined with macros for automatic image labeling, to identify important regions of the particular image the user is working with.

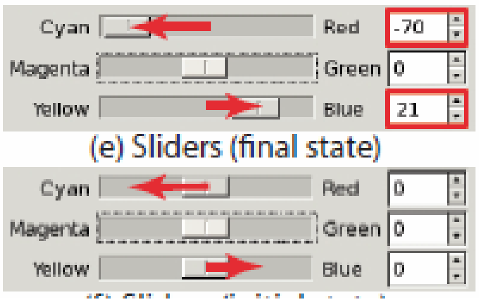


**Figure 3.4:** An example of highlighting relevant UI elements in a screenshot with *Photo Tutorials. Image source: [GAL+09]*

Through a user study, the authors verify the effectiveness of their tutorials by observing that users perform significantly better and make less errors compared to tutorials based on text and video. However, some problem areas are identified: even better tutorials can be created by providing feedback to users as they are performing the steps of the tutorial.

### 3.1.5 Toolclips

The authors of the paper *ToolClips: An Investigation of Contextual Video Assistance for Functionality Understanding* [GF10] explore the *learnability* of software products, more specifically relating to the concept of *understanding* how to properly use functionality (see [GFA09]). They identify problems with existing approaches based on both text and videos, where information is provided outside the context of the UI in question. The authors also assess that regular tooltips, which provide the user with a short in-context description of what a UI element does, do not provide a sufficient level of detail for complex tools.

Attempting to provide the best of several worlds, the authors suggest their *ToolClips* approach, enhancing regular tooltips with additional documentation and video content, in a more context-sensitive manner, as seen in Fig. 3.5.

Through the results of two user studies, the authors show that the *ToolClips* approach significantly improves users' understanding of how to use elements, and additionally

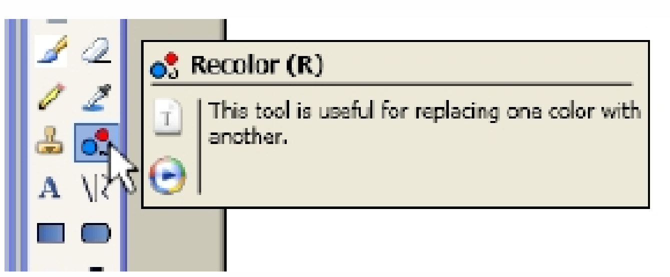


**Figure 3.5:** An example of a *ToolClip*, appearing as a regular tooltip, but with buttons allowing access to additional media. *Image source: [GF10]*

has a positive impact on retention of this understanding, for applications that are *highly graphical.*

### 3.1.6 A Summary of Good Practices for Tutorials

The previous sections in this chapter, as well as Sect. 2.3, describe various research efforts on the creation of good tutorials and introductions for computer-based procedures. Different aspects of the learning process are considered, and it is apparent that there can be many paths to success. In this section, I attempt to summarize the various aspects that should be considered when making a tutorial for a computer-based procedure.

**Interactivity and Active Learning** The most important aspect of a good tutorial is likely that it should be interactive. Active learning is one of Chickering and Gamson's *Seven Principles of Learning* [CG87], in addition to being recommended for tutorial design by various other sources [BEH$^+$99, ABH$^+$00]. A common way of making a tutorial interactive is to provide exercises the user must complete.

**Feedback** If a user is presented with tasks and exercises during the tutorial, it should also be possible to see the results of these exercises, and if applicable, check if the results are correct [CG87]. The feedback should also give the user some indication of the user's performance compared to the learning objectives [BEH$^+$99].

**Motivation** It is important for users to know not only why they should learn the processes described in the tutorial (i.e. how these processes are useful to them), but also why they should complete more advanced tutorials on the subject that may optimize their work [GFA09]. Users should also be informed of the scope of the tutorial, and what they are supposed to learn [BEH$^+$99].

**Reasonable Teaching Order** If a tutorial teaches multiple concepts, a reasonable teaching order should be established, particularly if these concepts build on each other [BEH+99]. When deciding on a starting point, user' background must also be considered.

**Context-Sensitivity of Information** All new information presented in the tutorial should be provided at the point where it is actually needed, as opposed to being presented in an introduction part before the tutorial starts [AOL+12].

**Visual Mapping of UI Elements** If UI elements are referenced in the textual part of a tutorial, it should be easy to map these references to the actual UI. This can be done by adding (partial) screenshots of the UI [HT07], or highlighting the elements that are relevant for the current step [KP05, BCLO05, GAL+09].

**Multimedia Content** A tutorial that consists of text alone is rarely adequate, and can be improved by adding multimedia content such as images, sound, or video [GFA09]. This kind of multimedia information content can also be present in the application itself, providing the user with more resources when help is needed [GF10]. Animations can also be advantageous, but should be used only where natural, for example to describe events in time [MTB00].

**Help-on-Demand** While the user generally shouldn't have to read through too much text before starting the interactive part of the tutorial, additional information resources should be available for when the user gets stuck, or for other reasons needs to know more about a specific topic [AOL+12, GF10].

**Relevant Tasks** When a tutorial contains tasks and exercises, these should be as relevant as possible to the user, as opposed to fictional tasks created only for learning purposes [KP05].

**High Expectations** Users should be presented with high expectations from the beginning, so that they are prepared to make an effort in understanding the concepts taught by the tutorial [CG87].

**Address Misconceptions** If common misconceptions exist within the topic that is being taught, the tutorial should do its best to address these misconceptions, preferably before they even have a chance to form [ABH+00].

**Multiple Perspectives** Generally, we want the user to actually think reflectively about the topic being taught, and not just passively absorb the facts. One way of doing this is by offering multiple perspectives of various concepts [ABH+00].

**Freedom** When introducing something like a game or a software product through a tutorial, we have to consider the amount of freedom users are given in the tutorial. There are issues in both allowing users complete freedom to explore, and restricting them to a single path [AOL+12, BSG+09]. The optimal solution likely lies somewhere in between, by for example offering *directed exploration* [EFG13].

## 3.2 Learning Programming through Games

Section 2.4 covered some of the reasons why educational games are becoming popular, and offered a few examples. In this section, a few more examples are explained in a little more detail. Rather than games for general school subjects, these games are designed to teach students about programming and thinking in the terms of software development.

### 3.2.1 Karel the Robot

Karel the Robot [Pat81] is a game-like programming language and environment designed to teach basic programming concepts to beginners. It was developed by Richard E. Pattis in 1981, who used Karel to teach programming courses at Stanford University.

The motivation behind Karel was to be able to teach students the basics programming without having to worry about the more complex and less important (to beginners) details. In Pattis' own words: *"The careful omission of variables and data structures from Karel's language ... allows the immediate exploration of the rich domain of abstraction and control structures."* [Pat81]. This allows students to focus on learning how to solve problems through programming concepts.

Karel was originally designed as a Pascal-like procedural programming language, but the concept gained wide popularity, and has been extended to Java [Rob05], Python,[1,2] Karel++ (object-orientation) [BSRP96], REALbasic,[3] and Scratch.[4] Karel was inspired by the LOGO project,[5] and has in turn inspired games like RoboMind[6] and C-Sheep [AM06].

The purpose of the game is to control a robot (Karel) by giving it a set of commands, and perform tasks. Initially, only a small set of commands are available, but as part of the learning process, users must learn how to extend these commands. The

[1]The Guido van Robot Programming Language (link)

[2]rur-ple: an environment designed to help you learn computer programming using the language Python (link)

[3]rbKarel: REALbasic adaption of Karel the Robot (link)

[4]Karel the Robot in Scratch (link)

[5]LOGO Foundation website (link)

[6]RoboMind website (link)

simplest task Karel is asked to perform is to pick up a beeper, seen as the diamond shape in Fig. 3.6.

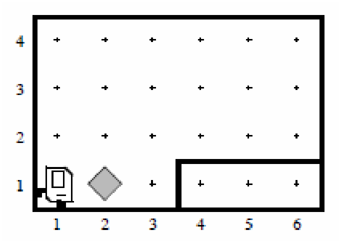


**Figure 3.6:** A simple *Karel the Robot* level, where Karel (bottom left) must move to and pick up a single beeper (diamond shape). *Image source: [Rob05]*

Karel the Robot has now been around for more than 30 years, and the teaching paradigm described has proved successful in introducing students to the art of programming.[7] Students are allowed to explore advanced concepts in a safe environment with less relevant complexities removed, and in a way that makes them *want* to learn. The concept has been adapted and refined in many ways over the years, but the core paradigm remains.

### 3.2.2 Josef the Robot

Josef the Robot [Tom82] is a game-like programming environment similar to Karel (Sect. 3.2.1). Unlike Karel, Josef more closely resembles "real" programming languages by being rich in structures and operations. Like Karel, Josef is also inspired by the LOGO project.

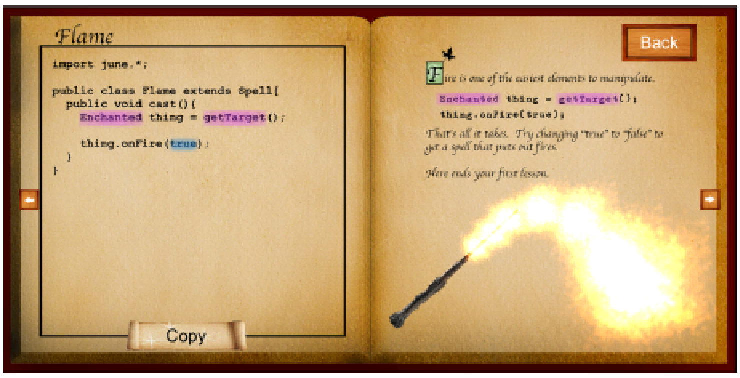
The author of Josef, Ivan Tomek, provides ample motivation for creating a more novice-friendly environment for learning how to program. Firstly, potential learners should find the problems they can solve with programming to be interesting, which is likely not the case (for the average person) with problems like sorting a sequence of numbers. Furthermore, novice programmers should not have to worry about the more complex rules that are not directly related to problem-solving, such as syntax and data handling.

### 3.2.3 CodeSpells

*CodeSpells* is a project from the University of California, San Diego, that aims to teach Java programming through a wizardry game [EFG13]. The CodeSpells team draws inspiration from the *epistemic games* concept, and their games immerses

[7]See for example [KSG82], [Unt90], [Bec01]

novice programmers in a world that connects abstract code (Fig. 3.7) with visual and “physical” effects in the environment (Fig. 3.8).


Flame

```
import june.*;

public class Flame extends Spell{
  public void cast(){
    Enchanted thing = getTarget();

    thing.onFire(true);
  }
}
```

Fire is one of the easiest elements to manipulate.

```
Enchanted thing = getTarget();
thing.onFire(true);
```

That's all it takes. Try changing "true" to "false" to get a spell that puts out fires.

Here ends your first lesson.

**Figure 3.7:** An example of the CodeSpells spellbook, with Java code for the *Flame* spell. The second page adds a short description, and a tip on how the spell can be modified. *Image source: [EFG13]*

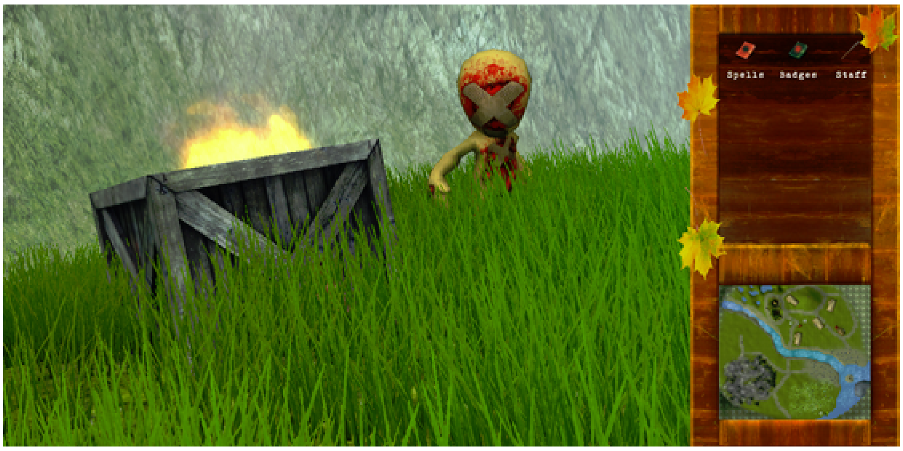


**Figure 3.8:** The game environment presented to the user in *CodeSpells*. In this particular quest, the player must modify the *Flame* spell (Fig. 3.7) in order to extinguish the fire by setting *thing.onFire(false)*. Image source: *https://sites.google.com/a/eng.ucsd.edu/codespells/home/level-1-quests*

An important goal for the CodeSpells project is for students to gain a deep understanding of the programming language they use and problems they solve, and retain this understanding after completing the game. They would also like the students to be able to play the game without instructor assistance, and have focused extensively on designing quests that provide the appropriate scaffolding, as well as encourage exploration beyond the minimum required to move on [EWF+14].

In order to verify and improve the usefulness of their game for teaching programming, the CodeSpells team have conducted some user studies. Through these studies, they discovered that by immersing users in the game, the users developed a determination and a positive outlook on solving programming challenges [EFG13]. The team also discovered some principles that can be used to improve quest design [EWF+14]:

- Provide examples that can be tested and used as a starting point, allowing users to see their effects.
- While the introduction should make it possible to perform complex actions with little effort, expectations for more effort should be provided from the beginning, letting users know that they have to build on and modify the examples.
- Provide directed exploration through quests, so the user can get a sense of complexity ordering and a choice of which challenge to overcome.

These principles can likely also be used to improve design of other serious games.

### 3.2.4 A Summary of Good Practices for Educational Games

While educational games by default generally incorporate some good learning practices (interactivity and feedback), some ways of designing and implementing such a game are likely better than other. Based on the work described in the previous sections as well as Sect. 2.4, this section summarizes some additional aspects that should be considered when designing an educational game.

**Tutorials** Many games start by guiding the player through a tutorial in order to learn the basics of the game, which is similar to completing a tutorial in order to learn about a topic like programming language. Games, particularly complex ones, may benefit from having a tutorial introduction, in the form of increased player engagement [AOL+12]. Since educational games are often used to teach complex concepts, creating a tutorial is generally justified. The principles listed in Sect. 3.1.6 also apply to tutorials for educational games, since these should be at least as good as tutorials for teaching a specific topic. Because of their immersive nature, video games likely makes a some of these principles easier to follow, such as providing good visual mapping of UI elements and suitable multimedia content.

**Immersion** By providing an *immersive* experience, educational games can provide better motivation and encouragement for players to interact with and explore the concepts taught [EFG13]. Providing an immersive experience may include giving the player a sense of *identity* in the game, enhancing commitment, or provide *fish tanks* or *sandboxes* where players can discover, explore, and experiment with concepts in safe and possibly simplified environments [Gee05]. Immersive game experiences also provide players with meaning for verbal concepts, through visualizations and experiences [Gee05].

**Exploration and Guidance** The discussion of exploration versus guidance becomes even more important for educational games than for tutorials. Good educational games often provide a safe environment where concepts can be explored and experimented with, even without any guidance. In many cases, too much guidance may even prevent exploration and discovery to some degree [BSG+09]. Depending on the complexity of the concepts taught, having no guidance at all is also not a good solution. Some players will have more trouble than others understanding certain concepts, and subtleties are not always intuitively understood. The optimal solution is to evaluate the game's complexity, and provide some kind of middle ground (directed exploration, [EWF+14]) tailored not only to how difficult the concepts are, but the player's previous knowledge and ability to understand them. Users may even be allowed to choose their own difficulty level based on their own perceived experience level and skill.

**Simplicity and Challenge** Irrespective of their background, any users within the target audience should be able to get started and see results early in the introduction [EWF+14]. However, the problems presented must eventually become difficult enough to allow players to gain deep understanding of the concepts taught, and expectations of this should be present from the beginning. Additionally, players can progressively be presented with an environment that is more and more like a real environment. This can help make the transition from practicing skills in a video game, to using these skills in real situations smoother [Unt90].

Chapter 4

# A Tutorial for UML Activities in Reactive Blocks

As the first part of my own work, I set out to make an improved tutorial for learning to create UML activity diagrams in Reactive Blocks. This chapter begins by describing the motivation behind and goals for this set of exercises. A tutorial design is then proposed, followed by a description of and results from a user study, and a brief evaluation of the tutorial.

## 4.1 Motivation

Before setting out to create an improved tutorial for UML activities in Reactive Blocks, it is important to determine whether there is real motivation and need for this. If good tutorials already exist, another will likely be redundant. While it is certain that *some* tutorials exist, we also need to check if these adhere to the principles of good tutorials discussed in Sect. 3.1.6. If there is motivation for creating a new and better tutorial, we should establish some goals and guidelines in advance.

### 4.1.1 Existing Tutorials

The major part of the motivation for creating a set of tutorial exercises comes from looking at the tutorials that already exist for UML activities and Reactive Blocks. While these tutorials generally present the information required to get started, it is not necessarily presented in an optimal way to newcomers.

#### Tutorials for UML Activities

The official UML website[1] links to four sources of tutorials for learning about UML:

– **No Magic *MagicDraw*:** Not really a tutorial, but a commercial software product for software modeling. A quick inspection of the trial version does not reveal any tutorial functionality other than some tips for using the software.

[1]Unified Modeling Language Resource Page (link)

– **OMG's List of Training:** Also not a tutorial, but a list of companies that offer UML training. Mostly as n-day sessions on-site.

– **Mario Jeckle - UML Tutorials:** A German website with a lot of links to other sources of information, many of which do not even work.

– **Sparx Systems:** The only link that actually leads to something resembling a tutorial. However, information is presented in a more documentation-like way, ignoring most practices for good tutorials.

A quick Google-search lists some additional tutorial sources, but most follow a documentation-like approach similar to the one from Sparx Systems listed above. In short, there is a real lack of interactive UML tutorials, where the user gets to learn a few concepts at a time, and to use and understand these concepts in the context of examples and exercises.

**The Reactive Blocks Tutorials**

Reactive Blocks has a set of tutorial exercises available to new users of the software.[2] These tutorials are created in an exercise-like manner, where the user is presented with some new information, and must use this in examples. However, the tutorials are mostly focused around the capabilities of the software and how to use it, rather than teaching good modeling. Previous familiarity with UML activities seems to be assumed.

If one is to learn about UML activities through Reactive Blocks, an additional set of tutorials must likely be made, with primary focus on the modeling aspect. In addition, like with the existing tutorials, it is likely that some information about using the software must be included.

### 4.1.2 The Target Audience

Making a tutorial that works well for *everyone* is difficult, maybe even impossible. The interest for a tutorial like this is also likely to be limited to a relatively small number of people, more specifically those who want to learn how to use UML activities, or even Reactive Blocks, to improve or simplify their software development process. This provides a rough frame for our target audience.

It is important to note that while people in the target audience likely have similar goals, their background and previous knowledge may vary. Users may be complete beginners in the field of programming and software development, experts simply seeking to add another tool to their process, or anything in between. The whole

[2]Reactive Blocks Tutorials (link, requires registration)

range of backgrounds and previous knowledge should be taken into consideration when designing and implementing the tutorial.

### 4.1.3 Goals

This section outlines the goals I set before creating the tutorial.

**Focus on UML Activities** If the user wants to learn how to work with Reactive Blocks, learning about UML activities is a start, but far from sufficient. The scope of this project is however learning modeling languages, so the tutorial should focus on the modeling aspect, and deal with topics specific to Reactive Blocks only where absolutely needed. This includes dealing with Java code; all operations required in a model should be predefined.

**Avoid Irrelevant Complexities** Reactive Blocks is a full-fledged modeling and code generation tool, with capabilities that go way beyond simply modeling UML activities. This gives the tutorial some advantages, such as actually creating runnable code from the models created, which provides the user with relevant feedback. However, it also introduces complexities that are not as relevant when learning about UML activities. Ideally, Reactive Blocks should have a separate tutorial mode that handles these additional complexities, and lets the user focus on the modeling.

**Difficulty and Challenges** The exercises presented to the user should be easy enough to allow most users to complete them without much difficulty and frustration, while still communicating the lesson properly. In addition, there should be challenges, perhaps as supplementary exercises, that force the users to really think about how an element can be used, and lets them take on a different perspective for solving the problem.

**Everything in One Place** With online tutorials, users often have to change between the application window, the tutorial (often in a web browser), and any other necessary resources. This increases the short-term memory workload, since users have to remember a lot of information between the windows. One of the golden rules of UI design is to reduce the short-term memory load [SP10], so for the tutorial, I would like to provide all the information and parts of the tutorial within Reactive Blocks, letting users find the resources they need while still being able to see the current problem. This should also help create some sense of immersion, though a standard tutorial will most likely not be nearly as immersive as a video game in any case.

**Follow the Principles of Good Tutorials** Section 3.1.6 summarizes various practices for making good tutorials. I would like to follow these as much as possible, while at the same time acknowledging that I likely will not be able to cover all. The ones I consider most relevant for this tutorial are:

- **Interactivity and Active Learning** The tutorial should first and foremost be interactive. For each new piece of information introduced, users should be presented with exercises they have to solve.
- **Feedback** Users should be given feedback on their exercises. Fortunately, using Reactive Blocks is a big help here, since the tool makes it possible to actually run the models created. In this way, users can see if their programs behave as expected.
- **Reasonable Teaching Order** Instead of learning everything about UML activities in one step, I would like to introduce one new element at a time, starting with the most basic elements like edges and operations. Then I will add a few more elements at a time, while allowing the users to experiment a little with the new elements for each step.
- **Context-Sensitivity of Information** Instead of providing a complete documentation for UML activities at the beginning and then start off with exercises, I would like to document each new element in the exercise where they are introduced, i.e. in-context.
- **Help-on-Demand** It is not a good idea to force readers to read through every detail about something before they start getting familiar with it, but I would like information about every detail to be available on-demand should the user need it. This applies to UML concepts learned, as well as information about UI elements. *Toolclips* (Sect. 3.1.5) offers some inspiration for the latter.
- **Visual Mapping of UI Elements** Eclipse is a tool used in many aspects of software development, and most parts of it will not be relevant to this tutorial. Whenever parts of the Eclipse and Reactive Blocks interface are referenced, the tutorial should make them easy to find. The examples listed in Sect. 3.1, particularly *Stencils* and the *Photo Tutorials*, offer some inspiration towards this goal.
- **Multiple Perspectives** UML activities and Reactive Blocks can be used to model many different kinds of systems, and it is important to understand their capabilities. The tutorial should provide exercises that use the various elements in different contexts and with different purposes, when applicable.
- **Freedom** Because the tutorial is made with Reactive Blocks, it is likely a good idea to limit the users' freedom when working with the tutorial. It is easy to get confused by the many capabilities of Eclipse and Reactive Blocks,

and clicking the wrong thing can easily lead to unexpected errors. Having a separate tutorial mode is one possible way of doing this. At the same time, we do not want to restrict the users to one specific way of thinking when solving the tutorial exercises. Ideally, it should be possible to solve these exercises in more than one way, and the users should be made aware of this.

These goals provide a basis for the design of the tutorial, and will hopefully lead to a tutorial that introduces UML activities to novices in a better way than the existing tutorials described in Sect. 4.1.1.

## 4.2 Tutorial Design and Implementation

The tutorial design consists of three parts:

- An enhanced Reactive Blocks user interface, i.e. a *tutorial mode.*
- A teaching order for the various elements and concepts, with an introduction for each new element and concept.
- An exercise for each step in the tutorial, implemented in Reactive Blocks. Some steps additionally have a challenge exercise.

The following sections describe these parts in more detail.

### 4.2.1 Proposal for a Reactive Blocks Tutorial Mode

The first part of the tutorial design is a proposal for a *tutorial mode* in Reactive Blocks. Figure 4.1 displays a visual mock-up of the proposed UI design for the tutorial mode. In the middle, we see the familiar Reactive Blocks modeling canvas (10), with some elements in place for demonstrative purposes. Surrounding the modeling canvas, we see various new elements that will be explained below.

**Element Buttons (1):** Instead of the current right-click menu selection scheme for adding elements to the canvas, buttons for each type of element is placed on the side, allowing more *direct manipulation.* The modeling canvas allows direct manipulation of elements already present, but allowing user to add new elements in the same way is likely to make an introduction to the software smoother and quicker for novices [SP10].

We should also note that only a few of the possible modeling elements are present. This is meant as a way of hiding extra complexity, by only displaying the elements

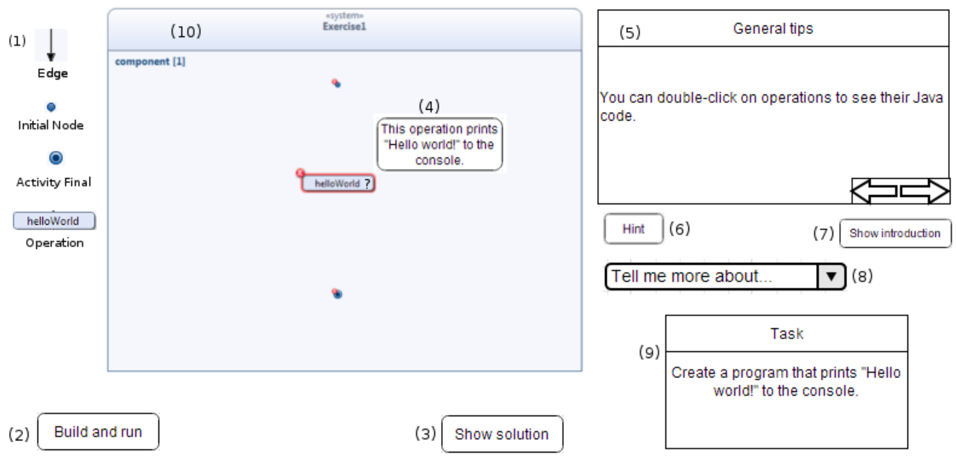


**Figure 4.1:** A visual mock-up of the UI presented to users when running Reactive Blocks in the suggested tutorial mode. The regular modeling canvas is shown in the middle (10), surrounded by utility buttons and three information windows.

that have been introduced so far in the tutorial. Other elements are still available via the right-click menu.

***Build and Run* Button (2):** In order to see the results of running a program, users have to first *build* their model (generate executable code) and then run it. Learning how to do this involves several steps, such as navigating drop-down menus and choosing Java platform.[3] While highly relevant when learning to work with Reactive Blocks, this process adds complexity that is less relevant for learning to work with UML activities. The purpose of the button is to remove the complexities by automatically performing the needed operations to run and build the model, with some default options that work with the tutorial. Like the element buttons, this button also allows more *direct manipulation*.

***Show Solution* Button (3):** In some cases, users may be unable to progress in a tutorial because they get stuck at certain steps. In this case, it is natural to ask for help from an instructor, but we would like users to be able to complete the tutorial without outside help. One way of doing this is to allow the users to see the solution to an exercise (in a pop-up window) when they are unable to complete it. Ideally, this button should only become visible after a certain time has passed, so as not to

[3]See for example the *Building and running the exercises* part of Appx. A

tempt the user into checking the solution before a serious attempt at finding it has been made.

An additional advantage is that when multiple solutions are possible, the user may discover a different approach than the one proposed, offering multiple perspectives to the same concept. To this end, a solution can be provided to the user even in cases of success, allowing comparison of solutions and additional reflection.

**Information about operations (4):** One of the goals for the tutorial was to avoid users having to deal with Java code, as this adds complexity that is less relevant to learning about UML activities. The users still need to know what the predefined operations they are using in their models actually do, and the name is not always enough of a description. The tutorial mode adds a clickable "?" on the operation icon, which opens a sticky note providing a short description of what the operation does. The sticky note is then removed when the user clicks somewhere else.

***General Tips* Window (5):** This window provides the users with some general tips on how to work with Reactive Blocks and UML activities. These should not be necessary to complete the tutorial, but may provide more interested or advanced users with some additional information about what goes on behind the curtains. Their purpose is also to ease the transition towards working with Reactive Blocks without the training wheels provided by the tutorial, for users to whom this is relevant.

***Hint* Button (6):** Like the *Show Solution* button, the purpose of the *Hint* button is to provide users that are unable to complete the tutorial on their own with some help. Unlike the solution however, this button is always visible. The hints provided should not give away the solution, but rather offer some insight about the task, concepts, or elements presented, that may not be obvious to all users.

***Show Introduction* Button (7):** An important part of the tutorial is the introduction, which is provided at each step. The introduction gives the user some basic information about the new concepts and elements introduced at the current step, allowing the user to understand and use the elements and concepts to complete the task. Since activity steps in Reactive Blocks describe events in time, it is natural to use animation to communicate this information [MTB00], perhaps supplemented by audio or textual descriptions. This animation is displayed when the user first enters the step, and the button allows the user to review the animation when this is needed.

***Tell me more about...* drop-down menu (8):** This menu serves as the *help-on-demand* part of the tutorial. In the menu, the user can select various concepts and elements to see more extensive and detailed information about them, either in an embedded pop-up window or through external resources, such as online documentation.

***Task* Window (9):** This window simply shows the task the user needs to complete for the current step.

Unfortunately, Reactive Blocks is a closed source product, meaning I was not able to implement and test this tutorial mode design. It is however still highly relevant to the thesis, as it was supposed to cover several of the good practices for tutorials, and serves as a basis for the UI presented in the game in Ch. 5.

Since I was not able to implement this design in Reactive Blocks, some of the goals that were set for the tutorial were not met. I was, for example, not able to provide *everything in one place.* The tasks and exercises were still implemented in Reactive Blocks, but the introductions and information about concepts and elements had to be provided in a separate document (see Appx. A) and as text and images only, not animations. In addition, it was difficult to provide *solutions* to the exercises, and instead of abstracting away the process of *building and running*, a description of how to do it manually was added. In the end, there was more focus on learning processes specific to Reactive Blocks than originally intended. Help-on-demand was only offered as a reference to the official online documentation.

### 4.2.2 Teaching Order

Teaching users to model systems with UML activities and Reactive Blocks includes teaching how to use the various modeling elements as well as introducing more abstract concepts, like activity steps, looping and modularity.

Finding a good teaching order was challenging, but after a lot of consideration and testing of exercise ideas, I settled on the order displayed in Tab. 4.1. Each step teaches a new concept that is central to UML activity Modeling, and introduces some elements that can be used to implement this concept.

The reasons for the choice of ordering are explained below:

- **Step 1:** This step introduces the core concepts and elements that are required for an application that does something the user can see the result of.
- **Step 2:** Timers were introduced in the second step, because they provide a simple and intuitive way of dividing a program into more than one activity step. They also need to be introduced before flow breaks.
- **Step 3:** This step introduces alternate branches and decisions. Branches must be introduced before loops, to provide a way of breaking the loop. Decisions

| Step number | Concepts | Elements |
|---|---|---|
| 1 | Control Tokens, Activity Steps | Initial Node, Operation, Activity Final, Edge |
| 2 | Stable Position | Timers |
| 3 | Alternate Branches | Decision, Object Flow |
| 4 | Looping | Merge, Flow Break |
| 5 | Parallelism and Concurrency | Fork, Event Reception |
| 6 | Synchronization | Join |
| 7 | Modularization | Local Block |

**Table 4.1:** The teaching order for the UML Activities tutorial. Each step teaches a concept, and introduces one or more elements that can be used to implement the concept.

also require the users to know about object flow, but this is also the first instance where this type of flow is needed.

- **Step 4:** This step introduces loops with merge nodes. Merge nodes are also very useful when working with concurrency. Loops must also be split into more than one step, which can be done elegantly and without adding delay by using the flow break.

- **Step 5:** Users should now be comfortable enough with basic concepts to start working with concurrency. Concurrent branches are also required before join nodes become useful.

- **Step 6:** The last core element to be introduced is the join node.

- **Step 7:** Modularization and local blocks are without doubt the most advanced concepts taught by this tutorial, because of how they are implemented in Reactive Blocks. Using a local block requires the user to learn an additional concept, ESMs, and if the user wants to make a new local block, input and output pins must be considered.

For each new concept and element, a short textual introduction is provided. I attempted to make these descriptions as short as possible, while making sure they still covered the concepts adequately. “Adequately” was entirely based on my own judgment, but the user test conducted in Sect. 4.3 offers some insight into whether this was true.

The complete tutorial document, with introduction text for each concept and element, is included in Appx. A.

### 4.2.3 Tutorial Exercises

For each of the seven steps described in Sect. 4.2.2, I designed and implemented an exercise in Reactive Blocks. The exercises were designed as unfinished blocks with predefined operations, which the user had to complete and then run to see if the result was correct. Some steps additionally had extra challenge exercises, which were optional but encouraged. This section describes each exercise and its purpose. The exercise project is also available online for download.[4]

#### Exercise 1 - Hello World!

Everyone who has learned programming in any way has likely created a "Hello World!" program at some point. It is one of the simplest programs one can make, also in Reactive Blocks, and used as the first step of programming introductions almost universally. To students, this is likely to be a familiar example.

**Task:** Create a "Hello World!" program.

**Purpose:** Teach users about *activity steps*, *tokens*, *edges*, and *operations* with a very simple example. Users also have to use an *Initial Node* to start the application. Use of the *Activity Final* is optional, but users will hopefully discover that they have to terminate the program manually if this element is not included. Note that this exercise only demonstrates a single activity step, and likely does not offer sufficient understanding of this concept for later use.

**Challenge exercise:** The challenge exercise asks users to create a program that prints one message, then a second message, then the first message again. The purpose is for users to (hopefully) discover that operations can be used more than once in the same UML activity diagram.

#### Exercise 2 - Delays

A very important aspect of UML activities is that activity steps are essentially atomic, meaning that logically there is zero delay between operations in the same step. Operations are also not allowed to wait or block, and while in reality each activity step takes some time to complete, they logically happen instantly. Delays are then used to provide waiting capabilities without blocking the application, allowing other steps to run in the meantime.

[4]Tutorial project on GitHub (link)

**Task:** Create a program that displays a light, sets its color to red, then changes it from red to green after 5 seconds. Make sure you have time to see that the light has changed before the program terminates.

**Purpose:** Teach users about *timers*, *stable positions* and multiple activity steps. This exercise only illustrates how timers can be used to implement delays, and not how they allow other parts of the program to run in the meantime.

**Challenge exercise:** The challenge exercise asks users to create a more advanced traffic light, with several more steps and transitions. While strictly speaking equally complex and with no new perspective added, it gives the users more practice in working with timers. Users may also notice that the larger model requires a little more mental effort to set up and possibly debug.

#### Exercise 3 - Choices

Branching is an important concept in computer programming. Depending on specific data or events, one may want the program to behave differently. Reactive Blocks and UML activities also implements this concept, by allowing an activity step to take different paths, depending on some data that is carried by the active token.

**Task:** Create a program that generates a random number between 0 and 200, and prints “Small!” if the number is smaller than 100, and “Big!” if the number is greater than or equal to 100.

**Purpose:** Teach users about *alternate branches*, *decisions*, and *object flow.* Users must learn how to pass data between elements in the diagram, and to set guards on the edges from a decision to decide which path the program should take.

**Challenge exercise:** The challenge exercise asks users to find a given number by implementing a binary search tree. It requires some additional information about how to work with decisions, encouraging users to become familiar with their documentation.

#### Exercise 4 - Looping

Another important concept in computer programming is loops; repeating the same procedure many times. This concept can also be implemented in UML activities and Reactive Blocks, with the help of merge nodes.

**Task:** Create a program that generates a random number between 0 and 200 until that number is greater than 100 (big).

**Purpose:** Teach users about *loops*, *merge nodes*, and *flow breaks*, which are timers with zero delay. An important point in this exercise is to learn that a token may only pass through an element **once** in each activity step, meaning loops must be split into separate steps for each iteration, for example by using a flow break.

**Challenge exercise:** This step has no challenge exercise.

**Exercise 5 - User Input and Parallelism**

One of the big advantages of creating software with Reactive Blocks is that it is fairly easy to implement parallelism and concurrency. This is a central concept in UML activities, and visualized by forking a flow. Concurrency is particularly useful for receiving user input, where one part of the application may wait for input, while another part performs a different task.

**Task:** Create a program that changes the color of a light every time you press one button, and exits when you press another. It should be possible to change the light an arbitrary number of times before exiting (even zero).

**Purpose:** Teach users about *concurrent branches*, *forks*, and *events*. This exercise should hopefully improve users' understanding of activity steps and waiting, as introduced with timers in exercise 2, and additionally adds indefinite waiting with events.

**Challenge exercise:** The challenge exercise for this step is to implement a simple game, where the user must navigate through a set of doors (full description available in Appx. A). The purpose of this challenge is to teach users to model "larger" systems, and combine most of the concepts learned so far. It is also meant to show users the power of being able to receive user input, and that it is possible to create programs that do not seem completely artificial, but actually do something *useful*.

**Exercise 6 - Synchronization**

In many cases, we want to perform a task only when several concurrent tasks have been completed. In order to do this, we need some functionality that waits for all this tasks to complete before continuing. In Reactive Blocks, this is implemented with a *join* node.

**Task:** Create a program that prints a message and then terminates when three buttons have been clicked (in any order).

**Purpose:** Teach users about *synchronization of flows* and the *join* node.

**Challenge exercise:** This step has no challenge exercise.

**Exercise 7 - Modularization**

One of the most important concepts in UML activity modeling is modularization. When dealing with large and complex programs, we generally want to separate parts of the functionality into smaller modules, making the application as a whole more orderly, and allowing reuse. However, in the context of Reactive Blocks this introduces some additional complexities like ESMs, which may arguably be the most difficult concept to grasp for Reactive Blocks novices.

**Task:** Use the *Counter* block to make the *CoinFlipper* block "flip a coin" 10 times, and then return the result.

**Purpose:** Teach users about *modularization* and *local blocks*. This exercise is a huge step up from the previous exercises, since it introduces users to many subtleties in Reactive Blocks related to concepts like activity steps and ESMs.

**Challenge exercise:** Since this tutorial was tested in the context of the TTM4115 course at NTNU, students were directed to the first lab exercise of this course as the challenging part of this step. In this lab exercise, students must create their own local blocks with ESMs, and connect them to form a complete application.

## 4.3 User Testing and Feedback

In order to verify the usefulness of this tutorial, I attempted to conduct a user test. It is not easy to find a sufficient number willing test subjects for a test like this, but being a student assistant in the TTM4115 course at NTNU, where the students learn about UML modeling and additionally need to use Reactive Blocks for their semester assignment, I had a unique chance to expose my tutorial to a small group of users that actually needed it.

### 4.3.1 Testing Method

The testing method used was rather straightforward. I arranged a two hour session where the subjects could work the tutorial on their own, but I would be available for help if needed. Each test subject also received a feedback form they would anonymously fill out in the end, and hand back to me.

I convinced the TTM4115 lecturers to let me use one of the weekly exercise sessions for the testing session, since the students needed to learn about Reactive Blocks for their assignments anyway. The students knew in advance that the particular session would be used for the tutorial, which actually led to more students showing up than usual. Initially I took this as a good sign, but it turned out many of them did not even know about the weekly exercise sessions before I showed up during a lecture and asked them to participate in my test. In addition to the testing session, the tutorial was made available online, for those who wanted to do it, but were unable to attend the session.

Ideally, I should have conducted a standard A/B split test, where some users would complete my tutorial while others did the standard tutorials. This did not happen for several reasons, the primary reason being an overall low number of test subjects. Additionally, it would have been difficult for me to measure and compare results, because most of the test subjects were there in their own interest, and did not want to spend a lot of time on tests and surveys after having completed the tutorials. Students did have a choice in whether they wanted to do my tutorial or the standard tutorials, but because of endorsement from the course lecturers (mine ended up being the one featured on the course website), they all chose to do mine.

During and after the testing session, I was able to gather data in three different ways:

**Feedback forms** All the students who attended the testing session were handed a feedback form. Around 20 students attended the session, but only 11 handed in the form at the end. In addition, 2 students answered the online form. The complete feedback form is available in Appx. B.

The feedback form consists of 7 questions. The expected number of participants for the testing session was low, so the questions were not intended to provide data for quantitative analysis, but rather attempt to identify problematic areas in the tutorial, and if anything could be improved. Since the test subjects were present in their own interest, the form was kept brief in order to not scare them away from filling it in (a too long questionnaire may require more extra effort than the participants are willing to give).

The first question asked whether the user found the introductions useful in solving the exercises. Positive answers were expected, but only negative answers would really have been interesting, since they could indicate that the introductions are in some way flawed, such as being redundant or lacking in information.

The second question was intended to give some indication on how the subjects thought about learning concepts like these from a game, for future reference. If people think they will enjoy learning these concepts from a game, there is a stronger incentive to explore this way of learning.

The third question asked whether the subjects by their own judgment felt that they understood the concepts presented in each exercise, rated by a degree of understanding for each exercise. Like with the first question, the interesting results would have been negative ones.

The fourth question asked whether the subject completed the challenge exercises, which were optional. This was meant to measure the subjects' motivation and willingness for gaining a deeper understanding of the concepts presented.

The fifth question asked the subjects to give an indication of the challenge exercises' difficulty. This question was slightly flawed, as it should have asked for separate feedback for each exercise. However, if most of the test subjects were to answer that the challenge exercises were, for example, too easy, this would give me something to consider for improvement.

The sixth question asked why, if not, the subject did not complete any of the challenge exercises. If motivation and willingness to gain a deeper understanding was lacking, it would be useful to know why, in order to attempt at providing better and clearer incentive.

The seventh and final questions simply asked for additional comments, so that subjects could add their own thoughts.

**Observations during the session** Since I was present for help during the whole session, I was able to make various observations about what people seemed to struggle with, and which parts needed refinement.

**Observations in the following weeks** As a student assistant in the course, I was present during all the following exercise sessions, where the students had to use Reactive Blocks to complete assignments. This allowed me to make some additional observations about what they had learned from the tutorial, but I did not know which of the students had actually completed the tutorial, so this was by all counts less reliable data. It did however give me some indication about what they *needed* to learn from a tutorial.

### 4.3.2 Test Results

With the total number of test participants being low, a quantitative analysis of the tutorial's quality was out of the question. Instead, I use the results to look for areas with potential for improvement, or indications of what works well.

### Test Subject Profile

All of the participants in the user test were students of the TTM4115 course at NTNU.[5] Some of the participants were 3rd year students of the 5-year MSc in Communication Technology at NTNU, while others were students of the 2-year MSc in Telematics. All were assumed to have some knowledge of computer programming or software design. Prior to the user test, most of the students had a 2-hour introduction lecture for UML activities in Reactive Blocks.

### Problems During the User Test Session

Not everything went smoothly during the testing session, which caused some delays and some additional trouble for the participants. First of all, very few of the participants came prepared, and had to go through the process of installing Eclipse and Reactive Blocks before starting the tutorial. Some had trouble with this part, and needed help from me or my co-instructor. Additionally, a license is required in order to use Reactive Blocks, and not all participants had received one. Fortunately they were able to get one during the session, but at the cost of extra delay. Finally, there were some errors in the Reactive Blocks tutorial project that I had not discovered, which had to be fixed on the fly and caused some additional delay.

The testing session was only 2 hours, and as a result of the various delays, many of the participants were not able to complete the whole tutorial. This is likely the reason why many of the participants chose to not fill out the feedback form, and even from the few who did, there were several who did not complete the whole tutorial.

### Data from the Feedback Form

I received a total of 13 filled out feedback forms during and after the testing session. This section summarizes the answers, and attempts to highlight the information we can infer from these results.

**Question 1:** Figure 4.2 displays the results from question 1: *"Did you find the provided information about elements and concepts useful?"*. All participants answered that they found the provided information about elements and concepts to be useful. This tells us that at least there is nothing that is apparently wrong with the introductions to the exercises, however it does not necessarily mean they are very good.

**Question 2:** Figure 4.2 also displays the results from question 2: *"Do you think you would find the exercises more interesting if they were designed as a game, where you had to create a program to complete each level?"*. Here the answers varied a little

[5]TTM4115 course website (link)

more, with 2 participants answering that they would not be interested in learning these concepts from a game, while 5 answered that it would depend on the type of game. The remaining 6 gave positive answers. In hindsight, it would have been interesting to know which types of games the 5 participants were interested in, but this is also the kind of question one might not know the answer to without seeing some examples. Overall, the results indicate that a well-designed game may improve motivation for at least some of the students. For the remaining, it is hard to say whether the game approach will reduce motivation or simply leave it unchanged.

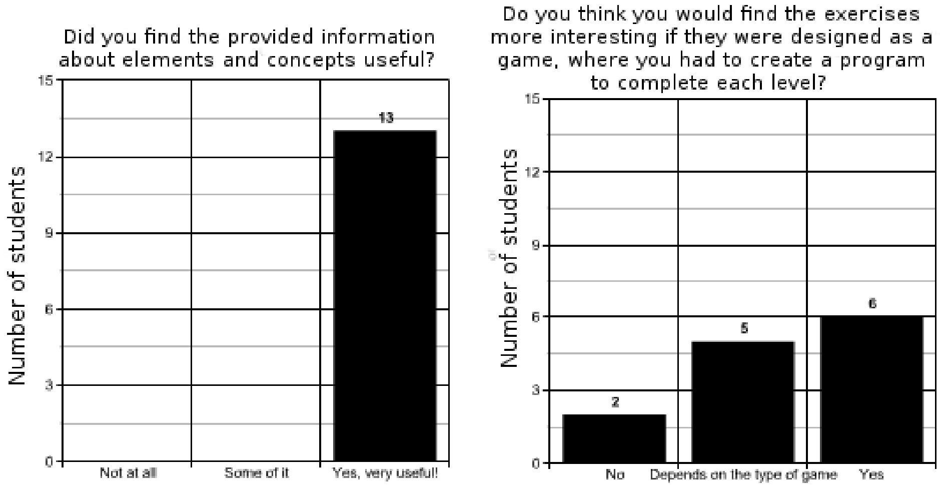


**Figure 4.2:** Results from questions 1 (left) and 2 (right) of the feedback form (Appx. B). The total number of students was 13.

**Question 3:** Figure 4.3 displays the results from question 3: *"Did you feel that you understood the concepts presented in the exercise?"*. The results are displayed as a separate graph per exercise, where the participants had to rate their own understanding. The most important point to notice from these results is that more than half of the participants did not have time to complete exercise 5 through 7, and 3 of these also did not complete exercise 4. It is possible that these are students in the "lower end of the bell curve", who have more difficulty and take more time in grasping the concepts presented in the exercise, but would have understood these concepts given sufficient time to explore them. Looking at the results for exercise 1 and 2, all or nearly all of the participants felt that they completely understood the concepts presented. As we move on to exercise 3, some of the participants are beginning to reduce their perceived level of understanding to "pretty good". From the participants who completed exercise 4 through 6, all measured their own understanding to be complete, but in exercise 7 the results are slightly worse. It appears that the most challenging exercises are 3 and 7, and these may have to be revised and improved.

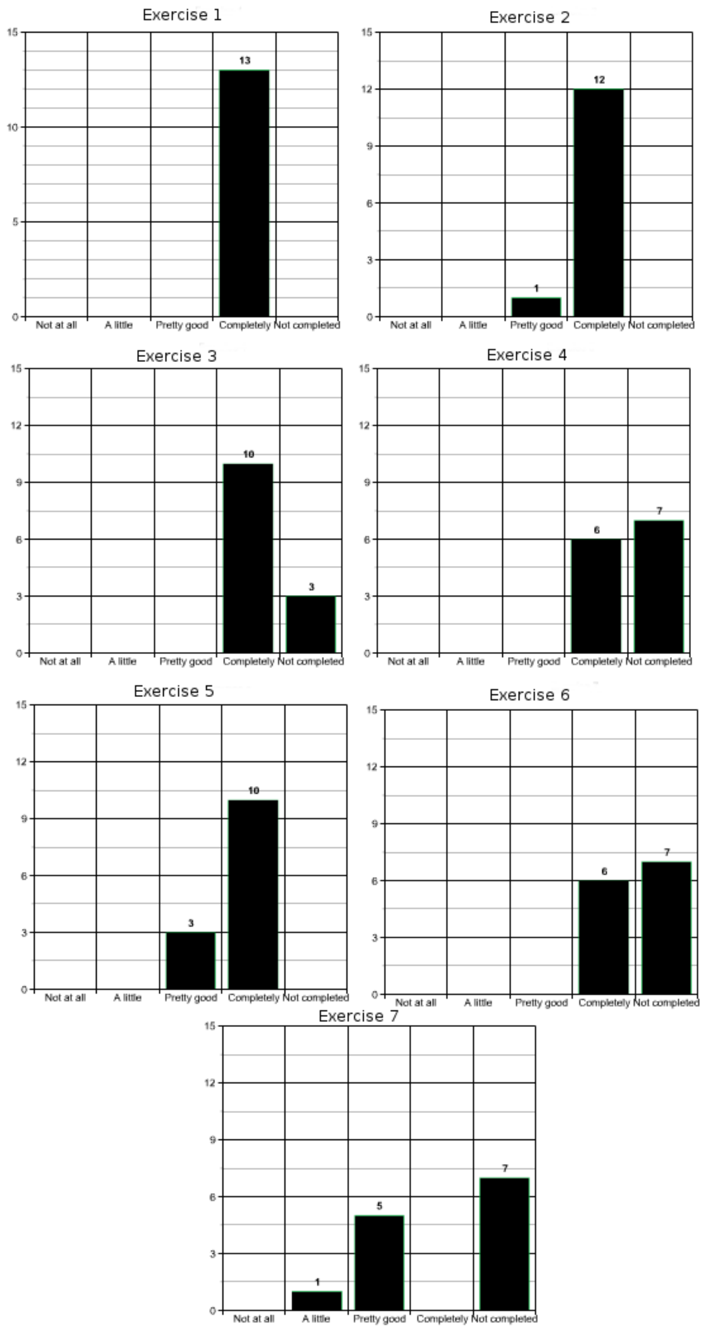


**Figure 4.3:** Results per exercise from question 3 of the feedback form (Appx. B). The value on the y-axis for all graphs is the number of students who chose that answer. The total number of students was 13.

**Question 4:** Figure 4.4 displays the results from question 4: *"Did you complete any of the challenge exercises?"*. The graph shows that most of the participants chose to complete challenge exercises 1 and 2, half completed exercise 3, and none completed exercises 5 and 7. Comparing this with the results from question 3, we see that all the participants who were able to complete exercises 4 through 7 had also completed challenge exercise 3, and the rest likely did not have time. In any case, the participants seemed motivated for completing the challenge exercises and gaining additional understanding of the concepts presented, but were constrained by time. Challenge exercises 5 and 7 were quite extensive, and would likely have required another 2-hour session.

**Question 5:** Figure 4.4 displays the results from question 5, which asked the participants to rate the overall difficulty of the challenge exercises. 1 participant answered "too easy", 1 felt that the challenge exercises were of "very mixed" difficulty, while the remaining found them to be "OK". Since these were not meant to be very difficult, and there is likely to be variations in how different participants perceive the difficulty of a given task, this information provides no grounds for changing the difficulty of these exercises.

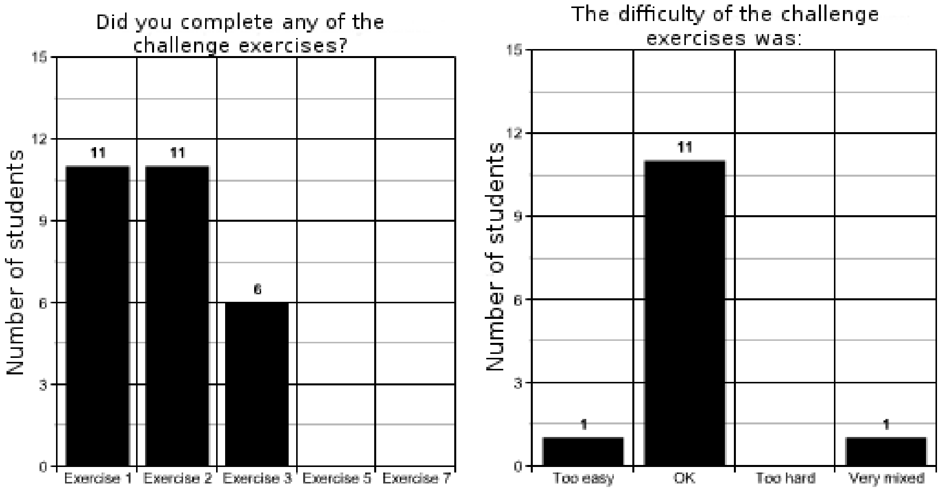


**Figure 4.4:** Results from questions 4 (left) and 5 (right) of the feedback form (Appx. B). The total number of students was 13.

**Question 6:** Since all participants completed at least one of the challenge exercises, there were no answers to this question.

**Additional comments:** While only a few of the participants bothered to submit their own thoughts, all of the comments received were positive. It was mentioned that this was a more fun way to learn than regular exercises, and that being able to complete several smaller exercises gave a sense of achievement.

### Observations during the Testing Session

In addition to the feedback form, I made some observations during the testing session about what the participants struggled with, and discovered some potential problem areas. Most of the observations are a result of the questions the participants asked directly to me.

**Technical problems** Most of the problems the students seemed to have were pure technical, and not related to the learning aspect of the tutorial. Some had trouble installing Eclipse and making it work properly, some had trouble setting up Reactive Blocks within Eclipse. While the tutorial document handed out to each participant contained a brief guide on how to set up and prepare for the tutorial, it appears this guide was not clear or detailed enough for some of the participants, who were unable to complete this process on their own.

**Loops and Flow Breaks** One of the main purposes of exercise 4 was to understand that loops needed to be split into more than one activity step, for example by the use of a flow break. While I thought this was well explained in the introduction, quite a few participants did not understand this part of the exercise, and had to ask for help. Most seemed to get it after a brief verbal explanation, so it is likely that the introduction part of this particular exercise can be improved.

**Decisions and Guards** During the session, I received quite a few questions about how to set guards on decisions in exercise 3. Participants understood that they had to set the decision values somewhere, but they had no idea how. This was not very well explained in the introduction part, and could clearly be improved. This issue can also be observed in the feedback form results, where some people felt that they lacked some understanding of these concepts.

**"Too long, did not read"** Sometimes, the students asked questions about things that were actually explained in the introduction, and they even seemed to understand it when I just repeated the information they already had. It is hard to know the reason behind this, but one possible explanation is that they did not bother to read the introduction because they felt it was too long.

**Tips Section** Since I had to provide a document instead of the tutorial mode I originally wanted in Reactive Blocks, it was difficult to provide the participants with general tips for solving the exercises in a reasonable way. I had compiled a list of tips at the end of the document, that may have helped the participants with some of the exercises, and even encouraged them to have a look when they encountered trouble. However, they seemed to forget or not know about the tips section, and instead asked me quite a few questions that were more or less directly answered there.

**Error messages** When the participants did something wrong, they were generally presented with error messages describing the problem. Very few seemed to understand these error messages however, as they were often presented in the form of Reactive Blocks concepts and terms they were not yet familiar with.

**The Reactive Blocks UI** In addition to questions related to the tutorial, I received a lot of comments about the Reactive Blocks UI. Many found the software awkward to use in various ways, and sometimes needed help finding particular elements and functions. Overall, there was a lot of frustration related to things that were outside the scope of the tutorial, such as being able to undo actions in Reactive Blocks.

#### Observations after the Testing Session

During exercise sessions in the following weeks, I was able to make some additional observations based on the questions I received from the students who were working with Reactive Blocks. While I recognized some faces from the testing session, I could not know for sure who had completed the whole tutorial, but some actually made references to it in their questions. The observations made after the testing sessions are mostly based on general impressions, since I did not take note of specific issues. Consequently, this is not very reliable data, but included anyway in order to add some additional perspective.

The most prominent problem the students encountered was that they were poorly prepared for working with Reactive Blocks outside of the tutorial context. They were not comfortable with creating their own local blocks with input/output pins and ESMs, and still had trouble connecting these to each other in a way that was consistent enough to be accepted by the Reactive Blocks model checker. Often, the problems were directly related to ESMs, which had not been covered by the tutorial, but sometimes the problems were actually very similar to those presented in the tutorial exercises, just in a different context. The lack of understanding of these concepts is also reflected in the results of the feedback form for exercise 7 (Fig. 4.3).

## 4.4 Evaluation of the Tutorial

This section provides an evaluation of the Reactive Blocks/UML activities tutorial, with respect to the goals that were set in advance, and the actual usefulness for learning about UML activity modeling. Finally, some suggestions are made as to how the tutorial could be improved.

### 4.4.1 Fulfilled Goals

Section 4.1.3 lists the goals that were set before creating the Reactive Blocks/UML activities tutorial. This section covers the goals I consider to be fulfilled by the tutorial that was tested.

**Interactivity** For each new concept introduced, users have to become familiar with this concept and some modeling elements by completing an exercise, and possibly an additional challenge exercise. Users become active learners, and have to actually think about the concepts.

**Feedback** Thanks to Reactive Blocks, users are able to execute their models, to see if they behave as intended.

**Reasonable Teaching Order** The teaching order of the concepts and elements was carefully considered to let the users start with the most basic concepts, and then build on these to create more advanced programs. None of the exercises required knowing about concepts that had not been previously introduced.

**Context-Sensitivity of Information** The new information required to complete each exercise is presented together with the exercise itself. People also have to retain some information from the previous exercises, but they have the chance to practice and really become familiar with it first.

**Multiple Perspectives** While some of the exercises present a different perspective on some of the concepts through challenge exercises, the contexts are fairly simple. This is a point where the tutorial has room for improvement, so this goal is considered only partially fulfilled.

**Difficulty and Challenges** During the user test, most of the users were able to complete the tutorial exercises. While there were some problems in understanding a few concepts, most of the problems encountered were of a more technical nature, and related to the use of Reactive Blocks. Additional challenge exercises were also available.

### 4.4.2 Not Fulfilled Goals

Unfortunately, not all of the goals listed in Sect. 4.1.3 were fulfilled. Below are the goals I consider to **not** be fulfilled by the tutorial that was tested.

**Help-on-Demand** I was not able to offer any help-on-demand within the context of the tutorial. Whenever users needed more information, they had to visit the official Reactive Blocks documentation online, or ask an instructor for help.

**Visual Mapping of UI Elements** Since I was unable to make changes to the Reactive Blocks environment, it was difficult to communicate to users how to properly work with the UI. During the testing sessions, some participants had trouble finding certain elements.

**Freedom** This goal was also not fulfilled because I was not able to make changes to the Reactive Blocks environment, and thus could not create a proper tutorial mode. However, some of the exercises could be solved in more than one way, giving users some freedom in how to think about the exercises.

**Focus on UML Activities** Also because of a lack of tutorial mode in Reactive Blocks, the tutorial had to teach the users quite a few Reactive Bocks -specific skills for them to be able to work with it.

**Avoid Irrelevant Complexities** No tutorial mode meant no abstraction of tasks like *build and run*, users instead had to learn to do these things manually. Additionally, users were frequently presented with error messages they did not understand, because they had not yet learned enough about Reactive Blocks.

**Everything in One Place** The lack of a tutorial mode forced me to distribute the tutorial over more than one platform. The exercises were completed within Reactive Blocks, but introductions and tasks were presented in a separate document. Additionally, users had to use online documentation to find additional information about a subject.

### 4.4.3 Notes on the Quality of the Tutorial

Despite not fulfilling all the goals that were set in advance, the tutorial still has some value. Based on the results from the user test, we attempt to get an impression of the actual quality of the tutorial. With *quality*, we consider the tutorial's ability to teach the various concepts and aspects of UML activities, its ability to motivate the users into wanting to learn, and the tutorial's overall usability. A tutorial with a low degree of usability is likely to waste some the users' time on issues that are irrelevant to the learning process.

### Ability to Teach Concepts

Judging by the results of the user test feedback form, comments received from the participants, and observations of the participants' subsequent work with Reactive Blocks, it seems that this tutorial does a fairly good job of teaching the concepts of UML activities to the target audience. The students appeared to mostly understand the concepts of activity steps and control tokens, as well as how the various model elements worked. They were less equipped to deal with problems related to ESMs and more advanced modularization, but this was mostly outside the scope of the tutorial. Since I was unable to conduct any comparative studies, we can not measure this tutorial's ability to teach against that of other similar tutorials.

### Motivating Users

In the comment section of the feedback form, some of the test participants suggested that the tutorial was a fun way to learn about UML activities and Reactive Blocks, as well as mentioning that completing several smaller exercises gave a sense of achievement after each one. This indicates that the tutorial gives the users some kind of motivation for exploring the topic. Again, it is difficult to compare with other approaches without additional data, but I find it unlikely that a group of students would have classified simply reading through documentation as a *fun way* to learn.

### Usability

The usability of the tutorial was severely hamstrung by the fact that I could not create a separate tutorial mode in Reactive Blocks. Most of the tutorial resources had to be provided outside the context of Reactive Blocks, which meant the tutorial offered less direct help, and users had to go through a lot of extra effort to find additional information. Users received error messages they could not understand, and even the provided information was in some cases insufficient for users to be able to efficiently complete the tutorial. A prime example is with decisions; many test participants spent a lot more time than they should have trying to figure out how to set guards on branches. With these points in mind, I consider usability to be the tutorial's weakest property. It is possible that in a larger user group, some of these usability issues could prevent a number of users from completing the tutorial without instructor help, or cause them to become too frustrated and lose motivation.

### 4.4.4 Suggestions for Improvement

After the tutorial has been tested with a small group of relevant users, some issues have been discovered and briefly discussed. This section suggests how the tutorial can be improved to account for these issues, and thus hopefully provide a better learning experience.

- Have a complete and immersive tutorial mode. This mode should provide the user with the information needed, both introductions as well has help-on-demand and tips, inside the application and in-context, and additionally abstract away processes like choosing Java platform and managing generated code. It should also ideally provide error messages that are better suited to the users' level of experience, and either hide elements that are *less* or highlight elements that are *more* relevant.
- In addition to the tutorial mode, the usability of the Reactive Blocks UI has some room for improvement, especially for novice users. Making elements available through buttons instead of menus and allowing *undo*-actions are some of the issues that require attention. Improving Reactive Blocks is a little outside our scope, but a better environment is likely to improve the tutorial experience.
- The introduction for decisions should be improved to properly explain how guards are set on branches.
- Supplement the introductions with animations that illustrate some of the more subtle aspects of UML activities in Reactive Blocks, such as tokens not being able to pass through a given node more than once in each activity step.
- If the tutorial will be used to teach users not only about UML activities, but how to work with Reactive Blocks, some additional steps should be created to better prepare users for the real environment. These steps should for example teach users more about different types of blocks, ESMs, and how to interpret error messages.

Chapter 5

# A Tutorial Game

As the second part of my own work, I designed and implemented a prototype for an educational game for UML activities in the context of Reactive Blocks, nicknamed simply *The Reactive Blocks game.* Based on the tutorial described in Ch. 4, this game teaches the same concepts in similar ways, but in a different environment. This chapter covers the motivation for creating the game, as well as a description of the design and implementation of the prototype. Additionally, a user test was conducted in an effort to uncover usability issues with the game prototype, supporting an evaluation of the game prototype.

## 5.1 Motivation

*Game-based learning* is a concept that is becoming more and more popular. Sections 2.4 and 3.2 describe how and why this approach to learning is becoming popular, with examples of some more or less successful learning games, and it is clear that games can successfully be used to teach various subjects and concepts.

Given the success of other educational games, it may be interesting to see if we can create a good game for learning UML activities, perhaps in the context of Reactive Blocks, in order to increase student motivation and understanding of how UML modeling works. In Sect. 4.3.2, we saw that most of the test subjects that provided feedback answered that they might prefer learning about UML activities through a game than a tutorial, so there could be a real interest for a game like this, given that it is well-designed.

### 5.1.1 Goals

Like with the tutorial in Ch. 4, I initially set some goals for the design and implementation of the game. These goals are primarily based on experiences from the tutorial implementation and test, as well as the good practices from Sect. 3.2.4.

**Immersion** The primary goal for the game is to provide an environment where players can immerse themselves in the learning experience. This involves giving the players a sense of identity within the game, and providing them with challenges that are easily visualized and comprehended within the given environment.

**Exploration and Guidance** UML activities can be used to model very complex systems, and learning how to do this is not a trivial process. The game is thus likely to benefit from providing some guidance to players, as opposed to complete freedom to explore [AOL$^+$12]. Some freedom is still necessary, to facilitate creativity and deeper learning [BSG$^+$09]. The goal is then that the game should be primarily focused around a tutorial-like guided path for learning about concepts within UML activities, but with the possibility for players to find their own solutions to problems. Players should also be encouraged to explore other perspectives, and maybe find different or even *better* solutions to the same problems.

**Level of Difficulty** The target audience for the game is the same as for the tutorial (Sect. 4.1.2). The game should be easy enough for less knowledgeable users to get started without frustration, but eventually provide challenges that feel relevant also to the more experienced. Players should also have the option to skip challenges they feel are less relevant to them, or that involve concepts they have already mastered, in order to avoid the game experience becoming tedious.

**Lessons from the Tutorial** The design and implementation of the tutorial in Ch. 4 fulfilled some, but not all of the goals that were set. The approach seemed to provide a decent learning experience, but with some potential for improvement. A goal for the game is to build on the parts of the tutorial that were successful, such as the teaching order, and additionally include some of the parts that were not implemented or successful, such as having everything in one place. Since the game still will be based on Reactive Blocks, many of the same challenges will be present, but with a game, there should be more options for providing resources within the same platform, such as help-on-demand.

## 5.2 Game Design

When developing the concept of the game, a few points were considered. First of all, it should involve some sort of main character, giving the player a sense of identity. Secondly, the game concept must accommodate a range of different types of challenge that are suited for learning about concepts like concurrency, modularization and reuse. Finally, the concept must be relatively easy to prototype, as the available time to implement the game was fairly limited.

### 5.2.1 The Final Concept

With the above design points in mind, I settled on a two-dimensional maze-navigating game, where the player controls a character or set of characters through operations and logic in Reactive Blocks. The main character, Malcolm, must perform one or more goals to complete the level, such as gathering stars. In order to get to these stars however, Malcolm must perform some other tasks, such as gathering keys, or invoke the assistance of other characters in the game.

This concept was chosen for the following reasons:

- Two-dimensional games are quick and easy to implement, both graphically and programming-wise, given an appropriate framework.
- Since the purpose of the game is to teach UML activities, it makes most sense to provide a game environment where the player must program their characters' path in advance, and then see if they modeled the desired behavior correctly. A two-dimensional maze-navigating game makes it easy for players to get a clear overview of what they are supposed to do.
- Despite being relatively simple to implement, a two-dimensional maze-game offers a lot of possibilities in the form of obstacles and challenges the player must overcome in order to complete each level.

Even with such a flexible concept in place, it was no trivial task to create levels and challenges that let the user learn about UML activity concepts in an intuitive and reasonable way. In addition, I had to consider how each concept should be introduced within its level, so that the user would know what to expect.

### 5.2.2 The Tutorial Part

One of the most important goals for the game was to provide an environment where the player could find all the information needed to complete a level, instead of having to switch between contexts or search through external sources. Players would be creating the logic for each level in Reactive Blocks, generate code from these models, and run it to see if it was correct. Thus, additional information, such as concept introductions, level maps, and help-on-demand, had to be presented inside Reactive Blocks. Unfortunately, because I was unable to alter the Reactive Blocks environment, I could still not provide a tutorial mode, just as with the tutorial in Ch. 4.

Instead of a tutorial mode in Reactive Blocks, a different solution presented itself. I could create a separate program with a window that would contain the most basic information needed for each level, with internal links to additional resources. This would of course not be completely within the context of Reactive Blocks, but hopefully a decent substitute for the lack of a real tutorial mode.

Designing the tutorial window presented some challenges of its own. The window needed to be small enough for players to be able to keep it on their screen together with the Eclipse/Reactive Blocks window, preferably also for smaller screens like on laptops. At the same time, I needed to include all the info a player would realistically need to complete each level. Obviously I would not be able to fit all the information in one small window at the same time, so I had to decide which elements to display by default, and what should only be displayed upon the player's request.

After some consideration, I settled on a 600x800 pixels window, which should fit pretty good with the Reactive Blocks modeling canvas on screen resolutions down to 1280x800. When the screen resolution becomes smaller than this, there is little room for anything but the Eclipse window anyway. Given more time for development, it would likely be better to have a variable size window depending on the available screen resolution, but for now, this size is fixed.

Figure 5.1 shows a mock-up of the introduction window design. The layout is the same for all levels, but the content varies. At the top is simply the title of the level, with a "tagline" giving a short description of what the current level is about. Below the title is a slide show, which the player may go through to get a quick introduction of the concepts taught by, and the goals of, the current level. These introductions are generally kept short, except for in level 1, where the whole game must be introduced in additional to all the most basic concepts. The player controls the slide show with the arrow and play/pause buttons.

Below the slide show, we see a list of the goals for the current level. The goals, together with the level map at the bottom, is the most important information of the level. The level map and the goals are always visible, so that the user has easy access to these when working on the solution for the current level. The level map varies in size, but is always scaled to fit the small frame in the bottom left corner. However, since the player sometimes has to check details of the map, it is possible to make it bigger by clicking on it.

Finally, the buttons on the right side of the window represent the *help-on-demand* information that is not visible by default. The *Tips* button provides some tips that are good to know either for solving the current level, or for learning about modeling in general. The hope is that if players get stuck or are unsure about how to proceed, they will see and click this button in an attempt to find help. The *Tell me more about...* buttons provide more detailed information about each concept presented in the current level, available for players who need more information in order to

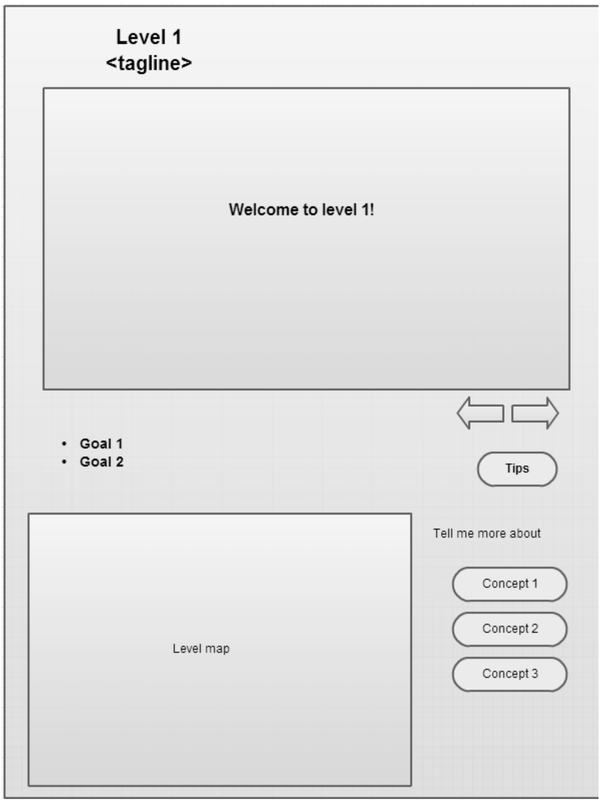


**Figure 5.1:** The introduction window design, with the introduction slide show at the top, followed by the goals for this level, a button for tips, the level map, and some buttons for retrieving additional information about the concepts presented.

understand the concepts sufficiently, or simply want to learn more.

### 5.2.3 Game Levels

For the first version of the prototype, only 5 levels were designed and implemented. This was because I wanted to test the prototype with a few users as soon as possible, in order to uncover any serious flaws with the UI or the way the introductions were presented as early as possible.

The prototype levels roughly follow the teaching order described in Sect. 4.2.2. The first version of the prototype covers steps 1 through 3, with level 5 introducing part of step 4. The game levels are briefly described below.

### Level 1

Figure 5.2 displays the map for level 1. The purpose of this level is to introduce the game concept, the main character Malcolm, and the most basic modeling concepts in Reactive Blocks. The goal is pretty much identical to exercise 1 of the tutorial (create a *Hello World!* program), but the game additionally visualizes this task through the game character. The player is provided with the *sayHello* operation, which makes a speech bubble appear next to the character on the level map.

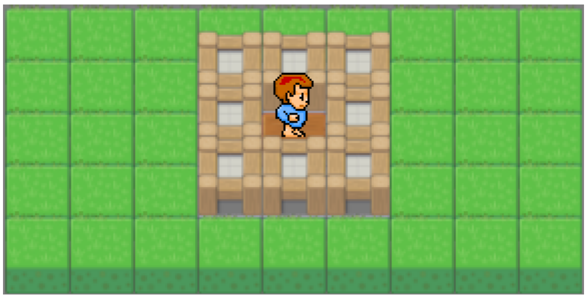


**Figure 5.2:** Level 1 of the Reactive Blocks game. The player must make the character speak, which makes a speech bubble appear beside him.

**Goal:** Make the game character speak the famous words “Hello world!”

**Concepts introduced:** Activity Steps, Edges, Initial Nodes, Operations, and Activity Final

**Operations:** sayHello

### Level 2

Figure 5.3 displays the map for level 2. The purpose of this level is to introduce the player to moving the game character around, and learn about timing delay with *timers*. The goal is to make the character start moving forward, and then add correct timing so that the character will stop moving on top of the star, allowing him to pick it up. The time it takes to move from the starting position to the star is directly proportional to the distance, as it takes 500 milliseconds to cross one tile.

**Goal:** Make the game character pick up the star

**Concepts introduced:** Moving forward, Timers

**Operations:** moveForward, stop, pickUp

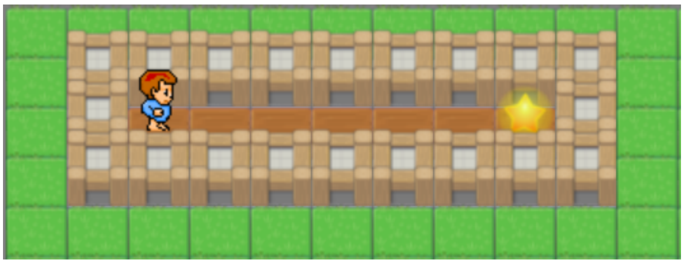


**Figure 5.3:** Level 2 of the Reactive Blocks game. The player must make the character move 6 tiles forward by timing the movement correctly, and then pick up the star.

### Level 3

Figure 5.4 displays the map for level 3. Modeling wise, this level does not introduce any new concepts, but lets the player become more familiar with timers, and use operations to also change the direction of the game character. Level 3 is a follow-up challenge for level 2.

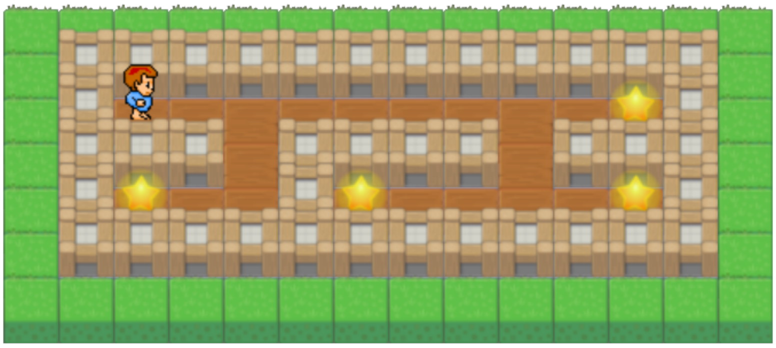


**Figure 5.4:** Level 3 of the Reactive Blocks game. The player must move the character around the maze, and pick up all the stars.

**Goal:** Make the game character pick up all four stars

**Concepts introduced:** Moving in different directions

**Operations:** moveForward, stop, pickUp, turnLeft, turnRight, turnAround

### Level 4

Figure 5.5 displays the map for level 4. The purpose of this level is to introduce the player to *decisions*, *alternate branches*, and handling things that may not be known in advance. Players also learn to pass data from operations to decisions, allowing *guards* to be set on outgoing edges.

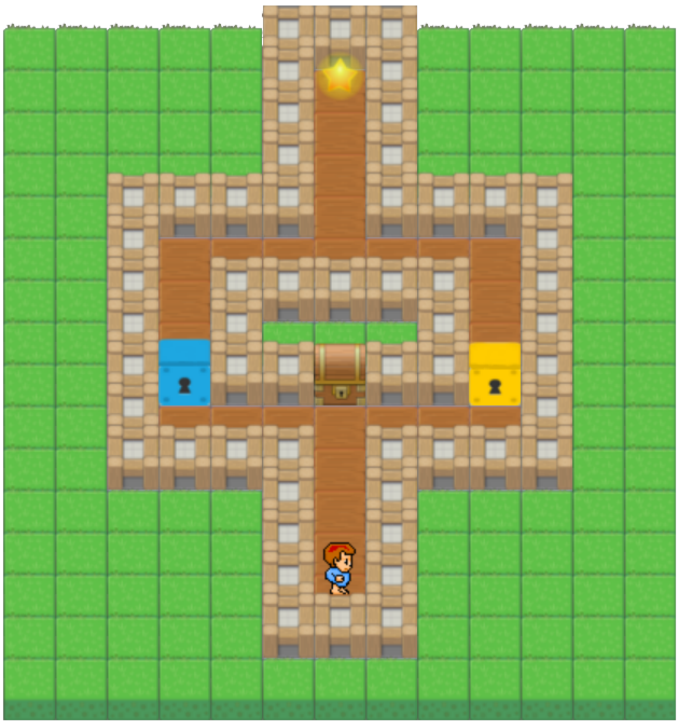


**Figure 5.5:** Level 4 of the Reactive Blocks game. The player must open the chest to reveal an either blue or yellow key, which allows one lock to be unlocked. The character can then pass through to the star.

**Goal:** Open the chest to find a key, unlock the lock matching the key, and pick up the star

**Concepts introduced:** Alternate branches, Decisions, Object flows

**Operations:** moveForward, stop, pickUp, turnLeft, turnRight, turnAround, interact

### Level 5

With the introduction of alternate branches, activity diagrams can become large and messy. Logic has to be (re)created for each branch, and if more branches follow, we quickly see an explosion of the decision tree. Fortunately, there may be ways of simplifying this, depending on what happens in each branch.

Figure 5.6 displays the map for level 5. There are 4 locks with different colors, meaning that there are 4 possible branches/paths after the chest has been opened. However, the final part of each path is identical, meaning we can use the same sequence to complete the logic. The purpose of this level is to introduce *reuse* to players, by using the same sequence to complete all alternate branches with one or more *merge* nodes.

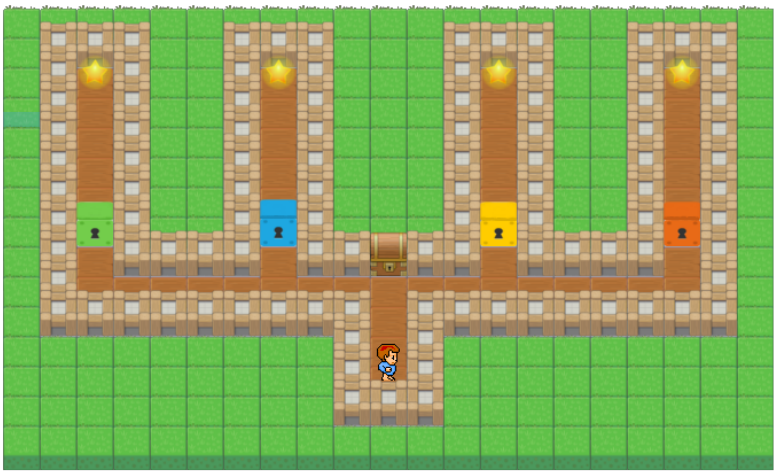


**Figure 5.6:** Level 5 of the Reactive Blocks game. The player must open the chest to reveal an either green, blue, orange, or yellow key, which allows one lock to be unlocked. The character can then pass through to the star.

**Goal:** Open the chest to find a key, unlock the lock matching the key, and pick up the star

**Concepts introduced:** Reuse of sequences, Merge nodes

**Operations:** moveForward, stop, pickUp, turnLeft, turnRight, turnAround, interact

## 5.3 Implementation

With the game concept in place, it was time to start working on the prototype implementation. Since Reactive Blocks is Java-based, the natural choice was to start with a Java framework for creating games. I settled for *libGDX*,[1] a Java game development framework licensed under Apache 2.0.[2]

[1]LibGDX: Java game development framework (link)
[2]Apache License, Version 2.0 (link)

Even with a framework as a starting point, implementing the game prototype required considerable work. The game itself was implemented as a JAR to be included with a Reactive Blocks project containing blocks for each level. Graphics are based on or retrieved from various collections of free graphics on the web. For the interested reader, both the full source code[3] and the Reactive Blocks project[4] is available online.

The result is a game prototype with 5 levels, and an introduction part for each level. The level maps were designed using the *Tiled Map Editor*.[5] Below, each component is described in more detail, using level 4 as an example.

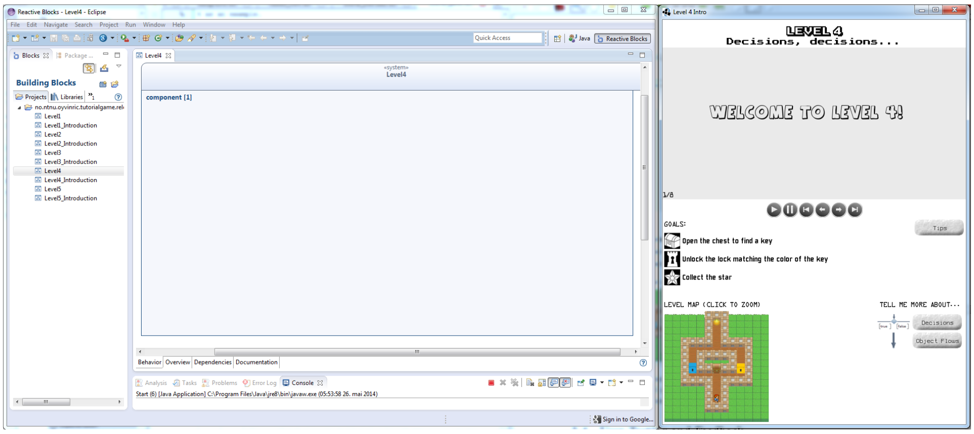
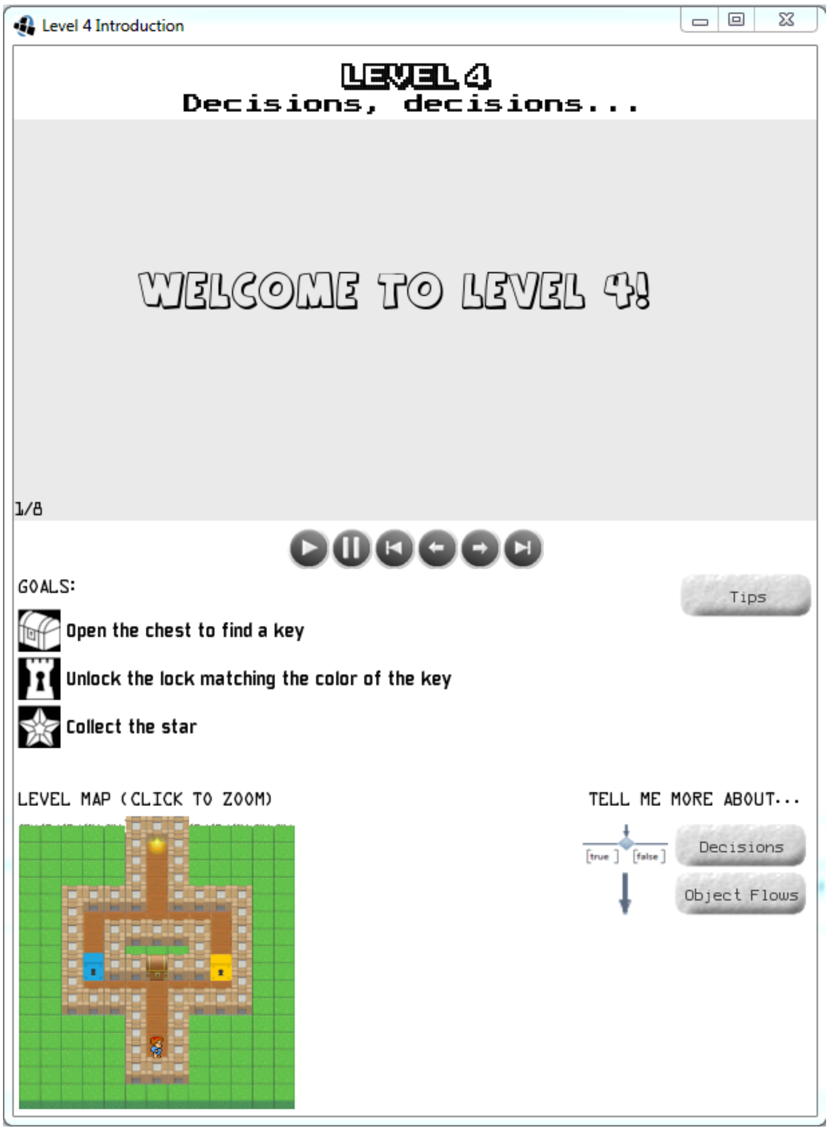


**Figure 5.7:** The Reactive Blocks game interface. On the right side is a regular Eclipse window running the Reactive Blocks perspective with the block for level 4 open. On the right side is the introduction window for level 4.

**The Game Interface** Figure 5.7 shows the interface of the game on a wide screen (1920x1080 pixels), with the Eclipse window running a Reactive Blocks perspective on the left, and the introduction window on the right. The player starts playing the game by "building and running" the *Level1_Introduction* block, which makes the introduction window pop up. A more detailed view of the introduction window implementation is displayed in Fig. 5.8. The player can then open the *Level1* block and start modeling, by right-clicking the empty canvas and adding elements. The required operations are already implemented, referencing the JAR with the game logic, and the player can simply add them to the model. An example solution for level 4 is displayed in Fig. 5.9. When the model is complete, it is time to "build and

[3]Tutorial Game source code project on GitHub (link)
[4]Tutorial Game release project on GitHub (link)
[5]Tiled Map Editor website (link)

run" the *Level1* block to see if the solution is correct. If this is the case, the player moves on to *Level2_Introduction* and repeats the process.



GOALS:

Open the chest to find a key

Unlock the lock matching the color of the key

Collect the star

**Figure 5.8:** The implementation of the introduction window for level 4 of the Reactive Blocks game. This implementation matches the design displayed in Fig. 5.1.

The introduction window is described in more detail in Sect. 5.2.2, and matches the design mock-up. Like the rest of the game, it is implemented within the *libGDX*

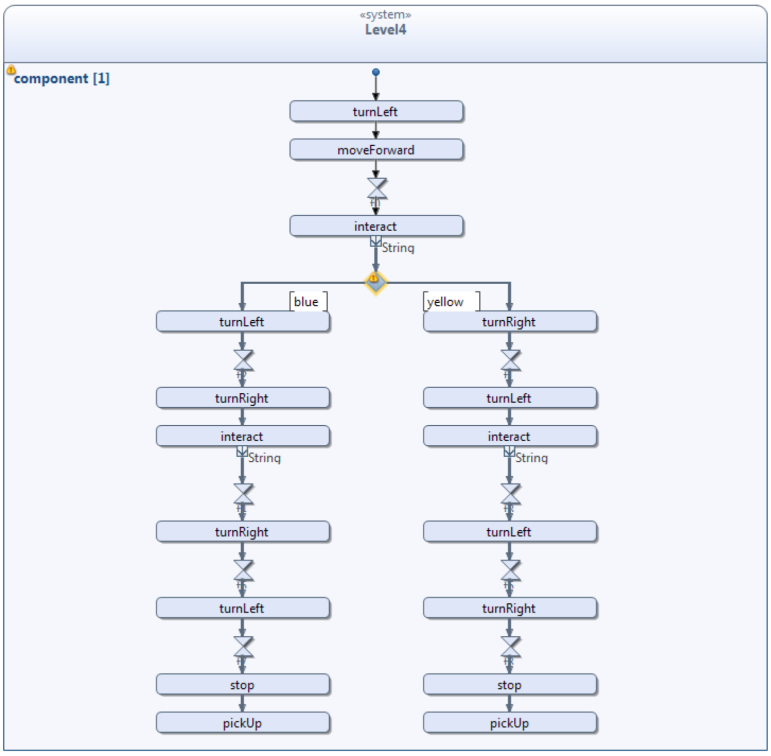


**Figure 5.9:** An example solution for level 4 of the Reactive Blocks game.

framework. The introduction slide show for each level is available on YouTube.[6]

**The Game Window** Figure 5.10 shows the main game window after level 4 has been completed. This window is opened when the player "builds and runs" the *Level4* block after completing the model, and assuming the model is correct, displays the desired behavior of the game character. A short video of a successful run through each level is available on YouTube.[7]

The main part of the window is the level map, where we can see the game character moving around and interacting with various objects. On the right side is a small HUD, which displays the current status of the level, such as how many stars have been picked up, how many need to be picked up in total, and which keys have been found.

[6]Reactive Blocks Game introductions (YouTube, link)
[7]Reactive Blocks Game level completion recordings (YouTube, link)

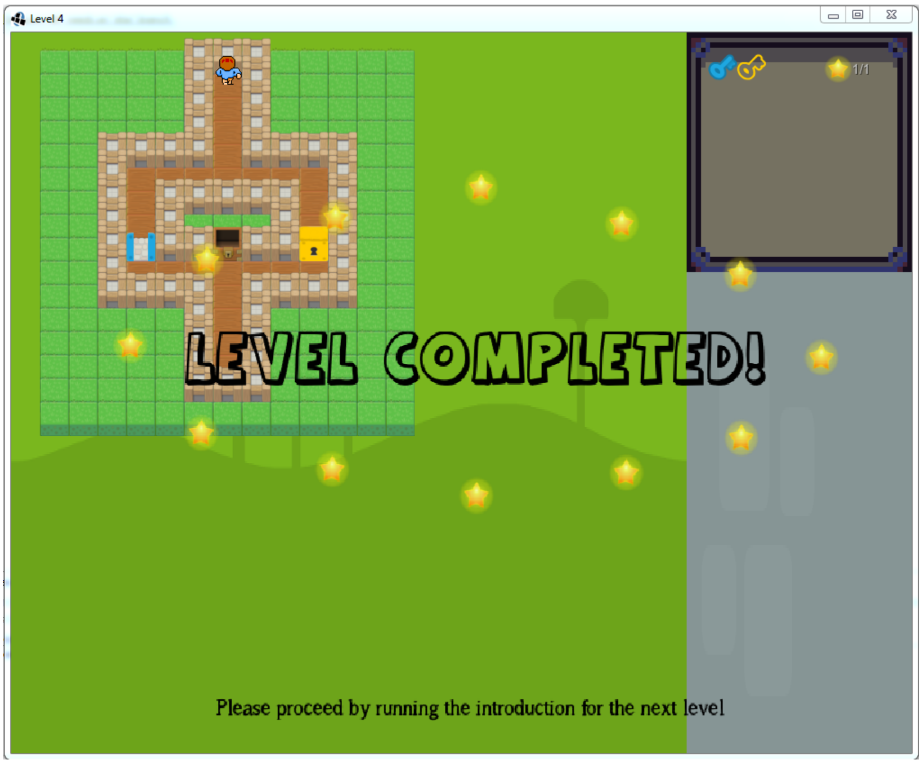

Level 4
1/1
LEVEL COMPLETED!
Please proceed by running the introduction for the next level

**Figure 5.10:** The game window for the Reactive Blocks game after level 4 has been completed. The level map is displayer on the left, while the right side contains a HUD informing the user about the current status of the level.

## 5.4 Usability Testing of the Game

With a prototype of the game ready, it was time to do some usability testing to see how players are able to work with the UI, and grasp the concepts presented sufficiently to complete the levels without help. Suitable test subjects were even harder to come by than with the tutorial in Ch. 4, but I was able to recruit three volunteers. Two of the test subjects fit within the target audience, having some previous experience with programming and system engineering, while the third subject offers a slightly different perspective, being an experienced gamer.

The primary goal for the usability test is to uncover issues about the game that may decrease the quality of the learning experience. Such issues may include poor affordance in the game UI, or insufficient, missing, or confusing information about concepts.

### 5.4.1 Testing Method and Collection of Results

The testing session is carried out as a fairly standard discount usability test [Nie94], with the significant difference that a functional prototype is used instead of a simple mock-up. The user is presented with some scenarios to go through, in this case starting with instructions on how to set up the game, followed by a challenge to solve for each level. While working on the tasks, the user is encouraged to think aloud in order to give the supervisor (the author) additional information about their choices.

Supervisor intervention is kept to a minimum. The supervisor will only intervene when a problem with the game has been clearly established, and there is no point in watching the user struggling further.

In an attempt to measure the users' understanding of the concepts presented in each level, the users are asked to fill in a feedback form[8] rating their own understanding of these concepts after the level has been completed. The purpose of the questions in this form is to get an impression on how difficult the game is to understand, if there is anything missing from the introductions, and if the users feel that they have understood the concepts.

In accordance with the discount usability test approach, a brief heuristic evaluation of the overall game UI is also performed. The UI will be evaluated against *The Eight Golden Rules of Interface Design* [SP10].

### 5.4.2 Test Subject 1

Test subject 1 is a 25 year old male with a university degree in petroleum engineering. The subject has some previous programming experience with numerical computations in MATLAB or Fortran, but no experience with system engineering. He is however a very experienced player of various video games within different genres.

While observing test subject 1 playing the game, I discovered both some problems with the game UI and some bugs that needed fixing. I will not go into details about the bugs, but attempt to highlight some of the UI issues. Subject 1 spent a total of 83 minutes on the 5 levels of the game.

The first thing I noticed was that the subject had trouble following the slide show, which would start automatically and change slide every 10 seconds. It is possible that having a slide show that runs automatically is not a very good idea, considering that users read and understand concepts at different speeds.

[8]See Appx. C

Throughout the whole game experience, the subject consistently had trouble navigating the Reactive Blocks UI, making some tasks unnecessarily difficult to complete, and slowing down the overall progress. This includes things like trouble finding the correct elements, or trying to connect edges to nothing. At one point the subject accidentally double-clicked an operation, bringing up the Java code for the current level, which added some confusion.

At level 1, the subject thought that an Activity Final node was required to complete the program. This was a completely logical observation, however unfortunate, since adding an Activity Final node to the application will terminate the game before the user can see that the level is completed. The tip that said an Activity Final node was not needed, was apparently not sufficient, as the subject eventually gave up on understanding what the problem was, and asked for help.

In more than one case, the subject was able to complete a level without having correct timing. The game character would just keep walking into a wall until finally the timer expired, and a star was picked up.

On level 4, the subject struggled with understanding decisions and guards, particularly which data type should be used. String values *blue* and *yellow* were mentioned in the introduction, but the subject believed the guards should be set to *true* for blue, and *false* for yellow. The subject was also confused by the option of adding an *else* branch from decisions, which was not really necessary in this case. Additionally, it was not clear to the subject where the data would come from, and he started experimenting with *variables*. In the end, the subject needed some help with understanding which data type should be set on the guards.

While struggling with level 4, the subject started actively looking for ways to discover what the problem was, and discovered the *analyze* functionality of Reactive Blocks. This could have helped him solve it, but he did not understand the error messages presented.

After the subject had understood how to set guards from decisions in level 4, he noticed that the branches could be merged for the final part, which is the learning goal for level 5. Since merge nodes had not yet been introduced, the subject thought the *join* node would be the correct choice, which resulted in some error messages the subject did not understand (in fact, these error messages did not really give any real information).

When working on level 5, the subject tried to add multiple outgoing edges from an operation, and set guards on these, without adding a decision node in between. The subject also got confused by the *connector merge* node in Reactive Blocks, as he tried to use this node instead of the regular *merge* node, which he did not discover until later. Finally, the key/lock value corresponding to *red* actually look orange within the game, which added some confusion.

Each time the subject would get stuck or have trouble understanding a concept, he would check the tips for that level, or the help-on-demand resources for that concept. The subject needed extra information on levels 1, 4 and 5, but was able to complete 2 and 3 with the information from the introduction alone. Levels 2 and 3 were additionally completed on the first try, while the other levels required some trial and error.

### 5.4.3 Test Subject 2

Test subject 2 is a 24 year old male, working on a university degree in Computer Networking and Signal Processing. The subject has some previous programming experience from various university courses, ranging from basic object orientation to microcontroller programming, and some HCI implementation.

Test subject 2's playing session was also far from flawless. Subject 2 encountered many of the same problems with the game as subject 1, in addition to some new issues. Subject 2 actually uncovered more bugs during his session, because his approach to solving the levels touched more edge cases. Subject 2 spent a total of 66 minutes on the 5 levels of the game.

First of all, subject 2 struggled with many of the same things related to the Reactive Blocks UI. The difference was that subject 2 was familiar with some other software development tools, and thus had some expectations that were not met with the UI. However, because of the initial awkwardness and annoyance experienced when working with the Reactive Blocks UI, the subject later discovered some actions that would simplify the modeling process on his own, such as duplicating elements or sequences of elements.

Subject 2 experienced the same problems in trying to follow the introduction slide show. Sometimes it progressed too fast, and he had to go back to finish previous slides.

Like test subject 1, subject 2 also experienced confusion about the *activity final* node. He commented that since the element had been introduced as a way of "finishing" a program, it was implied that this element should be the final part of all programs, and was actually *required* to end an activity step.

Test subject 2 additionally experienced some confusion about the goals for some levels, such as believing he had to return to the starting point in order to complete.

Unlike subject 1, subject 2 was able to better understand decisions and guards, and was thus able to complete levels 4 and 5 more easily. However, since the *object flow* was introduced as an individual concept, he believed this was an element separate from control flow edges, and spent quite some time looking for it.

Likely due to his previous programming experience, the subject attempted to use the logical operation *OR* to simplify the alternate branches in level 5. This is unfortunately not possible, as guards in Reactive Blocks only accept *literal* values.

Unlike subject 1, subject 2 was able to create the correct logic on his first try for all levels (disregarding the use of *activity final* in level 1). Subject 2 also actively used the *tips* functionality to find help, but rarely needed to check the help-on-demand resources.

Subject 2 also mistook the red key/lock pair to be orange.

### 5.4.4 Test Subject 3

Test subject 3 is a 24 year old male, with roughly the same background as test subject 2.

Like subject 2, subject 3 also uncovered quite a few bugs during his session, also some new ones. Otherwise, the issues encountered were largely the same as with subjects 1 and 2. Subject 3 spent a total of 74 minutes on the 5 levels of the game.

Subject 3 began the testing session by commenting right away that he did not prefer the introduction slide show to start automatically, and progress in fixed intervals. The subject would then go on to pause the slide show for each level right away, or go back to the start if he forgot to pause, and then progress manually.

With many of the same expectations for the Reactive Blocks UI as subject 2, subject 3 also faced the same kind of awkwardness and annoyance when learning to work with it. Having already established that this was an issue, I gave the subject some tips on working with the UI in order to lessen his frustration.

Like the other two subjects, subject 3 found it natural to include an *activity final* node to complete the program. He also explained that he did not realize the various parts of an activity step were performed without intermediate delay, and thus expected to see the result of completing the level even if the program was terminated (after a delay).

Like subject 1, subject 3 was confused about how to add guards to decisions, not realizing he was supposed to first create outgoing edges. He attempted to work around this by using the “Add else branch” option in Reactive Blocks, but this proved a more confusing approach, as it created *control flow* edges which may only have *else* guards.

Unlike the other subjects, subject 3 preferred to solve some of the levels incrementally, particularly level 3 and 5, by completing part of the logic and checking if everything was correct so far. Subject 3 also spent more time reviewing *tips* and help-on-demand sources than the other subjects.

Subject 3 also attempted the use of the logical operator *OR*, and mistook the red key/lock pair to be orange.

### 5.4.5 Data from the Feedback Forms

In addition to my observations, each test subject filled in the feedback form included in Appx. C. This section summarizes the data gathered from these forms.

The first question asked, for each level, if the introduction part gave enough information to solve that level. All three subjects answered consistently *yes* for all levels, except subject 2 for level 1. Subject 2 further commented that he thought the introduction part gave misleading information about the *activity final* node, causing his solution to level 1 to sort of be both right and wrong at the same time. It was difficult to understand what was going wrong, and subject 2 eventually had to ask for help from the supervisor. As we can see from my observations in the previous sections, this actually also happened with the two other subjects. Since subjects 1 and 3 answered that the introduction part gave them enough despite their trouble, they may have considered this as an oversight of their own, rather than missing information.

Question 3 asked the subjects to rate how difficult they found each level to be, and the results are displayed in Tab. 5.1. From the table, we see that all three subjects found levels 2 and 3 to be easier than the others. The two subjects with more diverse programming backgrounds, subject 2 and 3, also consistently experienced each level as being easier than subject 1.

Question 4 asked the subjects to rate their own level of understanding for each concept presented, and the results are displayed in Tab. 5.2. There is a lot of variation, but we should notice that none of the subjects felt they gained a *complete* understanding of *activity steps*, *activity final*, and *timers*. Subjects 1 and 3 actually did not rate their own understanding as complete for any of the concepts presented.

The fifth question asked, for each level, whether the subjects thought they would be able to use the concepts they had just learned about to solve other problems. All three subjects consistently answered *yes* to this question for all levels.

| **Level** | **Subject 1** | **Subject 2** | **Subject 3** |
|---|---|---|---|
| 1 | Difficult | OK | OK |
| 2 | OK | Easy | Easy |
| 3 | OK | Easy | Easy |
| 4 | Difficult | OK | OK |
| 5 | Difficult | Easy | Easy |

**Table 5.1:** The *perceived level of difficulty* for each level and each test subject. Available options were in ascending order *Too Easy*, *Easy*, *OK*, *Difficult*, and *Too Difficult*.

| **Concept** | **Subject 1** | **Subject 2** | **Subject 3** |
|---|---|---|---|
| Activity Steps | Partly | Mostly | Mostly |
| Edges | Mostly | Completely | Partly |
| Operations | Mostly | Completely | Mostly |
| Initial Nodes | Mostly | Completely | Mostly |
| Activity Final | Partly | Not at all | Mostly |
| Timers | Mostly | Mostly | Mostly |
| Decisions | Partly | Completely | Mostly |
| Object Flow | Mostly | Completely | Mostly |
| Merge | Mostly | Completely | Mostly |

**Table 5.2:** The self-rated *level of understanding* for various Reactive Blocks elements by each test subject. Available options were in ascending order *Not at all*, *Partly*, *Mostly*, and *Completely*.

In addition to the question asked for each level, there were some final questions. The first question asked whether the subjects *enjoyed* playing the game, to which subjects 1 and 3 answered *yes*, and subject 2 answered *a little*. As a follow up, they were asked to comment on what they did or did not like. While subject 1 left this field empty, subject 2 mentioned that it was a good way of learning concepts, and that it gave him a sense of achievement. Subject 3 mentioned that the UI had been somewhat annoying to work it. Finally, the subjects were asked if they would prefer the game as a way to learn about UML activities and Reactive Blocks, compared to other options. All three subjects answered in favor of the game.

### 5.4.6 A Side-Note on an Informal Experiment

In addition to the formal usability tests conducted with the three volunteer subjects, I thought it might be interesting to test the game experience on someone completely outside the target audience, just to see what happened. I decided my girlfriend, an elementary school English teacher with no experience even remotely related to programming (she does not like working with computers), would be a suitable subject.

The experiment was conducted in an informal way, where she would simply do her best to absorb the information provided in the introductions, and then try to solve the exercises. I helped with some of the parts that were more technical and less relevant, such as preparing Eclipse and Reactive Blocks, and building and running the blocks. If she got completely stuck, I would also offer some hints, primarily about where she could find more information.

To my slight surprise, my girlfriend actually handled the problems presented very well. She read the introductions carefully, and easily understood the concepts of operations, tokens and control flow. She handled timers without trouble, struggled a little with decisions (I had to explain what *Strings*, *booleans*, and *integers* were), and used *Merge* nodes perfectly to simplify the logic in level 5. She did spend notably more time on each level than the three subjects in the formal tests (she even got a little more help with UI issues), but this could simply be because she was not used to building this particular kind of mental models, added to the fact that she read the instructions extra carefully.

Despite her aversion for working with computers and lack of interest for software development, she found the game fun to play because it let her get the sense of mastering a skill she thought was far beyond her abilities. When starting both level 4 and 5, she sighed and exclaimed that *"it looks really difficult!"*, but attacked the problems with determination, reading and re-reading the introductions until things started to make sense.

I should also include that while my girlfriend was able to understand the concepts presented sufficiently to solve problems within the context of the game, she admitted to having no idea about how they could be used for other purposes.

While I found this experiment to be quite interesting, its informal nature prevents me from drawing any real conclusions. It is likely that a large part of my girlfriend's motivation was to support my thesis work, as opposed to wanting to learn about UML activities. Also, despite not having any interest in working with computer-related topics, she could simply be the type of person who has a knack for understanding these things (without even knowing it). All in all, much is left to speculation, but the experiment provides an interesting anecdote for the potential of educational games.

## 5.5 Evaluation of the Tutorial Game

Following the usability test of the game prototype, a formal evaluation of the prototype is prudent. This evaluation is done in three parts, starting with a heuristic evaluation of the game UI, followed by a discussion of the usability test result, and finally some suggestions for improvement.

### 5.5.1 Heuristic Evaluation

As the first part of evaluating the tutorial game, I conducted a heuristic evaluation of the UI. The UI primarily consists of three elements, namely the Reactive Blocks modeling environment, the introduction window, and the game window. Since there is little interaction with the game window after it has been opened, this part is discussed in less detail. The Reactive Blocks UI is included in the evaluation despite not being designed as a part of this project, because it plays a significant role in the overall usability of the game.

The evaluation is performed with respect to *The Eight Golden Rules of Interface Design* [SP10], combined with the context of good practices for games.

#### Strive for Consistency

Similar processes should require similar actions, and elements with similar functions should display similar affordance, making the UI feel more consistent for the user.

In the introduction window, all buttons for information windows look the same, and perform the same action (though with different content). All these windows may be closed by a corresponding *close* button. Additionally, the buttons for controlling the slide show have similar looks, but different symbols according to their function. The game map displayed in the introduction window is identical to the one found in the game window. The remaining elements are mostly unique, so there are no real inconsistencies.

The Reactive Blocks interface is mostly consistent, with a few exceptions. *Edges*, unlike other elements, can not simply be added to the modeling canvas, but must be connected to other elements in both ends. There is no indication of the differing behavior in the UI, and the users must discover this on their own. All elements are found in the same menu, and the only thing separating them is a very subtle line. A similar line also separates some other elements in the same, though there is no indication of what the differences between these are.

### Cater to Universal Usability

The UI for the game should be easy to comprehend for any user within the target audience, which may involve detailed explanations for novice users, or useful shortcuts for the more experienced.

In the case of the introduction window, this is well-covered. Only the most important information is displayed by default, allowing "expert" users to start working on the task right away. Novice users have the option of going through the introduction slide show, viewing tips for the current level, or reading more detailed descriptions of specific concepts.

The Reactive Blocks UI is likely designed with more advanced users in mind. There are some useful UI tools and shortcuts, such as duplicating elements or sequences of elements, but the UI is lacking in features for novice users. Most of the possible actions are hidden behind a right-click menu, with no explanations or tooltips. The consequences of this are apparent from the usability test results in Sect. 5.4, where all test subjects struggled with learning the Reactive Blocks UI.

### Offer Informative Feedback

It is important that users understand the consequences of their actions within a given UI, and this is highly relevant also for games. Players will want to understand exactly how they can manipulate the game environment in order to overcome challenges and make progress.

The introduction window will react to any interaction by changing its content, making the results of any action clear to the user. This includes resizing of the map when it is clicked on, opening sub-windows when clicking buttons, or changing slide when interacting with the slide show controls.

For this particular principle, the game window provides some help. When players try to run their model, they will see the result of their logic as movement and actions in the game world. This mapping of created logic to visible result is less obvious within the Reactive Blocks environment, where the only option is to go through activity steps with the *analyze* function.

Being a modeling tool, the Reactive Blocks UI provides some visual feedback when users are modeling by simply displaying the elements on the canvas. Selecting these elements for manipulation is additionally visualized by highlighting. Building and running a model will generate a project with executable code, and the interface changes to display this project. Some feedback is however less intuitive, such as adding guards to edges. The visual representation of a guard with a *String* value is a small text box, which by default is too small to display the *String*, which is visually replaced by “[...]”. When creating *else* branches, the visual representation of the guard is often placed in a completely different place on the canvas, such as the upper left corner. Error messages when the user creates invalid models or does something unexpected, are also not always very informative, depending on the user’s familiarity with Reactive Blocks and Java.

### Design Dialogs to Yield Closure

When working with a UI involves sequences of actions, they should be organized in a way that makes it clear where the sequence begins and ends. When playing the Reactive Blocks game, most actions are individual, and there are few sequences.

In the introduction window, the sequences of actions present consist of no more than two steps. One example is resizing the map, where clicking once makes the map bigger, and clicking again shrinks the map to its original size. The UI is then returned to its original state, making it clear that the “sequence” has been completed.

The Reactive Blocks environment offers a few sequences of actions. One such sequence is building and running the model, which is a sequence of dialog windows the user must go through. After the final window, the environment is changed to display the generated project, indicating that the sequence was successful. There are however sequences of actions that do not yield closure in this way, such as adding edges to the model. When the user selects the *Edge* element, the mouse pointer is changed to an icon indicating edges can be added. Clicking anywhere within the canvas will attempt to add an edge to or from this point, blocking most other actions. After the desired edges have been added, the user must press the *Esc* button in order to clear this functionality and again allow other actions. The user may feel that the sequence has been completed after the edges have been added, when in fact further actions are required.

The game window essentially consists of one sequence of actions, with two alternate endings. The player starts the window, and then watches to see if the model was correct. If the model was incorrect, the player will hopefully discover this as a result of what happens in the game world, and simply close the window to continue working on the model. If the model was correct, the player is provided with a message that says “Level completed!”, indicating that nothing more will happen in the game window. In this case, the user will hopefully also close the window, and then proceed to the next level.

### Prevent Errors

More important than allowing users to recover from errors, is to prevent users from provoking such errors in the first place. And if errors do appear, they should be possible to recover from, and the UI should help the user with this recovery.

With the introduction window, provoking errors is virtually impossible. The only interaction available is to click on elements with predefined actions, and these have been thoroughly tested for bugs. The same principle applies to the game window, however bugs are more likely to be present here. Even in the case of errors due to bugs, these should only appear when the player's solution is wrong, causing unexpected behavior, and are thus easily recoverable. Recovery is simply done by closing the game window, fixing the model, rebuilding the system and running again. Of course, this assumes that the player is able to discover the problem after the error has occurred.

The Reactive Blocks is a completely different story when it comes to error prevention, likely because it is a relatively young software product, and with vast interaction possibilities. Fortunately, errors are generally recoverable, given that the user understands what is causing the error. For expert users, the error messages provided are generally sufficient, particularly for the cases where models are invalid. In this case, error messages are presented both on-the-fly on the modeling canvas, or summarized in a small window when the user tries to build the model. For novice users however, these error messages may be less informative, since they often assume knowledge of various more advanced concepts. Additionally, there are some error messages that offer no real information at all, such as the one given if the user tries to build a model that is already running.

### Permit Easy Reversal of Actions

Allowing actions to be undone provides a safety net for the user, encouraging experimentation and speeding up processes.

The introduction windows allows easy reversal of any possible action. If the map is zoomed, it can simply be reduced to its original size by clicking on it again. Information windows can be removed simply by closing them, and in the slide show, the user can easily navigate both forwards and backwards at any time.

Missing reversal of actions is likely the part about the Reactive Blocks UI the test subjects found most annoying (Sect. 5.4). There is no way of easily reversing any action done within the modeling canvas of Reactive Blocks. The user may work around this by for example using the clipboard to temporarily store elements that are to be deleted, or reload any of the "backups" Reactive Blocks periodically creates, but neither of these options are good enough for effective use.

### Support Internal Locus of Control

Users, particularly experienced ones, want to feel in control of the UI. This means designing the UI in a way that avoids surprises and repetitive, tedious sequences of actions.

With the possible actions being quite limited in the introduction and game windows, there is little room for going wrong here. The user can control the slide show, or open additional info windows, neither of which should give the user any surprises.

Reactive Blocks fares a little worse here. Always having to go through the right-click menus to add or change elements will likely feel tedious and repetitive to most users, and this was indeed the case for the test subjects in the usability test (Sect. 5.4). Additionally, having to always go through right-click menus could make it difficult for some user to find the information they need, or the right options to produce the result they want.

### Reduce Short-Term Memory Load

Most users have limited short-term memory, and UIs must be designed with this in mind. This includes, among other things, not having to remember information between different screens.

The introduction window for the Reactive Blocks game was intentionally made small with this principle in mind. A small window can be placed beside the Reactive Blocks/Eclipse window, meaning the player does not have to remember things like the task or the layout of the map. However, since this is a learning game, some remembering of information is expected. Players have to remember concepts from one level to the next (though it is possible to review previous introductions), and from one slide to the next in the introduction slide show. Additionally, if the player's model is incorrect, it might be necessary to remember what went wrong in order to fix it.

Reactive Blocks also does not require users to strain their short-term memory. The model is generally visible on the screen, and in the case of modularization, nested blocks serve as “black boxes” with (hopefully) well-defined interfaces. Some memory might be required in the case of operations and knowing what they do, but generally these will have sufficiently descriptive names. Most tasks are relatively simple, and do not require the users to remember long sequences of actions.

### A Summary of Problem Areas

From the analysis in the previous sections, we see a kind of duality in the adherence to many of the principles. The introduction and game windows present no obvious issues, while the Reactive Blocks modeling environment has quite a few, particularly with respect to consistency, novice user friendliness, reversal of actions, and feedback. The introduction window offers some help with the novice user friendliness part, but it may not be enough.

The many issues with the Reactive Blocks UI raises the question whether it is the right tool to use in an educational game. Players should not have to spend time learning UI quirks that are not necessarily relevant to what they are supposed to learn, or experience frustration with tasks that should be simple and intuitive. This essentially leaves us with two choices: either the Reactive Blocks UI must be improved, or the game should be based on a different modeling environment with a more user-friendly interface.

### 5.5.2 Discussion of the Test Results

The purpose of the usability test (Sect. 5.4) was to uncover in which parts of the game there were issues, and possibly how they could be improved. Both the observations made by the supervisor (the author) during the testing sessions and the feedback forms yielded results that uncovered quite a few issues.

**The Reactive Blocks UI** The most prominent issues in the game were related to usability in the Reactive Blocks modeling environment, which caused a lot of frustration. All three test subjects struggled with learning how to work efficiently with the UI, and frequently attempted actions that were not possible in the current state. Some of these usability issues are also analyzed in Sect. 5.5.1. In short, even with results from only 3 test subjects, we can conclude that the Reactive Blocks UI needs some work in order to be good enough for this type of game (this is also supported by the heuristic evaluation).

**Activity Final** The test subjects also uncovered some issues with the introductions that will require attention. First of all, it was not clear to any of the users how the *Activity Final* node would affect their programs. Since the game is kind of a special case, all 3 subjects made the “mistake” of adding this node at the end of their logic. Players likely need clearer information about what the *Activity Final* node does, and why the game is a special case with regard to this.

**Automatic Slide Show** Having an side show that starts automatically, and then changes slide after a fixed delay, might be a bad idea. People read and understand at different speeds, and will probably prefer to control the slide show themselves. All 3 test subjects had to either pause the slide show or go back one or more steps, because it progressed too fast.

**Incorrect Timing** In several cases, the test subjects were able to complete a level despite having set the timers to wrong values. This could be interpreted as a kind of "cheating by laziness", and should be considered a potential problem. If found to be disruptive of the learning experience, punishing wrong timing by not allowing the level to be completed could be an option.

**Decisions and Guards** All 3 test subjects had some trouble figuring out how to work with decisions and guards, some more than other. The main issues involved figuring out how and where to place the various elements, such as understanding that guards had to be set on the outgoing edges from decisions. Additionally, there was confusion about data types, and what the values of the guards should be. This is actually something that touches programming a little bit more than modeling, but it is a part of working with Reactive Blocks, and thus needs to be made clear to the player in some way.

**Tips and Help-on-demand** These resources were available for the purpose of helping players figure out how to proceed if they got stuck. All 3 test subjects ended up reviewing either the tips or the help-on-demand resources at some point, because they were unsure how to proceed. These resources were successful in the sense of being discoverable and available when the test subjects got stuck, but it is unclear whether they provided the information the test subjects were looking for.

**Expert User Limitations** Test subjects 2 and 3 both attempted to use the logical operator *OR* in guards. This is likely a result of the test subject being more advanced users in the realm of programming and software development, and thus found this to be a natural solution. This is not necessarily a problem, since more advanced users are more likely to quickly figure out what works and what does not.

**Incremental Problem Solving** While test subjects 1 and 2 preferred to complete their logic before getting visual feedback from the game, test subject 3 chose to implement part of the logic, and then get feedback on the correctness of the logic so far. This shows that the game allows multiple problem solving strategies, accommodating different types of players.

**Colors** All 3 test subjects mistook the red key/lock pair in level 5 to be orange (even the author has to agree that it does in fact look orange). Such inconsistencies inadvertently make the game more difficult, as players may spend time trying to fix mistakes they think are caused by something else.

**Difficulty Level** The answers from the feedback forms indicate that the difficulty of the levels varies. More specifically, levels 2 and 3 may be easier for most players to understand and complete than levels 1, 4, and 5. It is inevitable that for many players, some levels will be more difficult than others, since some concepts will be more difficult to understand, and this is not a problem, so long as the level does not become *too* difficult. The test subjects also experienced the difficulty levels to be different, but the fact that none of them rated any of the levels as *too difficult* or *too easy* is taken as a good sign.

**Information in Introductions** Apart from the previously mentioned issues with the introductions, there seem to be no significant pieces of information missing, and no information present that is redundant or irrelevant. All 3 test subjects actively used the introduction parts to complete the levels, and found these to be useful overall.

**Complete Understanding** From the results of the feedback forms, we can observe that two of the test subjects did not feel that they gained *complete* understanding of any of the concepts presented. Additionally, there were some concepts that not even the last test subject gained complete understanding of. This is certainly undesirable, as we want players to learn these concepts thoroughly, though within the realms of what is actually possible with just a short game.

**Enjoyment** Finally, we should observe that none of the test subjects answered that they did *not* enjoy the game. They collectively thought the game would be the preferred way of learning about UML activities and Reactive Blocks. The game appears to do a good job of providing learners with an immersive experience and a sense of achievement.

### 5.5.3 Fulfillment of Goals

Section 5.1.1 listed some goals that were set in advance for the game. With a prototype in place, it is prudent to review the goals, and see whether they have been met, or if any require more attention. It is also taken into consideration that we are only working with a *prototype* that is not yet complete.

The game certainly provides a sort of immersive learning experience for players, allowing them to learn concepts and solve problems within the game environment. The immersion is however slightly impaired by the game environment being distributed over different platforms, with the introductions, game world and modeling environment being separate entities. This is also unlikely to change as prototype development continues, since we are dependent on the Reactive Blocks environment. This goal is thus considered to be only partially fulfilled, with potential for improvement.

Players are given freedom in the game by being able to use any available elements to find their own solutions, and they can also choose how much information they want to absorb before starting on the tasks. There are hardly any limits to the players' freedom, however. Players are guided through a "learning path" where they are introduced to only a few concepts at a time, limiting the amount of information they need to process at a given time. However, the entire Reactive Blocks modeling environment is available to the players, even though many parts of it are less relevant, especially for the first levels. This may add some confusion and complexity to the learning experience, which may slow down progress for a some users. This goal is then also only partially fulfilled, as there could be additional advantages gained from providing a more limited modeling environment, and is also unlikely to change as development continues.

All test subjects (even in the informal test) were able to get started on solving level 1 relatively quick. At the same time, the test subjects had different impressions of the difficulty of various levels. The current levels are still likely too easy to really be challenging for more experienced users, but this can be taken into consideration as development continues. In addition to adding more concepts, we can design additional challenge levels for each concept that require deeper understanding.

A lot of the issues from the tutorial in Ch. 4 were unfortunately still present in the game. These were mainly issues related to Reactive Blocks, such as no real limits on freedom, insufficient visual mapping of UI elements, and exposure to quite a few complexities and processes that were less relevant to the players at their current experience level. Some additional efforts were made to help players find the correct UI elements in Reactive Blocks, but test subjects still had trouble with some parts, particularly decisions. It was also still not possible to really have everything in one place, but this was significantly improved from the tutorial, with the introduction window. The one goal that was really improved from the tutorial was to offer help-on-demand resources, which was done in the form of tips for each levels, and additional information pages for each concept. These were used to various degrees by the test subjects when they got stuck on something.

It is clear that there is still a long way to go before all these goals can be met, and we potentially have an *excellent* learning tool (as opposed to only a *good* one). The primary obstacle appears to be the Reactive Blocks environment, and the limitations it poses on implementing several of these goals. Currently, the game itself is actually developed completely independent of Reactive Blocks, which is simply used to control it though an API. This means that it would be trivial to change to a different modeling environment. The challenge in this is that there are few options available, and creating a UML activity modeling framework from scratch would require considerable effort.

### 5.5.4 Suggestions for Improvement

The current version of the prototype serves as a pretty good starting point for a UML activity learning game, but there are still some things that can, and should, be improved.

**Better Modeling Environment** The Reactive Blocks modeling environment seems to currently be the greatest obstacle for improving the game experience. One option is to modify the modeling environment in the context of the game, so that it provides users, particularly novices, with a smoother learning experience. This requires the Reactive Blocks development team to implement this, which is unlikely to happen any time soon, since they likely have other priorities. The other option is to use a different modeling environment, for example one created specifically for the purpose of being used with the game. The feasibility of the latter option is however unclear, as implementing a UML activity modeling framework is no trivial task.

**Refining of Introductions** The usability test uncovered some issues with the introduction part that should be improved, such as confusion about the *Activity Final* node. The introductions need to be refined with these specific issues in mind, without inadvertently introducing other issues. Additionally, introduction parts for additional levels within the game should not only be carefully designed, but thoroughly tested to verify that they provide the information required in a clear and simple way.

**More Levels** Since this is only a first prototype, it only covers part of the UML activity topic, with many concepts left untouched. The prototype should be extended with more levels that introduce players to all of these concepts, and possibly also with levels that add additional challenges allowing deeper understanding to some of the concepts.

Some level ideas to introduce additional concepts follow:

– **Concurrency and Forks:** Add more characters to the game, allowing the player to control several characters simultaneously.

– **Events:** Instead having to perfectly time cooperation between several game characters, make interaction operations send *events* that other characters can listen for.

– **Joins:** Similar to events, levels can be designed so that the player will benefit from synchronizing actions between multiple game characters with the join node.

- **Modularization:** Design the level so that the same actions are repeated in several places. The player can then create an “action module” in the form of a *local block* that can be reused in several places.

**Minor Fixes** In addition to these three major points, I compiled a long list of minor fixes following the testing sessions. These will not be listed here, but include things like bug fixes, not starting the slide show automatically, and changing the color of the red key/lock pair to orange.

Unfortunately, limitations on time prevent me from continuing the development of the prototype or conducting additional tests within the scope of this project. However, a faculty representative at ITEM, NTNU has expressed some interest in using the game as an introduction to the world of software for first-year students later this year, so I will eventually make some adjustments to prepare the game for this purpose, in case it becomes relevant.

Chapter 6

# Discussion and Conclusion

The work of this thesis has been focused around creating a better environment for learning about software modeling languages, more specifically focusing on UML activities within the context of the *Reactive Blocks* modeling tool. Inspired primarily by methods used to teach players how to play video games, two different learning environments were explored.

## 6.1 The Tutorial

The first teaching method explored was the *tutorial*, a concept widely used to teach various topics by offering interactive introductions, often with exercises. This approach to teaching is also used in many video games, where players are exposed to a tutorial that teaches them the basics of the game before they are allowed to explore the game on their own. The design, implementation, testing, and evaluation of the tutorial is covered in Ch. 4.

When creating the tutorial, various learning practices were taken into consideration. These were a mixture of good practices for video game tutorials and other types of tutorials, from both formal and informal sources. A summary of these is provided in Sect. 3.1.6.

A few iterations of going through UML activity concepts and figuring out their interdependencies, finding the least amount of information required to describe them accurately, and designing suitable exercises for understanding their semantics, resulted in a seven-step tutorial. Each step gave a short introduction of one or more concepts, and challenged learners with an exercise to solve using these concepts. The tutorial and its seven steps is described in more detail in Sect. 4.2.

Unfortunately, not all of these practices were feasible to implement within the given frame, particularly because of the constrains provided by the proprietary Reactive Blocks modeling environment. This meant some of the goals that had initially been set for the tutorial had to be forgone, and some adjustments had to be made to the tutorial design. The most prominent change was that all concept and task information had to be provided outside the modeling environment, in a supplementary document. Sections 4.4.1 and 4.4.2 provide a more detailed overview of the extent to which each goal was fulfilled by the tutorial design and implementation.

In order to uncover any major flaws or issues with the tutorial, and additionally get an impression of its teaching potential, a user test with a small number of test subjects was conducted. The testing session revealed a number of issues that required some additional consideration, and the main bulk of these were related, directly or indirectly, to the use of Reactive Blocks as the modeling environment. The session also provided some indications that the tutorial could indeed be a very good way of learning about UML activities, particularly with respect to providing motivation for learners. The tutorial may not be an adequate tool for learning to use Reactive Blocks however, as follow-up observations of the test subjects revealed that many of them struggled when working with Reactive Blocks, on account of the concepts that were not covered in the tutorial.

Although I was unable to implement and test several of the tutorial design principles, the tutorial served as a good starting point for establishing a teaching order for concepts in UML activities, and figuring out how much information users need in order to solve certain types of exercises related to each concept. These results laid the foundation for the *Reactive Blocks game*, which was the second teaching method explored.

## 6.2 The Game

The second teaching method explored in this thesis was to teach UML activity concepts inside a game. Learning games are becoming very popular in many levels of education because they offer a more immersive and interactive learning experience, among other things. Teaching UML activities with a game gave more potential for utilizing game strategies for introducing and teaching new concepts, such as giving learners a sense of *empowerment* and *identity.* The design, implementation, testing, and evaluation of the learning game is covered in Ch. 5.

Like with the tutorial, the learning game is based on a set of principles for designing this type of game, and these are summarized in Sect. 3.2.4. Additionally, the game would have a lot in common with a tutorial, so consideration was made with respect to the principles for good tutorials, and the lessons learned from the design and testing of the tutorial in Ch. 4.

The design and implementation process resulted in a first prototype version of a game with 5 levels, covering only part of the seven steps from the tutorial. The game was meant to have more levels, but we wanted to conduct a usability study in order to discover any serious issues as early as possible. The concept of the game is a two-dimensional world, where the player controls a character, performing various tasks. The character is controlled entirely through UML activity/Reactive Blocks models, which the player has to set up prior to launching the game level. Each level also has an introduction part, similar to the introductions given with the tutorial, but more extensive and with additional resources. The complete design and implementation of the game is covered in Sections 5.2 and 5.3.

While the game format allowed consideration and implementation of some additional "good" practices, there were still issues that were difficult to deal with because of the Reactive Blocks modeling environment. A usability test with three test subjects revealed that all three, who were novice users, had difficulties with correctly understanding and using a number of UI processes and elements in Reactive Blocks. This was despite the fact that some additional UI help had been added to the introduction part of the game, as a result of some of the same issues appearing in the tutorial test. Apart from those related to Reactive Blocks, the usability test revealed only minor UI issues that could be taken into consideration for the next version of the prototype.

In addition to usability-related results, the testing sessions also revealed that the test subjects gained a lower understanding of the concepts presented than desired. Some hints of potential misconceptions were also observed. This could mean that 5 levels is not sufficient to really learn these concepts, and that more practice is required. If this is the case, it can likely be solved by adding additional levels and challenges, for the players to practice their skills and knowledge more, and with different perspectives. In any case, more testing is needed to properly verify and understand the issue, preferably on a more complete version of the game.

On a brighter note, all three subjects stated that they enjoyed the game as a tool for learning about UML activities and Reactive Blocks, and would likely prefer it over other ways of learning. However, more than three opinions are likely needed in order to verify a real interest in this type of learning resource. It also remains to be seen whether a game like this will instill deeper learning in players, but research supports that this is the case for well-designed educational games (see Sect. 2.4).

The game is still in a relatively early stage, but the results so far are encouraging. Further development of the game will however not continue within the scope of this thesis, because of time constraints.

## 6.3 Concluding Remarks

The work of this thesis has been about exploring the use of game-related learning principles and strategies for teaching software modeling with UML activities. Two different teaching approaches were explored: the first approach simply incorporated many of the learning principles in a stepwise modeling tutorial, and the second approach involved designing a game with levels teaching the same steps, but in a more immersive environment.

Some minor tests of the two approaches were conducted, yielding mostly indicative but interesting results. Both approaches were found to be flawed in some ways, but they also showed great potential for teaching the concepts in question to the target audience, in an interesting and motivating way. It is however difficult to draw any conclusions about how the two approaches would measure up against other learning resources for the same concepts, without data from comparative studies.

It is also worth noting that with regard to established theoretical usability and learning principles, both approaches measure up quite well by *design*, despite the *implementations* lacking some important features. Whether implementing all of these features will actually be beneficial to the learning experience or not is however a little uncertain, as users have been known to behave unpredictably [AOL+12].

In the end, there is no reason to believe that learning strategies from games will *not* be appropriate also for learning modeling languages, as there have been no results indicating this. At the same time, there is not enough ground in this thesis alone to conclude that they *definitely will be* appropriate either. There are however several other research efforts endorsing this approach for teaching programming, which is a very closely related topic (see Sect. 3.2).

The conclusion to be drawn from this thesis is the following: using learning strategies from games to teach modeling languages is definitely worth exploring, but designing and implementing a good learning environment is not trivial, and requires a considerable amount of work and effort. Hopefully, the work and examples provided here can serve as a starting point for others.

## 6.4 Future Work

As the final part of this thesis, some suggestions for future work are included. There are several possible paths to take, and some are described in the following sections.

### 6.4.1 Complete Game

The most prominent path to take would be to continue development of the Reactive Blocks game, improving on the prototype and moving closer to a “finished” game. The current prototype is a little lacking in content, in addition to needing some minor fixing. There is also great potential for improving the modeling environment, particularly in order to improve usability. Section. 5.5.4 contains some more concrete suggestions on how to continue the development of the game.

### 6.4.2 Comparative Studies

The weakest part of this thesis is likely that I was unable to perform proper comparative studies, pitting the *Reactive Blocks game* against other similar learning resources in order to measure factors like player engagement and levels of understanding. This was not feasible because of constraints on time and resources. Comparative studies would however have been prudent in this context, and should thus be the second priority for any future work (after fixing the simpler issues and adding more content to the game).

More specifically, the game can be studied in comparison to the tutorial in Ch. 4, any of the existing tutorials mentioned in Sect. 4.1.1, or the lazier approach of just letting learners dig through documentation.

### 6.4.3 Other Modeling Languages

This thesis has been focused around teaching UML activities, more specifically in the context of Reactive Blocks, using teaching principles and strategies from games. It could also be interesting to use the same approach with other modeling languages, such as UML state machines, sequence diagrams, flowcharts, or even class diagrams. These will likely provide different challenges with respect to game and exercise design, but allow use of the same teaching principles.

# References

[ABH$^+$00] Christopher L. Aberson, Dale E. Berger, Michael R. Healy, Diana J. Kyle, and Victoria L. Romero. Evaluation of an Interactive Tutorial for Teaching the Central Limit Theorem. *Teaching of Psychology*, 27(4):289–291, 2000.

[Ada11] Ernest Adams. The Designer's Notebook: Eight Ways To Make a Bad Tutorial. Website/blog, 2011. http://www.gamasutra.com/view/feature/134774/the_designers_notebook_eight_.php?page=2, retrieved 2014-06-01.

[AM06] Eike F. Anderson and Leigh McLoughlin. Do Robots Dream of Virtual Sheep: Rediscovering the "Karel the Robot" Paradigm for the "Plug&Play Generation". In *Proceedings of GDTW2006*, pages 92–96, 2006.

[AOL$^+$12] Erik Andersen, Eleanor O'Rourke, Yun-En Liu, Rich Snider, Jeff Lowdermilk, David Truong, Seth Cooper, and Zoran Popovic. The Impact of Tutorials on Games of Varying Complexity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 59–68. ACM, 2012.

[Ass14] American National Education Association. Preparing 21st Century Students for a Global Society: An Educator's Guide to the "Four Cs". Official document, 2014. http://www.nea.org/assets/docs/A-Guide-to-Four-Cs.pdf, retrieved 2014-06-01.

[BCLO05] Lawrence Bergman, Vittorio Castelli, Tessa Lau, and Daniel Oblinger. DocWizards: A System for Authoring Follow-me Documentation Wizards. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, UIST '05, pages 191–200. ACM, 2005.

[Bec01] Byron Weber Becker. Teaching CS1 with Karel the Robot in Java. *SIGCSE Bull.*, 33(1):50–54, February 2001.

[BEH$^+$99] Chris Bowerman, Anders Eriksson, Mark Huckvale, Mike Rosner, Mark Tatham, and Maria Wolters. Tutorial Design for Web-based Teaching and Learning. In *MATISSE-ESCA/SOCRATES Workshop on Method and Tool Innovations for Speech Science Education*, 1999.

[Bræ04] Rolv Bræk. On Methodology Using the ITU-T Languages and UML. *Telektronikk*, 4:96–106, 2004.

[BSG+09] Elizabeth Bonawitz, Patrick Shafto, Hyowon Gweon, Isabel Chang, Sydney Katz, and Laura Schulz. The Double-edged Sword of Pedagogy: Modeling the Effect of Pedagogical Contexts on Preschoolers' Exploratory Play. In *Proceedings of the Thirty-first Cognitive Science Society*. Cognitive Science Society, 2009.

[BSRP96] Joseph Bergin, Mark Stehlik, Jim Roberts, and Richard E. Pattis. *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming*. John Wiley & Sons, Inc, 1996.

[CG87] Arthur W. Chickering and Zelda F. Gamson. Seven Principles for Good Practice in Undergraduate Education. *AAHE Bulletin*, pages 3–7, March 1987.

[DAJR11] Damien Djaouti, Julian Alvarez, Jean-Pierre Jessel, and Olivier Rampnoux. Origins of Serious Games. In *Serious Games and Edutainment Applications* [MOJ11], pages 25–43.

[dFL11] Sara de Freitas and Fotis Liarokapis. Serious Games: A New Paradigm for Education? In *Serious Games and Edutainment Applications* [MOJ11], pages 9–23.

[DSN+11] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. Gamification. Using Game-design Elements in Non-gaming Contexts. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 2425–2428. ACM, 2011.

[EFG13] Sarah Esper, Stephen R. Foster, and William G. Griswold. CodeSpells: Embodying the Metaphor of Wizardry for Programming. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 249–254. ACM, 2013.

[ESA14] ESA: Essential Facts About the Computer and Video Game Industry. Survey report, 2014. http://www.theesa.com/facts/pdfs/ESA_EF_2014.pdf, retrieved 2014-06-07.

[EW02] Rik Eshuis and Roel Wieringa. Verification Support for Workflow Design with UML Activity Graphs. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 166–176, 2002.

[EWF+14] Sarah Esper, Samantha R. Wood, Stephen R. Foster, Sorin Lerner, and William G. Griswold. Codespells: How to Design Quests to Teach Java Concepts. *J. Comput. Sci. Coll.*, 29(4):114–122, April 2014.

[GAL+09] Floraine Grabler, Maneesh Agrawala, Wilmot Li, Mira Dontcheva, and Takeo Igarashi. Generating Photo Manipulation Tutorials by Demonstration. *ACM Trans. Graph.*, 28(3):66:1–66:9, July 2009.

[GBCB07] G. Gweon, L. Bergman, V. Castelli, and R. K E Bellamy. Evaluating an Automated Tool to Assist Evolutionary Document Generation. In *Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on*, pages 243–248, Sept 2007.

[Gee05] James P. Gee. Learning by Design: Good Video Games as Learning Machines. *E-learning*, 2(1):5–16, 2005.

[Gee07] James P. Gee. *What Video Games Have to Teach Us About Learning and Literacy. Second Edition: Revised and Updated Edition.* Palgrave Macmillan, 2007.

[GF10] Tovi Grossman and George Fitzmaurice. ToolClips: An Investigation of Contextual Video Assistance for Functionality Understanding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1515–1524. ACM, 2010.

[GFA09] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. A Survey of Software Learnability: Metrics, Methodologies and Guidelines. In *CHI '09: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 649–658. ACM, 2009.

[HT07] Jeff Huang and Michael B. Twidale. Graphstract: Minimal Graphical Help for Computers. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 203–212. ACM, 2007.

[KBH09] Frank Alexander Kraemer, Rolv Bræk, and Peter Herrmann. Compositional Service Engineering with Arctis. *Telektronikk*, 105(2009.1), 2009.

[KP05] Caitlin Kelleher and Randy Pausch. Stencils-based Tutorials: Design and Evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pages 541–550. ACM, 2005.

[Kra07] Frank A. Kraemer. Arctis and Ramses: Tool Suites for Rapid Service Engineering. In *Proceedings of NIK 2007*. Tapir Akademisk Forlag, Nov 2007.

[KSG82] Kenneth L. Krause, Robert E. Sampsell, and Samuel L. Grier. Computer Science in the Air Force Academy Core Curriculum. *SIGCSE Bull.*, 14(1):144–146, February 1982.

[MOJ11] Minhua Ma, Andreas Oikonomou, and Lakhmi C. Jain. *Serious Games and Edutainment Applications.* Springer London, 2011.

[MTB00] Julie B. Morrison, Barbara Tversky, and Mireille Betrancourt. Animation: Does It Facilitate Learning? Technical report, AAAI, 2000. http://www.aaai.org/Papers/Symposia/Spring/2000/SS-00-04/SS00-04-009.pdf, retrieved 2014-04-28.

[Nie94] Jakob Nielsen. Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier. In Randolph G. Bias and Deborah J. Mayhew, editors, *Cost-Justifying Usability*, pages 245–272. Academic Press, Inc., 1994.

[Pat81] Richard E. Pattis. *Karel The Robot: A Gentle Introduction to the Art of Programming in Pascal.* John Wiley & Sons, Inc, 1981.

[Pau11] Christopher A. Paul. Optimizing Play: How Theorycraft Changes Gameplay and Design. *Game Studies*, 11(2), May 2011.

[RMMH$^+$09] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for All. *Commun. ACM*, 52(11):60–67, November 2009.

[Rob05] Eric Roberts. Karel the Robot Learns Java. Technical report, Stanford University, Department of Computer Science, 2005. http://people.reed.edu/~jerry/121/materials/karellearnsjava.pdf, retrieved 2014-04-28.

[Sel03] Bran Selic. The Pragmatics of Model-Driven Development. *IEEE Softw.*, 20(5):19–25, September 2003.

[SG05] David W. Shaffer and James P. Gee. Before Every Child Is Left Behind: How Epistemic Games Can Solve the Coming Crisis in Education. *WCER Working Paper*, (7), 2005. http://www.wcer.wisc.edu/publications/workingpapers/Working_Paper_No_2005_7.pdf, retrieved 2014-05-11.

[SP10] Ben Shneiderman and Catherine Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Pearson Higher Education, 5th edition, 2010.

[Stö05] Harald Störrle. Semantics and Verification of Data Flow in UML 2.0 Activities. *Electronic Notes in Theoretical Computer Science*, 127(4):35 – 52, 2005. Proceedings of the Workshop on Visual Languages and Formal Methods (VLFM 2004) Visual Languages and Formal Methods 2004.

[Tom82] Ivan Tomek. Josef, Programming for Everybody. *SIGCSE Bull.*, 14(1):188–192, February 1982.

[Unt90] Roland H. Untch. Teaching Programming Using the Karel the Robot Paradigm Realized with a Conventional Language. Technical report, Department of Computer Science, Clemson University, 1990.

# Appendix A
# UML Activities Tutorial

The following pages contain the complete document for the Reactive Blocks UML Activities tutorial described in Ch. 4.

# Introduction to Reactive Blocks

This set of exercises is meant as a second option to the standard Reactive Blocks tutorials. It is only the first step towards a more interactive and immersive tutorial, but these exercises will hopefully provide a better introduction to UML Activities and working with Reactive Blocks.

We also hope you will take the time to answer a few simple questions about the exercises when you are finished, so we can use your feedback to improve these exercises. You can fill out the paper form, or answer here. Additional comments are also welcome!

This assignment is made up of 7 smaller exercises, where each exercise introduces some new elements and concepts. For each exercise, you need to design a program that uses the new elements introduced (as well as some of the previous ones). The new concepts are briefly described in the exercises, but if you need more information, you can always check out the reference pages (http://reference.bitreactive.com/). Please let me know if you feel any information is incomplete, missing or even incorrect. You only need to think about modeling the system, all *operations* are already implemented.

In addition to the basic exercises, there are some challenge exercises that are a little more difficult to complete. These are completely optional, but may provide some useful additional insight into how you can utilize these concepts in an application. Some of these might even be fun to complete.
If you get stuck, there are some general tips on the last page.

## Preparation

1. Sign up for *Reactive Blocks* (http://blocks.bitreactive.com/login/signup.html)
    - Select the "Free plan"
    - In "About yourself", write "TTM4115".

2. Install the *Reactive Blocks SDK* (http://reference.bitreactive.com/install_arctis)

3. Open the *Reactive Blocks Perspective* (http://reference.bitreactive.com/doc/the_arctis_perspective)

4. Import the tutorial project
    - Join the *TTM4115 Tutorial Experiment* team (http://blocks.bitreactive.com/#/group/Gcm7ufo7fb67h0i8a)
    - Click *Import new libraries and examples* in the *Blocks* window
    - Select *no.ntnu.item.tutorials* and click *Finish*

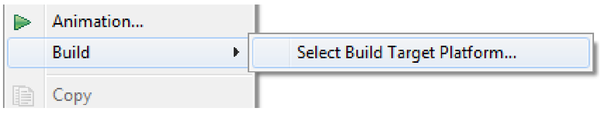
☑ no.ntnu.item.tutorials 1.0.0 (13) no.ntnu.item.tutorials

5. Use the provided blocks to complete the exercises

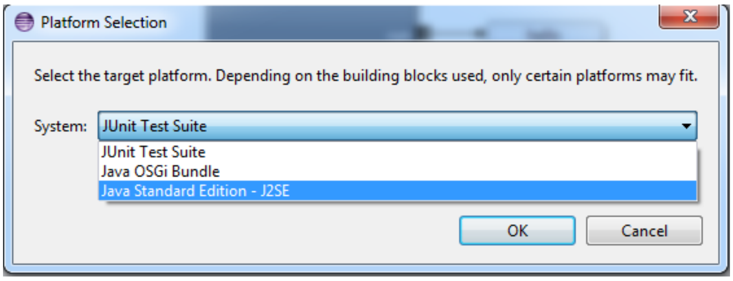
## Building and running the exercises

When you have completed modeling the exercise, you may want to run it in order to see if it does what it's supposed to. To build and run an exercise block, do the following steps:

1. Right click -> *Build* -> *Select Build Target Platform...*



2. Select *Java Standard Edition - J2SE*



Platform Selection

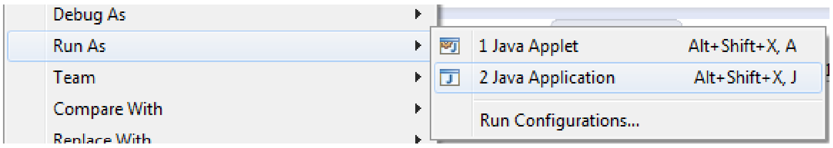
Select the target platform. Depending on the building blocks used, only certain platforms may fit.

System: JUnit Test Suite

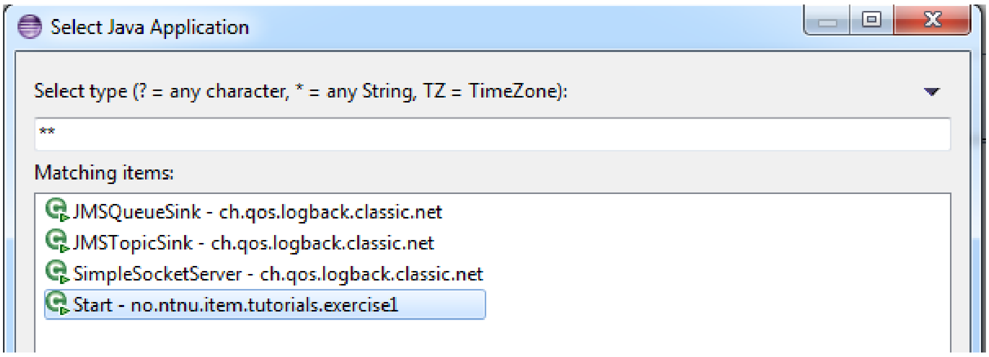
JUnit Test Suite
Java OSGi Bundle
Java Standard Edition - J2SE

3. Click *OK* without changing anything. This will generate a new executable project.

4. Right click the newly generated project (*no.ntnu.item.tutorials.exerciseX_exe*) in the *Package Explorer* and select *Run As* -> *Java Application*



5. Select *Start - no.ntnu.item.tutorials.exerciseX* and click *OK*



Select Java Application

Select type (? = any character, * = any String, TZ = TimeZone):

**

Matching items:

JMSQueueSink - ch.qos.logback.classic.net
JMSTopicSink - ch.qos.logback.classic.net
SimpleSocketServer - ch.qos.logback.classic.net
Start - no.ntnu.item.tutorials.exercise1

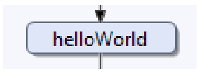
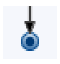
# Exercise 1 - Hello world!

**Task:** Create a “Hello world!” program.

**Introducing concepts: Tokens and control flow**
In *Reactive Blocks*, an application is executed by creating one or more *control tokens* that is passed through the various *blocks* and *nodes* in the program in a series of *steps*. Whenever a token passes through a node, some application logic is performed, and the token continues to the next node. For example, if this node is an *Operation*, the application will run the code associated with the operation.

When the application is first started, one token is created in each of the *Initial Nodes* present. The token then moves along the *Edge* connected to the *Initial Node* to the next element. Finally, when any token reaches an *Activity Final* node, all tokens stop flowing and the application is terminated.

New elements/nodes

| | |
|---|---|
| | **Initial Node:** generates a *token* when the program first starts. |
|  | **Operation:** an operation is performed when a *token* passes through it. |
|  | **Activity Final:** when a *token* reaches this node, the program terminates. |
| | **Edge:** connects nodes, so *tokens* may pass between them. |

Information about operations
- **helloWorld**: prints “Hello world!” to the console.

**Challenge exercise:** Create a program that prints one message, then a second message, and finally the first message again.

## Exercise 2 - Delays

**Task:** Create a program that displays a light, sets its color to red, then changes it from red to green after 5 seconds. Make sure you have time to see that the light has changed before the program terminates.

**Introducing concepts: Activity steps and stable positions**
As mentioned in exercise 1, applications are executed by passing tokens through nodes in a series of steps. Most of the time, a token will pass through multiple nodes in a *single* step, but some nodes, such as the *timer*, will put a token in a *stable position*, ending the current step. In the case of the *timer*, a new step will be initiated when the set time has expired, and the token will continue to the following nodes.

New elements/nodes

| | |
|---|---|
| ⧗ | **Timer:** keeps incoming tokens in a stable position for the duration of the *timer*, then starts a new activity step when it expires. |

Information about operations
- **displayLight**: displays a “light” image in the application window.
- **setLightRed**: sets the color of the light to red.
- **setLightGreen**: sets the color of the light to green.

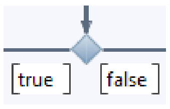
**Challenge exercise:** Create a program that simulates two traffic lights, one for cars and one for pedestrians.
- The car light should start at green, and the pedestrian light at red.
- After 10 seconds, change the car light to yellow.
- After 2 more seconds, change the car light to red.
- Wait another second, then change the pedestrian light to green.
- Give the pedestrians 10 seconds to walk, then change their light back to red.
- Wait 1 seconds, then change the car light to yellow.
- Finally, after 2 more seconds, change the car light to green.
- Let the lights shine for a short time before terminating the program.

# Exercise 3 - Choices

**Task:** Create a program that generates a random number between 0 and 200, and prints "Small!" if the number is smaller than 100, and "Big!" if the number is greater than 100.

**Introducing concepts: Object flow and Decisions**
In addition to providing a means of controlling the logic of the application, tokens may carry with them a single *object* (this can be any regular Java object). This data can be used by following decisions or operations to perform a part of the application logic. Most nodes, like timers and *decisions*, will pass the object on to the next node, but operations will not. Decisions will additionally check the value of the object, and select an outgoing edge depending on this value.

New elements/nodes:

| | |
|---|---|
|  | **Decision:** reads the incoming data object, and chooses an outgoing edge depending on their *guards* and the value of the data. |
| | **Object flow/edge:** *object flows* are like regular *edges* (flows), but tokens traveling along these edges also carry an object that may be used by the receiving node. |

Information about operations:

- **generateNumber**: generates a random number (int) between 0 and 200.
- **isNumberBig**: takes a number (int) as input, and returns *true* if the number is greater than 100, and *false* if not.
- **big**: prints "Big!" to the console.
- **small:** prints "Small!" to the console.

**Challenge exercise:** Create a program that generates a random number between (and including) 1 and 8, and then uses a binary search tree to find out which number it was. This exercise might require the use of variables and the Set/Get Variable Actions (read more at http://reference.bitreactive.com/doc/decisions_and_guards).

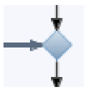
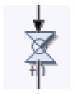
## Exercise 4 - Looping

**Task:** Create a program that generates a random number between 0 and 200 until that number is greater than 100 (big).

**Introducing concepts: Loops, flow breaks and Merge**
Sometimes, we want our application to repeat the same operation or sequence several times without adding many instances of the operation (imagine 100 repeats or more). In this case, we can create a loop in our model, inserting the token back at the start of the sequence with a *merge* node. The token will then pass through the same sequence again, repeating it.

However, a token may not pass through the same sequence more than once in a single *step*. In order to avoid this, we have to make sure the token reaches a *stable position* somewhere in the loop, letting it finish the current step, and then start a new one to run the sequence again. We previously saw that a timer will put tokens in a stable position, but in many cases we don't want to add a timer delay before starting a new step. In this case we may use a timer with *zero* delay, also known simply as a *flow break*.

New elements/nodes:

| | |
|---|---|
|  | **Merge:** merges two or more edges into a single edge, passing on tokens from any of the inputs to the output within the same step. |
|  | **Flow break:** a flow break is a timer with zero delay, used to divide a flow into more than one step without adding delay. |

Information about operations:

- **generateNumber**: generates a random number (int) between 0 and 200.
- **isNumberBig**: takes a number (int) as input, and returns *true* if the number is greater than 100, and *false* if not.

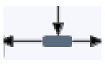
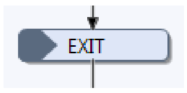
# Exercise 5 - User input and parallelism

**Task:** Create a program that changes the color of a light every time you press one button, and exits when you press another. It should be possible to change the light an arbitrary number of times before exiting (even zero).

**Introducing concepts: Forks and Events**
A timer is not always the appropriate solution for delaying execution of a step. Sometimes we don't know how long we should wait, as it may depend on something random or outside the program (e.g. user input). In these cases, we use the *Event Reception* node, which puts incoming tokens in a *stable position* until the appropriate *Event* is received by the program.

In many cases, we want our application to perform more than one task at the same time. We may want to perform some computations while waiting for user input, or maybe receive user input from more than one source. To make this possible, we need more than one token flowing in each step. There are essentially two ways of doing this: add more than one *Initial Node*, or place a *Fork* node where you want to add parallelism. The latter is typically more appropriate if you only want parallelism for certain steps.

New elements/nodes:

| | |
|---|---|
|  | **Fork:** when a token is received on this node, it creates a token on each o its outputs (two or more) within the same step. |
|  | **Event Reception:** Like *timers*, *Event Receptions* are stable positions marking the end of an activity step. However, instead of waiting for a timeout, a new activity step is initiated when the named *event* is received |

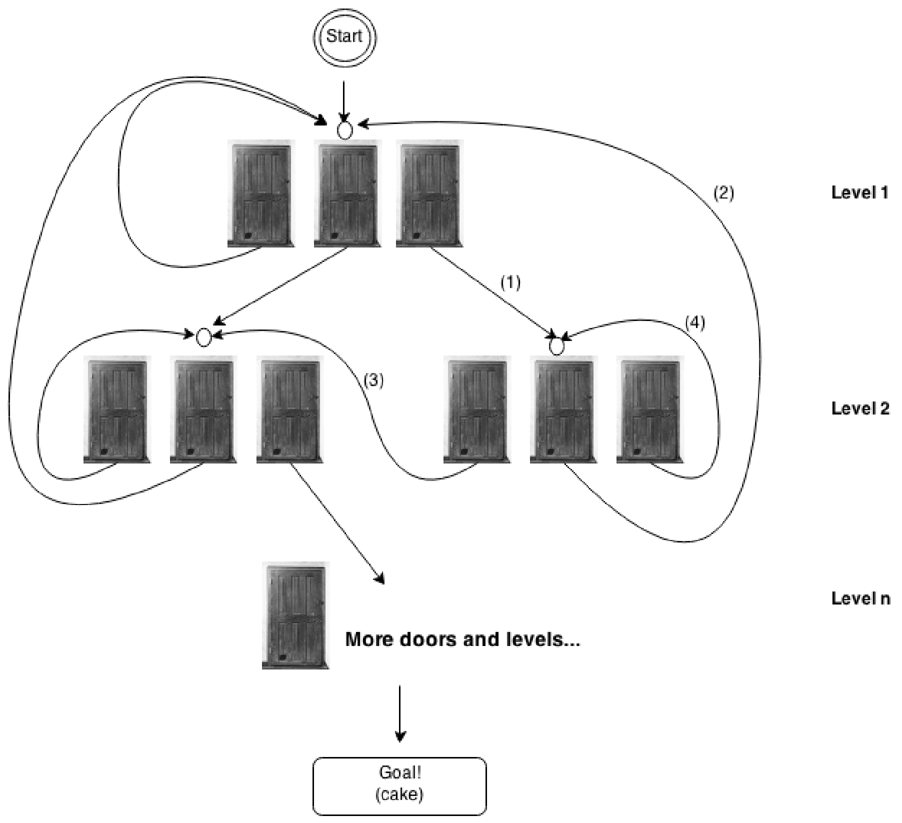
Information about operations:

- **showLight**: displays the light in an application window.
- **showButtons**: displays the "Change color" and "Exit" buttons in an application window.
- **changeColor**: changes the color of the light.

**Challenge exercise:** Implement *The Door Game* (see next page).

## The Door Game

The door game is a simple game where the purpose is to find the correct way through a set of doors to the goal, without knowing where each door leads. The player starts at level 1, and must then advance through the levels by opening doors until the final level (and the goal) is reached. At each step, the player is presented with 3 doors; blue, red and green. Depending on which door the player chooses, it will take him/her one step forward (1), one step back (2), one step to the side (3), or nowhere (4).



**Tips:**

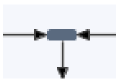
- It's entirely up to you how you create the door structure (i.e. which doors lead where).
- The events must be used with their corresponding operations (e.g. OPEN_DOOR1 and showDoors1).
- The *goBack()* etc… operations provide useful feedback info to the player.
- The events carry with them a *String* object, which is either "red", "green" or "blue".
- Code for 5 sets of doors is already implemented, but you can add more if you want. This will however require editing the Java code.

# Exercise 6 - Synchronizing

| |
|---|
| **Task:** Create a program that prints a message and then terminates when three buttons have been clicked (in any order). |

**Introducing concepts: Join**
In addition to timers and events, there is a third option to delaying steps. If you have several concurrent flows, with a sequence depending on all of them to be completed, you can use a *Join* node to synchronize the flows. The *Join* node will put incoming tokens in a *stable position* until a token has been received on *all* inputs, and then produce a *single* token on its output (within the same step as the last received token).

New elements/nodes:

| | |
|---|---|
|  | **Join:** The join node produces a *single* token on its output only when a token has been received on *all* its inputs. This synchronizes any incomin flows, forcing all of them to wait for the last token to arrive. |

Information about operations:
- **showButton1**: displays the first button.
- **showButton2**: displays the second button.
- **showButton3**: displays the third button.
- **printMessage**: prints a message to the console.

# Exercise 7 - Local blocks

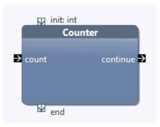
**Task:** Use the *Counter* block to make the *CoinFlipper* block "flip a coin" 10 times, and then return the result.

**Introducing concepts: Local Blocks**
When creating bigger applications, it becomes inconvenient and chaotic to have all the logic in a single block. Most of the time, many parts of the program can be separated into their own blocks, allowing for application design by composition of blocks. Separating functionality in this way is generally considered good design, and it is one of the core principles of Reactive Blocks.

So far we have been working with only the *System Block*, which is the top-level block of the application (it contains all other blocks). Inside the *System Block*, we can add an arbitrary number of *Local Blocks*, and these *Local Blocks* may themselves contain other *Local Blocks*. Each *Local Block* has a set of input and output pins, where tokens may enter or leave the block.

One important attribute of *Local Blocks* is the *External State Machine (ESM)*, which serves as a contract between the *Local Block* and its environment, deciding/limiting which signals may be sent or received in any defined *state*. If interaction with a *Local Block* violates its ESM, Reactive Blocks will produce an error. For more information about ESMs, see http://reference.bitreactive.com/doc/esm_basic.

New elements/nodes:

| | |
|---|---|
|  | **Local Block:** A *Local Block* is a reusable component providing part of th application logic. It has input/output pins for tokens, and an *External Stat Machine* defining how it may interact with its environment. |

Information about operations:

- **displayResult**: prints the input String to the console.

Information about blocks:

- **Counter**: The Counter block receives an *int* value when it is initialized, and counts down from this number towards zero every time a token is received on the *count* pin. When the counter reaches zero, the block terminates and releases a token on its *end* pin, otherwise a token will be released on its *continueCount* pin.
- **CoinFlipper**: Flips a coin every time it receives a token on its *flip* pin. When it receives a token on its *getResult* pin, it terminates and returns a token carrying the result on its *result* pin.

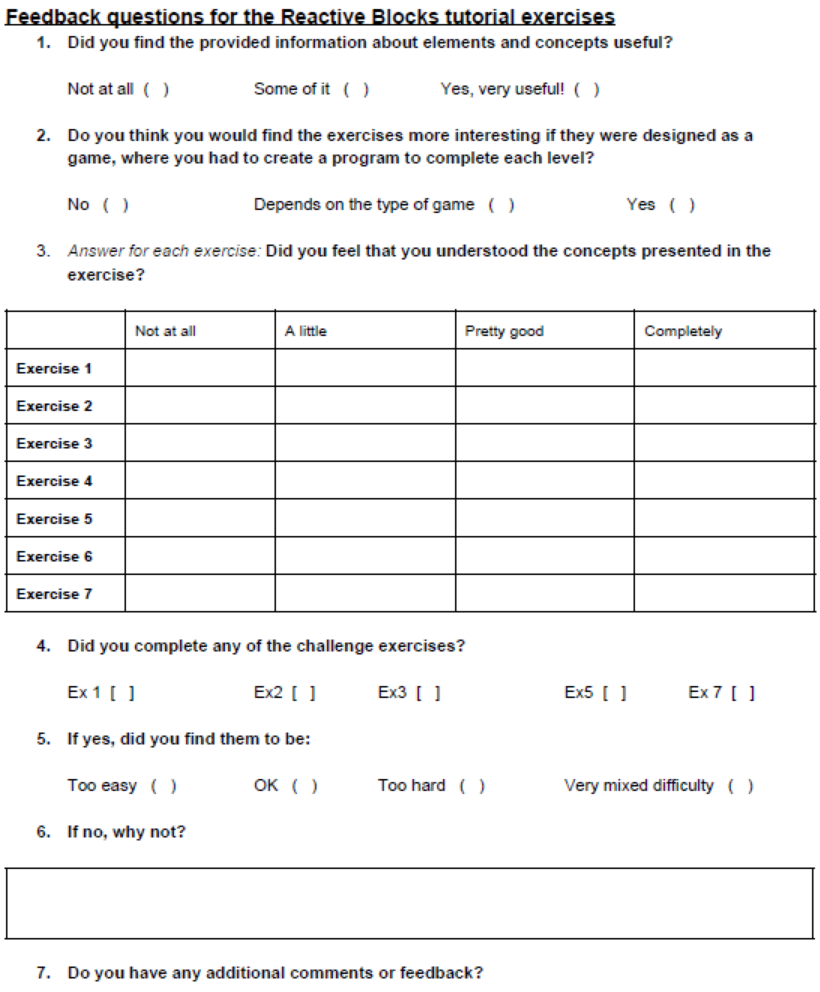
**Challenge exercise:** Taxi Order System (TTM4115 Lab Exercise 1).

## General tips

- Right click -> *Add…* to add more elements to a block.
- Right click on any element for more information and options.
- Double click on *operations* to see their source code.
- The same operation may be used more than once in a block.
- It is possible to have more than one *Initial Node*.
- The console window is at the bottom of your Eclipse window.
- Red color usually means something is wrong. You can view the error messages under *Analysis* at the bottom of your Eclipse window. Additionally, you can check if there are other problems with Right click -> *Analyze*
- If you want to see how tokens will flow in a program before running it, you can do so with Right click -> *Animation*
- There is often more than one way to solve the exercise!
- You can move the lines of edges around to make it look cleaner.
- In exercise 7, you may encounter the problem that the "Building block … is missing". If this is the case, simply right click the Local Block giving the error message, and select *Set Building Block…* Type the name of the block you want to link to (e.g. "Counter"), select the correct block and press *OK*. You can also simply delete the "missing" block, and drag-and-drop a new one from the *Building Blocks* window.
- If you get stuck, check the reference pages (http://reference.bitreactive.com/) or ask someone for help.
- Please take the time to answer the feedback questions! You can also provide feedback directly to me at oyvinric@stud.ntnu.no.

Appendix B

# Feedback Form for the Tutorial Testing Session



**Feedback questions for the Reactive Blocks tutorial exercises**

1. **Did you find the provided information about elements and concepts useful?**

   Not at all ( )  Some of it ( )  Yes, very useful! ( )

2. **Do you think you would find the exercises more interesting if they were designed as a game, where you had to create a program to complete each level?**

   No ( )  Depends on the type of game ( )  Yes ( )

3. *Answer for each exercise:* **Did you feel that you understood the concepts presented in the exercise?**

|  | Not at all | A little | Pretty good | Completely |
|---|---|---|---|---|
| **Exercise 1** |  |  |  |  |
| **Exercise 2** |  |  |  |  |
| **Exercise 3** |  |  |  |  |
| **Exercise 4** |  |  |  |  |
| **Exercise 5** |  |  |  |  |
| **Exercise 6** |  |  |  |  |
| **Exercise 7** |  |  |  |  |

4. **Did you complete any of the challenge exercises?**

   Ex 1 [ ]  Ex2 [ ]  Ex3 [ ]  Ex5 [ ]  Ex 7 [ ]

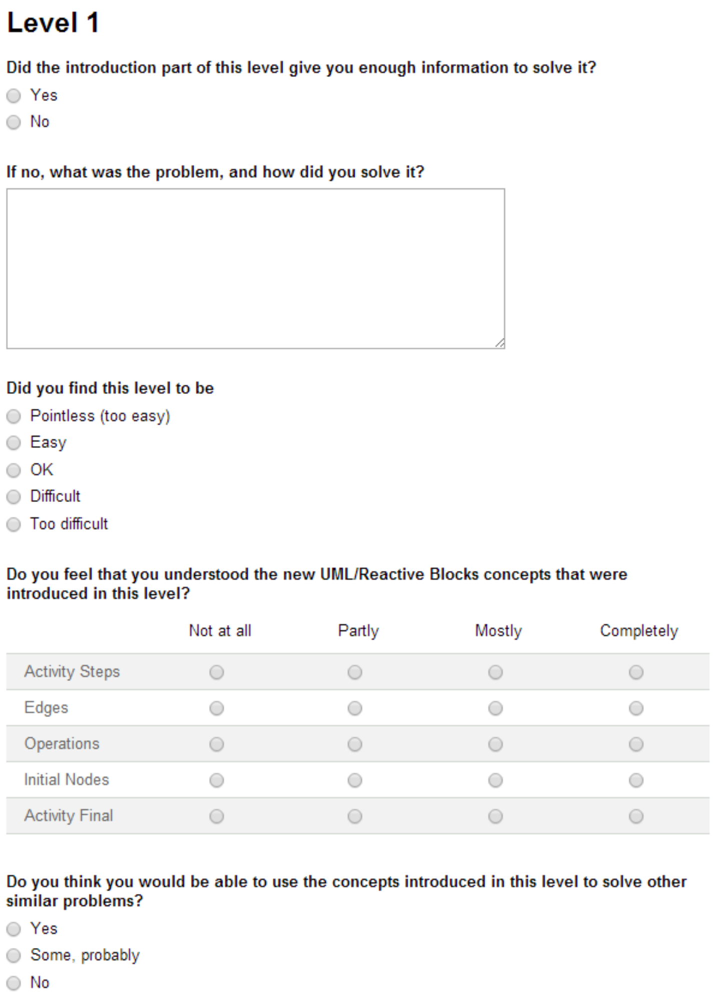
5. **If yes, did you find them to be:**

   Too easy ( )  OK ( )  Too hard ( )  Very mixed difficulty ( )

6. **If no, why not?**

7. **Do you have any additional comments or feedback?**

# Appendix C
# Feedback Form for the Reactive Blocks Game Usability Test



## Level 1

**Did the introduction part of this level give you enough information to solve it?**

○ Yes
○ No

**If no, what was the problem, and how did you solve it?**

**Did you find this level to be**

○ Pointless (too easy)
○ Easy
○ OK
○ Difficult
○ Too difficult

**Do you feel that you understood the new UML/Reactive Blocks concepts that were introduced in this level?**

| | Not at all | Partly | Mostly | Completely |
|---|---|---|---|---|
| Activity Steps | ○ | ○ | ○ | ○ |
| Edges | ○ | ○ | ○ | ○ |
| Operations | ○ | ○ | ○ | ○ |
| Initial Nodes | ○ | ○ | ○ | ○ |
| Activity Final | ○ | ○ | ○ | ○ |

**Do you think you would be able to use the concepts introduced in this level to solve other similar problems?**

○ Yes
○ Some, probably
○ No