NTNU
Norwegian University of
Science and Technology

# Educational Game Programming with JavaFX

2D Game Framework Integrating JavaFX with
Physics Engine

Kristian Juul Wannebo

Master of Science in Informatics
Submission date: November 2017
Supervisor: Hallvard Trætteberg, IDI

Norwegian University of Science and Technology
Department of Computer Science

## Abstract

Java@Kodeklubben Trondheim students have expressed wishes for making games, but programming physical movements and interactions proved too complex for the students. This research aimed to design a prototype of a game development framework that could be used as a tool for learning JavaFX, where movements and interactions were based on physics. The game development framework was aimed at both pre-university students and university students who are learning programming.

The research looked at existing frameworks for learning programming with games and learning theory as a basis for the design, then designed and implemented a prototype based on the derived information. The prototype was evaluated by creating a set of example games and performing a usability test with university students.

The research found that the students' learning process was affected positively by using the prototype in combination with real programming tools, and that by using real programming tools when learning, the knowledge learned was transferable to real world programming. The prototype also made it simple to make physical games.

## Sammendrag

I Java@Kodeklubben Trondheim har studenter utrykt ønsker for å lage spill, men programmering av bevegelser og interaksjoner har vist seg å være for vanskelig for studentene. Denne forskningen har hatt som mål å designe en prototype av et rammeverk for spillutvikling som kan brukes som et verktøy til å lære JavaFX, hvor bevegelser og interaksjoner er basert på fysikk. Spillrammeverket ble målrettet mot ungdomsskoleelever, videregåendeelever og universitetsstudenter som lærer programmering.

Forskningen så på eksisterende rammer som brukes under læring av programmering med bruk av spill og læringsteori som en basis for designet, så designet og implementerte en prototype basert på informasjonen som ble funnet. Prototypen ble evaluert ved lage et sett med eksempelspill og ved å gjennomføre en brukbarhetstest med universitetsstudenter.

Forskningen fant at studentenes læringsprosess ble positivt påvirket ved å bruke prototypen i sammenheng med programmeringsverktøy, og at ved å bruke disse verktøyene med læring, så ble kunnskapen som ble lært overførbar til programmering utenfor spill og læringssammenheng. Prototypen gjorde det også enkelt å lage fysiske spill.

## Preface

This research was done with the motivation to combine education, games and programming to create a learning environment for students in JavaFX. Having a background and interest in XML based GUIs made this research very relevant for me, and the idea to combine this interest with educational games where I could design and implement a prototype was the main reason I wanted to do this research. This research is also the final step in my Master of Science in Informatics. Doing this research while working has been one of the most challenging tasks I have ever done, but equally rewarding.

I want to thank my supervisor Hallvard Trætteberg who has gone above what I expected from a supervisor by supporting and guiding both me and the research from start to finish. I also want to thank my company Comsol AS and coworkers for making it possible for me to finish a Master of Science degree while working.

## Table of Contents

## Table of Figures

## Table of Tables

## Abbreviations

| | |
|---|---|
| **2D** | 2 Dimensional |
| **3D** | 3 Dimensional |
| **API** | Application Programming Interface |
| **CPU** | Central Processing Unit |
| **CS** | Computer Science |
| **CSS** | Cascading Style Sheets |
| **FXML** | JavaFX Extensible Markup Language |
| **GIF** | Graphics Interchange Format |
| **GPU** | Graphical Processing Unit |
| **GUI** | Graphical User Interface |
| **HTTP** | Hypertext Transfer Protocol |
| **IDE** | Integrated Development Environment |
| **JPEG** | Joint Photographic Experts Group |
| **NSD** | Norwegian Center for Research Data |
| **NTNU** | Norwegian University of Science and Technology |
| **PNG** | Portable Network Graphics |
| **RQ** | Research Question |
| **STEM** | Science, Technology, Engineering, and Mathematics |
| **SUS** | System Usability Scale |
| **UI** | User Interface |
| **WYSIWYG** | What You See Is What You Get |
| **XML** | Extensible Markup Language |

# 1. Introduction

This chapter introduces the motivation, problem and research questions for the research.

## Motivation

The motivation to pursue this research topic is to create an educational programming framework that integrated games into the learning process. This approach to learning programming has been researched a lot, and many different products have started from educational programming research, such as Logo, Alice, Scratch and Greenfoot. Combining programming, education and games is a tried and proved approach to learning, since many students who would like to learn programing also has an interest in games. Finding new ways to stimulate students who wants to learn programming by using games as an educational tool is an exciting approach to the learning process, and in the course for Java@Kodeklubben, participants who have an interest in STEM have been asking for game programming, showing that this is a field that interests both participants and researchers. Since this is a very broad field it could also be useful in a university setting, since learning Java is usually the first exposure students get to programming.

## Problem

Kodeklubben in Trondheim has held courses in JavaFX, SceneBuilder and FXML for pre-university students, where the students get to learn about making user interfaces and logos using SceneBuilder's drag and drop interface, and then modifying the graphical settings on the elements. During the courses, the students have wanted to create their own 2D games with JavaFX and SceneBuilder, but programming game objects with movement and interactions based on physical properties has proved to be too complex for the students.

This research will therefore look at the possibilities of making this part of the course easier for the participants, so they can learn programming by making their own games. This is also relevant for university students learning JavaFX as well, since these students are also exposed to JavaFX and might benefit from this research.

## Research Questions

The research will explore game frameworks and physics engines that can be used together with JavaFX and SceneBuilder in order to preserve the drag and drop capabilities of SceneBuilder, and then design and implement a working prototype that can be evaluated on pre-university students in Kodeklubben for JavaFX, or university students if pre-university students cannot be recruited.

RQ1: How should a framework for 2D game objects with movements and interaction based on physics with JavaFX be designed for educational purposes in a way that encourages students to learn programming?

RQ2: How and to what degree does the framework affect the learning process for the students?

# 2. Background and Related Work

In this chapter, the background for the research is described, Kodeklubben in Trondheim and the JavaFX assignments they have had, JavaFX at NTNU and an overview of JavaFX and SceneBuilder. The related work is also described by existing educational game frameworks and physics in 2D space.

## Kodeklubben

Code clubs have been very popular across the world, and aims at teaching children and teens programming, the clubs are usually run by volunteer work. Kodeklubben Trondheim is such a code club, and is connected to Lær Kidsa Koding which is a network of code clubs in Norway.

In Kodeklubben Trondheim, Java@Kodeklubben has held Java courses with the goal of providing an environment for learning programming while having fun, and builds on the previous courses Kodeklubben has had for learning Python. In the Java courses the students have learned to transition from Python to Java, making graphical user interfaces with JavaFX and tried to explore the framework.

### JavaFX assignments

After learning the transition to Java from Python, the students got to try JavaFX by doing GUI assignments. The assignments started by letting the student get familiar with FXML and SceneBuilder, by creating views in FXML without programming controller logic. After the students were familiar with FXML and SceneBuilder, they got to expand their views with controller logic by doing JavaFX controller programming. They learned how to connect JavaFX nodes to the controller, and program logic connected to those JavaFX nodes. The first assignments were Hangman, Calculator, Clock, Stick figures telling jokes, and a 2D platform game. Feedback from the students expressed that the assignments were too abstract and too hard when it came to the controller logic programming, and that the students would rather make games than regular graphical user interfaces. Since most of the students were also interested in STEM, they wished that the games could have more realistic movements and interactions.

## JavaFX at NTNU

At NTNU in the courses TDT4100 – Object-Oriented Programming and TDT4180 – Human-Computer Interaction have taught university students some JavaFX. TDT4100 is usually done in the 1. year and TDT4180 in the 2. year in the informatics and computer science study programs.

In TDT4100 the students have learned the basics of object-oriented programming through Java programming, but have been exposed to JavaFX through lectures showing JavaFX examples, but JavaFX has not been a teaching goal in this class.

TDT4180 has taught the students graphical user interface design, and JavaFX has been taught to the students for implementation of user interfaces by design principles, so the students who has taken this course has had some experience with JavaFX.

## JavaFX

JavaFX is a graphical user interface framework for rich and powerful desktop applications. JavaFX borrows ideas from HTTP and CSS in the way the user interfaces are set up and styled, and can be programmed in FXML in a similar way that HTML is structured, and supports CSS for styling graphical components. The way FXML is set up makes it easy for users to set up complex user interfaces.

JavaFX has a hardware accelerated graphics pipeline, so user interfaces and graphics are rendered by using the graphical processing unit in a machine(GPU), if a machine does not have a GPU, then software rendering is used which relies on the central processing unit(CPU) in a machine. This feature makes JavaFX have good performance for 2D games.

The Model View Controller design pattern is used for setting up a JavaFX application, and the View is connected to the Controller by injecting View components into the Controller by using the @FXML annotation on fields in the controller, and having the View referring to them via an fx:id xml attribute, thus making the Controller able to modify and read the content and behavior of the View as needed. It is also possible to connect listeners in the View to methods in the Controller by using the @FXML annotation on the method and referring to the methods by using a ‘#’ prefix in the View, this way the controller can react to events that happens in the View.

JavaFX also has support for effects, animation, and media. It is possible to define effects like blend, bloom, blur, and other effects on components. It is also possible to set up animations and transformations by defining transitions, timeline animations, and interpolators. The media functionality supports audio and video, so applications can have built in videos or play sounds when needed. It is also possible to load images for textures, JavaFX supports GIF, PNG JPEG image format file.

Geometric primitives are supported, and can be used for drawing shapes. The following geometric primitives are available in JavaFX; Arc, Circle, Cubic Curve, Ellipse, Line, Path, Polygon, Polyline, Quad Curve, Rectangle, and Text.

## SceneBuilder

SceneBuilder is a tool to design and build JavaFX user interfaces and graphical compositions by utilizing drag and drop functionality. You can drag graphical and UI components into a design surface to set up a view where what you see is what you get (WYSIWYG), and FXML is generated for the designed view. This makes it easy for a student who is learning FXML to get into user interface design.

SceneBuilder can only be used to set up the visual appearance of the desired user interfaces, if the interface needs business logic, it must be programmed in a Controller class associated with the FXML. The connection between FXML and Controller logic, can be set up using properties in SceneBuilder.

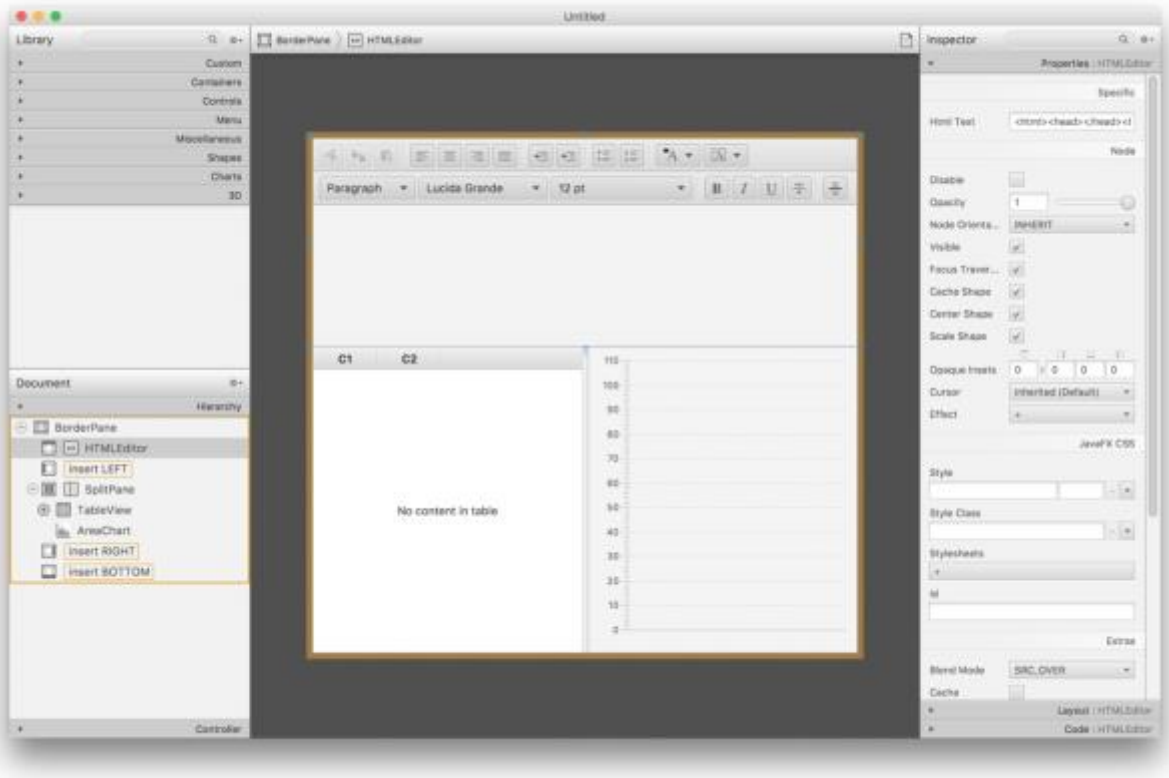


**Figure 2-1: SceneBuilder**

Figure 2-1: SceneBuilder shows the workspace in SceneBuilder, it is possible to browse UI components on the left side and drag them into the workspace, and then modify properties with the properties editor on the right.

## Game development as a learning tool for programming

Some research has been done on how well game development works in an initial learning environment for learning programming concepts. A central theme in these frameworks has been Constructivism and Constructionism.

Piaget's theory of constructivism (Ackermann 2001), describes a theory on how children think and do things at different levels of their development. Based on these views, Piaget describes an education model where a child learns when experiencing interaction with the world, people and objects. This idea stands in contrast to the normal education model where information is transferred

by written books, lectures and memorizing knowledge and then to be used and applied to appropriate fields.

Papert's constructionism (Ackermann 2001) builds on Piaget's theory of constructivism, and expands on it by saying that children especially learn when using their own ideas and building blocks, and that a good environment for such learning is through computers, where the children have many possibilities through media. With media on computers children can have the freedom to explore and learn natural science by creating simulations, media presentations or games.

## Existing frameworks

Multiple frameworks for educational programming that uses game programming as a learning tool has been made. The purpose behind the programming language Logo from the 80s was to make learning programming easier through visual games and simulations. Since then further progress has been made and the result of these is Alice, Greenfoot and Scratch, which are game development frameworks intended for learning.

### *Logo*

(Papert 1980) describes the ideas behind the programming language Logo. It is based on ideas from constructionism and is designed as a learning tool, but its goals are to enable people with different backgrounds and even young children, to program games, simulations and presentations via textual syntax.

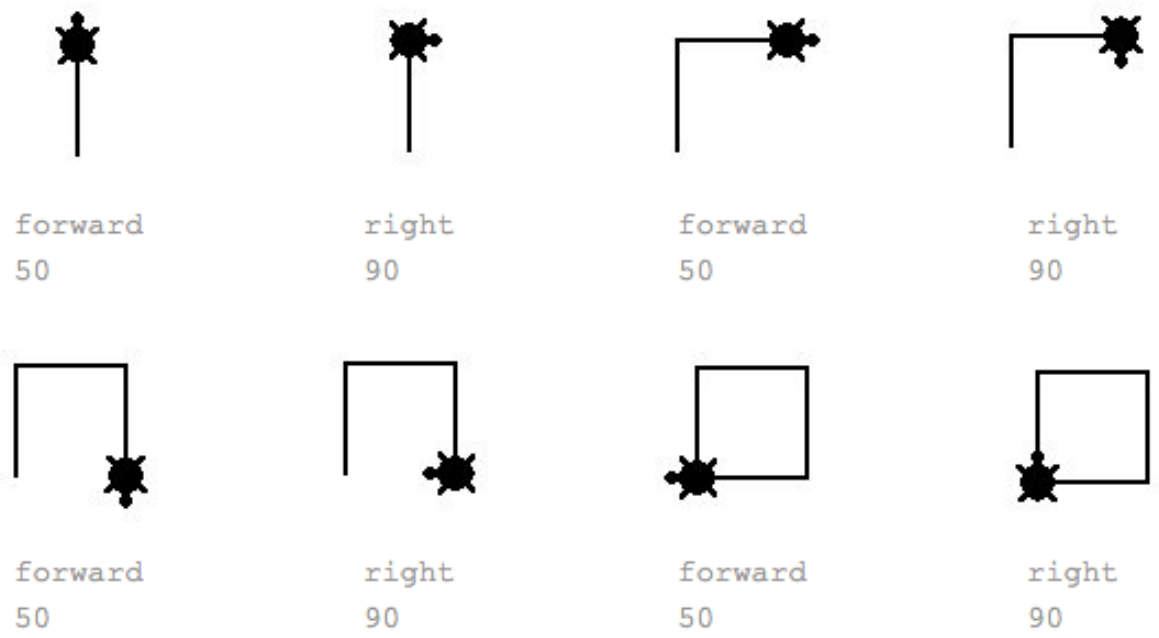


**Figure 2-2: Logo turtle command examples**

There are multiple versions of Logo, and the most popular ones has included a turtle that can be programmed to move and rotate in 2D space, viewing the turtle top down. The turtle can be programmed to draw a line where it moves, so it is possible to draw shapes by programming sequences of movement. This makes the users able to see and understand the effects of programming commands. Programming a turtle has been known as turtle graphics, and this concept has been transferred to modern programing languages like Java and C#.

Logo might not have been very successful at teaching children programming concepts. Empirical research showed that children who used Logo did not understand programming concepts better than children who did not use it (Pea 1987), and (Kurland, Pea et al. 1986) supported these findings but more programming languages were used in combination with Logo, so it is hard to draw a hard conclusion based on those data. The research points in (Pea 1987) are formulated in a way that suggests bias, for example; "the pedagogical fantasy that Logo can serve as a stand-alone center in classrooms for learning programming and thinking skills does not work".

(Resnick, Maloney et al. 2009) states that Logo never caught on in schools in the 80s because its programming syntax was too hard, and was used in a context where the activities were not interesting enough, and where nobody could provide guidance when users made a mistake.

### *Alice*

Alice is a teaching tool for object-oriented programming, and lets children create 3D scenes, games and stories via drag and drop interfaces. It was designed to teach object-oriented programming without using complicated semantics like a regular programming language like C++ does. Alice was designed based on many of the same principles as Logo, and Logo itself (Conway 1997).

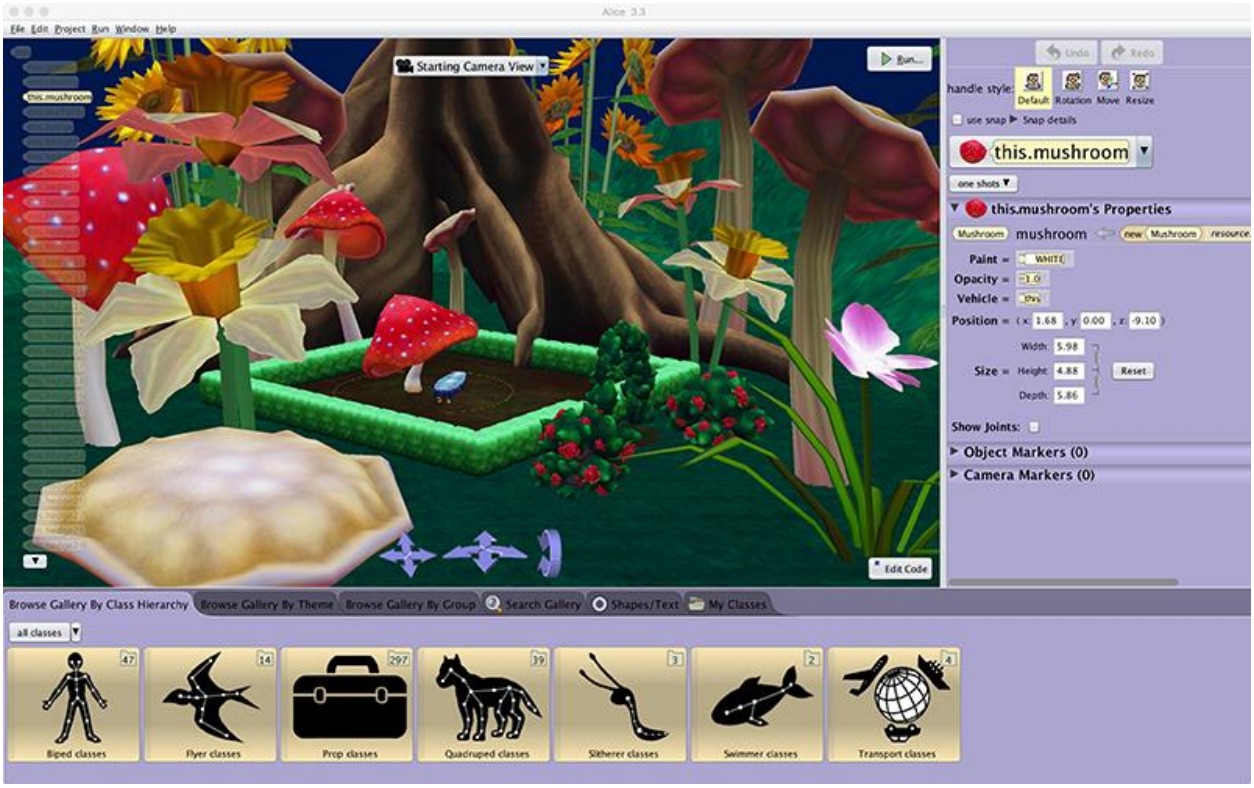


**Figure 2-3: Alice 3**

In Alice, you can drag and drop tiles that represents logical structures to create stories, animations and interaction, and when this logic is run, the user immediately sees how the logical structures affect the 3D objects in the world. This visual functionality is one of the key features that makes Alice user friendly, and can be compared to SceneBuilder's drag and drop functionality and property editor.

A recent study conducted found that using Alice in a CS1 course had significantly better test scores in Java for a CS2 course than students who had not previously used Alice (Dann, Cosgrove et al. 2012).

***Greenfoot***

Greenfoot is a learning tool for programming designed to empower the users to create games, simulations and graphical programs, it is designed around teaching object-oriented Java to pre-university students. Greenfoot is both textual and visual in regards that you can create actors and worlds visually, but the implementation beyond simple scaffolding must be implemented in Java. This is very relatable to SceneBuilder and controller logic. To help with Java programming, Greenfoot is an IDE with project management, syntax highlighting, auto completion and other common tools in IDEs.

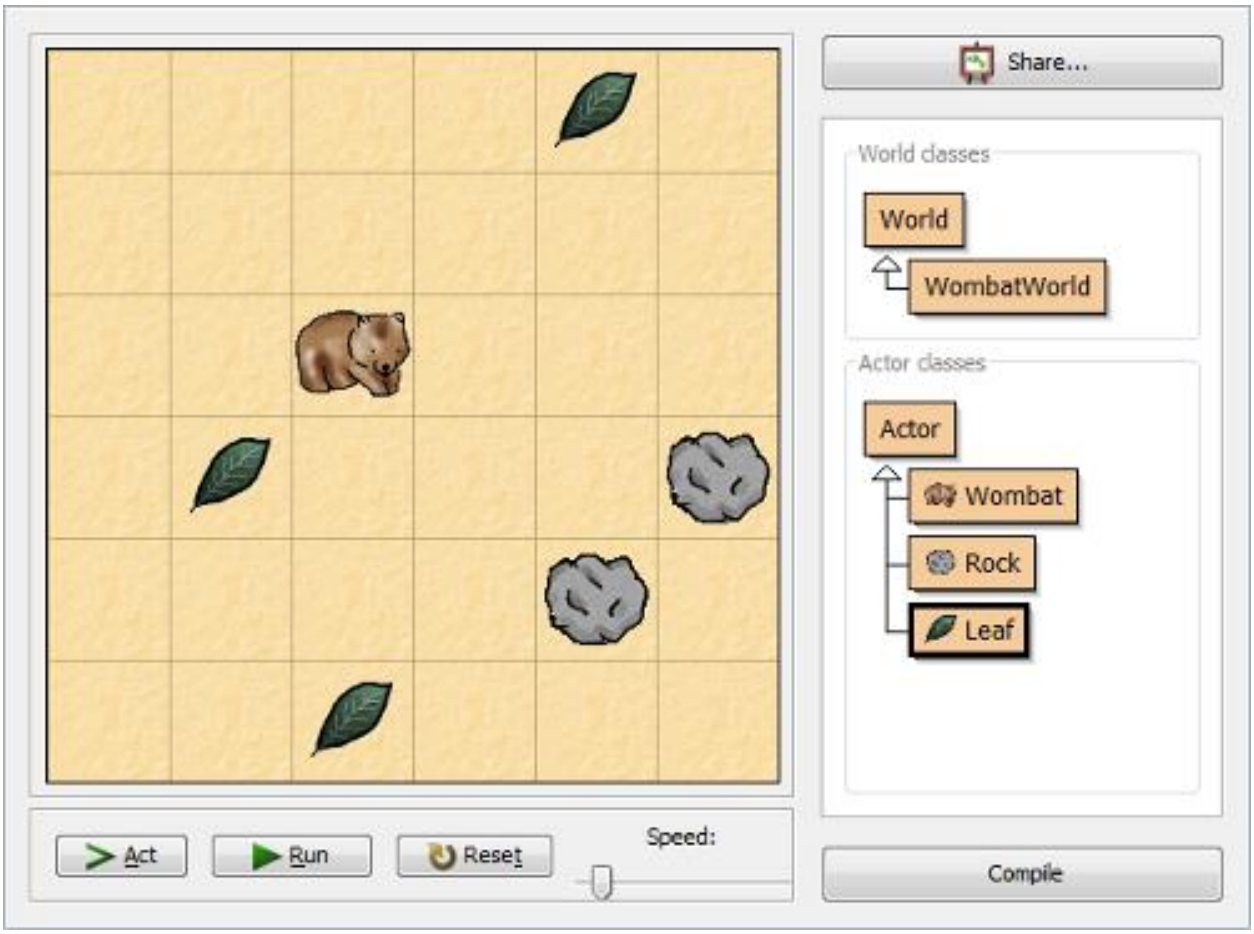


**Figure 2-4: Greenfoot**

Even though Greenfoot is textual, it has visual aids to help the user understand concepts like using brackets and parentheses to form blocks of code, it will frame the programming blocks created using color coding.

(Kölling 2010) describes Greenfoot, and reports that no big research studies have been performed with Greenfoot, so it is hard to conclude if Greenfoot works or not. But generally, feedback from teachers who has used Greenfoot in school have been positive, that the students are motivated and engaged when using Greenfoot.

### *Scratch*

Scratch was made to introduce programming to children and teens. It was inspired by existing learning tools such as Logo and Alice, but sought out to solve the problems of existing frameworks by making the floor for learning programming even lower (Resnick, Maloney et al. 2009).

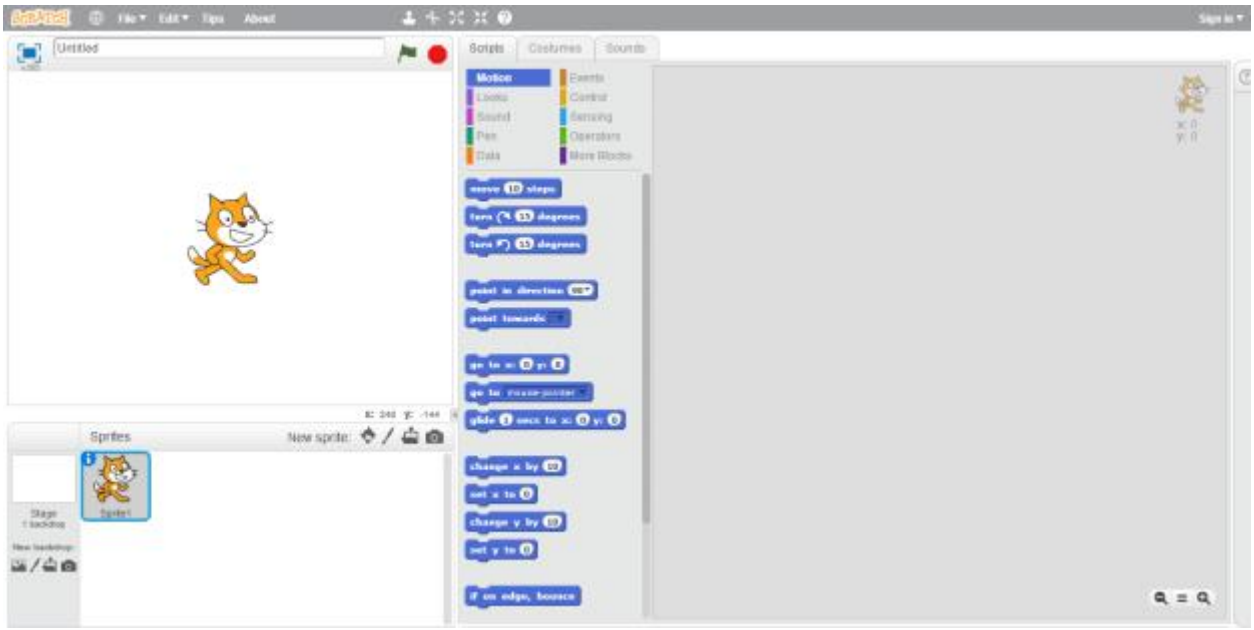


**Figure 2-5: Scratch layout**

Scratch is visually based where users use the Scratch interface to drag and drop logical blocks that fit together to form programming structures. Sprites can also be dragged and dropped into a scene, story or game. No textual programming is necessary in Scratch.

A key concept in scratch is utilizing social media and sharing to make the user more engaged and get feedback on their projects. Users can upload their project to the scratch website, and can also download other user's projects which can then be tested or remixed, which helps stimulate learning and engagement by seeing how other users form logical structures.

(Malan and Leitner 2007) did a study using Scratch where the goal of the study was to improve first time programmers' experiences, and then transition to Java. A couple of the participants had tried programming languages like Java and C++, but were disappointed when they found out that making a game with those languages was very complex. So, when Scratch could provide a game making platform, the participants were exited to program. 75% of the 25 participants reported that Scratch had a positive influence on their Java programing experience.

## Computational thinking

Computational thinking is a term from (Wing 2006), and is in Wings words "the thought processes involved in formulating problems and expressing its solution as transformations to information that an agent can effectively carry out". The term has become popular in pre-university education, and can be applied to more than programming. To keep this research relevant, results will be reported in terms used for computational thinking.

In relation to Scratch, (Brennan and Resnick 2012) describes the dimensions to computational thinking as computational concepts, computational practices and computational perspectives.

(Brennan and Resnick 2012) gives examples of the three dimensions as the following;

- Computational concepts: Sequences, loops, parallelism, events, conditionals, operators, and data.
- Computational practices: Being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing.
- Computational perspectives: Expressing, connecting, questioning.

## Physics in 2D space

Learning physics via programming and computational thinking is important because making games with realistic movements and interactions requires an understanding of 2D physics, it is also important since many of the students are interested in this field as well, and can learn physics by programming games or simple physics simulations.

Using physics simulations for learning and understanding physical concepts has been used in learning successfully, but not much research has been done on applying computational thinking to learn physics via programming.

## Physics simulations

Physics simulations as a tool for learning has been used in grade 6-12 science education where the users have gotten to play with parameters to see what effects a parameter has on the simulation result.

(Chang, Chen et al. 2008) did research on learning support for simulation-based learning systems and found that it was extremely important to let the users explore hypotheses on their own rather than provide step by step guidance to the hypothesis, even if it meant that the result made the hypothesis not accurate.

## Physics in 2D games

When describing physics in 2D games, we usually mean physical constraints we can put on game objects to make them behave more realistic in a 2D environment when moving and interacting with other objects.

There are lots of different physical constraints that can be applied to 2D objects, for example; *gravity*, *friction*, *weight*, and *joints*. Physical constraints are useful in 2D game development, since you usually want to create something that resembles the real world. Physical constraints can also be used to create game mechanics.

### *Geometric representation*

2D physics needs geometric representation of the objects they model. An ellipse in a game might just be represented as a rectangle in the geometric representation, depending on the behavioral needs of the ellipse. A performance limitation in geometric representation is that polygon vertices needs to be convex, and not concave, a way to overcome this issue is to define multiple convex shapes that together form a concave shape.

### *Physical representation*

A game object also has a physical representation to model properties like mass *density*, *friction*, and *restitution* which affects how external forces affects the game object when interacting with other objects.

#### *Collision handling*

Interaction between objects can be physical, and when two objects travel towards each other and eventually crash, they will apply forces on each other. This might send the objects in different directions and/or make the objects rotate, depending on how objects interacted with each other and how this interaction was simulated in the game.

#### *Joints*

A joint can be used to connect two physical objects together, a joint can be used to weld two objects together, set a point an object can rotate around at a fixed distance, or be used to set a max and/or min distance between two objects. A joint can also be used to create a suspension system for a car game where the wheel needs to rotate freely, but still be connected to the body of the car.

### Frameworks for 2D physics simulations

There are a couple of frameworks for 2D physics simulation in Java, and during the preparation for this project it was found that Jbox2D, Dyn4j and FXGL had a good maturity level and were most used in the Java 2D game community. All these frameworks are based on the same 2D physics algorithms, so their differences are minimal and mostly related to performance and design.

#### *Jbox2D*

Jbox2D is a Java implementation/port of Box2D. Box2D is a 2D rigid body physics engine based around manipulation with force, torque, and impulses. Box2D tries to model 2D physics in a realistic way, and uses units for mass, forces, torque and impulses in meters, kilograms and seconds. The calculations in Box2D are mostly approximations, because of the algorithms used are tuned for performance, and some of the problems does not have a solution, so approximations have to be used, but even though it is not perfect, it still offers realistic physics.

License: BSD 2-clause "Simplified" License, this license gives the user full freedom with the software as long as the BSD copyright notice is included.

#### *Dyn4j*

Dyn4j is a framework for 2D collision detection and rigid body physics, and was started after Box2D was ported to Java as JBox2D. Dyn4j is a pure Java implementation and is programmed in a way that most Java developers should be familiar with.

It is based on the same algorithms as Box2D, but with some differences. The project is Junit tested and has good documentation with many examples.

License: BSD 3-clause license, this license gives the user almost full freedom with the software as long as the BSD copyright and license notice is included.

## Game frameworks

The prototype being made is a game framework with focus on integrating physical properties into JavaFX nodes. A game framework or a game engine is a set of features adapted for making games, these features can be rendering, physics simulation, sound, level loading, and many other features that can be generalized for usage across many different games. In the planned prototype JavaFX will take care of the rendering and this research will focus on adding physics simulation, but the framework can also be expanded to support many other features that enhances the learning process.

### *FXGL*

FXGL is a JavaFX framework for games which has its own extended implementation of JBox2D included, this framework is similar to the one this assignment sets out to make a prototype of, but is designed around making the games programmatically and not with FXML.

License: MIT license, this license gives the users full freedom with the software.

# 3. Research Design

This chapter summarizes what the research will set out to do, and some of the elements

## This research

This research will use concepts from constructionism, existing learning tools and computational thinking to create a framework for making 2D games with physics based interaction. It will build on the existing knowledge in the field by making a specialized framework for students who are learning Java at universities, and students in Kodeklubben@Java who has had an introductory course for JavaFX, and knows how to use the basic functionality of a Java IDE and SceneBuilder.

Existing research uses their own learning tools to make an environment for learning, this research will try to use real programming tools to make a similar environment, so users of the framework can learn to use real tools and make the learning process using those tools more engaging. Using the framework should be simple, but since real programming tools and a real programming language is used, there is no roof for the user. The user can make their games as complicated as they want, but the framework will provide all the basic functions to create simple game logic for games like Pong, Breakout, Snake, Lunar Lander, Asteroids and Donkey Kong.

Since Kodeklubben students are interested in STEM, when creating games, a learning goal for the users should be to use computational thinking with mathematics and physics.

The prototype will focus on getting a physics engine to work with JavaFX, and how this should be designed in the best way possible for the intended use and users. The design and implementation will start from scratch, but might borrow ideas from an existing proof of concept prototype with limited functionality developed by Hallvard Trætteberg at NTNU.

## Design

The implementation of the framework in the research proposed will be designed around philosophy that Logo, Alice, Greenfoot and Scratch was designed around. The drag and drop functionality in JavaFX with SceneBuilder can be compared to dragging and dropping objects into the scene in Scratch and Alice, so even though no IDE will be developed in this research, it will still be comparable to existing learning environments. Preserving the functionality in SceneBuilder will be a high priority during the design decisions since this functionality can make the user experience

more visual than just writing text. For Java code and JavaFX functionality it should follow the code and functionality standards already established in the Java environment.

Going to participants from previous JavaFX courses in Kodeklubben for design requirements could be an option, but their needs have already been expressed through information in the assignment, and since they are not software developers, it will be hard to gain useful information on how to design the framework from them. It is better to design a framework they can use, get them to try it, and then do adjustments that fits their needs.

### Target users

Greenfoot is the most technically challenging learning environment because it uses textual syntax more than the other learning environments, and since this research will be based on an existing textual programming language syntax, it can be comparable to Greenfoot. Greenfoot was designed to be used by 14+ year old children, and when Greenfoot was designed, experiments showed that 10-year olds could not handle the syntax, but some 13-year-old children could, so generally after 14 years of age, children could handle the syntax (Utting, Cooper et al. 2010). This research will therefore use the same age restrictions as Greenfoot.

## Evaluation

Evaluation of the produced prototype will need to get Kodeklubben students to test the prototype. This testing can be performed in the JavaFX course for Kodeklubben, but if this is not possible another approach could be to get university students to test the prototype. The university students would need to be carefully selected, since their base knowledge of JavaFX must be similar to those who has participated in Java@Kodeklubben.

### Scaffolding

When assessing the implemented framework in accordance to computational thinking, (Lye and Koh 2014) suggests using scaffolding so the users have some basic computational concepts and practices to build on. They suggest using small sample examples that users can remix and expand, so when performing the evaluation of the implemented framework, examples will be made with computational thinking in mind.

### Data collected

Data collected during this research needs to conform to the guidelines provided by NTNU and Norwegian Center for Research Data (NSD). If personal data is expected to be collected during the evaluation of the prototype, then this project will need to be reported to NSD. It is not expected that collecting and storing personal data is required, so the initial decision is that this project will not be reported to NSD.

### Ethics

This research tries to study how pre-university students learn with the implemented framework, so interaction with the users is required, but disclosing that users are being monitored might make them behave differently. For this research, it is not expected that the participants will behave differently, so this research will be completely open towards the participants on the data collected and how the data is collected.

# 4. Methodology

The methodology used for the research.

## Strategy

To answer the research questions a Design and Creation research strategy (March and Smith 1995) was used. It uses an iterative software development process where the research activities; *build*, *evaluate*, *theorize* and *justify* are done for each of the research outputs; *constructs*, *model*, *method* and *instantiation*.

Other research strategies for design science research was also considered (Vaishnavi and Kuechler 2004) and (Peffers, Tuunanen et al. 2007), but these strategies were focused on the whole research process. Because of the scope of this research and the need for a strategy that targets artifact creation, (March and Smith 1995) was chosen as the strategy to answer the research questions, but elements from (Vaishnavi and Kuechler 2004) was also used for the research process as well.

## Design and Creation research

The Design and Creation research strategy offers research outputs as a contribution to knowledge. It defines a way to demonstrate analysis, explanation, argument, justification and critical evaluation. The iterative process of design and creation has the five following steps (Vaishnavi and Kuechler 2004);

1. Awareness: Finding areas for further research by searching the literature
2. Suggestion: Offering a tentative idea of how the research problem can be solved
3. Development: Implementation of the IT artifact, which can be the implementation of a new IT system
4. Evaluation: Evaluates the implemented IT artifact
5. Conclusion: Results of the evaluation are discussed and new knowledge is justified.

Since the main focus for this research is to create a computer based product, in order to justify that this research is not just using normal design and creation strategies, but design and creation research strategies, the research will focus on the academic qualities when creating new knowledge, it will also focus on risky parts of the artifact design to create new knowledge, which might have been avoided in a regular design and creation strategy.

### Research outputs

The design and creation research strategy creates different artifacts as output for the research. The types of output consist of the following research outputs (March and Smith 1995);

- Constructs: Domain description of the problem and solution, can be formalized as entities, attributes relationships, identifiers, constraints.
- Models: A description of how things are, and can be entity relations.
- Methods: A set of steps on how to perform a task, and can describe algorithms.
- Instantiations: Realization of the resulting artifact by instantiating them in a testing or actual target environment.

### Research activities for the outputs

The research activities from (March and Smith 1995) describe the iterative research activities that are done for each of the research outputs. The research activities are the following;

- Build: The creation of the research outputs.
- Evaluate: Evaluation of the built output.
- Theorize: Making theories about what happened during the evaluation.
- Justify: Making an argument for why these theories are relevant.

## Evaluation of the IT product

The IT system will need to be evaluated by certain criteria, this can be *functionality*, *completeness*, *consistency*, *accuracy*, *performance*, *reliability*, *usability*, *accessibility*, *aesthetics* or other criteria. The criteria chosen will depend on what kind of questions need to be answered for the research and why the product was developed in the first place, so the main criteria when testing the prototype should be *usability*. The evaluation can uncover issues or problems with the design process or with the implemented artifact, so further improvements can be made. It is therefore beneficial to do multiple iterations of the design and creation research strategy so those issues and problems can be further investigated.

### Data generation for evaluation

Interviews, observations, questionnaires and documents are options for data generation during the evaluation. Since there will be no way to compare potential results found with older data,

experiments cannot be used in this research. Interviews might be the best way to discover how the created framework works, and is understood by the user. This information can then be analyzed and improvements to the framework can be made.

The type of interview can be a one on one interview or a group interview if enough participants agree to test the prototype. A semi-structured interview is ideal for this research and will be used with questions prepared, but will let the interviewee trail off and ask unplanned questions if the interviewee brings up new issues around the prototype and usage, this will uncover research data for the research questions and issues around the prototype itself (Oates 2005).

The questions prepared must be related to the evaluation criteria and an assignment that is done by the students before the interviews.

The assignment given to the students will focus on testing the usability and functionality of the prototype. Scaffolding will be used so the students have something to start working on, and some directions will be given so the students can progress on their own. No step by step guides should be given so the students can freely explore how the framework works by themselves, since this might enhance the learning process according to (Chang, Chen et al. 2008).

Observing students while they work is possible. The observer will take part in helping students while they work on assignments, so the observation will be a participant observation where the observer will take notes on how the students are working with the 2D game framework. In order to keep the research ethical, the observations should be overt, so the participants will be informed about the note taking.

## Data analysis

Since qualitative data is generated, the data will be analyzed by identifying themes and relationships in the produced data. Since the data created is done by interviews, where the interviewees are not familiar with the research field literature, the data will be coded by open coding or selective coding if appropriate (Oates 2005).

# 5. Results

This chapter contains the prototype design, prototype implementation, user testing of the prototype, analysis and a discussion of the results.

## Prototype design

The design of the prototype is based on a requirement analysis, considering the needs of the target users, how the user should interact with the API, and how the API should be integrated with JavaFX. This process was able to tackle the design challenges and produce outputs for the Design and Creation research strategy in the iterative stages of the prototype design and implementation.

### Requirement analysis

The process of finding design requirements for the prototype API, was to look at the features of Alice, Greenfoot and Scratch, and then look at the game assignments Kodeklubben had previously done in their Scratch course and assignments in Java@Kodeklubben. The games analyzed were Pong, Breakout, Snake, Lunar Lander, Asteroids and Donkey Kong, but also keeping in mind general puzzle and platform games, since these were also a good fit for the prototype.

These methods of gathering design data should provide guidance for the design. Asking the target users directly was considered, but their needs have already been expressed, and they are not capable of giving relevant input in API design choices.

### Analysis of Kodeklubben games

The games in Kodeklubben for Scratch are typical games that Kodeklubben participants would like to make. Some of them are very simple, but games like Pong and Breakout gives a good starting point for deriving minimal requirements and functionality the framework needs.

***Pong***

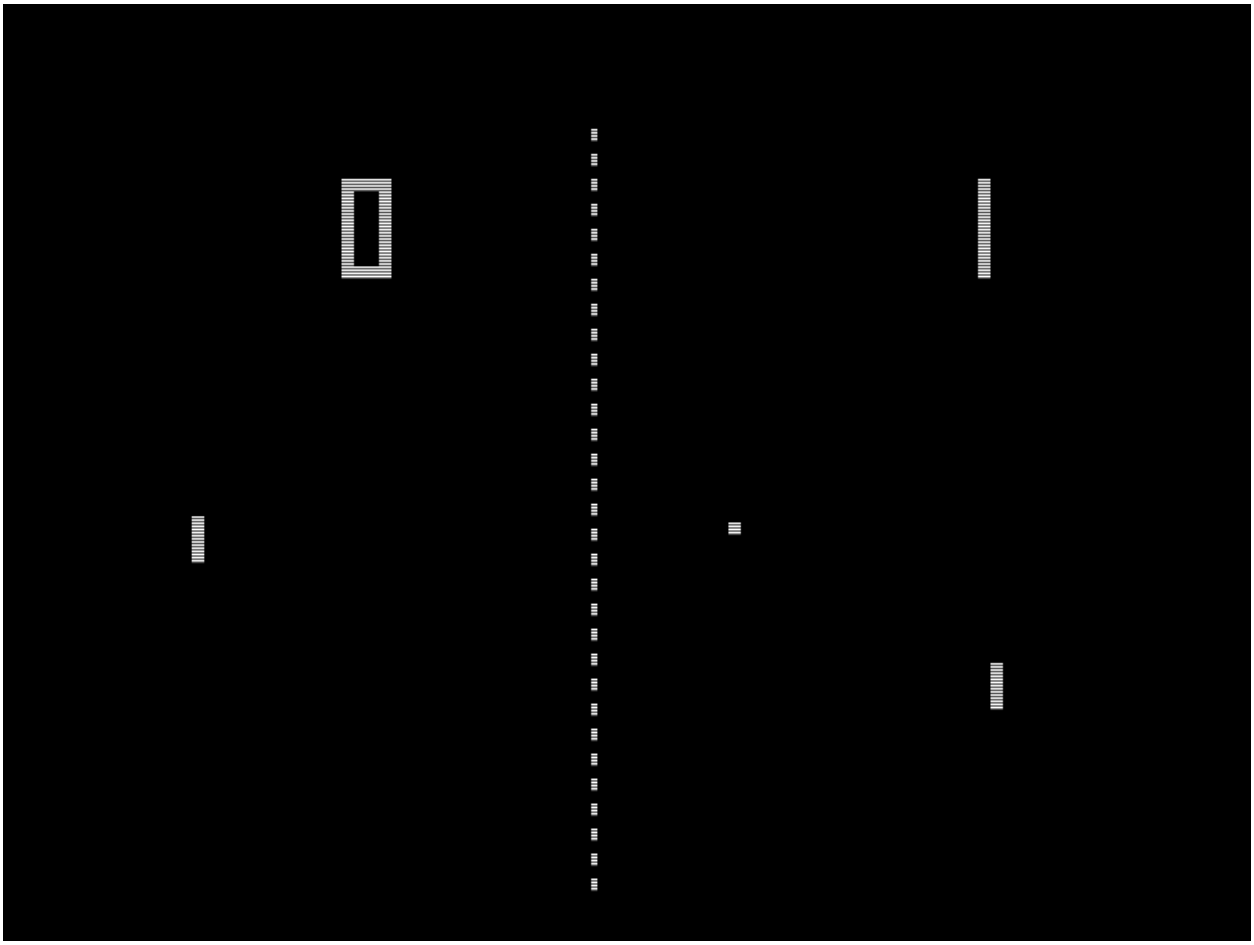

**Figure 5-1: Pong**

A two-player game where each player moves a paddle on the y-axis to block a moving ball, and the objective of the game is to block the moving ball from entering the player's area behind the paddle.

The ball in the game can be simulated with physics, and reacts when it collides with a wall in a way that its entire energy is conserved. When it collides with a wall or a paddle it bounces off the surface in a way that looks realistic. The paddles cannot be simulated with physics, and they are not affected by the ball, but can block the ball. The walls are non-moveable objects that will block the ball.

In order to build a game like pong, the framework must support the following;

1. An object that can move in all directions and react to its environment.

2. Movable objects that are not affected by other objects, but can collide and interact with other objects.
3. Non-moveable objects that are not affected by other types of objects in Pong, but can apply forces to other objects.
4. Collision detection between all types of objects.
5. Simple shapes like circle and rectangle.
6. Moving an object with an input device.
7. Setting the speed and direction of travel for a fully simulated object.

***Breakout***

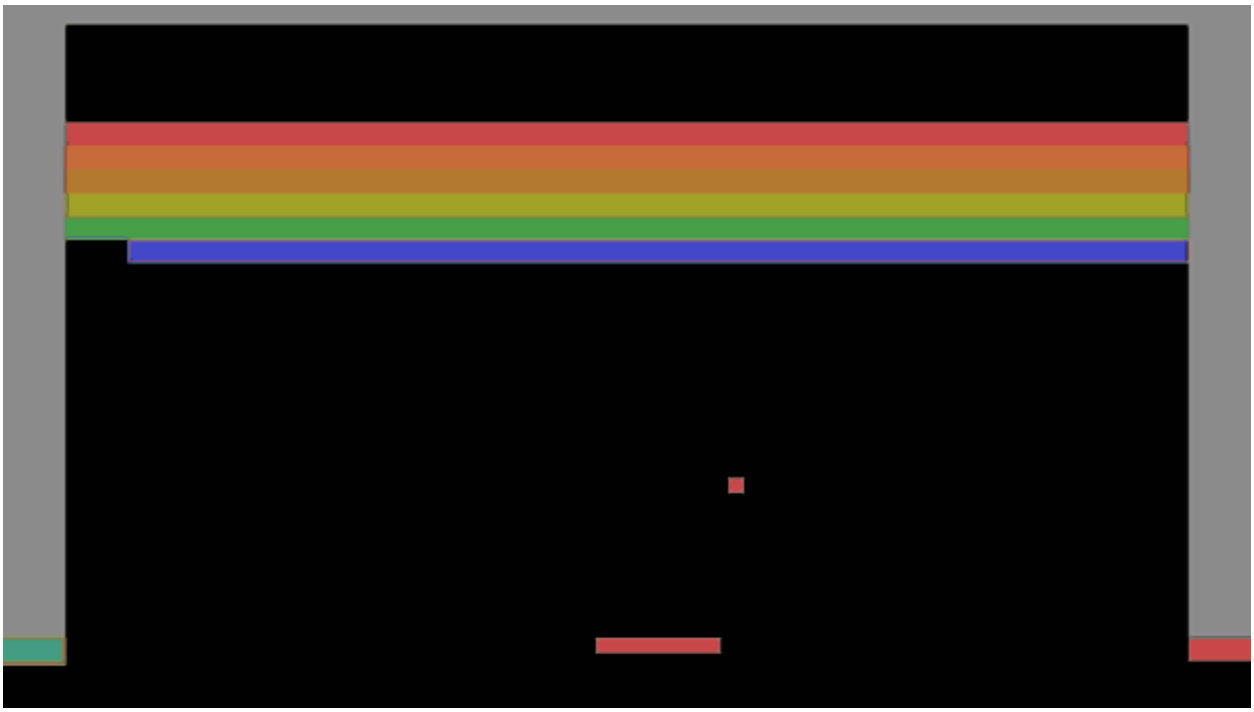

**Figure 5-2: Breakout**

One player game where the player moves a paddle on the x-axis to hit a moving ball. The objective of the game is to make the ball hit rows of bricks that takes damage or are destroyed on collision, when all blocks are gone, the player advances to the next level. The ball will increase speed at certain checkpoints and intervals in the game. The paddle will resize to half the size when the top brick row is breached. The player has 3 turns per level, if the ball moves behind the paddle and hits the wall, the player loses a turn and the ball is reset.

The paddle is similar to the paddle in pong, except that the paddle moves on the x-plane instead of the y-plane. The bricks in the game are non-moveable objects that are removed when hit. The game also has walls around the game world to keep the ball inside the game area.

In order to build a game like breakout, the framework must be extended with the following;

1. Removing an object from the world.
2. Loading the next level when current level is completed.
3. Resize an object in width.
4. Increase the speed in the direction an object is already traveling.

### *Snake*

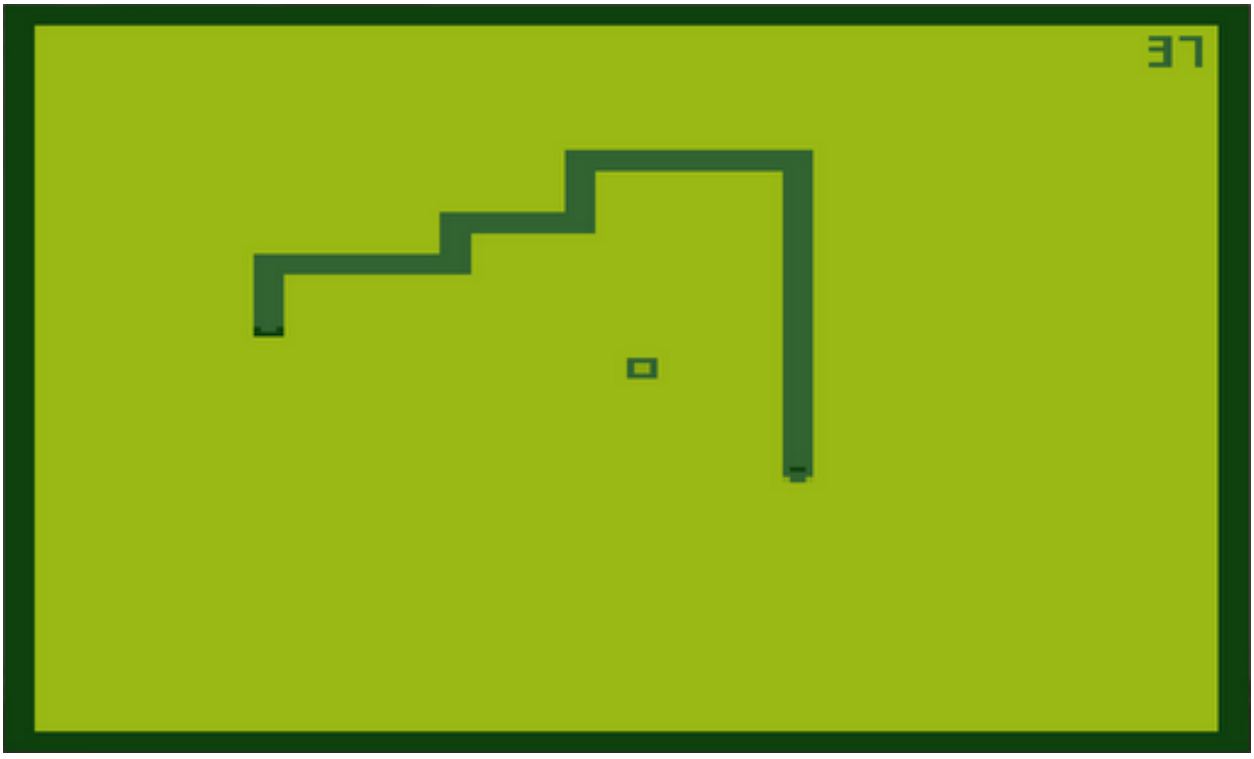


**Figure 5-3: Snake**

A player manipulates the direction of a snake that moves one block per time unit. When the snake moves forward one block, its tail follows, to keep the length. When the snake eats a fruit that is on the game board, the tail waits a bit, to make the snake longer. If the snake collides with the edges of the map or itself, the game ends. This game can be simulated in a physics world, but would require setting up constraints and joints that makes the snake behave in the expected way.

The classic version of Snake is not a physics-based game, and it would make the game complicated to make when using physics to model the snake movements, so no requirements are derived from this game.

### *Lunar Lander*

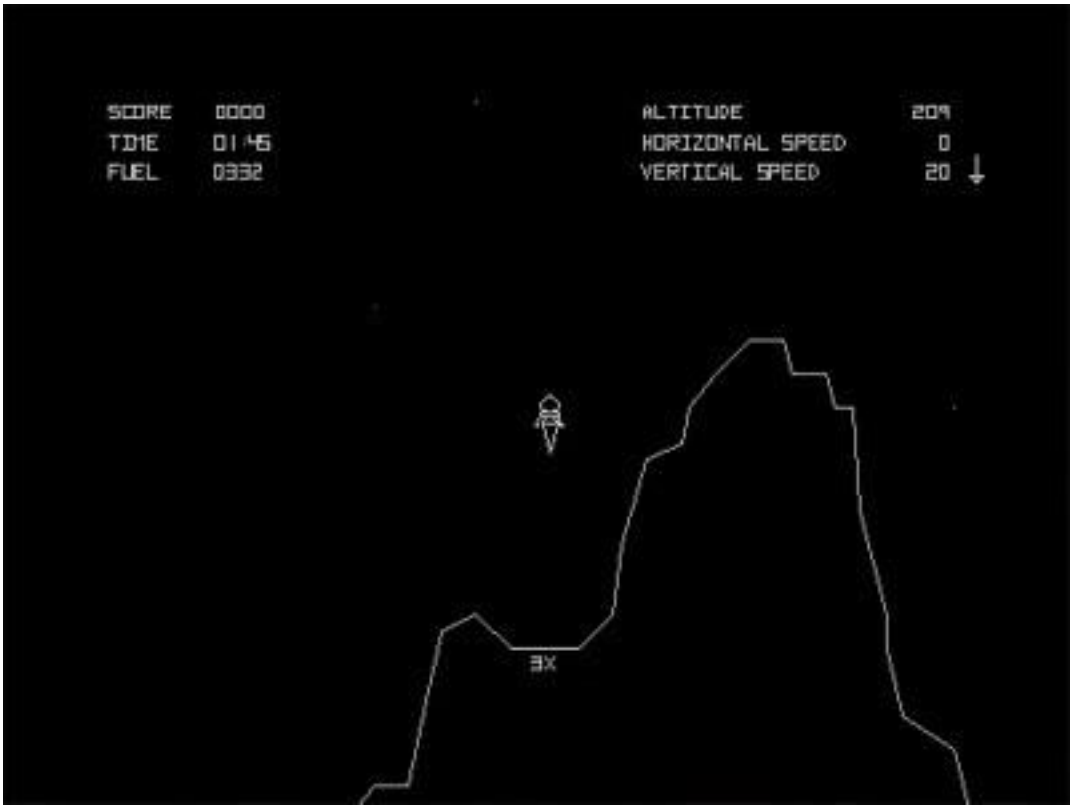


**Figure 5-4: Lunar Lander**

A player controls a lunar lander by using thrusters to manipulate rotation and apply force in the direction the moon lander is pointing. The objective is to land the moon lander on a landing pad. If the moon lander hits the surrounding landscape, the game ends. If the lunar lander lands on the landing pad, the next level is loaded.

In order to make Lunar Lander with the framework, it must be extended with the following features;

1. Lunar gravity.
2. Composition of shapes for the moon lander.
3. Apply forces at a shape on a position within the shape to simulate thrusters.

***Asteroids***

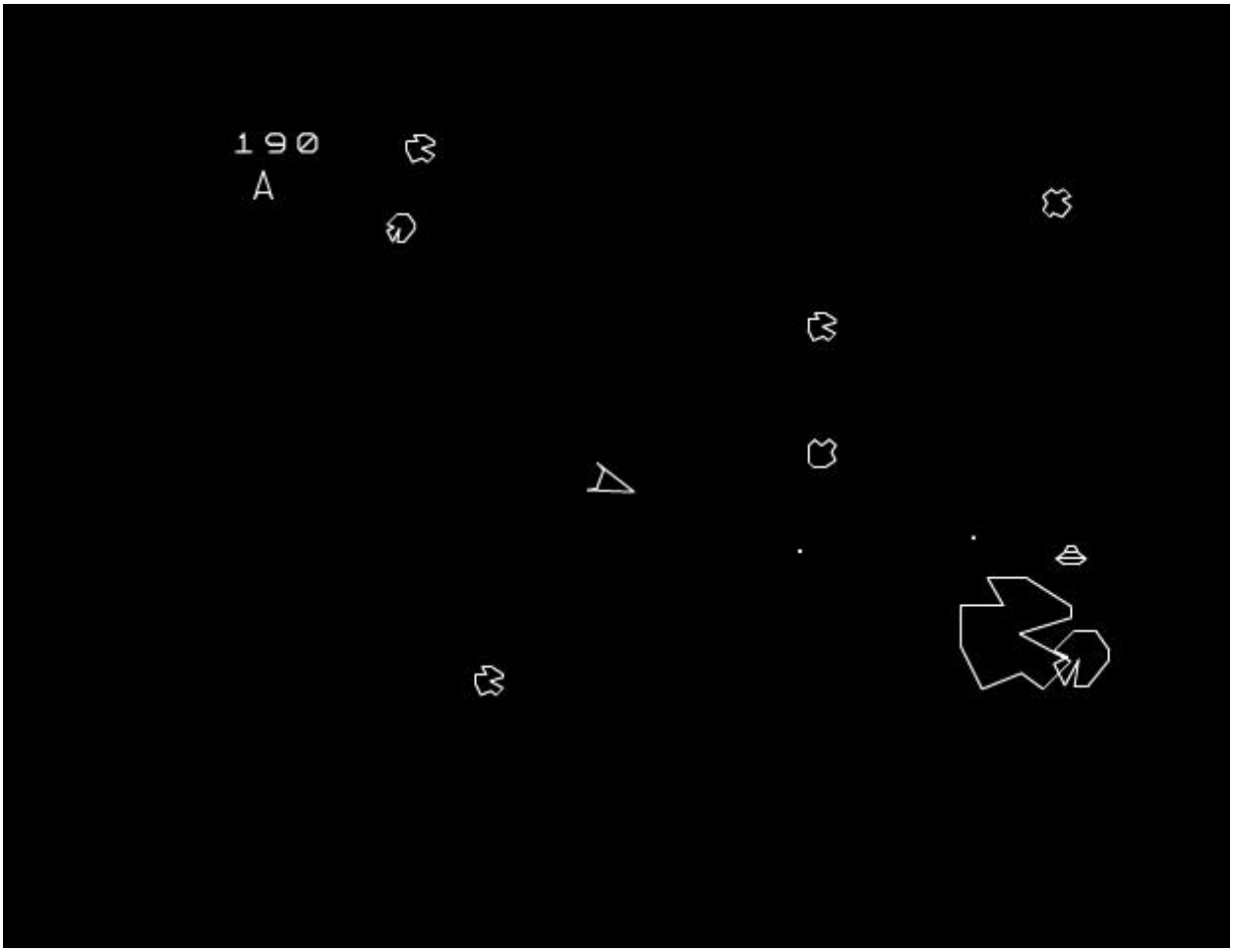


**Figure 5-5: Asteroids**

This game builds on Lunar Lander, but instead of landing a spacecraft on the moon, the goal is to avoid incoming asteroids that will break apart into smaller asteroids that when shot. If the spacecraft moves out of the screen, it will appear on the opposite side of the screen.

In order to make Asteroids with the framework, it must be extended with the following features;

1. Shooting bullets.
2. Break apart existing objects into multiple objects (This can be done with add and remove).
3. Teleport objects to the other edge of the screen when they go out of bounds.
4. Or make the same object appear on both edges as it moves out of bounds and into the opposite edge.

***Donkey Kong***


**Figure 5-6: Donkey Kong**

The player controls a character that can move forward and backwards, jump and climb ladders. The level contains multiple slopes, and there are ladders between each slope. An opposing player controlled by the computer throws barrels down the slopes, and the objective for the player is to make their way up all the slopes while avoiding being hit by a barrel.

In order to make Donkey Kong with the framework, it must be extended with the following features;

1) Collision filtering so the barrels can pass the ladder openings.
2) Trigger zones so player overlap with ladder areas can be detected.

## Framework design requirements

The design requirements for the framework was derived from the games and summarized in feature descriptions.

### *JavaFX physical object types*

It must be possible to define objects that is simulated differently, for example the paddle in pong and breakout should not be able to be affected by physical forces, but should be able to inflict physical forces on other objects, and still needs to move using player input. There is also a need for objects that act as a physical barrier in games. Pong Breakout is inside a bounding box and Breakout has a physical barrier on the left, top and right side of the game area.

Physical object

In order to satisfy the demand for simulated moving physical objects, there needs to exist objects that are fully simulated by physical properties in the x, y plane.

Non-movable object

There must also exist objects that cannot be moved or manipulated by other fully simulated physical objects, this is to satisfy the need for immovable objects like floor, roof and walls that are used to keep the player and game objects within the bounds of the game.

Movable object

Games like breakout need a moveable object that collides with fully simulated objects, but simulated objects should not be able to move this object.

### *Game world*

The different object types Physical, Non-movable and Movable also depend on the properties of the world they are simulated in, so there needs to exist a JavaFX representation of the game world and the physical parameters for this game world that affects the objects in the game world. An example of such a game world parameter is the direction and strength of gravity.

### *Interaction between objects*

When two objects collide with each other, the interaction and forces this collision creates will be simulated to appear as realistic as possible. It must also be possible to define which objects should interact and which objects should ignore each other.

### *Shapes*

Different shapes must be supported. Types of shapes that needs to be supported: Rectangle, Triangle, Circle, Line and Custom shape defined by multiple polygons or triangles. Collision interaction should be bound by the shape.

## Game logic

Functionality to support the game logic that the analyzed games require must be supported by the framework.

### *Game condition on a tick*

Game condition logic must be able to exist, and in order to simplify the game condition loop, it must be possible to add game conditions by defining the condition logic and how often the condition should be checked.

### *Collision detection and custom handling*

The framework must detect when two objects collide, this can be done by checking if two different objects overlap on the 2D plane. It must be possible to add logic attached to a specific object that defines what should happen when a collision occurs.

### *Add and remove objects*

While the game is running it must be possible to add and remove objects, Asteroids demands that a big object should be replaced with multiple smaller objects when destroyed. It must therefore be possible to add new objects to the world that are loaded and simulated when added. It must also be possible to remove objects from the game world.

### *Input*

All games described requires a player to control a game object, and in order to move objects or make things happen, it must be possible to attach logic to keyboard presses, this can be done by providing scaffolding for functionality like this, or create help functions that manage the keypresses. Actions could also be mapped to keypresses.

### *Apply forces to an object*

To move an object or rotate it, it must be possible to apply forces to the simulated objects. The force should be defined in a way that describes direction of the force, strength of the force and where the force is applied on the game object. It should also be possible to apply rotational forces to an object.

### *Levels*

Games like Breakout and Lunar Lander requires multiple levels, so it must be possible to load different levels.

#### *Shooting*

Asteroids needs to be able to shoot bullets. This can be covered by adding and removing objects and applying enough force to those objects, but Jbox2D also have a bullet function that adds special collision detection for fast travelling objects, so this function must be exposed to the game objects.

#### *Gravity scale*

While an object is on a ladder, the object must not be affected by gravity, so it must be possible to define logic that exempts an object of gravity forces.

#### *Visual effects, Textures and animations*

Simulated physical visual effects would be a feature that would make the game world seem more interactive and give a player more visual feedback when things happen. But defining such visual effects in a generic way is outside the scope of this framework.

Defining textures on objects other than just colors will make the game world feel more alive, and making games will be more satisfying. Being able to change textures and create animations when objects move would also make the framework more engaging for the user. This might be covered by JavaFX functionality already.

JavaFX does have some support for textures, visual effects and animations that can be used, but this research will not explore this functionality because of time constraints.

#### *Sound*

Playing sounds and background music is possible, and would help make the game more interesting, but there is nothing stopping a player from adding sounds when certain things happen, so it is not critical that the framework has its own functionality that covers sound.

### Framework design based on background, literature design and requirements

The design of the API builds on the ideas from constructivism and constructionism where a child learns especially well when they can use their own ideas to learn about a field, so the integration with SceneBuilder is especially important. It is also based on existing game making frameworks for learning, such as Scratch, Alice and Greenfoot. And takes inspiration from Java code standards and JavaFX standards, while also taking into account the target users.

### *Additional design considerations*

The physics game framework will be used in combination with SceneBuilder, so all game objects are set up using SceneBuilder via the drag and drop interface SceneBuilder provides. These objects must be a JavaFX Node object, and must expose settings and properties that configures their physical behavior in the simulated world.

The workflow with JavaFX and SceneBuilder is that you set up an FXML file and controller class, then load the FXML file in SceneBuilder where the user can edit the scene, which is saved to the FXML file as xml nodes. Logic for the FXML scene can then be programmed in the controller class by importing the FXML resources set up in SceneBuilder, logic can be defined in the `initialize` method that is automatically run when an FXML file is loaded or in event listeners in the controller that are connected to the FXML resources. Typical FXML resources can be a physical shape or the game world which have event listener properties.

The physics game framework will use an existing physics framework, exposing the physics framework to the user will make the physics game framework complicated to use, so in order to keep that dependency hidden from the user, the physics game framework will expose functions that work against the required features in the physics framework. How these functions are exposed to the user will affect the usability of the framework.

The JavaFX layer has coordinates used for setting up the physics world, and the physics layer also has its own coordinates, so translation between these coordinates must exist. In JavaFX, each container has its own zero-based coordinates, so in order to get true window coordinates for an object, all parents and their position must be included in the coordinate calculation. For a side scrolling type of game, a set of game coordinates could also be included where the screen coordinates are translated to where a camera is positioned, or how far the world has scrolled.

An important consideration for the design is how a game should be programmed or modelled. Since the game physics framework is based on JavaFX, it should follow the JavaFX standards. The business logic, or game logic that defines what happens when objects interact or the user presses a key, should then be defined in a controller that has access to the JavaFX objects via being bound to the view.

The physics engine calculates the movements, interactions and forces in steps that are run a certain amount of times per second, a normal value for a 2D platformer game would be 60 steps per second. After each step is completed the physical properties needs to be updated for the JavaFX objects' properties like velocity, rotation, position. While the physics calculation is running it is not possible to update the physics model, so when a JavaFX object has updated its property, the update to the physics model must be queued and performed when the physics calculation is complete.

When using a physics engine for physics the physics engine will act as a model for the physical properties, and when a physical representation of a JavaFX object is moved, it must be moved equally in the JavaFX visual model. The same is true for when a user updates the position of a JavaFX object from the controller, but reversed, the physical model must be updated so the physical model is in the same position as the JavaFX visual model.

### *Design alternatives for API interaction*

When designing the prototype, the following approaches to how the user interacts with the API from the controller was considered.

Mediator

A design where all API functions are put in a mediator object that can interact with both the JavaFX objects and physics objects. Each JavaFX and physics object will have its own mediator object which is retrieved by using the JavaFX object as a key.

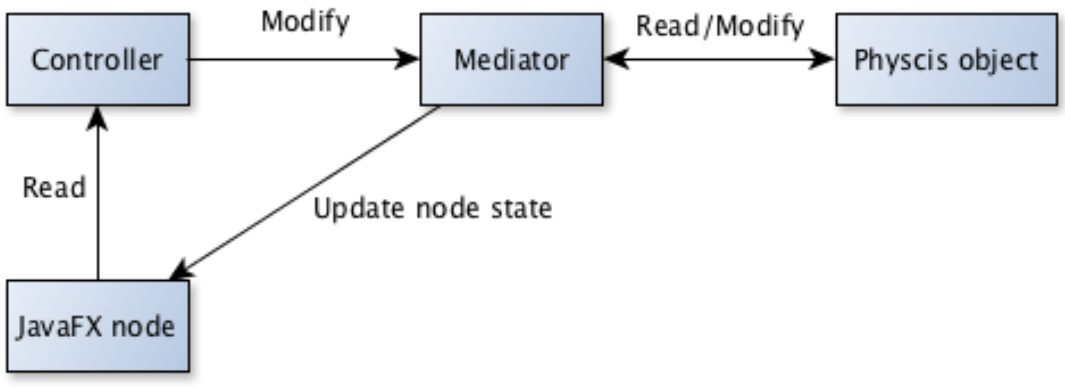


**Figure 5-7: Controller interaction with the mediator object and JavaFX node**

The Java code will look like this when retrieving and interacting with the mediator object inside a JavaFX controller class.

```
@FXML
Rectangle rectangle;

@FXML
GameWorld gameWorld;

public void initialize(){
   Mediator rectangleMediator = gameWorld.getMediator(rectangle);
   rectangleMediator.setPosition(20, 30);
   rectangleMediator.applyForceToCenter(5, 5);
}
```

**Figure 5-8: Code example of the mediator pattern**

The mediator will move the rectangle in the physics layer and update the JavaFX object.

This pattern has the advantage of keeping the functions that can be invoked on the game objects in one class separate from all other JavaFX functions. This makes the framework easier to read, but has the disadvantage of having to get a hold of the correct mediator object for a wanted JavaFX object, which can be a bit confusing since there is no way of communicating how to get the mediator and where to get it from without examples.

The mediator pattern makes the user modify the JavaFX object indirectly even though the JavaFX object already has a function for moving the visual location, this might be confusing for users.

JavaFX centric

In a JavaFX centric pattern the API functions are accessed via JavaFX objects. This is a more natural approach since it follows the rules of object oriented programming, and is easy to understand, since the users will just access the API directly via the JavaFX objects.

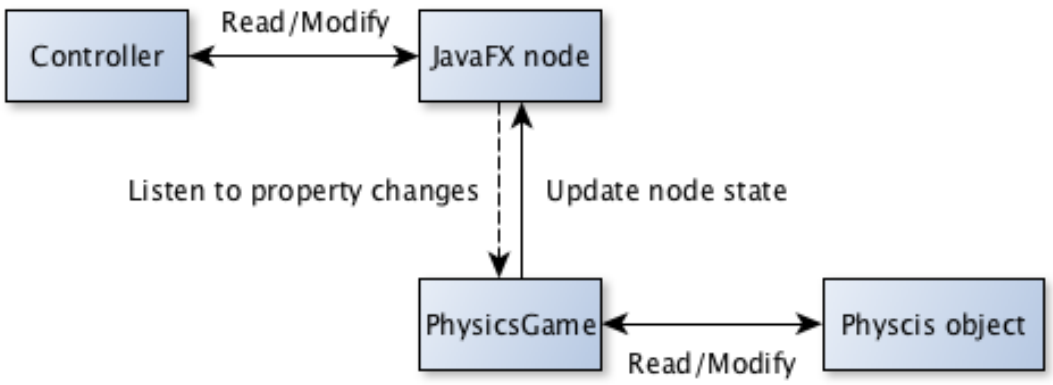


**Figure 5-9: Controller interaction with the JavaFX object**

The Java code will look like this when interacting with the JavaFX object inside a JavaFX controller class:

```
@FXML
PhysicsRectangle rectangle;

@FXML
GameWorld gameWorld;

public void initialize(){
   rectangle.relocate(20, 30);
   rectangle.applyForceToCenter(5, 5);
}
```

**Figure 5-10: Code example of the JavaFX centric design**

The code calls functions on the JavaFX object and uses the existing JavaFX functions, the game world is listening to changes on the JavaFX object and will update the physics objects to reflect the state of the JavaFX objects. It also expands the existing JavaFX class to include new functions needed for physical manipulation.

This has the advantage of getting access to the API functions easily, but a disadvantage is that the API functions will be mixed with existing JavaFX functions, which can make it harder for the user to find a wanted function, or just explore the API. It is possible to cast the JavaFX object to an implemented interface in order to limit the functions available to the API functions, but there is no way to communicate which interface to cast to without examples.

Single reusable mediator

In a single mediator pattern the API functions are accessed via an object that holds all object relations. An advantage to this is that the user can easily specify which objects the user wants to modify, and access a relevant function on the mediator, then the mediator will take care of updating the JavaFX objects and physics objects. The interaction between objects is the same as in the mediator pattern.

```
@FXML
Rectangle rectangle;

@FXML
GameWorld gameWorld;

public void initialize(){
   gameWorld.setPosition(rectangle, 20, 30);
   gameWorld.applyForceToCenter(rectangle, 5, 5);
}
```

**Figure 5-11: Code example of the shared mediator pattern**

In this example, the `gameWorld` field is acting as the mediator, and will perform interaction with the physical properties for the rectangle object. This approach has the same issues as the first mediator pattern, the JavaFX objects are indirectly manipulated, leaving the JavaFX object with confusing functions like `setLayoutX` and `setLayoutY` which cannot be used in this approach. It can also make the shared mediator hold functions that only work for certain JavaFX objects, which can be confusing for the user.

### *Design details*

Design details for decisions done during the design and implementation of the framework.

Design choice for API interaction

The chosen design from the alternatives described was the JavaFX centric design. This design was chosen because the users are used to working with JavaFX from the previous assignments they have done, and this design choice works well with SceneBuilder integration, since SceneBuilder has support for import of custom libraries.

JavaFX nodes

In order to expand a JavaFX shape with physical and geometrical properties, it can either be placed in a container that holds the physical properties, or the shape can be expanded to include the

physical properties. In order to keep the API as simple as possible, existing JavaFX shapes can be extended into new classes that also hold new physical and geometrical properties. Using a container might introduce a new layer that the users are not familiar with, so it might make the framework harder to understand.

Shape composition

In order to create complex shapes, it must be possible to combine multiple JavaFX objects into a single object. This can be achieved by creating a node that combines all the JavaFX shape nodes into a single object with multiple shapes in the physics model. This approach exposes geometric properties on the `GeometricComposition` node, and will ignore the same geometric properties on the JavaFX shape nodes when inside a `GeometricComposition`. This is a tradeoff for keeping the JavaFX shape nodes as simple as possible, an alternative could have been to always use a geometric composition when setting up a JavaFX shape, and the user could choose to use one or more shapes in the composition, then no properties would have been duplicated, but this would make the framework harder to understand and use.

Multiple levels

When changing a game level, it is preferred to share core game logic, but the FXML should be changed per level. The user has access to the `PhysicsWorld` object in the controller and the user has access to the `PhysicsGame` object outside the controller, a natural way to establish communication between these would be to create an event on the controller that the class that controls level logic listens to, but it is not expected that pre-university students understand how events work, so a solution where the user does not have to write their own events, is to call a `finishLevel` method on the `PhysicsWorld` which fires an event on `PhysicsWorld` that `PhysicsGame` listens to and fires its own event that users can connect logic to, like changing the level or ending the game.

Supplementary functions

Some games have complicated 2D space calculations that must be performed, and it is not expected for pre-university students to learn and understand these calculations, so in order to make these calculations and other supplementary functions available in the framework, static functions or exposing them on `PhysicsWorld` can be done. The users are used to importing static functions

from Kodeklubben's Python course, and it is possible to import static functions in Java, so this is a good approach. University students should also be familiar with this way of importing functions.

Directional vectors

In JavaFX coordinates for a point is given by X: distance from top to bottom, and Y: distance from left to right, so a vector that points up would be [0, -1] in this logic, this makes the coordinates and vectors consistent, but a user might not think about a vector the same way as coordinates.

A more user-friendly way to define coordinates would be to define up as [0, 1], this is consistent with the way of thinking about coordinate planes in mathematics, and is the most natural way for a user to define directional vectors, but this would break the way JavaFX defines coordinates in a coordinate plane, so in order to keep coordinates consistent to what they have already learned in previous assignments and keep the framework consistent with JavaFX, directional vectors should be based on the same system as JavaFX.

Alice has abandoned the XYZ system and instead uses the Logo inspired forward and back, left and right, up and down. This approach could also be more user friendly for younger users, but makes the compatibility with the existing JavaFX coordinate system troublesome. Overload functions that uses left and right, and the value, could be used as an alternative for `setLayoutX(double x)`. Same for `setLayoutY(double y)` with up and down.

```
@FXML private Circle ball;

@FXML
private void initialize() {
          ball.setLayoutX(Direction.LEFT, 10);
          ball.setLayoutY(Direction.DOWN, 15);
}
```

**Figure 5-12: Example of Logo style XY coordinates**

Naming classes and functions

The names of classes and functions directly impacts how understandable the framework is. The names used should communicate functionality that is understandable for users learning programming. Deciding what is an understandable name and not is not an easy task, so discussions with Hallvard Trættebærg when going through code decisions can be used to decide if a name is good enough. Names can also easily be modified after an evaluation of the framework based on feedback from the users.

## The Prototype: PhysicsWorldFX

The prototype produced was named PhysicsWorldFX, this name reflects that it is a framework for JavaFX and that it is a framework that offers physics functionality in a world. PhysicsWorldFX enables users to create small 2D games with physical behaviors and physics simulations by designing a physical world in FXML with SceneBuilder, and then programming game logic in the controller class associated with the FXML file.

### Physics

The physics library chosen for PhysicsWorldFX was JBox2D. FXGL had a lot of useful functionality, but it also had a lot of functionality not needed for this prototype, so in order to avoid complexity with functionality that was not needed, and to keep the physics integration simple, JBox2D was preferred. JBox2D was chosen over Dyn4J because of its maturity, performance and documentation. There was more documentation and examples available for JBox2D than Dyn4J. Benchmark tests also showed that JBox2D had an edge over Dyn4J.

### API Overview

The API provides classes and functions that the user can use in their JavaFX game. The game world with its physical properties is represented as a class named `PhysicsWorld` and is a necessary container for the game assets. The API provides game assets in the form of geometric shape nodes that define the interaction and collision bounds for the physics model, and these geometric shapes are rendered as shapes in JavaFX. The API also provides supplementary functions for difficult 2D space calculations by importing them using static imports.

#### *PhysicsGame*

In order to parse the components defined in FXML, the root node must be loaded and parsed by the `PhysicsGame` class, this will set up the physics model for the contents of the root node which should have physical properties. This class runs the game loop and wires up event listeners for forwarding events for dependent objects.

#### *PhysicsWorld*

The physics world is a representation of the game world, which can be defined in FXML as a node, and holds the physical properties for the world, like gravity and scale for the physics. It also provides access to the game loop that `PhysicsGame` is running when started via event listeners,

and has event listeners for collision handling that are raised when a collision in the game world occurs. The event listener properties are the following; `OnPhysicsStep`, `onBeginCollision` and `onEndCollision`.

#### *Physics shapes*

The physics shapes are the FXML nodes that get a physical representation in the physics model. It is possible to define the following shapes; Rectangle, Circle, Polygon and Polyline. The rectangle is defined by height and width, the circle by a radius, and the polygon and polyline by a set of points. All these shapes hold physical properties like; `Friction`, `Density` and `Restitution`. They also hold geometric properties like; `SimulationType`, `LinearDampening`, `LinearVelocity`, `AngularDampening`, `AngularVelocity`, `GravityScale` and `Bullet`. It is also possible to drag multiple shapes into a `GeometricComposition` node to make a complex shape.

#### *Supplementary functions*

The supplementary functions can be accessed by using a static import in the controller class.

```
import static framework.PhysicsWorldFunctions.*;
```

`PhysicsWorldFunctions` provides functions that makes programming with advanced 2D vector calculations easier, functions for loading FXML resources, functions for registering and unregistering key presses, geometric functions and functions for checking `StyleClass` attributes on nodes.

### Usage of PhysicsWorldFX

When the Integrated Development Environment (IDE) and SceneBuilder is set up, a normal workflow with the framework would be to start a JavaFX project with a JavaFX application class, an FXML file and a controller class. The application class must load the FXML root object using the `PhysicsGame load` function. SceneBuilder can then load the FXML file and then the user can then set up a `PhysicsWorld` root node, and drag and drop different physics shapes into `PhysicsWorld`. The game logic can then be programmed in the controller class by setting up game logic in the `PhysicsWorld onPhysicsStep` event listener, typical game logic that can be defined here can be logic that is connected to key presses, for example; if the key left is down,

then rotate object left with specified force. If the game logic requires collision handling the logic can be defined in the `PhysicsWorld onBeginCollision` event listener, or the `PhysicsWorld onEndCollision` event listener depending on if the user wants logic on initial contact or on separation.

### Working with SceneBuilder and PhysicsWordFX

PhysicsWorldFX can be imported into SceneBuilder which will make the physics nodes available in SceneBuilder, and can be dragged and dropped into the workspace. All physical and geometric properties are available in SceneBuilder.

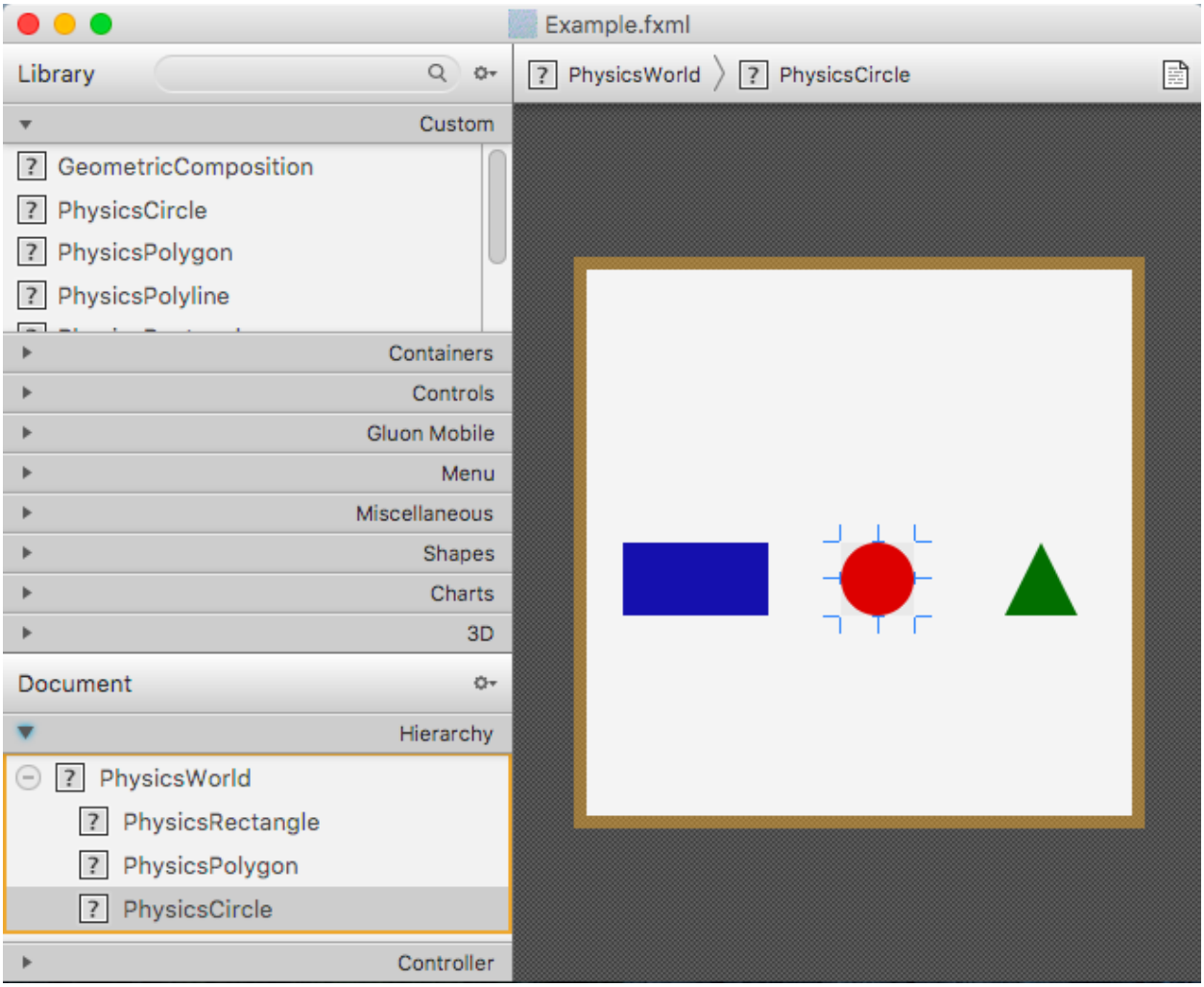


**Figure 5-13: PhysicsWorldFX in SceneBuilder**

SceneBuilder has support for custom libraries and Figure 5-13: PhysicsWorldFX in SceneBuilder shows PhysicsWorldFX JavaFX nodes in the Custom tab.

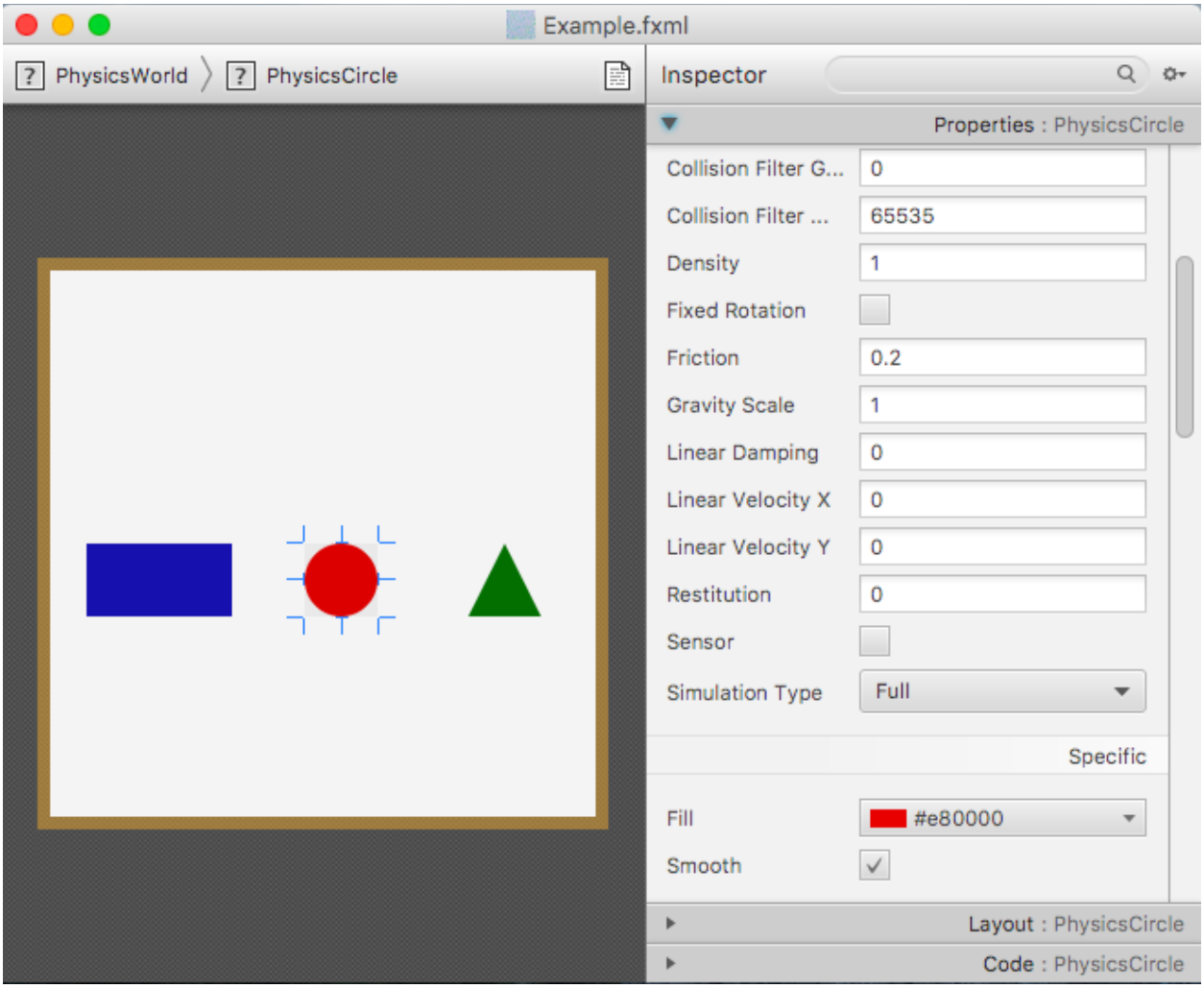


**Figure 5-14: Editable properties of a PhysicsWorldFX node in SceneBuilder**

SceneBuilder also can edit properties on a node, and Figure 5-14: Editable properties of a PhysicsWorldFX node in SceneBuilder shows the properties of a `PhysicsCircle` node.

## Example games as scaffolding

Example games were made to ensure that PhysicsWorldFX could support the types of games used to derive requirements for the design and to serve as examples on how to use the framework. The games made were Pong, Breakout, Lunar Lander, Asteroids and Jetpack. Pong demonstrated that a simple game with a ball bouncing inside a bounding box worked, and Breakout built on this by adding interaction logic that removed an object from the world. Lunar Lander showed that applying forces to objects based on key input could move them.

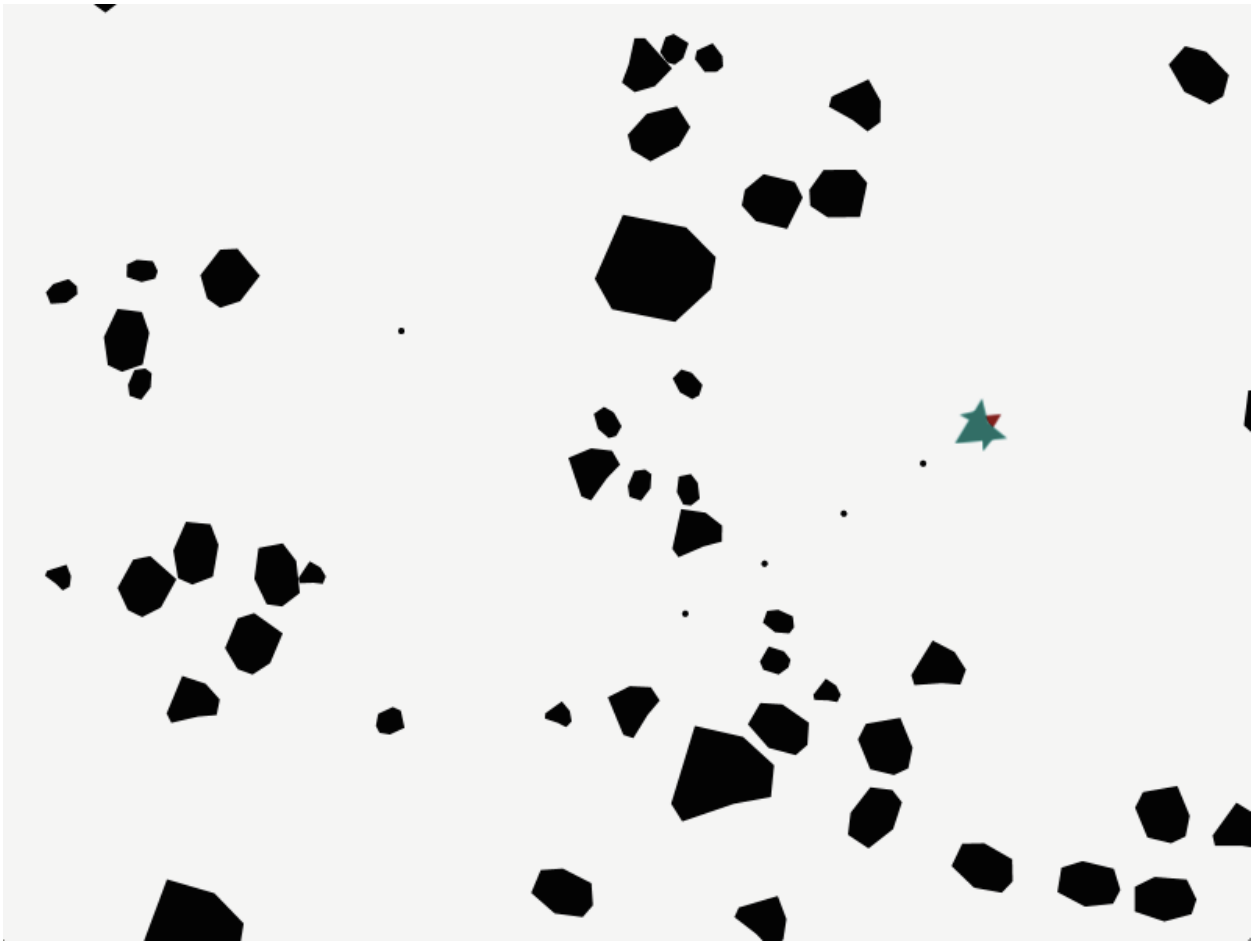

**Figure 5-15: Asteroids built with PhysicsWorldFX**

Asteroids built on Lunar Lander and showed that a spaceship could shoot at asteroids that break apart when hit, this required loading resources and functions for calculation of velocity vectors.

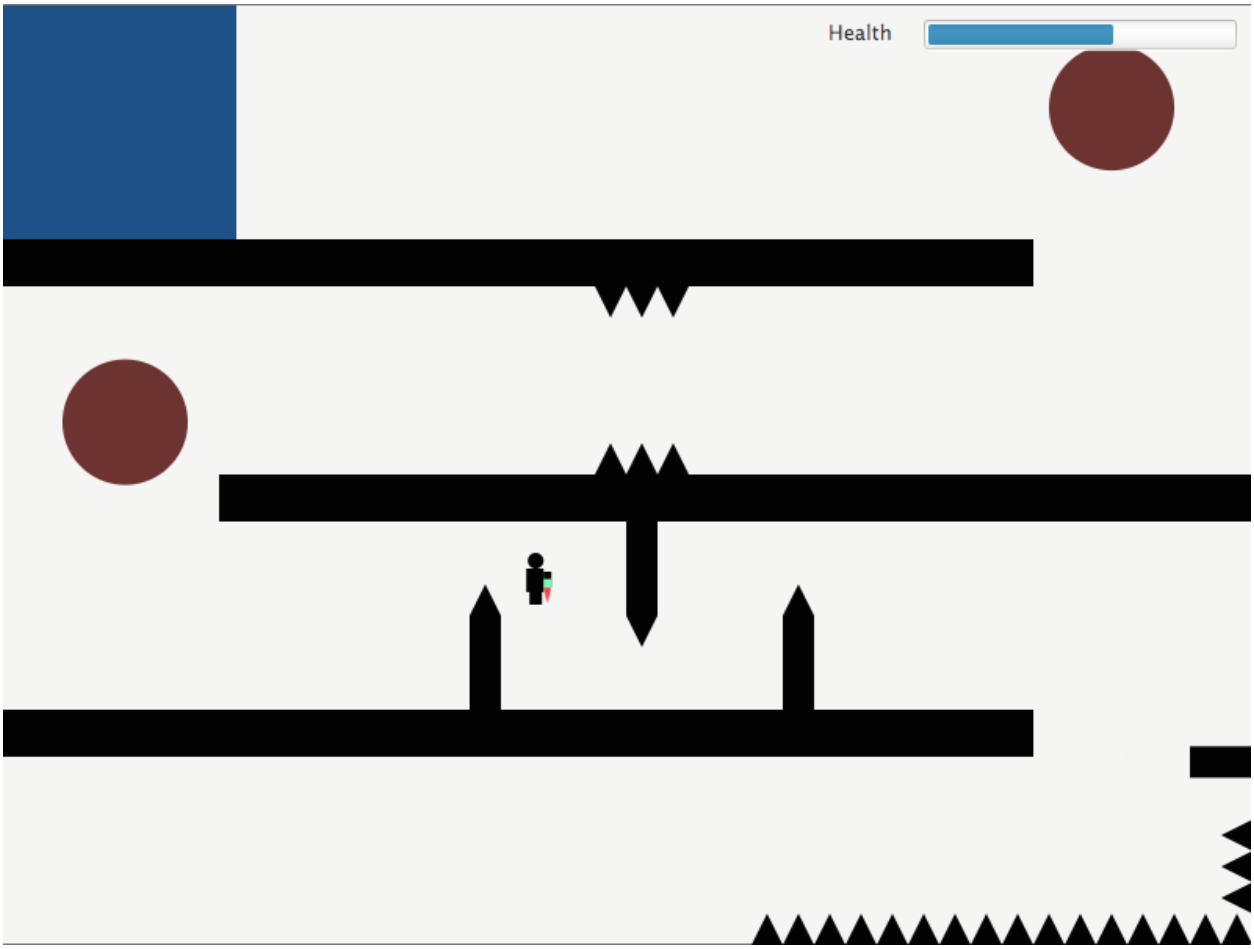


**Figure 5-16: A 2D platformer named Jetpack built with PhysicsWorldFX**

The Jetpack game demonstrated that a simple 2D platform game could be made where GUI elements were used as an overlay displaying the remaining health. The example games also serve as scaffolding where the user can reuse and remix the game.

## Analysis of the example games implementation

The example games were made to ensure that the requirements were both possible and straight-forward to implement, so this evaluation will look at how some requirements and features were implemented.

### *Pong*

In Pong, the main challenge was to define the different simulation types needed for the different types of behavior requirements of the game objects. Connecting key presses to movement was also a requirement. This was achieved by the following code;

```
<PhysicsWorld fx:id="world"
              onKeyPressed="#handleKeyDown" onKeyReleased="#handleKeyUp"
              onPhysicsStep="#handlePhysicsStep"
              onBeginCollision="#handleCollision">

    <PhysicsRectangle fx:id="paddle1" simulationType="Movable" />
    <PhysicsRectangle fx:id="paddle2" simulationType="Movable" />

    <PhysicsRectangle fx:id="wallLeft" simulationType="NonMovable" />
    <PhysicsRectangle fx:id="wallRight" simulationType="NonMovable" />
    <PhysicsRectangle fx:id="roof" simulationType="NonMovable" />
    <PhysicsRectangle fx:id="floor" simulationType="NonMovable” />

    <PhysicsCircle fx:id="ball" simulationType="Full" linearVelocityX="800"
                   linearVelocityY="50" />
</PhysicsWorld>
```

**Figure 5-17: FXML file for Pong (simplified to only show relevant properties for paddle movement)**

The code shows that paddle1 and paddle2 are of `Moveable SimulationType`, this means that the shape can be moved by setting a velocity, but will not move when another object collides with it. The walls are `NonMovable` types which can never be moved, only teleported, and the ball has the `Full` simulation type which is fully simulated and interacts with all simulation types.

```
@FXML
private void handlePhysicsStep() {
    setPaddleSpeed(paddle1, keyIsPressed(KeyCode.A), keyIsPressed(KeyCode.Z));
    setPaddleSpeed(paddle2, keyIsPressed(KeyCode.K), keyIsPressed(KeyCode.M));

    //keeps the ball speed constant
    setVelocityToCurrentTravelVector(ball, ballSpeed);
}

private void setPaddleSpeed(PhysicsRectangle paddle, boolean movePaddleUp,
                            boolean movePaddleDown) {
    if (movePaddleUp && !physicsNodesTouching(paddle, roof)){
        paddle.setLinearVelocityY(-500);
    }else if (movePaddleDown && !physicsNodesTouching(paddle, floor)){
        paddle.setLinearVelocityY(500);
    }else{
        paddle.setLinearVelocityY(0);
    }
}
```

**Figure 5-18: Controller logic for Pong paddle movement**

The function `handlePhysicsStep` is connected to `PhysicsWorld onPhysicsStep`, so the `setPaddleSpeed` function is called for both paddles on every physics step in the application, and

can be used as the game loop, the reason it is not called `onGameStep`, is to support physics simulations as well, and makes the user think of the world as a physics simulation. `setPaddleSpeed` is called with the keypress states for controlling the paddle, and shows how little code is needed to connect movement logic to a JavaFX node in PhysicsWorldFX.

### *Breakout*

In Breakout, the bricks were generated in the initialize function, but the main feature in Breakout was to react to collisions and remove the brick when the ball hit it. This was achieved with the following code:

```
<PhysicsWorld … onBeginCollision="#handleCollision" … >

       …

        <PhysicsCircle fx:id="ball" density="0" simulationType="Full" restitution="1"
                       linearVelocityX="10" linearVelocityY="400" fill="#7a1fff"
                       layoutX="320.0" layoutY="270.0" radius="10.0"/>

        <Pane fx:id="brickContainer" layoutX="20.0" layoutY="20.0"
              prefHeight="200.0" prefWidth="600.0" />

        <PhysicsRectangle fx:id="paddle" simulationType="Movable" fill="DODGERBLUE"
                          height="20.0" layoutX="270.0" layoutY="400.0" width="100.0" />
</PhysicsWorld>
```

**Figure 5-19: FXML snippet for Breakout**

```
    @FXML
    private void handleCollision(CollisionEvent collisionEvent) {
        handleCollision(collisionEvent.getObject1(), collisionEvent.getObject2());
        handleCollision(collisionEvent.getObject2(), collisionEvent.getObject1());
    }

    private void handleCollision(Node object1, Node object2) {
        if (object1 == ball){
            if (hasStyle(object2, "brick")){
                brickContainer.getChildren().remove(object2);
            }
        }
    }
```

**Figure 5-20: Controller code for handling collisions in Breakout**

The bricks are added to the `brickContainer` during the `initialize` function, and are automatically picked up by PhysicsWorldFX and gets a physical representation in the physics model.

The `handleCollision` function is connected to the `onBeginCollision` event, so when the physics engine detects that a collision has happened this event is raised. The `handleCollision(Node object1, Node objec2)` is called twice because there is no predictable way to know if `object1` or `object2` is the ball, so both must be checked. When the ball hits a brick, it is removed simply by removing it from the `brickContainer` (in practice there is no need for a separate container, but it was used in this case to test compatibility), the change is picked up by `PhysicsGame` who listens to changes is the container, and the brick is also removed from the physics model. Very little code is needed and the code written uses existing JavaFX standards.

### *Lunar Lander*

This game had the requirements of having a shape that was put together by a composition of shapes both physical shapes and visual shapes, and applying forces to a physics node.

```
<GeometricComposition simulationType="Full" layoutX="40" layoutY="40" >
    <PhysicsPolygon fx:id="leftThruster" layoutX="0" layoutY="18" restitution="0"
                    density="0.2" friction="1" >
        <points>
        …
        </points>
    </PhysicsPolygon>
    <PhysicsPolygon fx:id="rightThruster" layoutX="39" layoutY="18" restitution="0"
                    density="0.2" friction="1" >
        <points>
        …
        </points>
    </PhysicsPolygon>
    <PhysicsPolygon layoutX="12" layoutY="0" restitution="0" density="1"
                    friction="1" >
        <points>
        …
        </points>
    </PhysicsPolygon>
    <Polygon fx:id="rightFlame" fill="red" layoutX="39" layoutY="33">
        <points>
        …
        </points>
    </Polygon>
    <Polygon fx:id="leftFlame" fill="red" layoutX="0" layoutY="33">
        <points>
        …
        </points>
    </Polygon>
</GeometricComposition>
```

**Figure 5-21: Lander that is a composition of multiple objects (vertices in `points` has been omitted)**

The lander is built up by multiple `PhysicsPolygons` where if a force is applied on either `PhysicsPolygon`, the whole `GeometricComposition` will be affected by that force. The definition of points takes up a lot of lines, but this is the way points are defined in JavaFX.

```
@FXML
private void handlePhysicsStep() {
    if (keyIsPressed(KeyCode.RIGHT)){
        leftThruster.applyUpwardForceToCenter(0, singleThrustForce);
        leftFlame.setVisible(true);
        rightFlame.setVisible(false);
    }
    else if (keyIsPressed(KeyCode.LEFT)){
        rightThruster.applyUpwardForceToCenter(0, singleThrustForce);
        rightFlame.setVisible(true);
        leftFlame.setVisible(false);
    }
    else if (keyIsPressed(KeyCode.UP)){
        leftThruster.applyUpwardForceToCenter(0, fullThrustForce);
        rightThruster.applyUpwardForceToCenter(0, fullThrustForce);
        leftFlame.setVisible(true);
        rightFlame.setVisible(true);
    }
    else {
        leftFlame.setVisible(false);
        rightFlame.setVisible(false);
    }

    if (physicsNodesTouching(leftThruster, landingPad)
        && physicsNodesTouching(rightThruster, landingPad)){
        world.finishLevel(true,0);
    }
}
```

**Figure 5-22: Controller logic for Lunar Lander**

The controller logic defines how the lander should be affected and what the visual state should be on key presses. A bit more lines of code is needed for this game, but the code needed is very simple and depends on that the user understands what the `applyUpwardForceToCenter` function does, and this is explained by these examples and in the code documentation.

### *Asteroids*

In Asteroids, the most challenging requirement was to spawn new objects programmatically in the controller class.

```
<PhysicsCircle xmlns="http://javafx.com/javafx/8.0.111"
               xmlns:fx="http://javafx.com/fxml/1"
               bullet="true" fill="black" radius="2.0" styleClass="bullet" />
```

**Figure 5-23: bullet.fxml resource file**

```
private void shoot() {
    try {
        PhysicsCircle bullet = loadFxmlResource(getClass(), "/bullet.fxml");
        Point2D point = getVectorForDegrees(playerShip.getRotate(), 600);
        bullet.setLinearVelocityX(point.getX() + playerShip.getLinearVelocityX());
        bullet.setLinearVelocityY(point.getY() + playerShip.getLinearVelocityY());

        Point2D spawnPoint = getRotatedLayoutPosition(playerShip, 15, -4);
        bullet.setLayoutX(spawnPoint.getX());
        bullet.setLayoutY(spawnPoint.getY());

        physicsWorld.getChildren().add(bullet);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

**Figure 5-24: Controller code that spawns a bullet using bullet.fxml**

The code from the controller is run when the user presses or holds the space key and loads an FXML resource defined as its own file. `loadFxmlResource`, `getVectorForDegrees` and `getRotatedLayoutPosition` are functions from the supplementary functions, and by using these functions the code for spawning a bullet gets very simple. Similar code is used when breaking apart a big asteroid by spawning 2 or 4 smaller asteroids.

Asteroids is the most complex game that was made, but the logic needed was required because of the complexity of the game, and not because the framework required the user to code in a specific way.

### *Jetpack*

The Jetpack game uses many of the already established requirements and was a test to see how easy it was to create a 2D platform game where the user could add wanted game mechanics. It was also a test to see how easy setting up an overlay that showed the health of a player using existing JavaFX nodes could be.

```
<ProgressBar fx:id="health" prefWidth="200.0" progress="1.0" />
```

**Figure 5-25: `ProgressBar` used as a health bar overlaid using `HBox`**

```
@FXML private ProgressBar health;
private int healthValue;

private void setHealth(int healthChange) {
    healthValue += healthChange;
    health.setProgress(healthValue/100d);
}
```

**Figure 5-26: Controller code that modified the `ProgressBar`**

The code needed was very simple, and the `setHealth` function was called when the player collided with a spike or was crushed by a ball, the game ended when the `healthValue` reached 0.

### GitHub hosting

PhysicsWorldFX and the example games are hosted at the project repository site GitHub[1] as a public repository at https://github.com/kristianw87/PhysicsWorldFX . A guide to the code can be found in Appendix A: Guide to GitHub Contents.

### License

The license chosen for the project is the MIT license. This is the most open license, and since educational game programming in code clubs are volunteer work, then this research prototype should be available for anyone to reuse or further improve.

## Planning user testing of the prototype

The user testing is aimed at evaluating PhysicsWorldFX by testing it for usability, the international standard ISO 9241-11(ISO 1998), defines usability as the following: "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

The usability testing should measure qualitatively how effective, efficient and satisfactory the prototype is so it can be further improved and give data for RQ1 and RQ2.

1. Effectiveness: Can users achieve their goals with this prototype?
2. Efficiency: How much time does the user use to achieve their goal?
3. Satisfaction: What does the users think of the prototype?

[1] https://github.com/

Intended users of the system: Kodeklubben participants for JavaFX course and students who wishes to learn programming through game development.

Tasks: Making games and learning programming while having fun.

Environment: Teaching environment in a classroom or a group of Kodeklubben students in a smaller teaching environment, depending on how many Kodeklubben students there are.

Since there is no Kodeklubben course for JavaFX this year, users of the prototype will be recruited from university students who has some knowledge of JavaFX. The user's goals are to make a 2D game without having to program complicated math in order to achieve desired functionality. The users will get an introduction to the framework and some scaffolding to get started making their own games.

Ideally a second usability test for the prototype should be performed after the first to measure if the usability of the prototype has improved since the first test, but because of time constraints for this project, this will not be done at this time.

### Test users

According to (Nielsen 2000) 5 users should be enough in a usability test, and it is better to run multiple tests than using a large number of participants, but since this test only will run one test because of time constraints, more participants is preferred.

Choosing the correct number of students is important, according to (Faulkner 2003) the number of users used for usability tests affects how many of the problems are discovered, and with only 5 users, the number of problems discovered under testing can be in worst case only 55%. With 10 users, the number of problems discovered becomes much better at 80% at worst case, and Faulkner recommends getting as many users as possible since more users more problems are discovered, but also acknowledges that selecting the number of users for testing is complex and needs to be adjusted per project, since there are too many variables to account for. Because the data produced increases with number of users and more data takes longer time to analyze, a realistic number of students needed should be around 10, this number should produce good data and stay within the time constraints the researcher has around this assignment, but if recruiting students proves tougher than expected, a lower number of students can be used.

### Qualifications

The users for the usability test will be 2. and 3. year students in computer science recruited from courses where the students have already done a course where they have learned about and tried JavaFX. There are two courses at NTNU where the students have learned about JavaFX, TDT4100 – Object-oriented programming and TDT4180 – Human-Computer Interaction (HCI). In TDT4100 the students have gotten an introduction to JavaFX, but have not had any real experience with it and it was not a teaching goal in that course. In TDT4180 the students have performed exercises where they have programmed JavaFX user interfaces, so a combination of students who have had these courses should produce similar data to Kodeklubben students who would have had an introductory course to JavaFX before using the game framework, even though university students in their 2. and 3. year are more capable at programming, so minor issues a less experienced user might have picked up in the test, might go unnoticed in this test environment.

One drawback with using university students is that it is not possible to compare the design choices with how Scratch, Alice and Greenfoot works, since the university students have probably not had experience with those learning tools, but this is not critical to answer the research questions.

### Recruitment of test users

In order to get students to do the evaluation the students recruited will receive 100kr per hour of usability testing they go through. These payouts are funded by funds available to master's thesis assignments and from the researcher's pocket. (Sova, Nielsen et al. 2003) warns against using too much incentive when recruiting participants for a usability test, since they might become biased to give good test scores when the incentive is too high. So, the incentive to get the students to do the usability test should not be too high, and therefore the incentive was limited to 100kr per hour. 100kr per hour is not that much for a student, but still enough that it will be worth their time and get an honest opinion.

Students will be recruited by presenting a pitch during lectures in courses where the students have previously had TDT4100 or TDT4180.

### Test procedure

The test procedure should produce data so effectiveness, efficiency and satisfaction can be measured qualitatively, but also needs to be set in context of the purpose for this assignment.

In order to produce usability data that targets research question 1 and 2, and gather data in accordance to the background of this field, based on the usability questions from (McLellan, Roesler et al. 1998), the questions used to test this framework will be the following;

- How easy is the programming environment and prototype to use?
- How efficiently can the prototype be used for specified tasks?
- What misconceptions or errors do users make while using the prototype?
- How users perceive the prototype?
- How does the prototype affect the learning process for students?

In order to gather data for these questions, the test users need to be exposed to functionality by performing some pre-defined tasks, and further build on their code to make a game with the prototype, and then gather qualitative data from the test users.

### Pre-defined tasks

The pre-defined tasks will focus on design choices made to see if they are understandable for the user, if the documentation for the API is good enough and target how well working with SceneBuilder and an IDE works. The tasks will not measure if the users were able to complete the tasks without help, but rather to get the users exposed to the functionality so it can be discussed during the group interview. The tasks should also not be step by step guides, since this affects the learning process negatively, but rather just be tasks where the users can form their own hypothesizes according to (Chang, Chen et al. 2008).

### Making their own game

The part where the test users makes their own games by further building on the tasks performed will give further data on how the users experience the framework when working with it, and provide further basis for the group interview.

### Observational data collection

The test users will be informed that observational data will be collected during the testing. The observations will focus on events around the prototype where the observer will participate in the proceedings. The recording of observations will be simple note taking during the testing. The observations will be overt since there is no risk that the participants might act differently when knowing they are being observed, since this is already an artificial test with students in a university

setting. The observations can only discover what happened, but not why and what the user thought about what had happened. The observational data can be used to make up questions for the group interview to gain more insight about the events that happen.

In order to secure validity of the data collected, verbatim quotations will be noted and can also confirm what has happened by asking questions during the group interview. Reflection will also be used when analyzing and using the data (Oates 2005).

### System usability scale data collection

After testing the framework for 2 hours, the users will be asked to fill out a System Usability Scale (SUS) form. SUS data is a benchmark to differentiate between usable and unusable for a product, and can be used on a small sample size.

### Group interview data collection

In order to generate qualitative data on the framework, and discover problems and issues with the framework a group interview will be conducted after the usability testing. A group interview should be conducted with 3 - 6 people at a time, so if the usability testing gets 10 participants the group interview will have to be conducted twice with 5 participants for each session. The purpose of the interview is to discover how the prototype performs in regard to the questions listed in the test procedure, so the interview will be semi-structured. Questions will be prepared ahead of the interview, but if the interviewees bring up unforeseen issues, these can be explored with new questions.

The interviews will be recorder and some field notes will be taken during the interview if the audio recording cannot pick something up.

## User testing of the prototype

The evaluation was performed at a meeting room in the IT-building at NTNU Gløshaugen campus at the evening, the room was 20mˆ2 with a meeting table for 8 people and a 55” LCD TV screen for demonstrations, and had 3 windows with a view to green trees outside. The evaluation lasted for 3 hours.

### Recruitment

The students that agreed to do usability testing of the prototype, were recruited by pitching the evaluation of the prototype in two different lectures where users who were qualified for the evaluation were attending. The pitch included information on what requirements the participants needed to be qualified, details on the prototype, time and date for evaluation, the researchers email, a link to the prototype GitHub page. There were about 60-100 students at the first lecture for TDT4136, and 150 – 250 students at the second lecture for TDT4120, but it is hard to estimate how many students were qualified from this group. Getting students to do the evaluation proved harder than thought, but after communicating over email with a couple of potential participants, 6 students ended up agreeing to doing the evaluation. 3 of the students had only had TDT4100 and the other 3 had had both TDT4100 and TDT4180. All students were 2. and 3. year students in the Informatics and Computer Science programs. One student was female and five students were male.

### Overview of the test procedure

1. The researcher and research project was introduced
2. Information about the purpose of the usability test and the schedule for the test
3. Help set up the development environment on the participants' computers
4. A demonstration on how to use the framework was performed on the TV
5. Usability testing by letting the participants do pre-defined tasks
6. Had to stop testing after the pre-defined tasks because of time limits
7. Finished with a group interview with the participants
8. The participants each filled out the system usability scale form
9. Thank the test users for participating in the test
10. Pizza was served

### Introduction

The evaluation started with introducing the researcher and the research project, but no background on why the project was being done, or for who the research was aimed at was given. The participants were informed about the purpose of the usability test, which was to gather data for the research project. The participants were given a schedule for the tests, and were informed that they could ask for help at any time, but that it was preferred that they tried a bit on their own before asking. It was also informed that everyone participating in the testing would be anonymous and

that notes would be taken during the usability testing, and that audio would be recorded during the group interview but deleted after transcription. Everyone then set up their preferred development environment, and it was noted that 4 of the participants used IntelliJ[2] and 2 of the participants used Eclipse[3] as their preferred IDE, as the usability testing allowed the participants to use the IDE they were most comfortable with, but as a backup a guide on how to set up Eclipse to work with the prototype was provided on the project GitHub wiki page. Setting up the development environment did take a bit longer than planned.

### Demonstration

After everyone had their development environment set up the researcher introduced the participants to the documentation on the GitHub wiki page, how they could find documentation in the source code, and that they could look at the provided example games to gain insight on how to use the framework. A demonstration was done by live coding a small example which showed a physics simulation where a couple of shapes interacted with each other.

### Pre-defined tasks

The participants then got to test the framework by doing the pre-defined tasks. From the researcher's perspective, this seemed to go as planned, but there were some questions asked now and then about certain functionality in the framework, and how to use SceneBuilder. Something that was particular difficult to the participants was to define points on a polygon shape, this was partly because Eclipse did not import a dependency for double in the FXML file, and partly because the users had to switch back and forth between their IDE and SceneBuilder. The pre-defined tasks did end up taking longer time than planned. The pre-defined tasks can be found in Appendix B: Usability Tasks.

### Making a game or simulation with the prototype

Unfortunately, the evaluation spent too much time on setting up the development environment and performing the pre-defined tasks, so there was not enough time to build on the code to make a game or simulation with the prototype, but they were still exposed to the prototype by the tasks,

[2] https://www.jetbrains.com/idea/
[3] https://www.eclipse.org/

and some of the participants had tried out some of the example games and looked at the code, so they had an understanding of how a game could be made with the prototype.

### Group interview

Because of the time constrains the evaluation jumped to the group interview part. The participants were again informed that the audio recorded would be deleted after transcription, and everyone would be anonymous. The researcher also asked if everyone was ok with being recorded, and everyone responded that they were ok with being recorded. The group interview was prepared as a semi-structured interview, and one of the challenges was to translate questions to Norwegian, since the questions were prepared and carefully worded in English, so some of the planned questions did not get the expected response, but they still started discussion about the intended topic, and since this was a semi-structured interview this was fine. The prepared group interview questions can be found in Appendix C: Group Interview Questions.

### Ending the evaluation

After the group interview was done, the evaluation was finished and the participants were served pizza from a local pizza restaurant. The participants were also each given a System Usability Scale form, and was asked to fill it out as honestly as possible, they all filled out the form and delivered it to the researcher. The participants were also given the promised allowance for doing the evaluation. Before everyone left, the researcher thanked everyone for taking their time to do the evaluation.

## Analysis

An analysis of the qualitative data gathered during the evaluation.

### Planning the qualitative analysis

Planning the analysis was based on the qualitative analysis procedure described by (Oates 2005).

Categorize transcribed audio into three themes:

- Text segments that have no relation to the overall research purpose
- Text segments that provide descriptive information that provides the research context for readers
- Text segments relevant to the research questions

Each text segment that is relevant to the research questions will be categorized with categories related to existing theories from the literature and theories that are developed by the researcher. This is a deductive approach, but if other themes in the data is found, then these should not be ignored. Explanations for the results should be provided and there could exist multiple explanations for the results. It is not expected to produce significant amounts of data, so it will be hard to use tables and diagrams to explain the data, but if an opportunity for this arises, then it will be considered.

The transcribed interview and coding of the text can be found at Appendix D: Coding of Qualitative Data.

### Qualitative analysis of the data

The participants who used IntelliJ IDEA were probably a bit more skilled in programming and setting up their development environment than the participants who were using Eclipse. 5 of the participants had previous experience with JavaFX, FXML and SceneBuilder and one participant had only gotten an introduction to JavaFX, but had never used SceneBuilder before.

The observed complexity to start with PhysicsWorldFX was a bit larger than expected during the testing of the usability tasks, and the participants also wanted a bit more explanation on what the different nodes in PhysicsWorldFX could do in the documentation. This perception from the users could have been because of how quick the framework was demonstrated, but the users also gave conflicting information during the group interview, so this observation could have been exaggerated. The participants might have gotten more out of the prototype if they had gone through a step by step tutorial where they set up a small game or physics simulation, instead of starting straight on to usability tasks, where they had to explore PhysicsWorldFX on their own and find information from the documentation. But this goes against the advice from (Chang, Chen et al. 2008) where step by step guides should be avoided, but this advice was for simulations, so it is questionable if this is also applicable for game development with physics.

Concerning RQ2, the participants seem to enjoy using PhysicsWorldFX, and getting feedback on the code they are writing is motivating them to further use PhysicsWorldFX, this was both observed under the testing and asked about during the group interview, and all participants agreed

that working iteratively with PhysicsWorldFX was something that motivated them to learn with the framework.

The participants who had no previous experience with JavaFX successfully learned the semantics and how JavaFX worked when using PhysicsWorldFX. A participant who had not programmed in JavaFX before said in response to the question “Did you learn anything new about how JavaFX worked while working with PhysicsWorldFX?", "I have never used JavaFX before, so I got a decent introduction to how it works. Got it quickly up and running, and I could see the results immediately, did not have to hassle with dull things that probably would have taken more time if I did not have a framework to start with. Since I started from scratch, then I would say that I learned quite a bit".

The participants did not successfully learn mathematical and physical thinking while using PhysicsWorldFX, they said that they felt more like the values they used when setting up nodes were based on trial and error to get the wanted physical properties. They did not get much experience using PhysicsWorldFX to program more advanced properties, so they might have given a different answer if they had gotten more time with PhysicsWorldFX, but in order to improve their physics thinking, values used in PhysicsWorldFX should be based on real world numbers, for example gravity on earth can be expressed by g = 9.80665 m/sˆ2, so the value of 9.8 for gravity should give a realistic gravity value.

PhysicsWorldFX did help the participants work with the computational thinking dimension of computational practices by letting the participants being incremental and iterative in their work. One participant said that “It was very comfortable to see that you could get things to move, adjust properties and get things to fall and so on. That you could go step by step and see that it was working for each step.”, two other participants agreed to this point. Another element of computational practices used was reusing and remixing, PhysicsWorldFX has demo projects that the users can look at and reuse to further improve or remix into their own game. A participant pointed out that the demo projects also served as good examples when learning what the framework was capable of.

When designing PhysicsWorldFX the learning context for the framework was pre-university students from 14 years old and up. When the participants were asked if the framework could work

in a university learning environment, they thought that it was well suited for this purpose, since it was easy to understand, and you can quickly get the graphics up and running so the students can start with the controller programming in JavaFX where most of the programming learning happens. The participants also thought that PhysicsWorldFX also could work in the environment it was designed for. One participants said that it could be a good fit with younger students and that it could be perfect for a 13-year-old acquaintance, which is the lowest age that some children can understand programming in Greenfoot. The framework was also compared to Scratch by the participants, and said that PhysicsWorldFX was capable at making more advanced games. Based on these data, PhysicsWorldFX can work in both Kodeklubben and in early university courses.

The participants were asked if they learned more about JavaFX while using PhysicsWorldFX, but answered that they did not feel like they expanded on their knowledge, but that the way they worked with PhysicsWorldFX was in accordance to what they were used to. That PhysicsWorldFX was in accordance with how JavaFX worked, so that could mean that the design of PhysicsWorldFX does not break any JavaFX conventions, which again makes the transition to JavaFX user interface programming relatable. PhysicsWorldFX also uses CSS, and it is possible to style PhysicsWorldFX-objects using CSS, and at least one participant liked the idea of using that approach to making shared resources during the testing.

A couple of participants reported that understanding how the `gravity` properties worked was a bit hard. How a gravity value affects the simulated properties in the world. During testing the gravity properties were named `gravityX` and `gravityY`, which together formed a gravity vector, so by defining a vector of v= (0, 1000), you have a vector that points down with a force of 1000. This could be improved by renaming the properties, and using more realistic values for the force like described earlier.

An issue that challenges the usability in SceneBuilder with PhysicsWorldFX, is the need to set up points to define a polygon or polyline shape directly in the FXML via an IDE outside of SceneBuilder. The participants found that shapes using points were harder to understand, because when they dragged a shape which has its visual geometry defined by multiple sets of coordinates into the workspace, the shape was initially invisible in SceneBuilder, and to understand and see the shape, the participants had to manually define the points inside their IDE. A way to improve

this is to give the shapes hard coded default values in the code definition, this way the shape is visible and the users will have initial points that they can modify.

Some of the participants had trouble understanding how SceneBuilder worked with setting properties on nodes, and during testing the documentation did not have any instructions on how to use SceneBuilder to find relevant properties. This can be explained by that the participants had limited experience with SceneBuilder, and were more used to doing things programmatically by writing code or editing the FXML files directly, so they could have gotten more out of the framework if there was a quick guide to where relevant properties could be found, and how they could be edited.

When testing PhysicsWorldFX the users asked about features that were not yet planned or considered. One participant suggested to make a web site for PhysicsWorldFX, with the reasoning that it would expose the framework to the public, and provide information on the framework, which could be guides, tutorials, examples and other types of documentation for the framework.

Media support was also asked about, and this was thought about during the literature study in the beginning of this project, but this functionality was not prioritized because of time constraints. Media support like textures, animations and sound could be added by using third party libraries, but it could be interesting to see if this could be set up easily in FXML or in SceneBuilder.

### System Usability Scale

The System Usability Scale forms were analyzed by calculating a SUS score for each of the forms. This was done by recalculating each question in the form from 0 to 4. Odd numbered questions were recalculated to a score by the answered number minus 1. Even numbered questions were recalculated to a score by 5 minus the answered number. All ranged questions were then summed which gave a max score of 40. To get the scores in the range of 0 to 100, the score was multiplied by 2.5.

| *Participant* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| *Score* | 87.5 | 67.5 | 87.5 | 92.5 | 87.5 | 85 |

**Table 5-1: SUS results from usability testing**

The form scores from Table 5-1: SUS results from usability testing *84.58*. This score can be labelled as a subjective label by using a table from (Bangor, Kortum et al. 2009) where results

from 2324 SUS surveys in 206 usability tests were analyzed and put into perspective of a subjective label range.

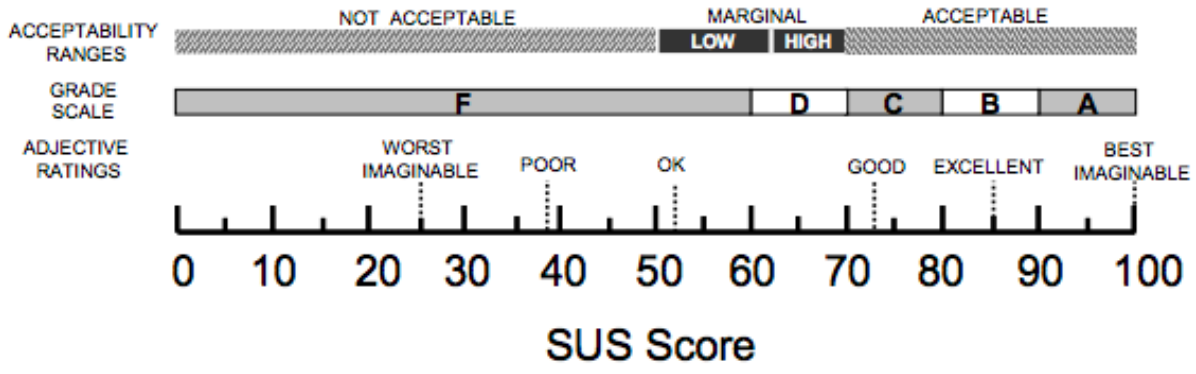


**Figure 5-27: SUS Score table (Bangor, Kortum et al. 2009)**

The table gives this test scores the subjective label "Good", the grade "B" and within the Acceptable range.

## Discussion

In order to answer RQ1: "How should a framework for 2D game objects with movements and interaction based on physics with JavaFX be designed for educational purposes in a way that encourages students to learn programming?" the design decisions must be discussed, and in order to answer RQ2: "How and to what degree does the framework affect the learning process for the students?" the analysis of the data gathered during the evaluation is relevant for discussion.

### Discussion of RQ1

The design of the prototype was based on the JavaFX centric design where users program game and physics logic in the controller class and call getters and setters on the JavaFX node they wish to update. This way of working with the framework worked well, and made the users work in a way they were already used to in JavaFX. The other designs; mediator pattern and single shared mediator could also have possibly worked just as well, but in addition to learning the framework the users would have to learn how the design pattern used worked with the framework, so it would not have been an optimal choice given that the users have already learned JavaFX basics before working with the prototype.

In a context where the users have not been exposed to JavaFX before trying the prototype either design pattern could be understandable, but the knowledge the users gain though using the prototype would not have been compatible with making graphical user interfaces with JavaFX, so with the JavaFX centric design the knowledge the users gain is directly transferrable to regular JavaFX programming. Using the prototype in SceneBuilder was also comparable to how Alice, Scratch and Greenfoot worked which were proven concepts that worked, and getting the workflow with the prototype as similar as possible required the prototype to be designed in the JavaFX centric design.

Judging on how successful the prototype was in capturing how JavaFX worked can be hinted to from the data collected during the evaluation, when asked if they learned more about how JavaFX worked, one participant answered, "Not really, it worked the way I was used to…" and another participant who had limited experience with JavaFX though that knowledge was gained, but since there is no way to validate that the knowledge gained was in accordance to the way JavaFX works, this does not necessarily speak for or against the prototype design.

During the design and implementation of the prototype several example games were made, and to discuss how complex the prototype is to use, the analysis of the example games can serve as data for this discussion. The games showed that using the prototype not much physics logic was needed to program the desired functionality in the JavaFX controller and that the prototype supported multiple levels by reusing the controller and changing the view. So, the user could focus on game logic, which was very simple for many of the games, therefore the programming needed to create such games as Pong, Breakout, Lunar Lander, Asteroids and Jetpack is not very complicated. Not many lines of code were needed either for a pre-university student or a university student who is learning JavaFX. Each game also increased a little in complexity as more functionality was needed, so a user could get a gradual exposure to the framework by testing functionality used in the example games. The example games can also serve as scaffolding where the users can remix and reuse the games to create new games.

### Discussion of RQ2

The evaluation of the prototype gave a lot of useful data on how the participants experienced the prototype and how it affected their learning process. In relation to RQ2: "How and to what degree does the framework affect the learning process for the students?", the participants enjoyed working

with the prototype and they liked seeing that their code adjustments could be seen visually by running their game or simulation. Their learning process was affected in a positive way, they were having fun while programming and their learning curiosity was stimulated when using the prototype. Another contributing factor to that the participants were enjoying themselves could have been that they were in a group setting where they were learning by doing the usability test tasks together.

The participants in the evaluation did have a lot of questions for the prototype after the group interview as well, indicating that they liked the idea of the prototype, and were thinking of ways to use it, one participant asked about media support and one participant suggested making a web site to promote the prototype so new users could discover it and to give information on how to use it.

The evaluation also uncovered some design problems with the prototype, the values used for parameters like gravity and force in the usability tasks had no units connected to them, so when the participants programmed the tasks, they did not think about the values as physical values, but just values they had to adjust to get the correct behavior. This is a complex issue with no apparent solution, but a way to make them think in a physics world, could be to always use units for the values used in examples and code documentation. The participants also reported that they did not have to think about mathematics when using the framework, but this could have been because they got limited time with the framework, and did not have time to program any complex functions where they would have needed to do math. When programming games with the prototype, some math will eventually need to be programmed.

The learning environment this prototype might fit well in could be in a Kodeklubben course for pre-university students which was the original target learning environment, but also in a university environment for learning JavaFX. The participants in the evaluation were university students, and they liked working with the prototype, they could also imagine this prototype being used in both a university environment and a pre-university environment like Kodeklubben. There is nothing that makes the validity of the qualitative data gathered during the evaluation questionable, but this research cannot exclude the possibility that the participants were biased in favor of the prototype since they tried the prototype in an artificial university setting, and because the researcher was one of their peers.

Working with the prototype in SceneBuilder had some issues, SceneBuilder's build-in logic for shape classes did not work for new physics shapes that inherited from the shape classes, the `PhysicsRectangle` did not get default width and height values like a normal `Rectangle` did. The `PhysicsWorld` class extended the `Pane` class, but the drag and drop functionality did not work for moving nodes within `PhysicsWorld`, instead a `Pane` had to be placed inside `PhysicsWorld` and nodes could be moved inside this `Pane`, or the user could just enter coordinates for the node placement manually, but this takes away from the functionality of SceneBuilder. This suggest that SceneBuilder should be modified to support classes that inherit from base classes with SceneBuilder specific functionality.

The usability says a lot about how well the design works and if the users liked working with the prototype. The analysis of the system usability scale results gave this prototype a letter grade "B" and a subjective label "Good" which was in the acceptable range for SUS results on whether the prototype was usable or not. An explanation for this result could be that the design worked as intended, but that there is room for improvements, and this is backed up by the analysis of the data from the evaluation as well. The usability form was filled out after the group interview, so the participants might have had their opinions of the prototype affected by this, and this could explain the high SUS score. The SUS result can be used as a comparison tool for further improvement by comparing new SUS scores to these results.

# 6. Conclusion

Conclusion to the research and recommendations for future work.

## Results

The discussion of the results showed that the design decisions done during the design and implementation stage of the research was based on how a student can learn and use the prototype optimally. These design decisions were based on letting the user explore freely using the tools the prototype provided where the user can see the code result visually. The discussion provided enough information to answer RQ1: "How should a framework for 2D game objects with movements and interaction based on physics with JavaFX be designed for educational purposes in a way that encourages students to learn programming?" and the conclusion was that it should be designed in a way where the student can use the full functionality of SceneBuilder, and where the user can learn JavaFX by making their own games, in an environment where the most difficult programming is already integrated into JavaFX. The prototype design achieved this, but with some flaws with SceneBuilder integration that has been acknowledged. The prototype and the example games demonstrated that games can be made with very little and simple controller logic.

The discussion also showed that the participants in the evaluation enjoyed working in the SceneBuilder and JavaFX environment to make games. Based on the results in the discussion, the research question RQ2: "How and to what degree does the framework affect the learning process for the students?" can be answered qualitatively based on observations and group interview data, that the prototype affected the learning process positively in the way that it made them work iteratively, and that the users had fun while working with the prototype where they could see the effects of their code visually. To what degree the prototype affected the learning process is hard to give a final conclusion to, but since the SUS scores interpreted that the prototype was acceptable, good and in the grade range of B, it definitely did not affect the learning process in a negatively way, which was also supported by the qualitative data.

## Recommendations for future work

This research has been through an iteration of the Design and Creation Research strategy, and to further ensure that the data from this research is valid, new research should go back to the field and further evaluate and improve on the prototype design. The prototype design should also be

tested in an environment with pre-university students. This will ensure that the design is thoroughly tested, and that the prototype design works for the whole range of the target users, not just university students.

It is also recommended to work with the compatibility of SceneBuilder with custom frameworks like the prototype. Functionality in SceneBuilder was not compatible with classes that extended classes that SceneBuilder had special functionality for. This can be improved by modifying SceneBuilder to work in such cases. SceneBuilder is open source, so anyone can contribute to the development, but there is no guarantee that the changes will make it into a SceneBuilder release.

This research shows that it is possible to use real world programming tools in educational game programming, and it might be worth looking into other programming languages with similar tools that can be utilized for educational game programming.

# 7. References

Ackermann, E. (2001). "Piaget's constructivism, Papert's constructionism: What's the difference." Future of learning group publication **5**(3): 438.

Bangor, A., et al. (2009). "Determining what individual SUS scores mean: Adding an adjective rating scale." Journal of usability studies **4**(3): 114-123.

Brennan, K. and M. Resnick (2012). New frameworks for studying and assessing the development of computational thinking. Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada.

Chang, K.-E., et al. (2008). "Effects of learning support in simulation-based physics learning." Computers & Education **51**(4): 1486-1498.

Conway, M. J. (1997). Alice: Easy-to-Learn 3D Scripting for Novices, Faculty of the School of Engineering and Applied Science, University of Virginia, Diss., Dezember 1997.

Dann, W., et al. (2012). Mediated transfer: Alice 3 to java. Proceedings of the 43rd ACM technical symposium on Computer Science Education, ACM.

Faulkner, L. (2003). "Beyond the five-user assumption: Benefits of increased sample sizes in usability testing." Behavior Research Methods **35**(3): 379-383.

ISO (1998). 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs). Part 11 Guidance on usability.

Kölling, M. (2010). "The greenfoot programming environment." ACM Transactions on Computing Education (TOCE) **10**(4): 14.

Kurland, D. M., et al. (1986). "A study of the development of programming ability and thinking skills in high school students." Journal of Educational Computing Research **2**(4): 429-458.

Lye, S. Y. and J. H. L. Koh (2014). "Review on teaching and learning of computational thinking through programming: What is next for K-12?" Computers in Human Behavior **41**: 51-61.

Malan, D. J. and H. H. Leitner (2007). "Scratch for budding computer scientists." ACM SIGCSE Bulletin **39**(1): 223-227.

March, S. T. and G. F. Smith (1995). "Design and natural science research on information technology." Decision support systems **15**(4): 251-266.

McLellan, S. G., et al. (1998). "Building more usable APIs." IEEE software **15**(3): 78-86.

Nielsen, J. (2000). Why you only need to test with 5 users, Useit. com Alertbox.

Oates, B. J. (2005). Researching information systems and computing, Sage.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas, Basic Books, Inc.

Pea, R. D. (1987). "Logo programming and problem solving."

Peffers, K., et al. (2007). "A design science research methodology for information systems research." Journal of management information systems **24**(3): 45-77.

Resnick, M., et al. (2009). "Scratch: programming for all." Communications of the ACM **52**(11): 60-67.

Sova, D. H., et al. (2003). "233 Tips and Tricks for Recruiting Users as Participants in Usability Studies." URL: http://www.nngroup.com/reports/tips/recruiting/234_recruiting_tips.pdf.

Utting, I., et al. (2010). "Alice, greenfoot, and scratch--a discussion." ACM Transactions on Computing Education (TOCE) **10**(4): 17.

Vaishnavi, V. and W. Kuechler (2004). "Design research in information systems."

Wing, J. M. (2006). "Computational thinking." Communications of the ACM **49**(3): 33-35.

## Appendix A: Guide to GitHub Contents

The source files for PhysicsWorldFX and the example games are hosted at the GitHub address: https://github.com/kristianw87/PhysicsWorldFX

This repository contains a readme, license, the source code and wiki pages.

The framework is in the PhysicsWorldFX/src/framework folder. This folder holds the `PhysicsGame` class and `PhysicsWorldFunctions` class and contains the following subfolders;

- events, the event listeners created for the framework
- geometric, geometric property factory and interface for geometric functionality
- nodes, the implemented JavaFX nodes which can be used in SceneBuilder
- physical, physical property factory and interface for physical functionality

The example games are in their own respective folders;

- Asteroids
- Breakout
- Jetpack
- LunarLander
- Pong

A testbed is also included where the most basic scaffolding is set up to get started with the framework. It is in the Testbed folder.

The wiki pages contain documentation used during the usability testing of the prototype;

- A getting started guide
- Documentation for PhysicsWorldFX
- The usability tasks used for usability testing

## Appendix B: Usability Tasks

Using the available documentation and examples perform these tasks.

1. **Creating a non-movable rectangle**

Add a rectangle at the bottom that covers the width of the world and is 10px high. The rectangle should not be able to move.

2. **Create a falling triangle**

Add a triangle in the middle of the world and make it fall onto the non-movable square when the application starts. Use a gravity value of 1200.

3. **Make a geometrical composition**

Add a geometrical composition in the middle of the world and use the triangle and a new rectangle as the geometrical composition children.

4. **Apply a force to the center of mass of the geometrical composition**

When space is pressed, apply a force in the v=(0, -1) direction with a force of 1500. Use the physics step event listener and check if space is pressed, then apply the force.

When left is pressed, apply a force in the v=(-1, 0) direction with a force of 500. When right is pressed, apply a force in the v=(1, 0) direction with a force of 500.

5. **Add a rectangle with a style class**

Add a rectangle at the bottom right corner of the world. Add a style class to the rectangle named “endblock”. The square should not be able to move.

### 6. End game on collision with endblock

When the geometrical composition collides with the rectangle containing the style class “endblock”, stop the game/simulation. The PhysicsWorld class has a method to end the game, and the PhysicsGame class has events that are raised when PhysicsWorld finishes the game/level.

## Appendix C: Group Interview Questions

1. What experience have you had with JavaFX before this trying PhysicsWorldFX?
2. What are your thoughts about PhysicsWorldFX?
3. Did you have any problems when using the framework?
4. Do you have any suggestions for improvements to the framework?
5. What did you think about working with SceneBuilder to set up the physics nodes?
6. Do you have any suggestions for improvements to SceneBuilder?
7. In what way did using the framework affect your learning process of JavaFX?
8. Have you learned more about the way JavaFX works by using the framework?
9. Have you learned more about math and physics by using this framework?
10. Does the framework make you work more iteratively?
11. In what context do you think this framework can be used?
12. Any additional thoughts?

## Appendix D: Coding of Qualitative Data

Green: Segments that are relevant to the research objectives
Yellow: Segments that provide context to the research
Red: Segments that are not related to the research

Categories:

1. Existing skills in programming
2. High complexity getting started
3. Learning
4. Learning context
5. Usability of the framework
6. Usability of SceneBuilder used for the framework
7. Improvements

Observasjoner under testingen:

**1: Existing skills in programming**:
16:30: Alle har kommet og jeg startet med å informere om opplegget som var planlagt, der etter sette opp utviklingsmiljø hos alle deltakerne. 4 stk brukte IntelliJ og 2 stk brukte Eclipse. De med IntelliJ fikk opp utviklingsmiljøet opp kjapt, mens de med Eclipse trengte mer hjelp og brukte lenger tid.

**2. High complexity in getting started:**
17:10: Alle har fått prototypen opp å kjøre, så begynte med å gå gjennom et kjapt eksempel på hvordan man setter opp og bruker prototypen. Hadde problemer med å påvirke et rektangel med en kraft, og det viste seg at kraften jeg satte på rektangelet var for lite. Men testet på et polygon, og da fungerte det.

17:30 Jeg la ut usability test tasks på wiki siden til prototypen, og deltakerne har fått instruks om å gjennomføre disse oppgavene. De ble også informert om at poenget med oppgavene var å eksponere de for funksjonalitet i prototypen og ikke teste om de fikk det til eller ikke.

**3. Learning**
17:40: Deltakerne har kommet i gang med oppgavene, og har ikke spurt om noe enda. Det ser ut som at de liker å se at fysikken fungerer, at dette gir en morsom tilbakemelding til brukeren.

**6. Usability of SceneBuilder used for the framework**
17:48: Spørsmål om hvordan man setter opp polygon koordinater. Det er litt problemer med å sette opp polygon koordinater, så jeg måtte forklare hvordan dette måtte gjøres. Det ble litt problemer hos en bruker da double ikke ble importert inn som en using i fxml filen automatisk, så måtte gjøre dette manuelt. Kun Eclipse brukerne som hadde dette problemet.

**5. Usability of framework**
**6. Usability of SceneBuilder used for the framework**

18:10: Oppfatning av SceneBuilder er litt vanskelig for enkelte av deltakerne, så jeg måtte forklare hvordan man setter properties med verktøyet. Noen av deltakerne hadde mindre erfaring med SceneBuilder.

**1: Existing skills in programming**

18:14: Noen klarer seg bedre enn andre, og de med IntelliJ har kommet litt lenger enn de med Eclipse. De med Eclipse hadde flere problemer med utviklingsverktøyet, mens de som har IntelliJ er godt drevne på bruk av verktøyet kan det se ut som.

18:24: Hjulpet til med else if logikk i spilløkken, var et spørsmål om hvordan man fikk logikk mot flere tastetrykk til å kjøre samtidig.

**5. Usability of framework**

18:28: Forklart hvordan styleclass fungerer og hva man kan bruke de til. Brukeren var veldig interessert i å kunne bruke styleclass til css, og ville gjerne teste dette. Men forklarte også at det kunne brukes til å identifisere en type objekt med en type funksjonalitet i controller klassen.

18:35: Testingen ble avsluttet, vi rakk ikke å gjøre noe kreativt med prototypen utenom å teste usability test tasks.

Gruppeintervju:
Spurte om noen hadde noen motsetning til at det ble gjort et lydopptak, og informerte om at lydopptaket kom til å bli transkribert, så slettet. Informerte også om at alle sammen kom til å være anonyme.

**1: Existing skills in programming**

I: Hvilke erfaringer har dere hatt med JavaFX før dere testet ut dette API-et?
M1: Har hovedskalig jobbed med UI-oppsett og sånt, ikke noe erfaring med spill i JavaFX.
I: Var det greit å komme i gang med API-et?
M1: Ja, det var ganske greit å komme i gang.
I: Hva med deg M2?
M2: Har brukt det litt i object orientert, og litt i MMI. Og hold på litt med XML i Java før, så det var litt kjent. Ja.

**2. High complexity in getting started:**
**7. Improvements**

I: Har dere noen umiddelbare tanker rundt bruken av API-et? Om det va tungvidt eller om det fungerte greit, noen generelle tanker?
M4: Hadde vært kult om du hadde en bedre get started guide hvor man gjør et enkelt prosjekt. Kan også ha en nettside for å tiltrekke folk, hadde gjort det litt lettere, for da vet man hva prototypen kan gjøre, i stedet for å gjøre oppgavene vi fikk.
K1: Det hjelper jo litt med de prosjektene som ligger der. Men ellers så..
M1: Ja, kunne hatt litt tydeligere forklaringer på hva de ulike objektene som ligger der gjør og hva man bruker dem til. Men det går mer på dokumentasjonsbiten.
M2: Tror det hadde vært kult med sånn som han sier, få en guide gjennom et prosjekt, så får du automatisk sett på hvordan du setter opp et prosjekt.

I: Høres lurt ut det.

**5. Usability of framework**

M5: Men selve prototypen virker greit, når du først greier å sette opp objektene og kan å bruke det, så kan du lage enkle spill og teste ut ting.
M2: Liten sånn lag når man begynner med prototypen, men når man først har kommet inn i det, så går det greit.

**6. Usability of SceneBuilder used for the framework**

M4: Hvis det skal brukes i læringssammenheng, så anbefaler jeg å lage en guide som kun går mot det grafiske, og ikke på xml biten. De som gjør sånt vil gjerne bruke kraftigere rammeverk.

**5. Usability of framework**

I:Hva var de største problemene dere opplevde med prototypen? Var det noe spesielt som skilte seg ut som var vanskelig, som godt kunne vært enklere?
M3: Det med tyngdekraften, at det var litt forvirrende å forstå hvordan tyngdekraften virket.
M2: Følte jeg hadde litt flaks med hvordan jeg fant ut av det. Prøvde litt forskjellige ting.

**6. Usability of SceneBuilder used for the framework**

M4: Når jeg lagde de fysikkobjektene, så kom de ikke opp på skjermen som at de var noe, og da måtte jeg inn i xml-en og sette opp en høyde og en bredde, og så gå tilbake igjen får å se de. (Refererer til SceneBuilder)
M1: Kunne hatt noen startverdier på de, så ser man i alle fall at man har dratt inn noe.
Jeg vet ikke hvor lett det er å sette opp points i SceneBuider, men man måtte sette de inn i rett rekkefølge, og hvilket point blir det her i forhold til det andre. Det var litt vanskelig.

**6. Usability of SceneBuilder used for the framework**

I: Litt mer spesifikt mot SceneBuilder biten, var måten man satte opp objektene med drag and drop funksjonaliteten grei?
M1: Følte det gav mening sånn som jeg var vandt til å bruke det.
Fra flere, litt utydelig: Mhm, ja.
M1: Sånn som hvor jeg fant layout og properties og sånt.
M2: Hvis elementene hadde hatt en default størrelse, slik at de ikke var usynlig når man satte de opp.
M4: Kanskje en liten get started guide der også på hvordan SceneBuilder fungerte.
M3: Syntes det var lett å skjønne hva de ulike klassene gjorde. Syntes det var oversiktlig.

**3. Learning**
**5. Usability of framework**

I: Lærte dere mer om hvordan JavaFX fungerte når dere brukte prototypen, i forhold til det dere var vant til fra før?
M1: Ikke personlig, det fungere sånn som jeg vandt til sammenkoblingen, så det blir bare å skrive funksjoner.
K1: Det kan være anderledes hvis man ikke har vært borte i det fra før.
M5: Jeg har jo ikke brukt det før, så jeg fikk jo en grei innføring i hvordan det fungerte. Fikk det kjapt opp og kjøre, fikk se resultatene med en gang, slapp å sitte å knote med ting som kanskje

ville tatt lengre tid hvis jeg ikke hadde hatt et rammeverk. Så siden jeg startet i null, så vil jeg si jeg lærte en del i alle fall.
M4: Var bra at det ikke krasjet når man gjorde feil i xml, det funket nesten samme hva man gjorde.

**3. Learning**
I: Måtte dere tenke litt mer med matematisk og fysisk tankegang når dere jobbet med prototypen?
M1: Følte det var sånn at man måtte prøve seg frem, og så den tredje gangen så funket det. Man kunne brukt tankegangen hvis man hadde hatt det i bakhode, men det var ikke nødvendig.
I: Noen andre som har noen tanker rundt den biten?
(Ingen videre kommentarer)

**3. Learning**
I: Jobbet dere iterativt med prototypen? Det blir kanskje litt som med det du sa om prøving og feiling, og ser at det fungerer. Syntes dere det var en morsom måte å jobbe på?
(Ja fra flere personer)
M1: Det var veldig behagelig at du fikk ting til å bevege seg, justere på properties og få ting til å falle og så videre. At man kunne gå steg for steg og se at det fungerte på hvert steg for å få alt til å fungere.
M2: Det å kunne teste at noe funker ved å se at det skjer noe på skjermen.
K1: Tror det er noe som kan motivere folk.
M5: Ja, er enig, det øker motivasjonen at man ser noe, at man får lyst til å lære noe når men får noe opp å kjøre. Ikke bare sitte å knote med en tutorial, og så har man glemt noe, og får ikke noe som helst opp og kjøre.
M4: Hente lavthengende frukt i starten, så ser man resultater. Så får man lyst til å lære mer og mer.

**4. Learning context**
I: Hvilken kontekst kan dere tenke dere til at prototypen passer inn? Nå vet dere jo at det er tiltenkt i læringssammenheng, men tror dere det ville passet godt til dette?
M1: Ja det er ganske lett å starte med, lett å komme i gang. Det er fort med å, du får veldig fort opp den spillbiten og fxml som er den kjedeligste biten å lage, så kan man gå rett på kontroller programmeringen som er litt mer direkte koding, jeg vil i hvertfall si det gjør det litt mer interessant. Når man kan hoppe over oppsettet med å sette opp en physics engine når man har det fra bunnen av, så går ting mye fortere.

**7. Improvements**
I: Noen som har flere tanker rundt prototypen som ikke er nevnt?
M4: Du må lage en nettside til prototypen, en skikkelig kul en. Siden det finnes mange rammeverk der ute, så vil en nettside synliggjøre prototypen bedre, og da gjøre et bedre førsteintrykk. Ta inspirasjon fra react. Lage noe lignende. Tror jeg vil slå an.

**4. Learning context**
K1: Jeg tror det kanskje ville passet bra inn hos yngre elever også, siden det virker ganske enkelt å komme i gang med.
M4: Lettere enn Scratch i alle fall, og det er jo rimelig stort.
K1: Ja, du ser mye mer enn, ja du ser kanskje mer på Scratch.
M4: Kult å kunne lage Asteroids, du kan ikke lage det med Scratch.

K1: Nei. Ja ikke sant.
M4: Og om man vil kan man gjøre det mer avansert også. Den blandingen der som er ganske fin.

I: Vi har vært innom de fleste punktene jeg hadde til gruppesamtalen, så da kan vi si oss ferdig med den formelle biten, men vi kan gjerne fortsette å snakke litt, siden vi har litt tid igjen.
M2: Hva slags læringsarena tenkte du på når du lagde prototypen?
I: Jeg tenkte jo på kodeklubben når jeg laget dette, og hadde egentlig planlagt å kjøre en test sammen med elever i kodeklubben. Og da e dæm fra ca ungdomskoleelever og oppover. Så det er tiltenkt de når jeg designet og laget dette prototypen. Men veilederen min fikk ikke til Javafx kurs til kodeklubben i år, så jeg måtte være litt kreativ og finne på noe alternativt, og da var det å rekruttere studenter for å prøve prototypen. Siden det blir en litt lignende situasjon hvor man har litt Javafx erfaring fra før, og så kommer inn i prototypen. Og det vil jo være varierende med kunnskap for slike elever i en kodeklubb, så fant ut at det var en god tilnærming å bruke studenter. Selv om studenter er litt mer dreven på programmeringsbiten. Tenkte også at det var en grei plass å samle data, om det kanskje kan brukes i en sammenheng med universitetsstudenter også.

**3. Learning**
K1: Tror du de må ha noe kunnskap om Javafx fra før, eller tror du de vil kunne lære Javafx gjennom prototypen?
M1: Det virker som en grei plass å starte, men må lære litt mer om kontrollerbiten hvor man trenger litt mer kunnskap om Java programmering. Det har jeg erfart med UI laging.
M4: Kult spill i alle fall (Referer til demoene som ble laget til prototypen).
K1: Likte Astroids
I: Asteroids er vel det mest komplekse spillet.
M4: Jetman var fancy også.

K1: Fikk ikke helt det til.
M3: Hvis du lander mellom spikesene, så kan du stå der til jetpacken har ladet opp igjen. (Referer til at man kan lande mellom to spikes, og at man ikke mister liv når man står der)
M2: Hvordan har du laget fysikken i spillet?
I: Fysikkmotoren er JBox2D, og denne ligger i bakgrunnen for prototypen

**7. Improvements**
M3: Finnes det muligheter for å legge til lyd også?
I: Det er ikke mulig med prototypen nå, men man kan bruke tredjeparts bibliotek for å legge til mediastøtte. Dette havnet utenfor scopet på oppgaven. Men absolutt noe som kunne fungert.

M1: Jeg rakk ikke komme så dypt inn i kontrollerbiten, men klarer den flere tastetrykk samtidig?
I: Ja, hvis du ser på Asteroids f. eks, så ser du at man kan holde inn space og skyte mens man flyr romskipet.
M1: Har du funksjoner for å registrere at en knapp er blitt trykket kjapt, slik at man kun kjører logikken en gang for tastetrykket?
I: Man kan gjøre dette via JavaFX sin key event, så prototypen tilbyr ikke egen funksjonalitet mot dette.
**Lydopptak slutt** tid 25 min.

Litt løs diskusjon om informatikk og data.
Gir deltakerene SUS skjema.
Fyller ut skjema mens brus blir hentet av I.
Pizza kommer og I går å henter.
Samler inn SUS skjema som er fylt ut.
M2 må dra.
Informerer om kompansasjon, og at de kan sende meg telefonnummer for overføring.
Takker M2 for at han kunne delta.
Opplegget er ferdig og vi blir sittende å spise pizza og snakke videre urelatert til prototypen.
Folk begynner å dra, takker alle sammen for at de kunne delta.

**4. Learning context**

K1 mens hun drar: Jeg har en nevø på 13 år som jeg tror dette prototypen kan være midt i blinken for.