A Stealth Learning Approach:

Teaching Programming Concepts by Building Games

Author: Michel Fiege

Student number: 1262955

MSc Science Education & Communication

July 12, 2011

TU Delft Delft University of Technology

Abstract

Building games in introductory programming is seen more often. Although it motivates pupils to learn, less is known regarding learning results. In this study the learning results on programming concepts in introductory programming are described. The study involved one group of 30 pupils in the age of 12-13 from secondary education in The Netherlands. Over a period of 10 lessons pupils build 6 small games by completion using new online learning material for Game Maker. They also created a game themselves. In the process of creating online learning material, an automated approach for creating Game Maker manuals was developed. The results show that pupils developed a basic understanding of some programming concepts in their first programming experience.

Preface

Games and education: I could not wish a better combination of subjects for my graduation project. In September 2007 I started teaching computer science in secondary education at the RSG Hoeksche Waard in Oud-Beijerland. I experienced the use of games in the classroom and especially the creating of games. Pupils like to build games and given the tools available nowadays it is easy to setup game making courses. Every week I now see pupils from 10 to 18 years old having fun building games.

So when in November 2009 I started thinking about a graduating project I contacted Dr. Rafael Bidarra to see if we could come up with an idea involving games. Over time the idea changed and developed into a web application as described in chapter 4. I would like to thank Dr. Rafael Bidarra for many things, but most of all for his patience as he has been involved from the very beginning.

In November 2010 my graduation project really started. I would like to thank my supervisor Drs. Martin Jacobs for his support, feedback and guidance. I learned a lot from him. I would also like to express my gratitude to Ir. Hans Geers and Drs. Martin Bruggink for the discussions we had. Those talks put me back on track and provided useful insights. Finally, I would like to thank my colleagues at the RSG Hoeksche Waard and in special Marcel Eggen. It really helped that I could interview pupils during their lessons.

Michel Fiege
Rotterdam, The Netherlands
July 12, 2011

Contents

Chapter 1

Introduction

This graduation project consists of two parts: in-class research and software development. In this chapter the motivation and problem description for both parts are given followed by an overview of the chapters in this thesis.

1.1 Motivation and problem description

Games are used more and more in education. Not only are games being played in an educational context, games are also being built. In the process of building a game the learner is confronted with different programming concepts such as *objects, conditional statements* and *variables.* As a teacher in computer science in secondary education it is encouraging to see that building games motivate pupils [26]. Motivated pupils learn better and a game building course as introductory programming is therefore attractive. Especially when in this way besides boys also girls are attracted to the area of computer science [14]. However, although building games motivate and stimulate pupils to use deep learning strategies [30], less is actually clear regarding learning outcomes. This raises the question: *what do pupils actually learn from building games in terms of programming concepts?*

In order to study these questions game-related learning material was created. However, creating learning material is a time-consuming task and it is often difficult to write and structure the material in a consistent way. Also, problems can arise when the material needs to be updated or delivered to the learners. Would it therefore not be easier if game-related learning material could be generated in an automated fashion?

1.2 Thesis overview

In chapter 2 related research is described and different tools for building games are compared. In the end of the chapter the research question is further elaborated. In chapter 3 the learning material that was created is described. In chapter 4 a new approach in creating game-related learning material is found. This is the software development part of the thesis. Chapters 5 and 6 describe the research method and results of the in-class study. In chapter 7 conclusions are drawn and the discussion can be found in chapter 8. Finally, a small reflection on the author's learning can be found in chapter 9.

Chapter 2

Related research

In this chapter research related to teaching introductory programming and tools and trends in doing so are described. Three frameworks for building games are then introduced and results of research regarding these frameworks on programming concepts are described. One framework is chosen for the introductory programming course and finally the research question is elaborated.

2.1 Teaching introductory programming

An objects-first approach in teaching introductory programming received a lot of attention in recent years. Compared to the more traditional imperative-first or functional-first approaches, objects-first starts with concepts like objects and inheritance. More traditional concepts like control structures are introduced later on in an object oriented context [7]. In teaching objects-first, programming is taught 'top-down' and emphasizes the design of a program. This reflects the way how humans solve problems [9]. At universities, colleges and high schools worldwide an objects-first approach is nowadays used most often [27].

Besides the object-oriented programming paradigm being used, there are many tools that visualize programming using graphics and animation. Visualization can help instructors in different ways. For example concepts can be explained easier, the attention of learners is easier grabbed and more material can be covered in less time [3]. In short, visualization can aid in learning to program. Especially the argument that programming requires both halves of the brain, thus also involving the artistic half and building mental images of concepts, is found in earlier work [3, 6, 18].

There are multiple ways to visualize programming [18]. Graphics can for example clarify the execution path or states of a program. The term used for this type of visualization is 'program visualization'. In this report the focus is on 'dynamic' visualization where a program is visualized in runtime, instead of 'static visualization' where snapshots of the program are taken at certain points in the execution. Examples of systems that dynamically visualize a program and have found their use in education are *Turtle graphics in Logo* and successors like *Karel the Robot, JavaLogo* and *PythonTurtle*. The main idea of these systems is that the learner controls an object, like a robot or pen, and sees the execution of the program being drawn.

Another type of visualization is 'visual programming'. Here graphics aid in building the program itself. Typing syntax is minimized as the learner uses graphical elements to create the program. For example boxes and arrows, lines, flowcharts, icons, forms or diagrams are used to assist the user. However, the program's execution is not visualized. *BlueJ* [15] is an example of a visual programming system that also takes the objects-first approach. The learner uses forms and class diagrams to start programming, focusing on the structure and design of the program.

A 'visual programming language' is a system that includes both 'programing visualization' and 'visual programming'. In education *Alice*, *Game Maker* and *Scratch* are examples of such systems. These systems, or frameworks as they are called from now on, provide an environment for learners to build computer games.

The objects-first approach goes hand in hand with programming games. In a game everything is an object: walls, coins, characters, etcetera. Programming the behavior of these *objects* is like thinking what happens to *objects* when certain *events* occur like pressing the arrow keys on the keyboard. Concepts like *objects, events* or even difficult concepts like *inheritance* come really natural in building games [20].

2.2 Building games in education using frameworks

The 'visual programming languages' Alice, Game Maker and Scratch are now introduced. These frameworks provide a drag-and-drop environment which makes programming more accessible as there is no need to learn any syntax. The frameworks are used widely in education and have large online user communities.

2.2.1 Alice

Alice 2.0[1] is an open source 3-dimensional interactive animation environment. Novice programmers learn the concepts of object-oriented programming (OOP) by creating 3-D movies and games. Alice is designed by a research group at the Carnegie Mellon University and is used for introductory programming in middle school [26] and higher education. [7].

The 3-D approach taken by Alice assists in providing a strong, appealing context for students to learn the concepts of OOP. In Alice the objects-first approach is clearly visible, because everything is an object with properties, methods and functions. Triggered by events, the different methods of objects can be executed. Alice comes with a large library of models which can be used directly in the animations. Models consist of different parts which can all be programmed by using the available properties and methods.

Shortcomings of Alice are found in its weak inheritance model. A class can be extended by modifying an instance and thereby a subclass is created. However, when changing the superclass these changes are not propagated to its subclasses. Here the benefits of inheritance are lost [4]. As of Alice 3.0, which is still in beta, the inheritance model will be improved. The choice for a 3-D environment can become a burden when users want to create their own models. Although Alice is bundled with 1200 models, the learner here is not completely free in personalizing his game.

From observations by Cooper et al. in 2003 [7] it followed that students in an elective Alice course prior to CS1 developed a strong sense of program design, *objects*, *inheritance, events*, boolean types and program state. Students collaborated by dividing tasks and they developed a an appreciation of trial and error. Students however did not develop a detailed sense of syntax, clearly because programming constructs are dragged and dropped. The

[1] `http://alice.org`

group of students that took the Alice course performed significantly better in CS1 than the students that did not followed the Alice course. Something similar is described by Moskal [17]. Her students' performances in CS1 also increased by taking an Alice course, although not significantly.

Although Brown [4] was able to teach a difficult concept like *recursion* with Alice, he mentions his concerns crossing the gap from an Alice CS1 course to Java CS2. He notes that the most positive reports of Alice deal with pupils in secondary education.

One of these reports is from Kelleher et al. [14]. She compared 'Storytelling Alice' (an adjusted version of Alice for building short animated movies) with 'Generic Alice' (the normal version). On learning basic programming concepts she concludes that pupils (all girls in this study) using 'Generic Alice' and 'Storytelling Alice' were equally successful. In this study data was collected from different instruments including the created Alice programs, a forced-choice programming quiz, Alice logs, an attitude survey and behavioral data. The use of programming concepts was analyzed by counting the number of pupils that used a certain concept including *methods, do togethers* and *loops*.

Another report of Alice used with pupils in secondary education is of Adams [1]. He describes the research that took place during an 8 day summer camp. The days were split in a morning session and afternoon session. In each session a different programming concept or Alice specific concept was explained. The pupils then produced an animated movie using the just learned concept. Programming knowledge of the pupils was assessed in a pre-test and post-test. Pupils were asked "Rate how much you know about programming" on a 0 (nothing) to 5 (expert) scale. The average shift on this scale was +1.8 and statistically significant.

The results of another summer camp are reported by Rodger et al. [26]. Over twenty tutorials were developed as learning material covering a variety of Alice constructs. The use of programming concepts was examined by counting the concepts that were applied in the projects. Percentages were then calculated. *Built-in function, events* and *loops* where used most often.

2.2.2 Game Maker

Game Maker[2] is an integrated development environment for the creation of computer games. Users without any prior computer programming experience are able to develop games by means of the so-called drag-and-drop functionality. Experienced users can make use of Game Maker's full potential by writing scripts in Game Maker Language (GML). Game Maker is developed by Mark Overmars[3] in 1999 and is currently maintained by YoYo Games. The latest releases are version 8 for Windows and version 7 for Mac OS. Game Maker 8 for Windows is available in a free (Lite) and charged (Pro) version. The Lite version suffices for creating most games. From now on *Game Maker* should be read as 'Game Maker 8 Lite for Windows'.

Game Maker uses an object-oriented design. Creating a game is a matter of defining *objects, events* and *actions*. Some objects have a visual representation, others are used to control certain game aspects and are not visible in the game [20]. *Instances* of *objects* are placed in a room, which is basically a level editor. Game Maker also provides conditional statements and executes a continuous loop during game execution. So both selection and iteration can be discussed [5].

Game Maker also provides an intuitive image editor to create and edit (animated) sprites. Using the community recourses however, thousands of sprites, sounds and backgrounds can be found for free.

Shortcoming of Game Maker in relation to programming concepts are the fact that arrays and lists are not easily supported using the drag-and-drop environment. GML however supports these data structures, but then the learner needs to learn some syntax.

Grgurina [11], computer science teacher educator at the University of Groniningen, says from own experience in secondary education that Game Maker "proved to be truly suitable to teach a large number of programming concepts in a limited period of time while appealing and motivation to students".

Besides these words by Grgurina little literature is found regarding programming concepts other then Overmars [20, 19], Chamillard [5] and Habgood [12] describing the possibilities of Game Maker.

[2]`http://yoyogames.com/gamemaker`
[3]`http://people.cs.uu.nl/markov`

2.2.3 Scratch

Scratch[4] is a 2-D environment for creating interactive stories, games, animations and simulations. It is developed by the Lifelong Kindergarten Group at the MIT Media Lab.

Three core design principles are applied: making programming more tinkerable, more meaningful and more social. Especially this social factor is interesting. With one click a project is shared online and others can view or edit it. And that is what happens a lot. There are children creating Scratch projects to teach others how to do certain things in Scratch [24].

Programming in Scratch is like playing with Lego. The 'programming blocks' fit into each other, giving programming a very playful character. A Scratch project starts with a sprite and then scripts (blocks), animations and sounds are added to it. It's all intuitive because blocks only fit on other blocks when this is appropriate, making margin for error small.

Scratch is bundled with a lot of images that can be used as in-game sprites or backgrounds. Images can also be edited with the build-in image editor, making the game more personal.

Scratch does not support object oriented concepts like *inheritance* and *defining classes of objects*. As a matter of fact the programming approach is 'bottom-up' and Scratch makes no attempt to teach OOP [28].

Maloney et al. [16] reported on programming experiences of youth working on their own at a Computer Clubhouse over a period of 18 months. In total 536 projects were analyzed on the use of programming concepts. This was done by checking if a concept was used within a project. 111 of these projects contained no scripts at all. Scratch was simply used as a media manipulation tool. The core concepts of Scratch, *sequential execution* and *threads*, were used most often. The 7 other concepts subject to study, in order of usage, were *user interaction, loops, conditional statements, communications and sync, boolean logic, variables* and *random numbers*. Maloney speculates that the concepts *variables* and *random numbers* are not easily discovered without guidance. Only after instruction pupils began to use *variables*.

[4]`http://scratch.mit.edu`

2.2.4 Comparison on programming concepts

Alice, Game Maker and Scratch were compared on the programming concepts that the frameworks support in table 2.1. Alice and Game Maker support a wide range of programming concepts. Scratch supports less concepts, partly because the framework does not support object oriented programming concepts like *inheritance* and *classes*. However, general concepts like *iteration*, *conditionals* and *variables* are supported, making Scratch suitable to use for a first programming experience. The lack of OOP-concepts also makes Scratch focus more on 'play and try' instead of 'design and try' as Alice and Game Maker do.

Considering programming Alice takes the most serious approach. This can also been seen from the fact that Alice is used in introductory programming courses at universities [7], whereas research to Scratch focuses more at younger children in secondary education [28]. Game Maker ends up somewhere in the middle when using only the drag-and-drop functionality. When using the full functionality of Game Maker by means of scripting in GML all programming concepts in table 2.1 are supported.

	Alice 2.2	Game Maker 8 Lite	Scratch 1.4
Sequence	Yes	Yes	Yes
Iteration	Yes	Yes	Yes
Conditional statements	Yes	Yes	Yes
Variables	Yes	Yes	Yes
Arrays	Yes	No	Yes
Lists	Yes	No	Yes
Event handling	Yes	Yes	Yes
Threads	Yes	Yes	Yes
Boolean logic	Yes	Yes	Yes
Parameter passing	Yes	No	No
Return values	Yes	No	No
Procedures / functions	Yes	Yes	No
Classes of objects	Yes	Yes	No
Objects	Yes	Yes	No
Inheritance	Yes	Yes	No
Recursion	Yes	Yes	No

Table 2.1: Comparison of Alice, Game Maker and Scratch on programming concepts

Although the 3-D approach of Alice looks appealing, a 2-D environment seems more attractive to use, because it allows for easy personalization of a game. The author prefers the objects-first approach over 'tinkering' in Scratch, because this way more attention can be given to the design of a program. Therefore Game Maker was chosen as the framework to work with.

The programming concepts subject to study are *object, instance, variable, event, condition, iteration, function, inheritance* and *recursion.* Using these concepts simple games can be build, whereas *functions* and *recursion* allow for more difficult but clever constructs. Note that due to naming in Game Maker, an *object* actually is a *class* and an *instance* is an *instantiated object of a class.*

2.3 Research question

Over the last years, like Grgurina [11], the author has seen pupils in secondary education successfully using Game Maker. The objects-first approach in combination with a 'visual programming language' let pupils easily build games. But what do the pupils actually learn about the programming concepts that they use in order to build their games?

Therefore the following research question was stated: *To what extent do pupils learn about programming concepts when building games as introductory programming?* To answer this question the following sub questions were stated:

- To what extent do pupils apply programming concepts in their own game creations?
- To what extent can pupils give examples of programming concepts used in a game?
- To what extent can pupils explain programming concepts in their own words?
- To what extent do pupils develop a systematic or structural thinking about programming?

Chapter 3

Learning Material

3.1 Teaching philosophy

The teaching philosophy is based on *constructivist* principles wherein learning is an active process in a relevant context with the learners being responsible for their own learning. Learning is stimulated by means of 'stealth learning', freedom, reward and challenge.

The learning material developed for this project starts with a new game each lesson. The games chosen are all game-classics. Therefore chances are high pupils are familiar with the games and that the games connect to the environment of the pupils. An environment wherein the pupil spends more than 100 minutes per day playing computer games [29].

Building a game over a long period is instructive, but can become tedious for the less motivated pupils. Pupils should experience moments of success often when building games. In each lesson therefore a game is to be build by completion. Pupils learn the basics of Game Maker by looking at the example games, modify them and extend them. This approach is also seen in objects-first programming [9] where concepts are taught 'top-down'. However, the difference here is that the focus in the learning material is always on building games and less on learning programming concepts. This is what Randy Pausch called 'a head fake'. Pupils are learning one thing while they believe they are learning something else [8]. In this case the pupils are building games while learning to program. This 'stealth learning' approach in which pupils ideally don't realize what they are actually learning [23], should keep the lessons fun and the pupils engaged and motivated, resulting in great learning experiences [22].

Once the game is completed at the end of the lesson, the pupil is given the opportunity to really play the game. This moment of success, or reward, is taken onto the next lesson, knowing that finishing the tutorial gives the pupil a playable game. The game however can always be improved. Thus the pupil is asked to extend the game, but this time with little to no instruction. This freedom and *challenge* to change the game is important to motivation [2].

The learning material is presented in the form of a website. This way the material is available anytime everywhere. Enthusiastic pupils can access the tutorials in their own time. For example to work ahead or create modifications of the games. For this reason all games can be build with the free version of Game Maker.

During the lessons the teacher mainly acts as a coach, a facilitator. The teacher assists pupils in *learning to learn* from the online material. This way pupils construct their own knowledge. Only a small part of the lesson is used for direct instruction. See section 3.3. The lesson is ended with a reflection on the work done as, according to Paras et al. [22], "learning is not fully realized unless the player [here pupil] reflects on the events that took place during the experience".

3.2 Structure of learning material

Each lesson adheres to the following structure. The learning material can be found online, see section 3.5.

1. A short background to the game as introduction
2. Learning goals stated in an understandable fashion
3. Text and images clarifying the first version of the game
4. Building the game by completion following step-by-step instructions
5. Summary, including references to the learning goals
6. A list of ideas for extending the game

The step-by-step instruction is inspired by 'The Game Makers's Companion' by Habgood et al. [13]. Each step, if applicable, is accommodated with an image corresponding to the action's icon in Game Maker. The instruction itself is written and directly followed by a clarification of the instruction. In addition to Habgood a screenshot of the current view in Game Maker is included as a link. Here the benefits of a web page clearly arise, making it possible to add visual help to every instruction. Clues about Game Maker and explanation about programming concepts are written in red text blocks. The blocks are positioned to the right of the actual content, making them stand out more. See appendix A for screen shots of the website.

3.3 Teaching strategy

The teaching strategy is based on the pupils working independently. The role of the teacher is to structure the lesson and provide assistance where needed. The lesson plan is as follows.

1. Introduction (5 minutes)
   (a) Rehearsal of the key slides of last lesson
   (b) Introduction to today's game, including a background story
   (c) Explaining important elements that already work in the first version of the game
2. Building the game (40 minutes)
   (a) Pupils login on the website
   (b) Pupils download the practice material
   (c) Pupils create the game by following the steps on the website while the teacher is available for questions
3. Reflection (5 minutes)
   (a) Reflecting on important elements the pupils encountered
   (b) Telling what game is next

Appendix C shows an example of the slides used in a lesson.

3.4 Programming concepts and learning goals

A programming concept is considered covered whenever the learner must apply a concept or when the learning material explains the concept. In table 3.1 an overview of programming concepts treated in the learning material is given. For each of the six lessons the learning goals are described. The goals are twofold. There are Game Maker specific goals and there are goals for learning programming concepts. The goals are described in the subsections below. Some learning goals naturally contain both aspects.

Concept	Sokoban	Breakout	Pac-Man	Asteroids	Snake	Super Mario
1. Object	X	X	X	X	X	X
2. Instance	X	X	X	X	X	X
3. Variable	X	X	X	X	X	X
4. Event	X	X	X	X	X	X
5. Condition	X	X	X	X	X	X
6. Iteration			X	X	X	X
7. Function		X		X	X	
8. Inheritance		X		X		X
9. Recursion			X		X	

Table 3.1: Programming concepts throughout the learning material

3.4.1 Lesson 1: Sokoban

In this first lesson the basics of Game Maker are treated, like installing and starting the program. Sokoban is completed by adding the character and making it move through the room. More difficult game-aspects like shoving the crates or checking whether a level is completed, are already implemented. At the end of the lesson the learner is encouraged to create more levels.

The learner...

- is able to download the practice files from the website
- is able to unzip a zip-file
- is able to install Game Maker on the computer
- is able to open a practice file in Game Maker
- is able to add and edit sprites and objects
- understands that a sprite can consist of multiple images, called subimages
- is able to edit a room by adding and removing instances of objects

- knows what a grid is when talking about rooms
- is able to explain what an event is
- is able to explain what an action is
- is able to count using a variable

3.4.2 Lesson 2: Breakout

In Breakout the learner starts to make the paddle move from left to right. Then the learner adds new stone objects. While making use of inheritance the learner implements the collision between the ball and the stones. The ball itself was already programmed to collide with the paddle in a logical manner. Finally the learner is asked to create more stones in different colors.

The learner...

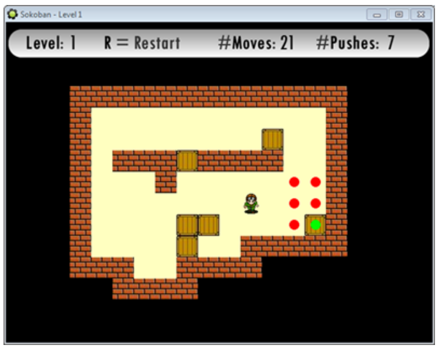
- is able to explain what a collision-event is
- is able to use inheritance to avoid creating multiple collision-events
- is able to adjust the speed of an object using a variable
- is able to explain how the function max() works
- is able to explain the difference between an object and an instance
- is able to explain the difference between an object variable and a global variable

3.4.3 Lesson 3: Pac-Man

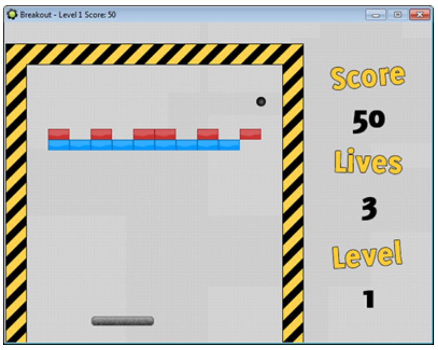
The goal in creating Pac-Man is to discover the very basics of artificial intelligence. Pac-Man itself is already created. The learner needs to implement the four different ghosts, each one being a bit smarter than the last one. At the end the learner is challenged to create yet another ghost that behaves different than the once created before.

The learner...

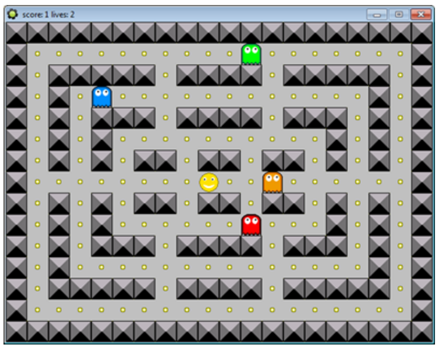
- is able to explain how recursion can be applied in a game
- is able to use the step-event
- is able to adjust the x and y coordinates of an object
- is able to implement chances in Game Maker



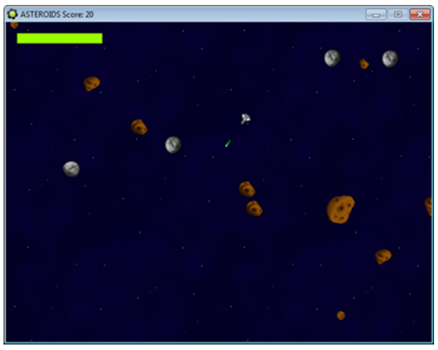
(a) Sokoban



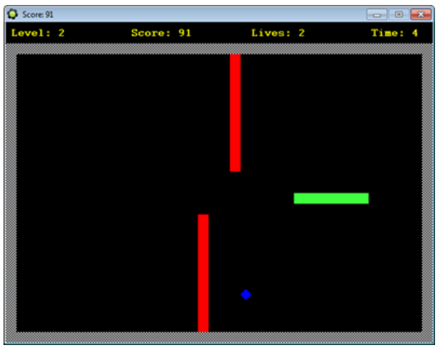
(b) Breakout



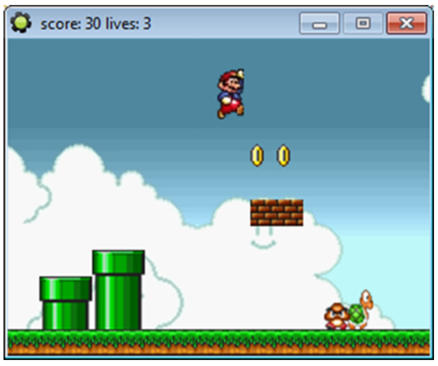
(c) Pac-Man



(d) Asteroids



(e) Snake



(f) Super Mario

Figure 3.1: Screen shots of the 6 games

3.4.4 Lesson 4: Asteroids

In this lesson the learner needs to implement the steering of the spaceship. By making use of subimages and variables the learner rotates and accelerates the ship. In the end the learner implements a laser gun. Extra challenges can be found in the creation of a protection shield or a computer controlled alien enemy.

The learner...

- is able to rotate an object
- is able to use inheritance
- is able to explain what parent-objects and child-objects are
- is able to explain how the function min() works
- is able to explain the very basics of modulo arithmetic

3.4.5 Lesson 5: Snake

This exercise is yet another approach in making an object move in Game Maker. The most difficult part of Snake is already created, namely the body following the head of the snake. The learner needs to implement the movement of the head using coordinates in the step-event. At the end the learner is asked to create more levels and to change the sprites and backgrounds.

The learner...

- is able to change the position of an object using variables
- is able to explain the difference between a non-persistent and persist object
- is able to explain how the create-event is used for recursion
- is able to state the numbers corresponding to the directions right, up, left, down
- is able to explain how many times the actions in the step-event are executed

3.4.6 Lesson 6: Super Mario

Super Mario, who can jump, duck and run is used to teach the learner about states and transitions. The learner implements parts of these transitions, focusing on the concept of gravity in Game Maker. Finally, the learner is challenged to make Mario enter a green tube.

The learner...

- is able to explain the different states Mario can be in
- is able to explain the state-transitions of Mario
- is able to implement gravity in Game Maker
- is able to explain what a view is when talking about rooms
- is able to explain why inheritance can be useful
- is able to use variables for multiple purposes

3.5 Website specifications

The website, publicly available at `http://gm.michelfiege.nl`, was created using *HTML*, *CSS* and *PHP*. In appendix A screen shots of the website can be found. For research matters an adjusted version was created. In that version a *MySQL* database was used to collect usage statistics such as page views, visits, downloads and clicks.

Chapter 4

Automatic generation of game-related learning material

Creating game-related learning material, for example as described in chapter 3, is a time consuming process. In its simplest form a game has to be built, the manual has to be written and the manual should be distributed amongst pupils. In this chapter a web application is introduced that makes the process of creating learning material for Game Maker easier: ‘GManualizer’.

4.1 Requirements analysis

Learning material for Game Maker is basically found in three ways. Professional books like ‘The Game Makers Companion’ [13] or ‘Games ontwerpen met Game Maker’ [21], tutorials by the creator of Game Maker[1] and material created by individual teachers. The books are full of information on game design, story development and describe the creation of the example games in full detail. The instructional directions are clarified with small *action-icon* images from Game Maker, but mainly text dominates. Sometimes screen shots of the program state are given. The tutorials written by Mark Overmars are bundled in a *zip-file* containing recourses like sprites and sounds, Game Maker files and a *pdf* file containing the actual manual. Like the books the manual contains a lot of explanatory text, but the in-

[1]`http://www.game-maker.nl/tutorials`

structions are now either in this text or in a screen shot of the program's state. Material by individual teachers[2,3] are structured in a variety of ways, but the approach by Overmars is often adopted, although not always Game Maker files and recourses are included.

The learning material basically consists of the following items:

- Instructions (with or without Game Makers' *action-icons*)
- Explanatory text
- Screen shots of Game Maker's state
- Resources like sprites and sounds
- One or more Game Maker (.gmk) files

The process of creating a manual for Game Maker consists of the following steps. The items between brackets are optional.

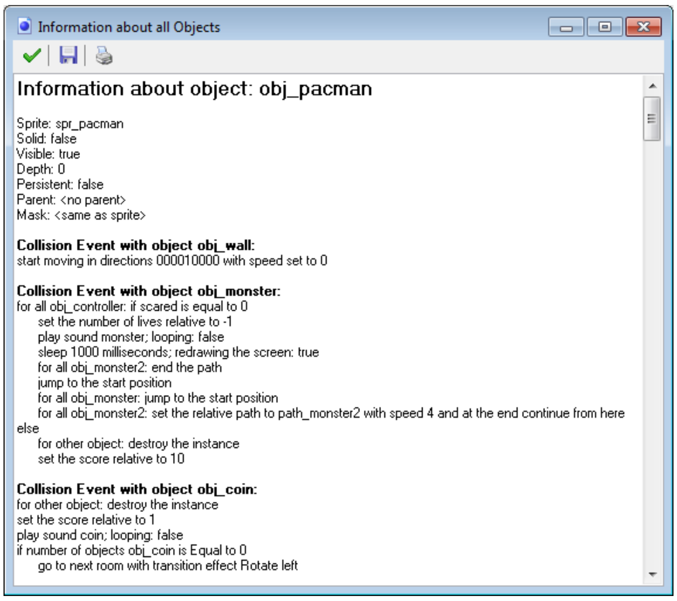
1. Create a game
2. Determine the order in which the game should be build
3. Open a text-editor and start writing
    (a) Write instructions one at a time
    (b) (Add action icon)
    (c) (Write explanatory text)
    (d) (Add screenshot)
4. Save the document
5. (Bundle the document, recourses and Game Maker files)
6. Distribute the manual

Throughout the writing process attention is given to a consistent writing style, a nice layout and correct placement of the images. Steps involved in creating a minimal manual thus are *creating a game*, *determining the order of instructions*, *writing the instructions* and *distributing the manual.*

The game itself actually possesses valuable information that can be used to write the instructions. In Game Maker there is an option to show the *Object Information* of the game. This is a small text file containing information about the *objects*, including *events*, *actions* and *action parameters* such

[2]http://www.game-maker.nl/educatief

[3]http://gamescool.nl/gratis_lesmateriaal.php



```
Information about object: obj_pacman

Sprite: spr_pacman
Solid: false
Visible: true
Depth: 0
Persistent: false
Parent: <no parent>
Mask: <same as sprite>

Collision Event with object obj_wall:
start moving in directions 000010000 with speed set to 0

Collision Event with object obj_monster:
for all obj_controller: if scared is equal to 0
    set the number of lives relative to -1
    play sound monster; looping: false
    sleep 1000 milliseconds; redrawing the screen: true
    for all obj_monster2: end the path
    jump to the start position
    for all obj_monster: jump to the start position
    for all obj_monster2: set the relative path to path_monster2 with speed 4 and at the end continue from here
else
    for other object: destroy the instance
    set the score relative to 10

Collision Event with object obj_coin:
for other object: destroy the instance
set the score relative to 1
play sound coin; looping: false
if number of objects obj_coin is Equal to 0
    go to next room with transition effect Rotate left
```

Figure 4.1: Screen shot of an Object Information file

as the name of the action and the values that were applied. In figure 4.1 a screen shot of a piece of *Object Information* from a small Pac-Man game is included. This file can be used to write the instructions automatically.

4.2 Design and implementation

The *Object Information* is the key component in automatically generating the learning material. This file is used as input for the web application. In figure 4.2 the process of creating a manual with *GManualizer* is outlined. In appendix B screen shots of the web application can be found that show the steps in creating a manual.

The user uploads the *Object Information*, which is a plain text-file, to the web application. The file is read line by line (parsed) and a manual is generated automatically.

In the parsing process the lines of the file are matched against a set of carefully crafted regular expressions. This way *objects*, *events* and *actions* are distinguished from each other. First a *GM_Manual* object is created. If a match is found one of the corresponding objects *GM_Object*, *GM_Event* or *GM_Action* is created and class variables are set. Each *GM_Object* is linked

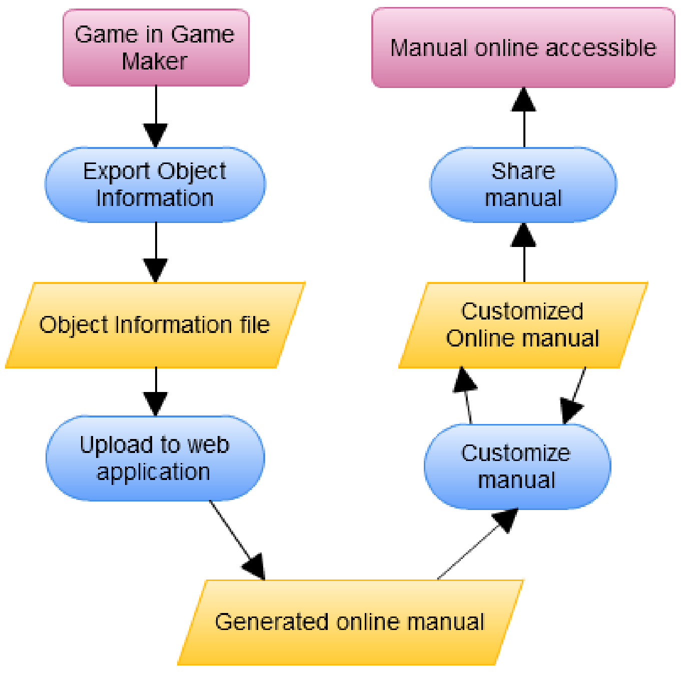


Figure 4.2: Flowchart of creating a manual with GManualizer

to *GM_Manual*, each *GM_Event* is linked to a *GM_Object*, each *GM_Action* is linked to a *GM_Event* and possible sub-actions (indents in the Object Information file) are linked to parent-actions. For each of the actions the parser then tries to automatically add some explanatory or supporting text based on the name of the action and the parameters that were set. The result of the parsing process is a *GM_Manual* object containing all the information from the *Object Information file* in a editable data structure. See figure 4.3 for a simplified object diagram.

After the parsing phase the user can customize the manual by adding text and explanations and by arranging the order of *objects* and *events*. The manual is a *PHP-object* which is displayed using *HTML* and styled using *CSS*. In the manual the *actions* are visually supported by the corresponding icons. Each time changes are made to the manual the *GM_Manual* object is serialized and stored in a session. The user can preview the manual anytime without sharing it.

The manual can be saved for later editing and shared online to make it accessible for anyone. When saving the *GM_Manual* object it is first serialized and then stored in *MySQL* database consisting of just one table.

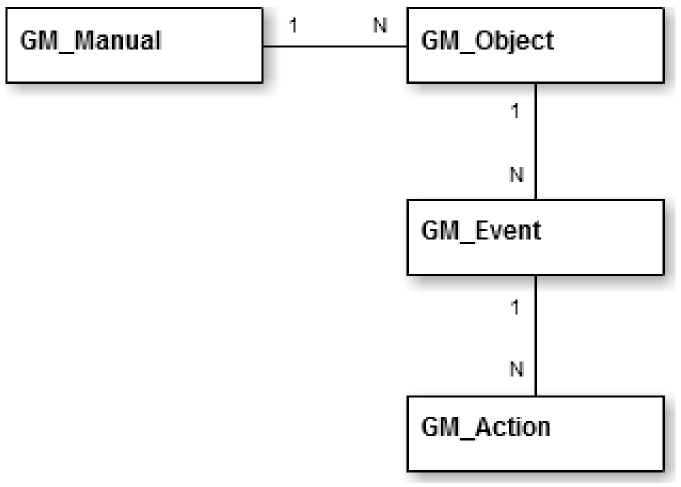


Figure 4.3: Simplified object diagram of the GManualizer data structure

Through serialization the amount of queries to save the manual is minimized. The user finally receives an email containing links to edit and share the manual.

The shared manual can be viewed in any browser. There is also an option to print the manual in case the learner prefers to read from paper.

The web application was custom build using *PHP*, *HTML*, *CSS* and *JavaScript* and can be found at `http://gmanualizer.michelfiege.nl/alpha`.

4.3 Evaluation and conclusion

GManualizer was thus far only used by the author. The *Object Information* from Game Maker is useful as a starting point for creating a manual. The basic steps in writing a manual, including adding explanations to instructions and distributing the manual, are covered. Moreover, every *action* in the game is displayed in a consistent way in the manual. This makes the creator of a manual focus on what's important: the actual learning contents. However, because it's an automated approach, as a creator you don't have as much freedom as for example in creating a manual in a text editor. You're somewhat tied to a fixed pattern. The author beliefs that with some future work and a thorough evaluation *GManualizer* could be improved and made available to a larger audience.

A paper to be published later by the author will cover thorough testing, evaluation and an improved version of *GManualizer*. Future work at this point can be summed up as follows.

- The set of regular expressions should be extended to also support Game Maker Pro functionality. Currently 85 actions from Game Maker Lite are supported.
- The parser needs to be adapted in order to parse a few multi-line actions. Currently those actions are skipped.
- The algorithm to automatically add supporting text could be improved.
- Easy localization support should be added. GManualizer is currently only available in English.
- Currently there is no support for adding custom images. Perhaps a little more freedom in creating the manual will attract a larger audience.
- GManualizer can be made more 'social', as in personal pages and overviews of recently created manuals.
- GManualizer's implementation could be improved by choosing a PHP-framework over the current custom build web application. This way future maintenance is easier.

Chapter 5

Research method

5.1 Design

A pre-test post-test single group design was used for this study. Multiple rather qualitative instruments were used to do the measurements.

This design in general is not strong in internal validity [25]. Single group threats like history, maturation, testing, instrumentation, regression and mortality may all occur. However, in this case not all of these threats applied. The history and maturation threats were low, because the time of the study was only 13 weeks. Also, because the curriculum was not graded, pupils were likely not stimulated by other events to perform better other then by the intervention.

The testing threat might have been an issue. However, the pre-test interview lasted only 10 minutes and the pupil was not told how he or she performed. The pupil was therefor not likely to really learn anything during this interview. On the other hand, the pupils might have spoken to each other about the interview, influencing each others prior knowledge.

The instrumentation threat was low when looking at the pre-test and pre-test interview. The interviews were conducted in the same fashion with as good as the same questions. However, the initial situation was measured with one instrument and the final situation was measured using multiple instruments. Those instruments together provide more data, but also difficulties comparing them to the initial situation. In section 5.4 each instrument is accounted for with regard to the reliability.

The regression threat was high. The pupils had little to no prior programming knowledge. It's thus likely that students performed better after the intervention.

The mortality threat was present. No pupils dropped out entirely, but some students received more intervention than others.

5.2 Sample

The study was conducted in a class of 30 pupils in secondary education, during a IT-course called *ICT-Extra*. The pupils were all around the age of 12 and they followed different streams of education: MAVO (pre-vocational secondary education), HAVO (senior general secondary education) and VWO (pre-university education). The class was formed at the beginning of the school year. Besides the regular curriculum pupils had to choose exactly one of the following elective courses: *Sport-Extra, Dance and Drama, ICT-Extra or homework counseling.* In table 5.1 details of the group can be found.

	Male	Female	Total
MAVO	4	5	9
HAVO	6	2	8
VWO	8	5	13
	18	12	30

Table 5.1: The sample in streams of education and gender

Most pupils had already worked a bit with Game Maker before this study commenced. Those pupils spent one day at primary school creating games as an introductory to *ICT-Extra*. In this one day crash course, about 8 to 12 months ago, the pupils created a simple maze kind of game in a very classical, step by step, direct teaching style. Only three of the pupils from the sample did not have any experience in Game Maker at all.

5.3 Procedure

The study was conducted in the 13 weeks between 10 January 2011 and 8 April 2011. In table 5.2 the time path of the study is given.

Week	Subject
0	Pre-test: Interview
1	Lesson 1: Sokoban
2	Lesson 2: Breakout
3	Lesson 3: Pac-Man
4	Lesson 4: Asteroids
5	Lesson 5: Snake
6	Lesson 6: Super Mario
7	Lesson 7: Building own game
8	Lesson 8: Building own game
9	Lesson 9: Building own game
10	Lesson 10: Completion and handing in own game
11	Post-test: Questionnaire and written test
12	Post-test: Interview

Table 5.2: Time path of the study

5.3.1 Week 0: Pre-test

The week before the course started the pupils were interviewed individually for 10 minutes by the author. During those interviews, which were recorded with a digital video camera, the pupils were first asked to tell about their experience with Game Maker. After that, they were shown a screen shot (appendix G) of the game-classic *Breakout.* If a pupil had used Game Maker before, the following questions regarding programming concepts were asked first.

1. Can you tell me the difference between an *object* and a *sprite*?
2. Can you tell me the difference between an *object* and an *instance*?
3. Can you tell me the number of *objects, instances and sprites* in the picture? Please explain.
4. Can you tell me what *events* and *actions* are? Can you give an example in Breakout?

5. Can you tell me what *variables* are? Can you give an example in Breakout?
6. Can you tell me what *inheritance* is? Can you give an example in Breakout?

Thereafter, they were asked to tell in their own words how Breakout works. Finally, they were asked to write down on paper the way Breakout works in a *computer kind of language* that felt right for them (in Dutch: "voor jouw gevoel computerachtige taal"). Note that during and at the end of the interview the pupil was not given any response on how he or she performed.

5.3.2 Weeks 1 to 10: Game Maker curriculum

In weeks 1 to 6 the pupils worked with the online learning material (see chapter 3) creating 6 different games. The teacher followed the teaching strategy as described in section 3.3.

In weeks 7 to 10 the pupils created their own game. The pupils were given the choice to work together in a small group or to create a game on their own. Pupils could also voluntary sign up for the Creative Game Challenge[1], a national game creation contest for secondary education organized by Utrecht University. The same teaching strategy was used.

During the curriculum the pupils were given the opportunity to stay an additional 50 minutes right after the lesson.

The games were handed in by email in week 10. The pupils that participated in the Creative Game Challenge also submitted their games on the website of the organization.

At first the idea was to also keep in-class notes using observation schemes created in advance, but this turned out to be too intensive a task. Some observation where however written down during the lesson, but this was only done when an observation was too important to ignore or when very few fingers were up in the air.

Each lesson was recorded using a digital video camera and a tripod. Also, after every lesson a short report was filed in the form of a post on my research blog[2].

[1]http://www.creativegamechallenge.nl
[2]http://michelfiege.blogspot.com

5.3.3 Week 11 and 12: Post-test

In week 11 the pupils were given a questionnaire (appendix D). The pupils answered 10 questions on a 5-point scale giving their opinions about the first 6 lessons (online learning material), the last 4 lessons (creating their own game) and more general points.

In the last week the pupils were interviewed again. The interviews were a near copy of the interviews in the pre-test, only this time the game of discussion was *Space Invaders* (appendix G). Once again every interview was recorded using a digital video camera. The questions asked were the ones stated below.

1. Can you tell me the number of *objects and instances* in the picture? Please explain.
2. Can you give some examples of *events and actions* in Space Invaders?
3. Can you give some examples of *variables* in Space Invaders?
4. Can you tell me how *inheritance* could be applied in Space Invaders?

After answering these questions the pupils were asked to describe Space Invaders in their own words. At last they were asked to write down on paper the way Space Invaders works in a *computer kind of language* that felt right for them (in Dutch: "voor jouw gevoel computerachtige taal"). Note that during and at the end of the interview the pupil was not given any response on how he or she performed.

5.4 Instruments

In this section the instruments used in this study are described in relation to the research questions. Multiple instruments were used including interviews, observations, created work, a questionnaire and a written test. The validity of the instruments is based on the fact that the instruments were developed in consultation with colleagues and experts in the area of computer science education. Their reliability is discussed as well. For sake of clearness, the research questions are labeled as below.

{apply} To what extent do pupils apply programming concepts in their own game creations?

{examples} To what extent can pupils give examples of programming concepts used in a game?

{explain} To what extent can pupils explain programming concepts in their own words?

{structure} To what extent do pupils develop a systematic or structural thinking about programming?

The concept investigated is *learning programming concepts*. The 4 dimensions are {apply}, {examples}, {explain} and {structure}. The way these dimensions are assessed and by which indicators is described in the subsections below.

5.4.1 Pre-test interview

To determine the pupils' knowledge about programming concepts before the intervention, every pupil was interviewed individually as described in section 5.3.1. This instrument was carefully created with the help of two educators in computer science and tested on 3 pupils not included in the sample.

The interview gave a first impression about {examples}, {explain} and {structure}. The reliability of this instrument is based on the interviewer and the interpreter of the answers. To strengthen the reliability the questions were written out and asked in the same order every time. The interview was timed with a stopwatch in order to question every pupil for the same amount of time. Also the interviews were recorded with a video camera.

The interviews were spread over multiple days. To conduct all interviews in the same fashion the interviewer prepared the interviews by watching the recordings of the day before. Analysis of the interviews started after the last interview was held. This way the risk of leading questions was minimized.

Each of the concepts discussed in the interview (*object, instance, variable, inheritance, events* and *actions*) were either clear ('Yes') or not clear ('No') to the pupils. The results were assessed by inspecting the notes taken during the interview and by carefully watching back the recordings. For *objects* to be rated correctly, at least the 'monsters', 'walls' and the 'spaceship' should be pointed to without making other mistakes (see appendix G. In order to receive a 'Yes' on *instances* it should be clear that the pupil is counting every *instance* in the screen shot of the *objects* he pointed out before. *Variables* are rated 'Yes' when besides for example saying "coordinates" at least 'lives' and 'score' are pointed to. *Inheritance* is given a 'Yes' when it is clear that the pupil does not confuse it with *instance* and that the answer contains one of the elements 'parent', 'child' or a description stating that *inheritance* "makes it easier to account for collision of multiple objects". *Events* and *actions* get a 'Yes' when the pupil can give an example of an *event* and *action* in Break Out.

Rating {structure} is also done on a 'Yes' or 'No' scale. The pupil either writes in a 'programming kind of structure' or not. The criteria to be given a 'Yes' were defined after the interviews, as little was known of what to expect. A 'Yes' was given when a pupil successfully wrote down *objects*, *events* and *actions* in a table- or line-by-line-style, as opposed to making a small story with long sentences.

The interpretation relies on the author and the reliability is therefor, as good as intentions might be, subjective. It's likely that there will be discrepancies between any two experts. To determine reliability the Spearman-Brown prophecy was used, using odd-even Split-Half Correlation. The internal consistency of the 8 items was 0.86 (n=26).

5.4.2 Observations and logbook

The observations, although not structured using schemes, were used to give an impression of the learning material and the course as a whole. During the lesson small notes where taken by the teacher and at the end of each lesson an entry was submitted to the logbook. The lessons were also recorded using a video camera. Attention was given at ambiguities regarding the learning material, pupils' commitment, the atmosphere in the classroom and questions regarding Game Maker or programming concepts. The logbook was later consulted to support the findings in the results section. The recordings give an impression of the attitude and behavior of the pupils during the lesson. The recordings don't give information as detailed as the recordings of individual pupils during the interviews. Reliability of the logbook and observation is low, because it is incomplete (the teacher could never have observed and reported everything) and thus only a selection of observations was reported.

5.4.3 Games built with Game Maker

The games created by the pupils were analyzed with regard to the use of programming concepts. This was done by counting the applied concepts, similar to what was done in Maloney for Scratch [16] and Rodger for Alice [26]. This number gives insight in {apply}. To strengthen reliability counting was done twice for each game. No differences where found in this process. However, the counting was done by the same person and therefor it is hard to say that this instrument is reliable.

5.4.4 Questionnaire

The questionnaire (appendix D) was used to evaluate the pupils attitude towards the learning material, building an own game, and the lessons series in general. Although the questionnaire does not answer any of the research questions, it provides valuable background information. The questionnaire consists of 10 questions rated on a 5-point scale, including a neutral option. Regarding the learning material the pupils were asked how 'fun', 'interesting', 'difficult' and 'clear' they found it. On building their own game they were asked how 'fun' and 'difficult' they found it. Finally the pupils were asked how much they had 'learned', to rate their 'skills' in Game Maker and to tell what they thought of the instruction and reflection at respectively the 'start and end of the lesson'. The reliability is measured using Cronbach's Alpha internal-consistency method: $\alpha = 0.68$ (n=30). This value is close to satisfactory.

5.4.5 Written test

The written test was created in consideration with a teacher in computer science in secondary education. The test consisted of 10 questions. The first 9 questions asked the pupils to explain and give an example of a given programming concept. Those questions respectively were used to measure {examples} and {explain}. The 10th question was used to find out more about {structure}. The questions regarding *giving examples* were rated 1 point for each good example. Most questions however only asked for one example. For all the *explanation questions* a maximum of 2 points was to be scored. For a blank or wrong answer 0 points were given, for an answer containing at most one wrong element 1 point was given and for a complete answer containing no wrong elements 2 points were given. Reliability of this instrument has not been strengthened in any way.

5.4.6 Post-test interview

To find out more about {examples}, {explain} and {structure} a second interview was conducted. The interview was held in the same way as the pre-test (see section 5.4.1). The setting, duration, kind of questions, interviewer, interviewees were all the same. In order to strengthen the reliability the interviewer prepared the interviews by watching some of the recordings of the pre-test.

Rating the conceptual knowledge ({examples}, {explain}) of the pupils was also done on a 'Yes' or 'No' scale as described in section 5.4.1. The same is true for {structure}. To determine reliability the Spearman-Brown prophecy was used, using odd-even Split-Half Correlation. The internal consistency of the 8 items was 0.67 (n=30).

Chapter 6

Results

In this chapter the four subquestions as stated in section 1.1 are answered. To put the level of conceptual knowledge in perspective, examples of quotes and answers from pupils are given.

6.1 General impressions

The interviews held at the beginning of this study went well. In general the pupils were comfortable answering the questions and only one pupil did not want to be filmed. Most of the questions the pupils could not answer correctly, but they felt somewhat obliged to guess anyway.

At the beginning of the course the pupils were all excited to start building games. They were focused on getting the game to work in order to play and some pupils therefore rushed through the manuals. Often this hasty attitude lead to questions, because instructions were misread or not read at all. Especially Ramon, a bright pupil who worked hard, showed this behavior. When I checked his games during the lessons things did not work as supposed, but he could not care less. He played and modified the game, because he liked that more than following instructions. Another group of pupils found it difficult to start and basically wanted a teacher to look over their shoulder. Unfortunately the group was too big to instruct each pupil individually. The teacher also noticed that the pupils worked mostly individually and that they did not bother to help each other in case of problems, although the teacher encouraged pupils to do so. When asked why the pupils did not help each other the argument given was all too often that they did not quit understand it themselves.

The creation of a completely new game brought new problems. The learning material was not really used anymore and pupils tried to figure out things all by themselves. For example in the learning material the way to make an object move using the arrow keys is explained in multiple ways. However, most of the pupils did not use these examples and created the control of an object themselves, often resulting in small errors and new questions. The creation of one's own game also showed that pupils really liked to edit images, such as sprites and backgrounds. The built-in tools in Game Maker to edit images were used without any problems and pupils spent lots of time in doing so. Sanne for instance, who worked together with Sven on their game, spent whole lessons creating new images.

At the end of the course the teacher felt that most of the pupils really wanted to start on a new subject. However, the pupils still needed to fill in a questionnaire and make a small written test. Only the pupils who really enjoyed building their own game would have wanted to continue. As a matter of fact 4 pupils kept making extra hours working with Game Maker long after the course was finished.

The questionnaire and written test were handed out to the pupils in the last lesson. Filling in the questionnaire went smoothly. The written test caused a bit of rumor in the class, because a lot of the questions were too difficult to answer. The fact that pupils knew that they did not get a grade was reason for some of the pupils to leave blank answers, instead of trying to write something reasonable.

The interviews held at the end of this study also went well. The pupils were even more comfortable as they knew what to expect. Besides, they now also told that they "just did not know that answer", instead of guessing anyway. Two pupils however did not want to be filmed.

6.2 Applying programming concepts

To what extend pupils apply programming concepts is measured by analysis of their created games and by observation.

Of the 19 created games 14 were inspected on the use of programming concepts. The 5 dropped games were mash-ups of the practice material where no new elements were programmed. In these games for example only new levels and new sprites were created. Similar findings were reported by Maloney [16], where in 21% of the Scratch projects the program was used as a media editing tool instead of a programming tool. In total 384 programming concepts were applied. In figure 6.1 the percentage of concepts applied in the 14 games is visualized.

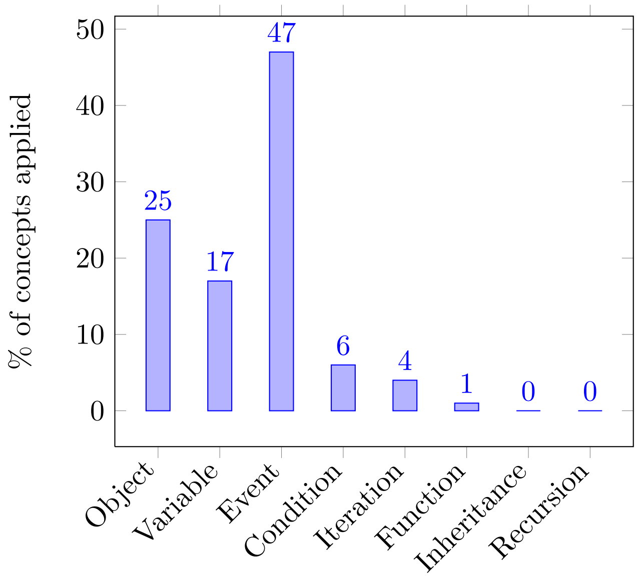


Figure 6.1: Concepts used in the created games

Since creating a game in Game Maker is primarily based on *objects*, *events* and *actions* it is not surprising to find high percentages for the concepts *object* and *event*. *Variables* were mainly used to keep track of scores and lives. These two *variables* each have an own action icon in Game Maker, making them easy to use. Only two groups used other *variables*. For example to adjust x- and y-coordinates or speed and direction of an instance. *Conditions*, including if- and else-statements were used in 7 of the 14 games. In particular to implement gravity or to check when to go to the next room. The event based approach of Game Maker makes the need for *conditions* low in these simple games. *Iterations*, implemented by the *Step- and Draw-event* were used in 6 games, mainly to handle gravity. Only in 1 game *functions* were used to limit speed and to set boundaries for x- and y-coordinates. *Inheritance* was used once (0.08%) and *recursion* was not used at all.

The quality of the games was low. Most of the games were unfinished and poorly playable. However, that was to be expected with approximately 200 minutes of work for a first attempt in creating an own game. See appendix F to get an idea of the quality of the games. Two games, named *Dolphin Island* and *De Spetterende Druppel Quiz (The Splashing Drop Quiz)* were really finished. Those games were created in groups of respectively 2 and 3 pupils and were also submitted to the Creative Game Challenge. Because the pupils of these groups often stayed an extra hour, they also received more help from the teacher compared to other pupils.

Dolphin Island was created by Sven and Sanne. Sven was responsible for programming and Sanne created the graphics. Sven was a pupil who always stayed an extra hour after the lesson. He worked approximately 10 hours on the game, including some time at home. Sven could not have created the game without the help of the teacher. He did not ask much, but he needed help on *variables* and *functions*. Those concepts he could not apply on his own using the online learning material or other tutorials found online. Just before the post-test interview started Sven said "please don't ask me about *variables*".

De Spetterende Druppel Quiz was created by Tessa, Anne and Sanne. Tessa was responsible for programming and Anne and Sanne created the questions and the graphics for the quiz. Tessa spent about 5 hours programming the game. She had help using the *Draw-event* and *variables*, as well as for creating a basic structure for the answering the quiz questions. Concepts like *iteration*, *variables* and *conditions* raised many questions.

The two games described above have in common that they were created by a group of pupils, not by a single individual. There was a clear division of tasks and the programmers of the teams were both highly motivated. This combination seems to work out well considering the timespan. The pupils who created a game on their own lacked the time to create graphics and to program the game. The result was often that both parts failed to be accomplished well.

As for Game Maker, creating *objects*, *events*, *actions* and *rooms* the pupils all learned and understood the basics. Difficulties were often found in applying concepts like *variables*, *conditions* and *iteration*. For example to count *instances of objects* or to check the position of an *instance* in a *loop*. Much individual help was required here. Difficulties regarding the basic concepts of the object-oriented event based approach (*object, instance, event*) were solved in the first 6 lessons using the online learning material.

In the process of creating an own game only questions were asked regarding what certain *events* were used for or whether some event existed.

In general the pupil is able to apply the concepts *object* and *event* without help. The concept of an *instance* is not included in figure 6.1. However, every game logically contained more *instances* than *objects* and the pupils were all able to place *instances* of an *object* in the game without help. As for *variables* the pupils could only use the in-game *variables* like score and lives without problems. *Variables* that need to be created using the variable-icon, as well as the concepts *condition, iteration, function, inheritance* and *recursion* are not as much applied for two reasons. The first being that there is often no need for it, because a lot of the game mechanics can be built using only *objects, events* and simple *actions*. Second, the pupil could not figure it out and asked no questions or a question was not answered (in time) due to the size of the group and the large amount of questions.

6.3 Giving examples of programming concepts

To what extend pupils can give examples of programming concepts is measured by a written test and an interview.

The written test was done by all 30 pupils. In figure 6.2 the percentage of correct examples per concept is given. In order to appreciate these results it is good to see some of the individual answers that were given. An overview of correct and incorrect examples for a given programming concept is given in table 6.1. The test itself can be found in appendix E.

From the written test it follows that pupils have difficulties stating examples on paper. An example of an *object* is given correctly by 70% of the pupils, but on all other concepts the pupils scored less than 50%. The concept *event* (41% correct) is sometimes falsely seen as an *action*. This is clearly a misconception. Examples of *variables* (33% correct) are vague. A lot of pupils answered this question by writing “x- and y axis”. It seems they are thinking in the right direction, but they can’t really give a right example. Considering all low scores, the fact that difficult concepts like *inheritance* (30% correct) and *recursion* (23% correct) score relatively well are remarkable. Again, the examples are not great, but the pupils were able to think in the right direction. However, the two concepts were also all too often mixed up. Meaning that an example of *inheritance* was given for *recursion* and vice versa. Good examples of *instances* (20% correct), the concept they applied most in their own games, are few. It seems like this concept has not been labeled into the minds of the pupils. Often there is given no answer or an example is given of some random concept. An answer containing the word *object* would be expected, but this is never the case. Examples of *conditions* (7% correct), split into *if* and *else* in the written test to make it more clear, were answered with a big question mark. Only three pupils had some idea that came close to testing *variables.* Examples of *iteration* and *functions* (both 0% correct) were not given correctly at all.

Similar results can be seen in figure 6.3, wherein the results of the interviews are displayed. In those interviews the pupils were asked to point out certain concepts in a screen shot of a game (See appendix G). Each concept is understood better in the post-test than in the pre-test. From the figure it follows that the pupils could give examples of *actions* in 67% of the cases. *Action* is however not considered a concept. It is tested in this interview to see how well the Game Maker trinity *object, event, action* is understood. *Objects* (63%) and *events* (47%) were pointed out correctly by almost a majority of the pupils. Again some of the pupils mixed up the concepts of *event* and *action.* The score on *inheritance* is again relatively high with 17%, but during the interviews a lot of pupils also gave examples pointing towards the concept of *instance.* Examples of *instances* (7%) were scarce, just as in the written test. In the interviews almost every pupil told that they did not know that term and wanted to skip that question. Sven however gave an answer that showed he did understand it. He argued that there were no *objects* to be seen in the picture, but that it were all *instances of objects.* None of the pupils could point out *variables.* Again the majority of pupils said they did not really understand that term.

In general the pupil is able to correctly give examples of *objects* and to a lesser extent of *events. Events* are too often mistaken for *actions.* Examples of other concepts, including *instance, variable, condition, iteration, function, inheritance* and *recursion* were only given correctly by a minority of the pupils. The relatively few correct answers concerning instances is peculiar, as the pupils worked with them a lot. On the other hand, examples given of difficult concepts like *inheritance* and *recursion* make it plausible that too little attention is given to the relatively easy concepts *variable* and *instance.*

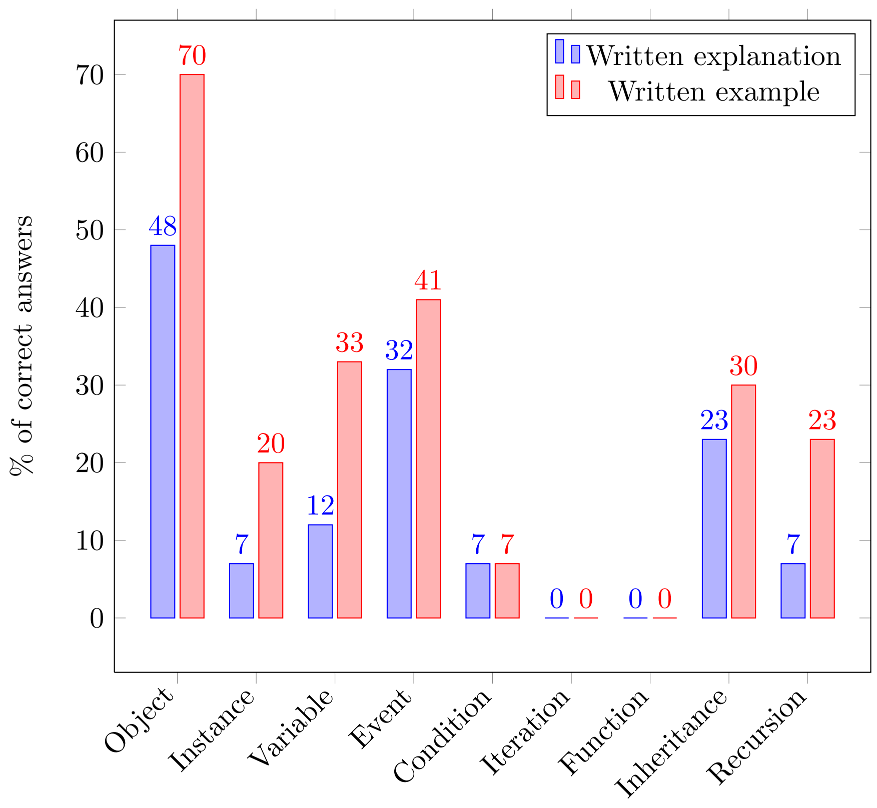


Figure 6.2: Results of written test

Example	**Correct**	**Wrong**
Object	"A dummy that moves back and forth"	"An image of a water drop"
Instance	"The Mario who you put somewhere (in the room)"	"Move forwards"
Variable	"To count steps"	"Event"
Event	"Create, Collision, Keyboard, Press, Step"	"The direction he is moving towards"
Condition	"If you're hit, die, else don't."	"For a number between 1 and 50"
Iteration	*no correct answers*	"If you create coins"
Function	*no correct answers*	"Walking, jumping, running"
Inheritance	"To give objects the same events"	"That they multiply"
Recursion	"To add multiple coins in a room at once"	"An object"

Table 6.1: Examples of correct and wrong examples of concepts

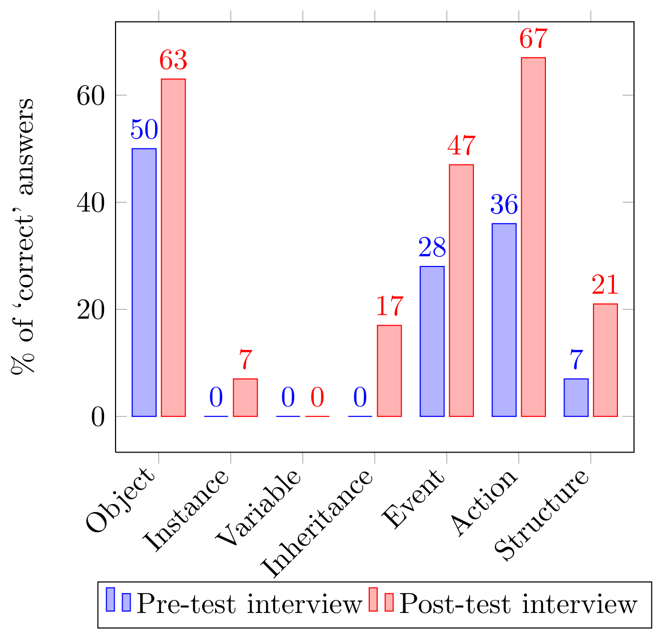


Figure 6.3: Results of the interviews

6.4 Explaining programming concepts

To what extend pupils can explain programming concepts in their own words is measured by a written test and an interview.

The written test was done by all 30 pupils. An overview of correctly explained programming concepts is given in table 6.2. From the figure it follows that the concepts object and event are best known. In table 6.2 some individual answers regarding the programming concepts are stated. Explaining a concept is more difficult than giving an example as can been seen in the table.

The concept *object* is explained well by 48% of the pupils. Some pupils mistake an *object* with a *sprite*, which basically is an image, and others find it just difficult to state it their own words. Descriptions of *event* (32%) and *inheritance* (23%) lack precision, but often the pupil heads in the right direction. As written earlier, there are also misconceptions in play. *Events* are considered *actions*, and *inheritance* is explained as the concept *instance*. The concept of *variable* (12%) is not clear at all. Pupils often guess by filling in some random answer. There is no general error to be found amongst the answers. Answers like "something that repeats itself", "the x- and y-axis" or "is a thing that walks" don't show any similarities. The concepts *instance* and *condition* (both 7%), as well as *iteration* and *function* (both 0%) are all explained poorly or not at all.

Description	**(Partially) correct**	**Wrong**
Object	“It’s connected to a sprite and you program stuff in it”	“Something you place in a room”
Instance	“The object that is placed in the room”	“Something your object does when you press a key”
Variable	“It adds/subtracts things”	“A button for the direction 0-360”
Event	An occurrence in the game	“Something you put the actions into”
Condition	“If you are working with variables”	“It’s the speed”
Iteration	*no correct answers*	“The counting of points”
Function	*no correct answers*	“Something he does”
Inheritance	“An object that has the same properties”	“That’s something that keeps the game running”
Recursion	“That an object (coin) splits itself”	“That the grid is filled with instances”

Table 6.2: Examples of correct and wrong description of concepts

6.5 Systematically thinking about programming

To what extend pupils can systematically think about programming is measured by a written assignment during an interview. In appendix H the writings of 4 pupils can be found.

From the pre-test interview it followed that 3 pupils (7%) could reason and write in a somewhat abstract and structural way. Meaning that they wrote down *objects*, *events* and *actions* in a structured way. They thought about interaction between *objects* in terms of *events* and *actions*. For the other pupils a generic trend was to be seen. These pupils could all orally explain the way Break Out works in a good fashion, but lacked the ability to write in a structured way as the other 3 pupils did. This formalization step went laborious, resulting in long sentences that described the game, rather than “programmed” the game. Compare for example the writings of Tessa and Justus with the writing of Kimberley in appendix H. In order to find out if the pupils knew what was most important in their sentences, they were asked to underline words. This step in the interview was improvised after seeing pupils struggle with the assignment. All pupils that were asked to do this did this surprisingly well. Words like "block", "bat", "wall", "if" and "then" were marked. Then they were asked to start over. They were now explicitly told that the sentences did not have to be well-formed, and that shorter was better. Shorter it got, but it were basically still sentences lacking a few words. Structure in the sense of line breaks or text indents were not applied.

In the post-test interview pupils showed more understanding of *events* and *actions*. Still however long sentences were formed. For example in Dylan's case who wrote: "you set the event left/right and space and the action causes [the object] to move left, right or shoot". Answers like this were not graded as correct, because these sentences show no, or at least little, systematic thinking about programming. Compare for example Kimberley's pre- and post-test as shown in appendix H. Her pre-test is a typical example of the way most pupils did both tests. However, her writings in the post-test were graded as correct, because there is a clear structure in *objects, events* and *actions* to be seen. For Sven the same is true. His pre-test writings were considered wrong and in the post-test he did correct. Tessa and Justus were the only two who structured their writing correctly in both the pre-test and post-test. In total 6 writings (21%) were graded as correct.

6.6 Course evaluation

All 30 pupils filled in the questionnaire. The results can be found in figure 6.4. The pupils on average assessed (on a 1 to 5 scale) the online learning material to be fun (3.5), interesting (3.2), easy (3.4) and clear (3.7). They found it more fun (3.7) to create their own game. And creating their own game was as easy (3.4) as working with the learning material. The pupils learned a lot (3.6) if you ask them, but they are critical at their own skills (2.7) concerning Game Maker. The first (2.7) and last (2.8) 5 minutes of a lesson, in which instruction was given and concepts were explained, were not seen as useful. Additional comments from pupils (19 in total) are listed below.

+ "It was super fun!"
+ "The teacher explained everything well."
+ "It was fun." (2x)

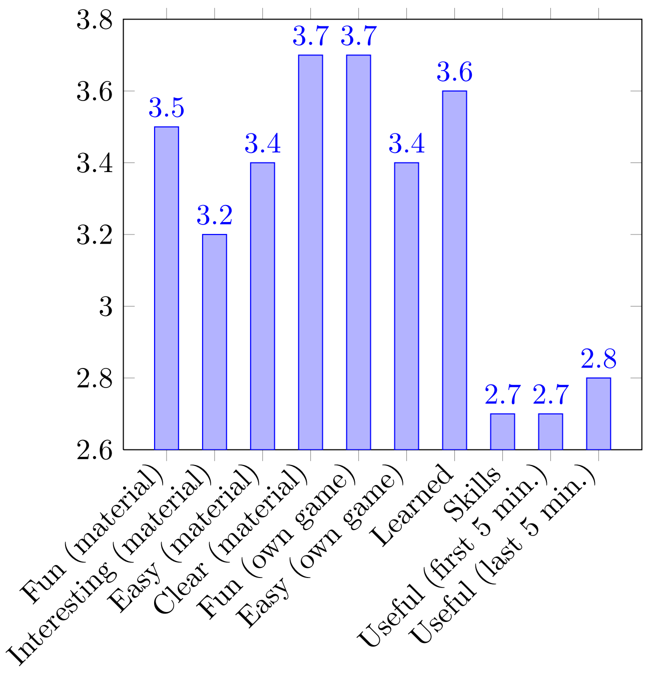


Figure 6.4: Results of questionnaire

+ "I've learned a lot in 10 lessons working with Game Maker."
+ "I learned a lot. I found it an interesting subject and I also spent time working at home."

+/- "I found it difficult to remember and apply things. Even though the lessons were fine and I hope my skills get better over time."

+/- "Sometimes I got bored, but overall it was fun."

+/- "I had more fun as the course developed over time."

+/- "Some lessons were fun, others a little less."

+/- "I'm not really clever in this kind of things, but the more easy stuff I could do and remember."

+/- "It was boring, but also fun."

- "Game Maker is too complicated."
- "I've learned just a little of Game Maker in 10 lessons."
- "Lots of things I could not find in the online material."
- "Six games were a little too much. I had too little time for my own game."
- "The first six lessons were too much. We had only little time to create

our own game. We did some work at home though.

- "I would suggest that you can start right away; that not everyone should listen to the introduction in the first five minutes of the lesson."
- "We had to do too much on our own."

The website containing the learning material is visited 230 times, resulting in 7.6 visits on average. A pupil spent on average 145 minutes on the site. The total time a pupil spent on the curriculum, including the time in the classroom and the time on the website, is on average 541 minutes. That's slightly more than 50 minutes per lesson. However, there are huge differences between pupils. The pupil that received the most intervention spent a total of 1071 minutes, while the least active student spent only 300 minutes. Notable is that only 2 pupils visited the website at home. Some pupils said they worked at home, but they must have done this without using the website.

From a teachers perspective the curriculum was fun and interesting, but not without comments. In total 35 blog posts were written to evaluate interviews and lessons, saving remarks and noting musings.

It took about 3 lessons before every pupil was able to independently download, unzip and open the practice file from the website. Direct instruction at the beginning of the lesson was more than once necessary on this subject, while this time was planned to instruct the pupils about Game Maker and programming concepts. The instructions in the online learning material were kept short on purpose, but still pupils did not read them well. Often the teacher was called quickly for help. As some pupil stated in the additional comments above, he had to do too much on his own. Given a class of 30 pupils with a lot of questions it was not weird that in the first 4 lessons only about half of the class managed to built the game of that day completely. The last 2 lessons working with the online learning material went smooth as the pupils really got the hang of Game Maker and understood were to put their attention to in the manuals.

The first and last 5 minutes of a lesson often were a struggle. Once a pupil entered the class all he or she really wanted to do was to continue with Game Maker. Waiting for some instruction only slowed things down in their opinion. The pupils did not seem to realize that they needed this instruction. At the end of the lessons, when the pupils were finishing up their game, it was difficult to get the full attention of the class. The results from the questionnaire make it clear that the short instructions were seen as obstacles, rather than as helpful.

The teacher expected more of the created games and the motivation of pupils. Normally a subject of *ICT-Extra* is given for 6 or 7 weeks. Maybe this is why some of the pupils got bored and did not perform at their bests. Another factor might be that the assignment of creating ones own game was stated too open. For intrinsic motivation to flourish a challenge should be 'pleasantly frustrating' [10]. The teacher thinks that the majority of pupils saw the creating of a game more as 'frustrating' than 'pleasant', because the group was too large to successfully assist each pupil individually. It might also had been better to just grade the work, even though no officials grades were to be given for the course. A little extrinsic motivation would not have harmed the pupils.

On the positive side, there were 8 pupils that stayed at least one extra hour in the 10 weeks. Two of them followed this extra lesson every week. In this hour more personal attention was given and better learning conditions were present. In these moments the pupils worked on their own game, finished or extended the game of that lesson and sometimes played each others games or played games found online.

Chapter 7

Conclusions

In this study the following question was addressed. To what extent do pupils learn about programming concepts when building games? Pupils' learning was investigated by assessing which concepts they *applied* in their games, to what extent they could give *examples* of programming concepts, to what extent they could *explain* programming concepts and to what extent they developed a way of *'structural thinking'* about programing. Pupils showed a basic understanding of some programming concepts while other concepts were poorly understood.

The results show that *objects* and *events*, together with *instances* and *actions* were most used. This is not surprising, because of the object-oriented event-based approach Game Maker takes. In order to build a simple game, like the pupils did, these four concepts suffice.

The pupils were best able to give examples of *objects, actions and events.* It seems that what they use most they know best. However, for *instances* this was not true. The term clearly was not labeled in their minds. One explanation might be that in Game Maker the word 'instance' is almost never used. Another explanation might be that the teacher spent more time on concepts like *inheritance* and *recursion*, as pupils were better able to give examples of these relatively difficult concepts. For *variables* something similar was observed. Although pupils used *variables* to set scores and lives, they actually did not label these as being *variables.* This can be explained by the fact that Game Maker uses separate action-icons for lives and score. Pupils are therefore not confronted with the word 'variable'. This seems to be a disadvantage of 'visual programming' regarding learning programming concepts. The pupils had lots of difficulties giving examples of *conditions, iterations and functions.* Most of the pupils simply had no idea. These concepts were rarely used in their games. It's therefore noteworthy that pupils could relatively well give examples of *inheritance* and *recursion.* None of the pupils used these difficult concepts in their own game, but from the post-test interview and written test it followed that the pupils could give simple examples of these concepts.

Explaining concepts was more difficult than giving examples of concepts. Again on the concepts of *objects, events* and *actions* the pupils scored best. The other concepts were all poorly explained.

On *structural thinking* it's important to note that no teaching whatsoever was done on the subject of structuring 'code'. Never in the whole intervention had the pupils been given examples or instruction on this matter. Building games with Game Maker and short instruction on programming concepts were the only interventions applied. In the post-test 6 pupils structured their writings using the trinity *objects, events* and *actions* of Game Maker. Also the 2 pupils who did structure their writings in the pre-test performed better in the post-test. The influence of the *objects-first* approach here is clearly visible, especially because concepts like *conditional statements* and *iteration* were not used in the writings. Pupils thus seem to develop a *top-down* thinking about programming.

Teaching 30 pupils on programming concepts without the 'stealth learning' approach of 'building games' would have been stricter and probably less fun to the pupils. The fact that pupils enjoyed working on their own game almost as much as working with the manuals is reason to believe that the course was not really seen as 'learning', but more like 'building a game'. Although some pupils could not cope well with the freedom that was given, the more motivated pupils produced playable games and learned quite a bit about programming concepts in doing so. The best example of this is Sven, who could not answer any of the questions in the pre-test correctly, but scored best in the post-test interview. He also created the best game and was among the best pupils in the written test.

Despite the poor summative results, the pupils did learn some things about programming concepts. The extent in which this happened can not be given a value, but can be best described as follows. *Objects, actions and events* were best known before and after the intervention. The increase in knowledge about these concepts was foremost on recognizing and giving examples. The better pupils were also able to explain these concepts in a basic way and they could think in a structured manner about programming. For the relatively more abstract concepts *inheritance* and *recursion* the pupils had some idea what it was, but applying these concepts themselves was too big a step. The terms *variables* and *instances*, although used in all games, were not labeled and therefore pupils could use them, but not explain them. For the concepts *condition, iteration* and *function* hardly any learning effects were visible. Quite possibly, when more instruction was given on *variables* and *instances*, instead of *recursion* and *inheritance*, the pupils' learning of concepts would have correlated with the usage of concepts. However, further research is needed to investigate this statement.

Chapter 8

Discussion

It is important to see the results in the context of secondary education and to take the age of the pupils into account. ‘Learning programming concepts’ is more like ‘learning the basics of programming concepts’ and in that light the pupils were also assessed. There was not a pupil that could explain for example the *inheritance* model of Game Maker in full detail, but notions of ‘sharing the same events and actions’ showed a basic understanding.

The fact that pupils were not graded during the course might have influenced the learning results. In a regular course the pupils are also extrinsically motivated by means of grades. In this elective course the absence of grades provided an opportunity to see how ‘stealth learning’ and intrinsic motivation can lead to understanding of programming concepts. For a majority of pupils this led to a decrease in motivation over time and looking back as a teacher the atmosphere in the class was not optimal for learning. This might explain the somewhat disappointing learning results.

It’s somewhat strange that relative easy concepts like *variables* and *instances* were less understood than *inheritance* and *recursion*. Clearly the terms ‘variable’ and ‘instance’ were not labeled in the minds of pupils, possible because less direct instruction was given too those concepts. Maloney et al. [16] confirm the need for guidance on *variables* as this concepts is caught on slowly by the pupils.

Misconceptions were also found. Pupils who could explain the basics of *inheritance* for example thought they were talking about *instances* and vice versa. Clearly applying concepts from a step-by-step manual and later on trying to build ones own game does not imply deep learning. Especially not in such a short time and with abstract entities like programming concepts. On the other hand pupils might have felt obliged to answer questions in the interviews and guessed certain answers instead of telling that they did not knew the answer.

The learning results might explain why pupils did not, although encouraged by the teacher, really help each other. The most obvious reason is that the pupils had a hard time figuring it out themselves, let alone help their fellow pupils. Possible the learning material was too difficult for their age.

The learning material, although well received by the pupils, was created in just 6 weeks and never used before in another setting. Only after the in-class research was conducted the learning material was made public. In the period from March 1st to June 30th the website was visited over 2600 times with more then 7000 pageviews and an average of 21 visitors a day. The author received numerous e-mails from teachers from other schools who were happy with the learning material. It would be interesting to conduct a follow-up study to see what a large number of pupils think of the material. Such a study might shed light on why the learning material was hardly used anymore when the pupils had to build their own game.

'GManualizer' might be an answer to the demand for Game Maker manuals. Although not yet used in education, the author beliefs that this web application can cause teachers (or even pupils) to create a variety of manuals, all bundled in one place. It would then be interesting to see which manuals are used most. Over time this might give insight in what teachers and pupils think are great manuals. For now 'GManualizer' needs to be tested by users and improved before made public to a large audience.

The notion that learning is correlated with the usage of programming concepts also suggests that working with another framework would likely produce other learning results. Projects produced in Alice and Scratch for example show a somewhat different use of programming concepts [26, 16]. Learning concepts therefor is likely to depend on the context in which they are taught. This research project could for example be extended with another group of pupils taking an Alice course.

It would also be interesting to research whether the conceptual knowledge of one framework is transferable to another. Would pupils who had Game Maker before perform better in an Alice course than pupils who have never used Game Maker nor Alice before?

The fact that there was only one group tested makes it impossible to generalize the results. Also the use of rather qualitative measures made it difficult to say things with certainty. In future studies a forced choice written test would therefore be recommended. The interviews provided a deep look in the minds of pupils and were found most useful in this study.

Chapter 9

Reflection

In this chapter I reflect on what I have learned and whether I have become a better, more researching teacher during my graduation project.

I learned that doing research, quite similar to giving a good lesson, is all about preparation and design. Without a well designed intervention and proper instruments research fails and without a lesson plan and learning material a lesson fails. I believe that this thesis helped me valuing preparation and design more than ever. Looking back for example at learning material that I created before this graduation project it often misses explicit learning goals and moments of reflection. I am working on new learning material for *ICT-Extra* and started with learning goals and summaries. The actual lessons will be written later. I believe that I have matured in creating learning material and lesson plans.

Interviewing pupils was new to me. From this I learned that pupils, although I knew this quite a bit already, are very honest and sincere in a personal conversation. Also I believe that interviews could be used well for assessing pupils. Basically I was testing what pupils knew and in 10 minutes time you can actually find out interesting things. I would like to find out more about oral assessments in the future.

From the interviews, but also because I created all the learning material, I now have a better view on what pupils in the age of 12-13 are capable of learning. Estimating the time pupils need for certain tasks is difficult. I learned that by means of additional exercises for the quicker pupils it's possible to keep all pupils equally busy during a lesson. I will try to keep this thought in mind while preparing lessons and creating learning material.

I learned a lot from writing this thesis. Structuring a large report, the style of writing, a supervisor questioning statements; it was a new experience and an experience I embrace. I believe that this experience can help me guide pupils in writing their (smaller) reports.

I also developed an appreciation for research and I learned to look critically at the work of others. I find it interesting to see new developments in the area of computer science education and I think it's important as a teacher to keep up to date. Especially regarding 'visual programming languages', which I believe are valuable in secondary education, I will keep myself informed. Perhaps I will even do some research myself once again in this area as I am planning to use more of these tools/frameworks/languages.

Although the research process matured me in thinking about preparation and design, I believe that regarding the research itself I still have a lot to learn. For example, I should have focused more on the design of the study and instruments. This could have made the results and conclusions more generalizable. And although I already had some experience in keeping a logbook of in-class events, I do this for all my lessons, for research matters it's better to have someone else report on observations. Being a teacher, researcher and observer at the same time is too much.

Then again, this was my first time doing research. Just as in creating learning material I believe that I will become better in doing research, because most importantly I find it fun and interesting to explore and create new things and write about it.

Did I become a better, more researching teacher? I think I did. However, a lot remains to be learned in the years coming. I think reflecting on learning processes by writing, for research or personal matters, is the key in getting better step by step.

Bibliography

[1] Adam, J. C. Alice, Middle Schoolers and The Imaginary Worlds Camps. *SIGCSE'07* (2007).

[2] Alessi, M., and Trollip, S. *Multimedia for Learning: Methods and Development*, 3 ed. Allyn and Bacon, 2001.

[3] Bergin, J., Brodie, K., Patiño-Martínez, M., McNally, M., Naps, T., Rodger, S., Wilson, J., Goldweber, M., Khuri, S., and Jiménez-Peris, R. An Overview of Visualization: Its Use and Design: Report of the Working Group in Visualization. *ITiCSE '96* (1996).

[4] Brown, P. H. Some Field Experience With Alice. *Computing Sciences in Colleges 24* (2008).

[5] Chamillard, A. Introductory Game Creation: No Programming Required. *SIGCSE '06* (2006).

[6] Cooper, S., Dann, W., and Paush, R. Alice: A 3-D Tool For Introductory Programming. *JCSC* (2000).

[7] Cooper, S., Dann, W., and Paush, R. Teaching Objects-first In Introductory Computer Science. *SIGCSE'03* (2003).

[8] Dann, W., and Cooper, S. Education: Alice 3: Concrete to Abstract. *Communications of the ACM* (2009).

[9] Decker, R., and Hirshfield, S. Top-Down Teaching: Object Oriented Programming in CS 1. *SIGCSE '93 25* (1993).

[10] Gee, J. P. What Video Games Have to Teach Us About Learning and Literacy. *ACM Computers in Entertainment 1*, 1 (2003).

[11] Grgurina, N. Teaching CS in the Netherlands with Game Maker. *Informatik und Kultur* (2011).

[12] Habgood, J. Passing On the Family Trade. *Develop Magazine* (2005).

[13] Habgood, J., Nielsen, N., and Rijks, M. *The Game Maker's Companion*. Apress, 2010.

[14] Kelleher, C., Pausch, R., and Kiesler, S. Storytelling Alice Motivates Middle School Girls to Learn Computer Programming. *CHI '07* (2007).

[15] Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. The BlueJ system and its pedagogy. *Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology 13*, 4 (2003).

[16] Maloney, J., Peppler, K., Kafai, Y. B., Resnick, M., and Rusk, N. Programming by Choice: Urban Youth Learning Programming with Scratch. *SIGCSE'08* (2008).

[17] Moskal, B., Lurie, D., and Cooper, S. Evaluating the Effectiveness of a New Instructional Approach. *SIGCSE'04* (2004).

[18] Myers, B. A. Taxonomies of Visual Programming and Program Visualization. *Journal of Visual Languages & Computing 1*, 1 (1990), 97–123.

[19] Overmars, M. Game Design in Education. 2004.

[20] Overmars, M. Learning Object-Oriented Design by Creating Games. *Potentials, IEEE 23* (2004), 11–13.

[21] Overmars, M., and Habgood, J. *Games ontwerpen met Gamemaker*. Van Duuren Media, 2008.

[22] Paras, B., and Bizzocchi, J. Game, Motivation, and Effective Learning: An Integrated Model for Educational Game Design. *DiGRA 2005 Conference* (2011).

[23] Prensky, M. *Digital Game-Based Learning*. McGraw-Hill, 2001.

[24] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. Scratch: Programming for All. *Communications of the ACM* (2009).

[25] Robson, C. *Real World Research*, second ed. Blackwell Publishing, 2002.

[26] Rodger, S. H., Hayes, J., Lezin, G., Qin, H., Nelson, D., and Tucker, R. Engaging Middle School Teachers and Students with Alice in a Diverse Set of Subjects. *SIGCSE'09* (2009).

[27] Schulte, C., and Bennedsen, J. What do Teachers Teach in Introductory Programming? *ICER'06* (2006).

[28] Utting, I., Cooper, S., Kölling, M., Maloney, J., and Resnick, M. Alice, Greenfoot, and Scratch - A Discussion. *ACM Transactions on Computing Education* (2010).

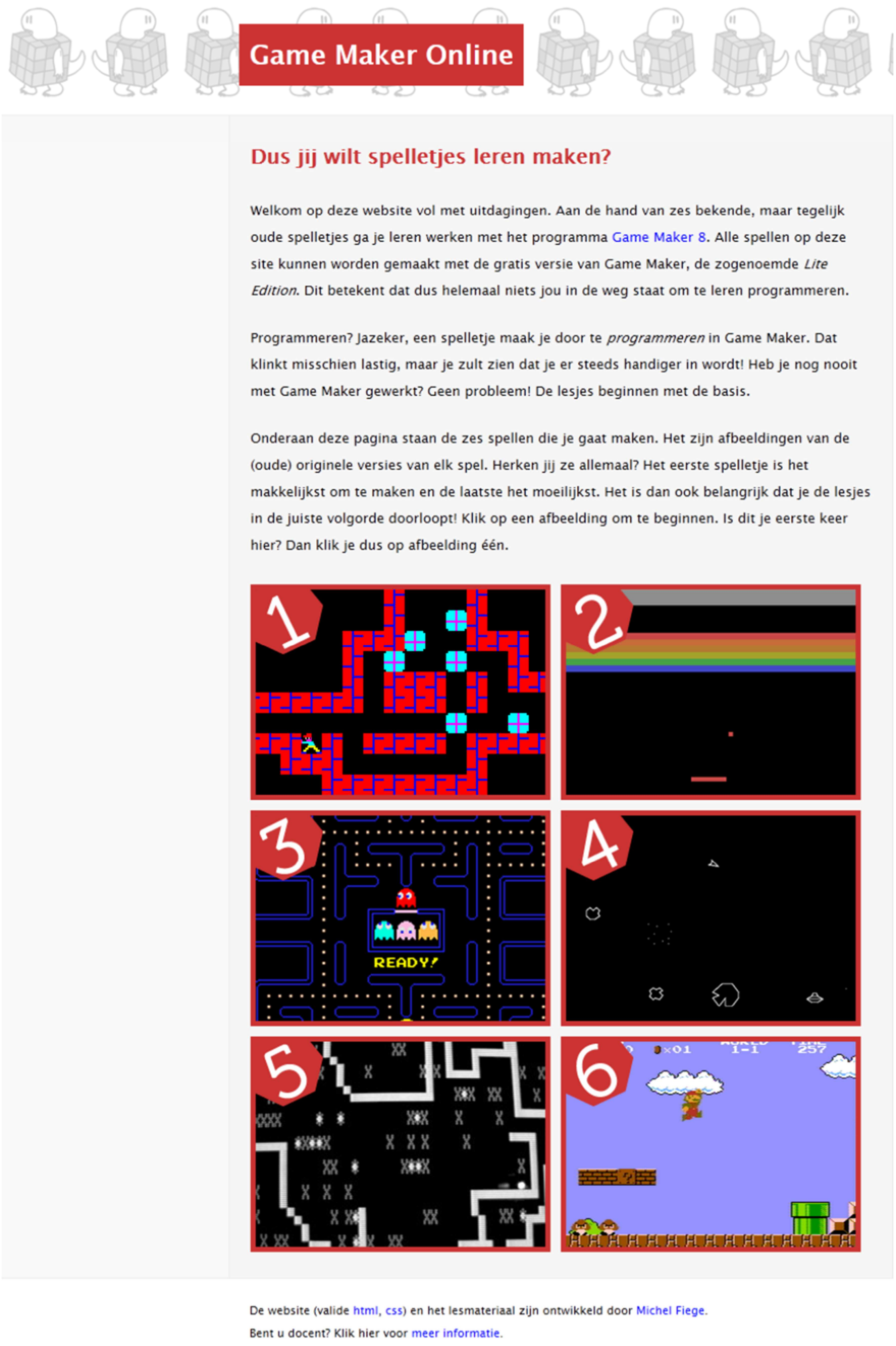
[29] van Dorsselaer, S., Zeijl, E., van den Eeckhout, S., ter Bogt, T., and Vollebergh, W. HBSC 2005 Gezondheid en welzijn van jongeren in Nederland. *Trimbos-instituut* (2007).

[30] Vos, N., van der Meijden, H., and Denessen, E. Effects of constructing versus playing an educational game on student motivation and deep learning strategy use. *Computers & Education 56* (2011), 127–137.

Appendix A

Online learning material

1. Homepage of the online learning material



Game Maker Online

Dus jij wilt spelletjes leren maken?

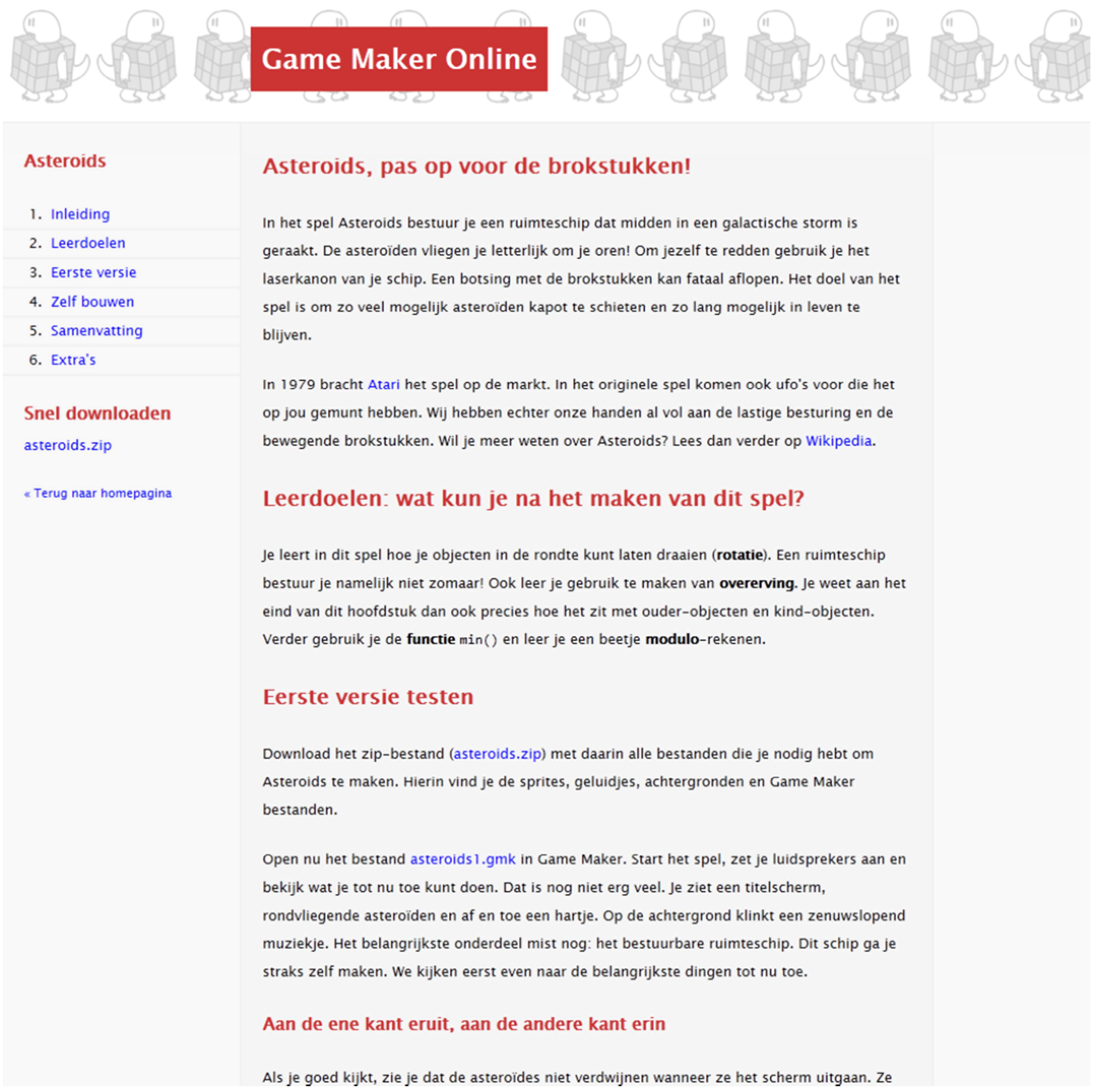
Welkom op deze website vol met uitdagingen. Aan de hand van zes bekende, maar tegelijk oude spelletjes ga je leren werken met het programma Game Maker 8. Alle spellen op deze site kunnen worden gemaakt met de gratis versie van Game Maker, de zogenoemde *Lite Edition*. Dit betekent dat dus helemaal niets jou in de weg staat om te leren programmeren.

Programmeren? Jazeker, een spelletje maak je door te *programmeren* in Game Maker. Dat klinkt misschien lastig, maar je zult zien dat je er steeds handiger in wordt! Heb je nog nooit met Game Maker gewerkt? Geen probleem! De lesjes beginnen met de basis.

Onderaan deze pagina staan de zes spellen die je gaat maken. Het zijn afbeeldingen van de (oude) originele versies van elk spel. Herken jij ze allemaal? Het eerste spelletje is het makkelijkst om te maken en de laatste het moeilijkst. Het is dan ook belangrijk dat je de lesjes in de juiste volgorde doorloopt! Klik op een afbeelding om te beginnen. Is dit je eerste keer hier? Dan klik je dus op afbeelding één.

De website (valide html, css) en het lesmateriaal zijn ontwikkeld door Michel Fiege.
Bent u docent? Klik hier voor meer informatie.

2. Introduction, learning goals and download first version



Asteroids

Snel downloaden

asteroids.zip

« Terug naar homepagina

Asteroids, pas op voor de brokstukken!

In het spel Asteroids bestuur je een ruimteschip dat midden in een galactische storm is geraakt. De asteroïden vliegen je letterlijk om je oren! Om jezelf te redden gebruik je het laserkanon van je schip. Een botsing met de brokstukken kan fataal aflopen. Het doel van het spel is om zo veel mogelijk asteroïden kapot te schieten en zo lang mogelijk in leven te blijven.

In 1979 bracht Atari het spel op de markt. In het originele spel komen ook ufo's voor die het op jou gemunt hebben. Wij hebben echter onze handen al vol aan de lastige besturing en de bewegende brokstukken. Wil je meer weten over Asteroids? Lees dan verder op Wikipedia.

Leerdoelen: wat kun je na het maken van dit spel?

Je leert in dit spel hoe je objecten in de rondte kunt laten draaien (**rotatie**). Een ruimteschip bestuur je namelijk niet zomaar! Ook leer je gebruik te maken van **overerving**. Je weet aan het eind van dit hoofdstuk dan ook precies hoe het zit met ouder-objecten en kind-objecten. Verder gebruik je de **functie** `min()` en leer je een beetje **modulo**-rekenen.

Eerste versie testen

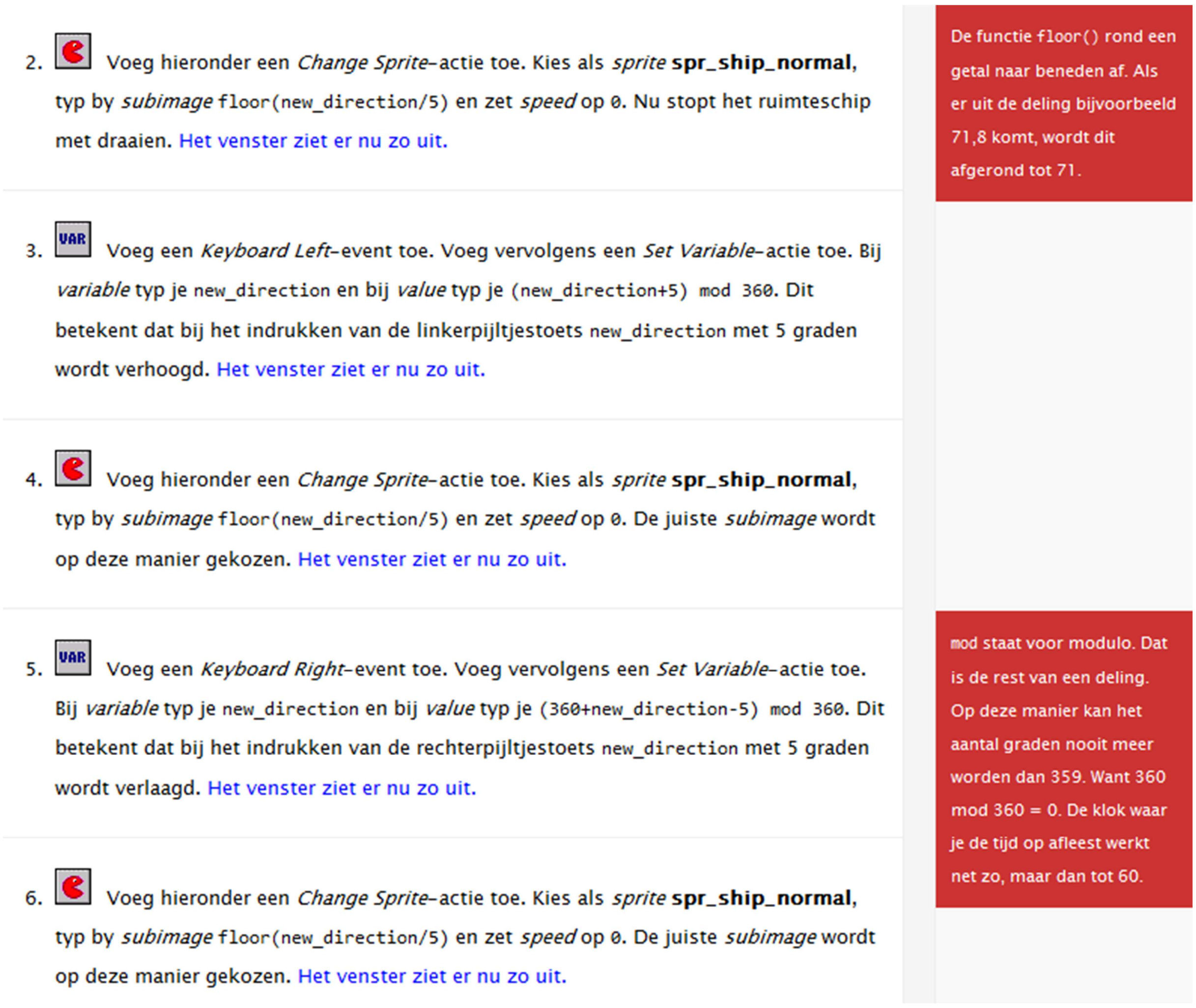
Download het zip-bestand (asteroids.zip) met daarin alle bestanden die je nodig hebt om Asteroids te maken. Hierin vind je de sprites, geluidjes, achtergronden en Game Maker bestanden.

Open nu het bestand asteroids1.gmk in Game Maker. Start het spel, zet je luidsprekers aan en bekijk wat je tot nu toe kunt doen. Dat is nog niet erg veel. Je ziet een titelscherm, rondvliegende asteroïden en af en toe een hartje. Op de achtergrond klinkt een zenuwslopend muziekje. Het belangrijkste onderdeel mist nog: het bestuurbare ruimteschip. Dit schip ga je straks zelf maken. We kijken eerst even naar de belangrijkste dingen tot nu toe.

Aan de ene kant eruit, aan de andere kant erin

Als je goed kijkt, zie je dat de asteroïdes niet verdwijnen wanneer ze het scherm uitgaan. Ze

3. Closeup of the instructions and explanations



2. Voeg hieronder een *Change Sprite*-actie toe. Kies als *sprite* **spr_ship_normal**, typ by *subimage* `floor(new_direction/5)` en zet *speed* op `0`. Nu stopt het ruimteschip met draaien. Het venster ziet er nu zo uit.

De functie `floor()` rond een getal naar beneden af. Als er uit de deling bijvoorbeeld 71,8 komt, wordt dit afgerond tot 71.

3. Voeg een *Keyboard Left*-event toe. Voeg vervolgens een *Set Variable*-actie toe. Bij *variable* typ je `new_direction` en bij *value* typ je `(new_direction+5) mod 360`. Dit betekent dat bij het indrukken van de linkerpijltjestoets `new_direction` met 5 graden wordt verhoogd. Het venster ziet er nu zo uit.

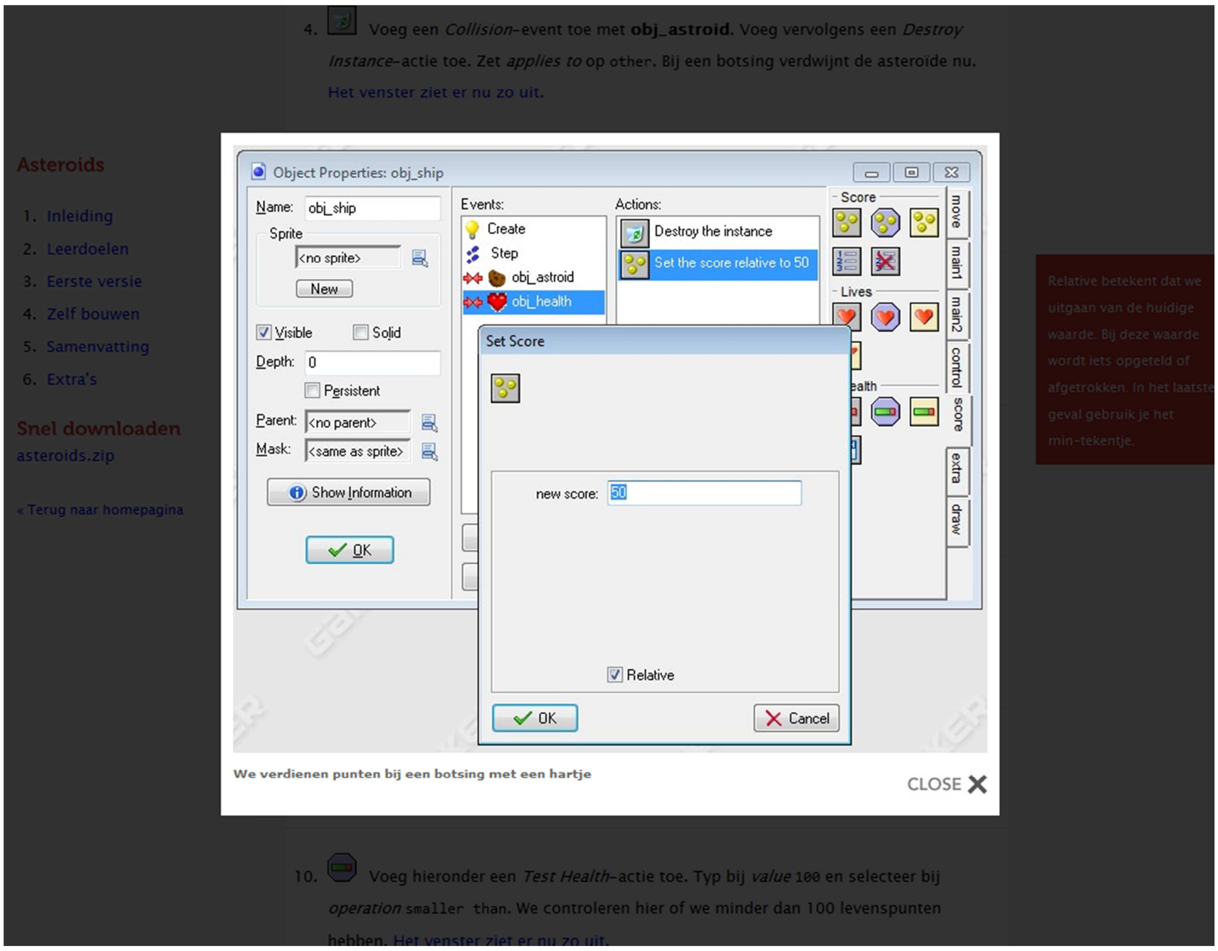
4. Voeg hieronder een *Change Sprite*-actie toe. Kies als *sprite* **spr_ship_normal**, typ by *subimage* `floor(new_direction/5)` en zet *speed* op `0`. De juiste *subimage* wordt op deze manier gekozen. Het venster ziet er nu zo uit.

5. Voeg een *Keyboard Right*-event toe. Voeg vervolgens een *Set Variable*-actie toe. Bij *variable* typ je `new_direction` en bij *value* typ je `(360+new_direction-5) mod 360`. Dit betekent dat bij het indrukken van de rechterpijltjestoets `new_direction` met 5 graden wordt verlaagd. Het venster ziet er nu zo uit.

`mod` staat voor modulo. Dat is de rest van een deling. Op deze manier kan het aantal graden nooit meer worden dan 359. Want 360 mod 360 = 0. De klok waar je de tijd op afleest werkt net zo, maar dan tot 60.

6. Voeg hieronder een *Change Sprite*-actie toe. Kies als *sprite* **spr_ship_normal**, typ by *subimage* `floor(new_direction/5)` en zet *speed* op `0`. De juiste *subimage* wordt op deze manier gekozen. Het venster ziet er nu zo uit.

4. Every instruction comes with a screen shot, showing the current state of Game Maker



4. Voeg een *Collision*-event toe met **obj_astroid**. Voeg vervolgens een *Destroy Instance*-actie toe. Zet *applies to* op `other`. Bij een botsing verdwijnt de asteroïde nu. Het venster ziet er nu zo uit.

Asteroids

1. Inleiding
2. Leerdoelen
3. Eerste versie
4. Zelf bouwen
5. Samenvatting
6. Extra's

Snel downloaden

asteroids.zip

« Terug naar homepagina

Object Properties: obj_ship

Name: obj_ship

Sprite: <no sprite>

☑ Visible ☐ Solid

Depth: 0

☐ Persistent

Parent: <no parent>

Mask: <same as sprite>

Events: Create, Step, obj_astroid, obj_health

Actions: Destroy the instance, Set the score relative to 50

Set Score

new score: 50

☑ Relative

We verdienen punten bij een botsing met een hartje

CLOSE

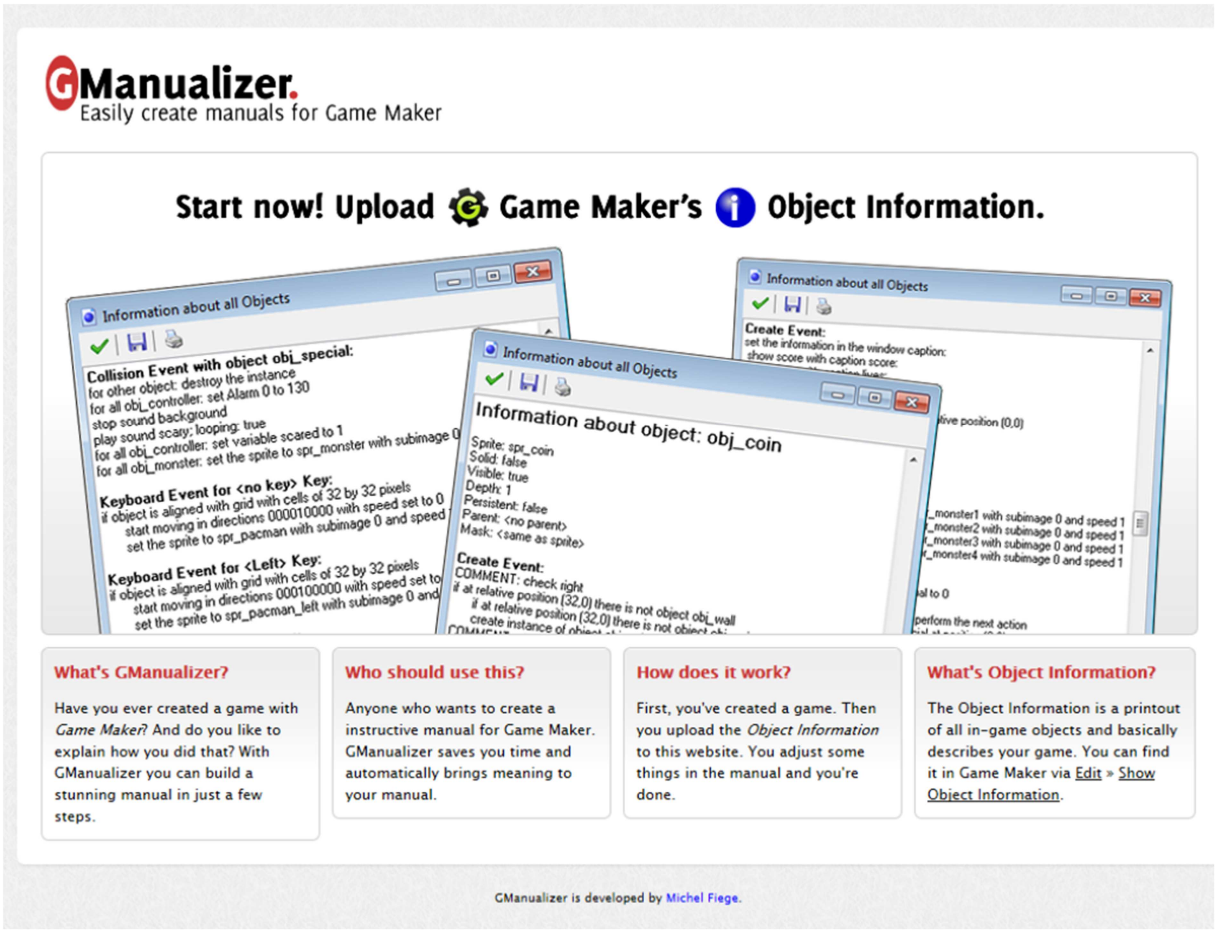
Relative betekent dat we uitgaan van de huidige waarde. Bij deze waarde wordt iets opgeteld of afgetrokken. In het laatste geval gebruik je het min-tekentje.

10. Voeg hieronder een *Test Health*-actie toe. Typ bij *value* `100` en selecteer bij *operation* `smaller than`. We controleren hier of we minder dan 100 levenspunten hebben. Het venster ziet er nu zo uit.

Appendix B

Web application: GManualizer

1. Homepage of GManualizer



GManualizer
Easily create manuals for Game Maker

Start now! Upload Game Maker's Object Information.

Information about all Objects
Collision Event with object obj_special:
for other object: destroy the instance
for all obj_controller: set Alarm 0 to 130
stop sound background
play sound scary; looping: true
for all obj_controller: set variable scared to 1
for all obj_monster: set the sprite to spr_monster with subimage 0
Keyboard Event for <no key> Key:
if object is aligned with grid with cells of 32 by 32 pixels
start moving in directions 000010000 with speed set to 0
set the sprite to spr_pacman with subimage 0 and speed
Keyboard Event for <Left> Key:
if object is aligned with grid with cells of 32 by 32 pixels
start moving in directions 000100000 with speed set to
set the sprite to spr_pacman_left with subimage 0 and
Information about object: obj_coin
Sprite: spr_coin
Solid: false
Visible: true
Depth: 1
Persistent: false
Parent: <no parent>
Mask: <same as sprite>
Create Event:
COMMENT: check right
if at relative position (32,0) there is not object obj_wall
if at relative position (32,0) there is not object
Create Event:
set the information in the window caption:
show score with caption score:

What's GManualizer?

Have you ever created a game with *Game Maker*? And do you like to explain how you did that? With GManualizer you can build a stunning manual in just a few steps.

Who should use this?

Anyone who wants to create a instructive manual for Game Maker. GManualizer saves you time and automatically brings meaning to your manual.

How does it work?

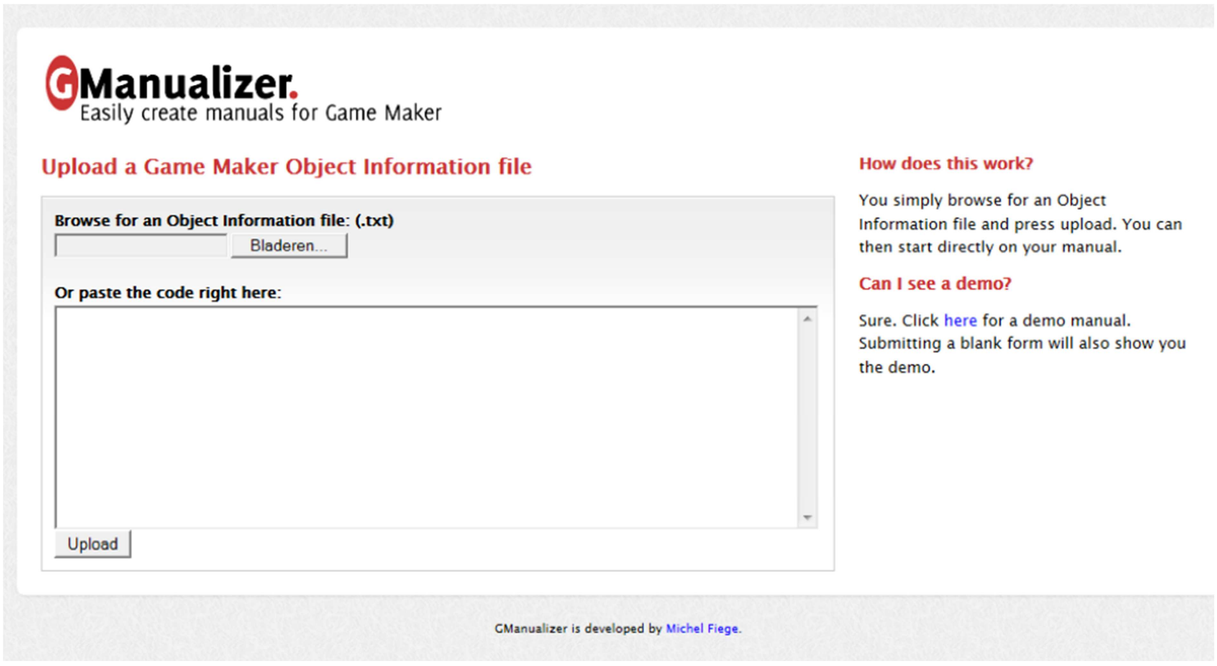
First, you've created a game. Then you upload the *Object Information* to this website. You adjust some things in the manual and you're done.

What's Object Information?

The Object Information is a printout of all in-game objects and basically describes your game. You can find it in Game Maker via Edit » Show Object Information.

GManualizer is developed by Michel Fiege.

2. Form to upload Object Information file



GManualizer
Easily create manuals for Game Maker

Upload a Game Maker Object Information file

Browse for an Object Information file: (.txt)

Or paste the code right here:

How does this work?

You simply browse for an Object Information file and press upload. You can then start directly on your manual.

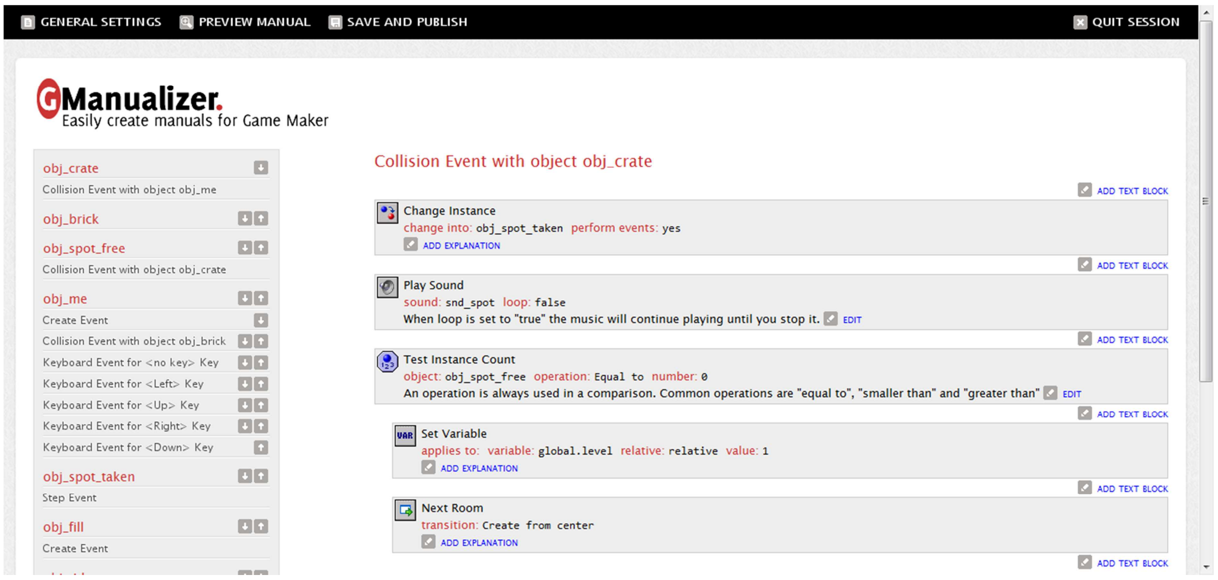
Can I see a demo?

Sure. Click here for a demo manual. Submitting a blank form will also show you the demo.

GManualizer is developed by Michel Fiege.

3. The generated manual in editing mode

Everything displayed in this image is untouched and automatticly generated. On the left the order of objects and events can be arranged by clicking on the arrow buttons.



obj_crate
Collision Event with object obj_me
obj_brick
obj_spot_free
Collision Event with object obj_crate
obj_me
Create Event
Collision Event with object obj_brick
Keyboard Event for <no key> Key
Keyboard Event for <Left> Key
Keyboard Event for <Up> Key
Keyboard Event for <Right> Key
Keyboard Event for <Down> Key
obj_spot_taken
Step Event
obj_fill
Create Event

Collision Event with object obj_crate

ADD TEXT BLOCK

Change Instance
change into: obj_spot_taken perform events: yes
ADD EXPLANATION

ADD TEXT BLOCK

Play Sound
sound: snd_spot loop: false
When loop is set to "true" the music will continue playing until you stop it. EDIT

ADD TEXT BLOCK

Test Instance Count
object: obj_spot_free operation: Equal to number: 0
An operation is always used in a comparison. Common operations are "equal to", "smaller than" and "greater than" EDIT

ADD TEXT BLOCK

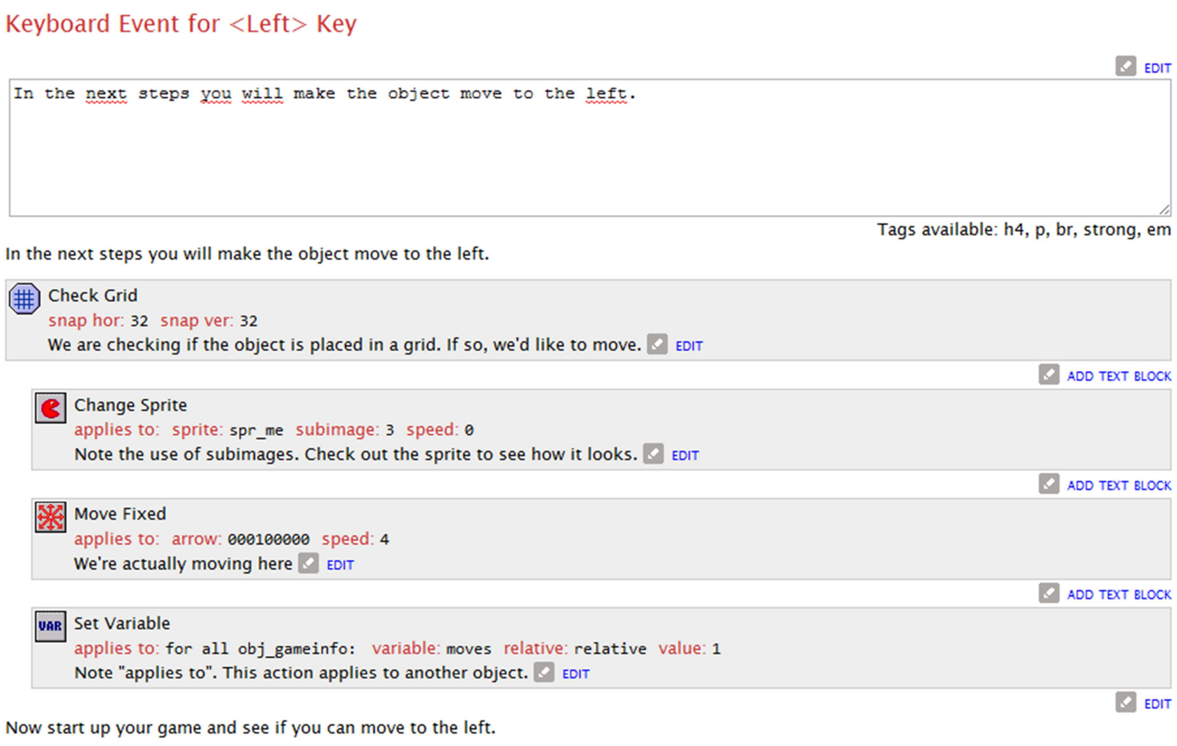
Set Variable
applies to: variable: global.level relative: relative value: 1
ADD EXPLANATION

ADD TEXT BLOCK

Next Room
transition: Create from center
ADD EXPLANATION

ADD TEXT BLOCK

4. Close up of adding text to the manual


Keyboard Event for <Left> Key

EDIT

In the next steps you will make the object move to the left.

Tags available: h4, p, br, strong, em

In the next steps you will make the object move to the left.

Check Grid
snap hor: 32 snap ver: 32
We are checking if the object is placed in a grid. If so, we'd like to move. EDIT

ADD TEXT BLOCK

Change Sprite
applies to: sprite: spr_me subimage: 3 speed: 0
Note the use of subimages. Check out the sprite to see how it looks. EDIT

ADD TEXT BLOCK

Move Fixed
applies to: arrow: 000100000 speed: 4
We're actually moving here EDIT

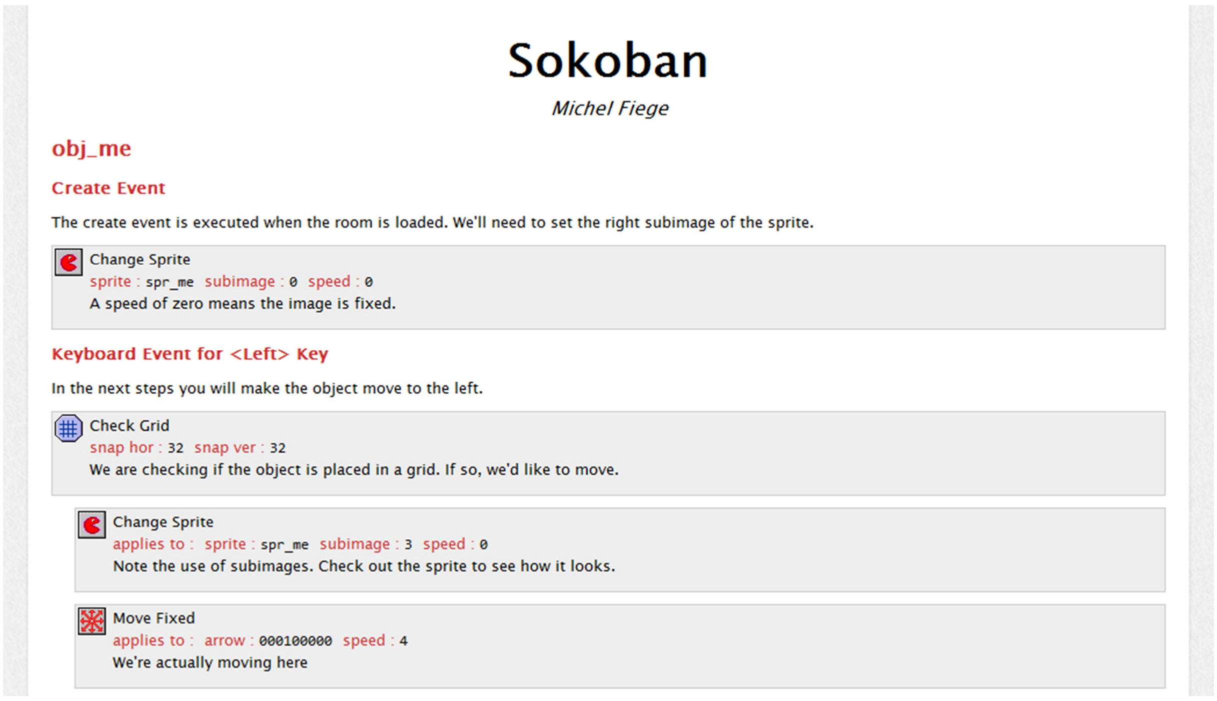
ADD TEXT BLOCK

Set Variable
applies to: for all obj_gameinfo: variable: moves relative: relative value: 1
Note "applies to". This action applies to another object. EDIT

EDIT

Now start up your game and see if you can move to the left.

5. A small part of the manual once exported



Sokoban

Michel Fiege

obj_me

Create Event

The create event is executed when the room is loaded. We'll need to set the right subimage of the sprite.

Change Sprite
sprite : spr_me subimage : 0 speed : 0
A speed of zero means the image is fixed.

Keyboard Event for <Left> Key

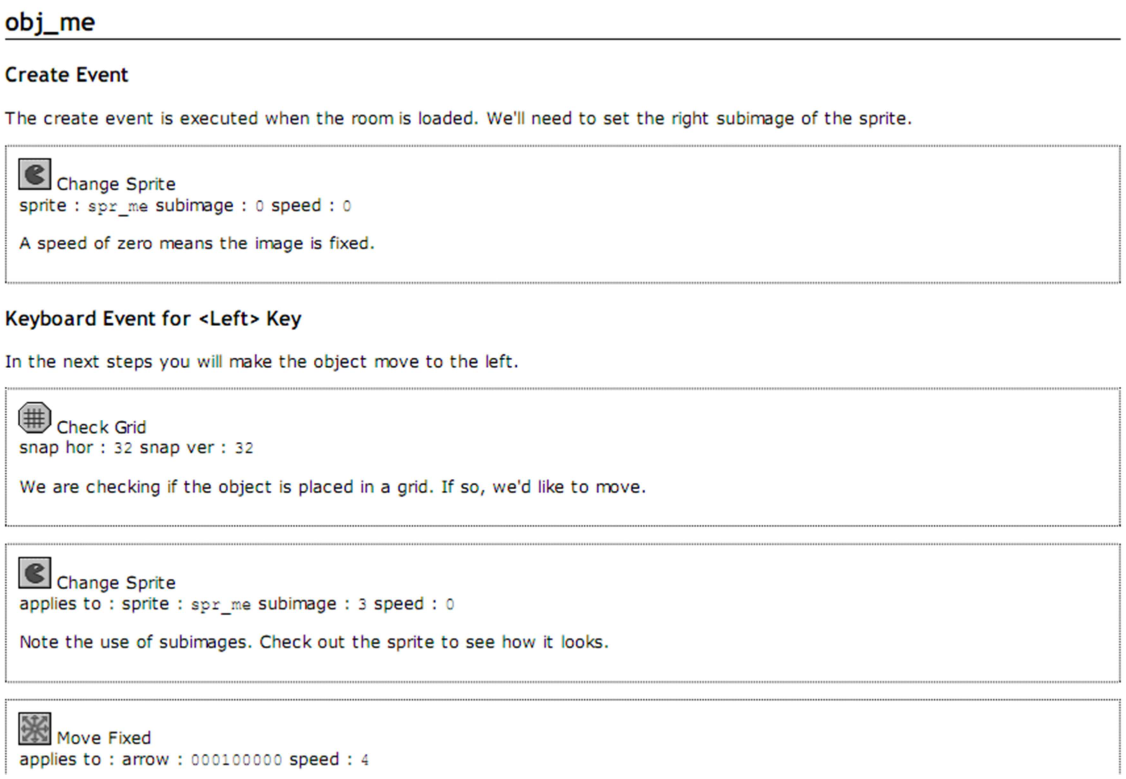
In the next steps you will make the object move to the left.

Check Grid
snap hor : 32 snap ver : 32
We are checking if the object is placed in a grid. If so, we'd like to move.

Change Sprite
applies to : sprite : spr_me subimage : 3 speed : 0
Note the use of subimages. Check out the sprite to see how it looks.

Move Fixed
applies to : arrow : 000100000 speed : 4
We're actually moving here

6. The manual in print style



obj_me

Create Event

The create event is executed when the room is loaded. We'll need to set the right subimage of the sprite.

Change Sprite
sprite : spr_me subimage : 0 speed : 0

A speed of zero means the image is fixed.

Keyboard Event for <Left> Key

In the next steps you will make the object move to the left.

Check Grid
snap hor : 32 snap ver : 32

We are checking if the object is placed in a grid. If so, we'd like to move.

Change Sprite
applies to : sprite : spr_me subimage : 3 speed : 0

Note the use of subimages. Check out the sprite to see how it looks.

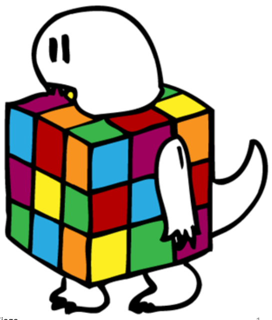
Move Fixed
applies to : arrow : 000100000 speed : 4

Appendix C

Slides Pac-Man

GAME MAKER ONLINE

Les 3 / Pac-Man



15-5-2011 Michel Fiege 1

Vandaag

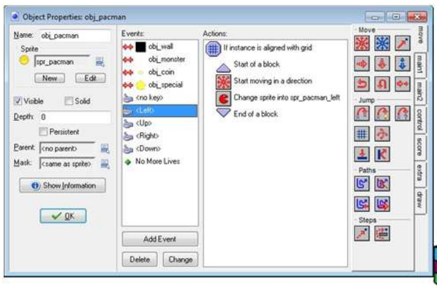
- Pac-Man
- Eerst een uitleg van wat er allemaal al werkt
- Daarna zelf aan de slag!

15-5-2011 Michel Fiege 2

Dus… Wat werkt er al?

15-5-2011 Michel Fiege 3

Pac-Man loopt…



Object Properties: obj_pacman

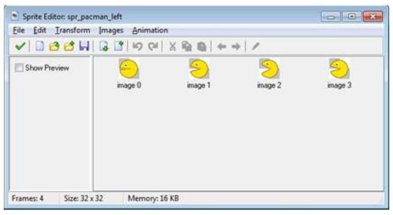
Name: obj_pacman
Sprite: spr_pacman
[✓ Visible] Solid
Depth: 0
Persistent
Parent: <no parent>
Mask: <same as sprite>

Events:
obj_wall
obj_monster
obj_coin
obj_special
<no key>
<Left>
<Up>
<Right>
<Down>
No More Lives

Actions:
If instance is aligned with grid
Start of a block
Start moving in a direction
Change sprite into spr_pacman_left
End of a block

15-5-2011 Michel Fiege 4

…en maakt gebruik van subimages

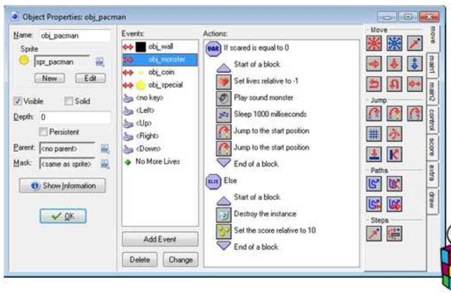


Sprite Editor: spr_pacman_left

Frames: 4 Size: 32 x 32 Memory: 16 KB

15-5-2011 Michel Fiege 5

Botsing Pac-Man en Monsters



Object Properties: obj_pacman

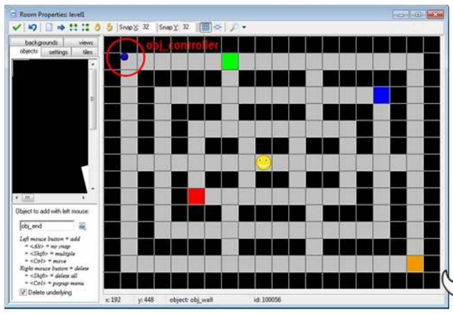
Name: obj_pacman
Sprite: spr_pacman
[✓ Visible] Solid
Depth: 0
Persistent
Parent: <no parent>
Mask: <same as sprite>

Events:
obj_wall
obj_monster
obj_coin
obj_special
<no key>
<Left>
<Up>
<Right>
<Down>
No More Lives

Actions:
If scared is equal to 0
Start of a block
Set lives relative to -1
Play sound monster
Sleep 1000 milliseconds
Jump to the start position
Jump to the start position
End of a block
Else
Start of a block
Destroy the instance
Set the score relative to 10
End of a block

15-5-2011 Michel Fiege 6

Object obj_controller



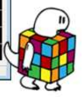
Object to add with left mouse:

obj_end

Left mouse button = add
+ <Alt> = no snap
+ <Shift> = multiple
+ <Ctrl> = move
Right mouse button = delete
+ <Shift> = delete all
+ <Ctrl> = popup menu

☑ Delete underlying

x: 192 y: 448 object: obj_wall id: 100056



Aan de slag: **Monsters** maken!

Website “Game Maker Online”

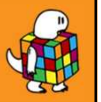
- http://gamemaker.michelfiege.nl
  - Zonder “www”
- Selecteer je naam uit de lijst
- Wachtwoord = “rsgict” of eigen wachtwoord
- Klik op inloggen

Zelf aan de slag

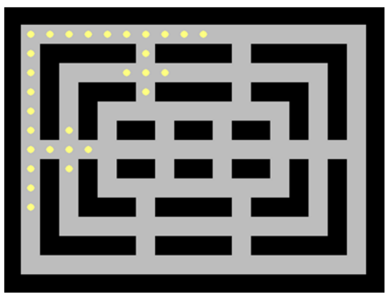
- Goed lezen
  - Gebruik de plaatjes!
- Altijd de oefenbestanden downloaden
  - In dit geval “pacman.zip”
- Help elkaar!

Succes!

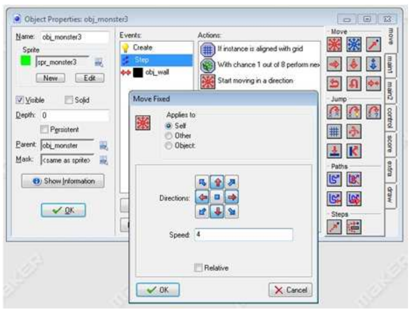
Wat heb je vandaag **geleerd?**



Recursie



Step-Event



Object Properties: obj_monster3

Name: obj_monster3

Sprite: spr_monster3

[✓ Visible] Solid

Depth: 0

Persistent

Parent: obj_monster

Mask: <same as sprite>

Events: Create, Step, obj_wall

Actions:
- If instance is aligned with grid
- With chance 1 out of 8 perform ne
- Start moving in a direction

Move Fixed

Applies to: [✓ Self] Other Object:

Speed: 4

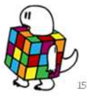
Relative

Variabelen x en y

- Elke instantie in een room heeft coördinaten:
  - x (horizontaal)
  - y (verticaal)
- Variabelen kun je opvragen en aanpassen
  - object.x = object.x + 10

Kansen

- Vergelijk met het gooien van een dobbelsteen
- Een dobbelsteen met 6 zijden:
  - Kans van 1 op 6 dat we het juiste getal raden
- Een dobbelsteen in Game Maker met 10 zijden:
  - Kans van 1 op 10 dat de onderstaande actie wordt uitgevoerd



Volgende keer…

Appendix D

Questionnaire

Enquête: Wat vond je van Game Maker?

1. Wat is je naam? ……….

2. In welke klas zit je? ☐ B1B ☐ B1C ☐ B1E ☐ B1F

Online lesmateriaal

3. Hoe leuk vond je de eerste 6 lessen Game Maker? (online lesmateriaal)
☐ helemaal niet leuk ☐ niet leuk ☐ neutraal ☐ leuk ☐ heel leuk

4. Hoe interessant vond je eerste 6 lessen Game Maker? (online lesmateriaal)
☐ helemaal niet interessant ☐ niet interessant ☐ neutraal ☐ interessant ☐ heel interessant

5. Hoe moeilijk vond je de eerste 6 lessen Game Maker? (online lesmateriaal)
☐ te moeilijk ☐ moeilijk ☐ het ging wel ☐ makkelijk ☐ erg makkelijk

6. Hoe duidelijk vond je het online lesmateriaal?
☐ helemaal niet duidelijk ☐ niet duidelijk ☐ neutraal ☐ duidelijk ☐ heel duidelijk

Eigen spel maken

7. Hoe leuk vond je de laatste 4 lessen Game Maker? (eigen spel maken)
☐ helemaal niet leuk ☐ niet leuk ☐ neutraal ☐ leuk ☐ heel leuk

8. Hoe moeilijk vond je de laatste 4 lessen Game Maker? (eigen spel maken)
☐ te moeilijk ☐ moeilijk ☐ het ging wel ☐ makkelijk ☐ erg makkelijk

Algemeen

9. Hoeveel heb je geleerd van de afgelopen 10 lessen Game Maker?
☐ heel weinig ☐ weinig ☐ neutraal ☐ veel ☐ heel veel

10. Hoe goed vind je jezelf met Game Maker?
☐ erg slecht ☐ slecht ☐ middelmatig ☐ goed ☐ erg goed

11. Hoe nuttig vond je de 5 minuten aan het begin van elke les?
☐ helemaal niet nuttig ☐ niet nuttig ☐ neutraal ☐ nuttig ☐ heel nuttig

12. Hoe nuttig vond je de 5 minuten aan het einde van elke les?
☐ helemaal niet nuttig ☐ niet nuttig ☐ neutraal ☐ nuttig ☐ heel nuttig

Tot slot

13. Heb je nog opmerkingen over de afgelopen 10 lessen Game Maker?

Appendix E

Written test

Overhoring Game Maker

1. Wat is een **object**? Geef een voorbeeld in Game Maker.

Een object is…

Bijvoorbeeld...

2. Wat is een **instantie**? Geef een voorbeeld in Game Maker.

Een instantie is…

Bijvoorbeeld...

3. Wat is een **variabele**? Waarvoor gebruik je variabelen in Game Maker?

Een variabele is…

Je gebruikt variabelen voor…

4. Wat is een **event**? Geef 5 voorbeelden van events in Game Maker.

Een event is…

Vijf voorbeelden van events zijn…

5. Waarvoor gebruik je een **if** en een **else**? Geef een voorbeeld in Game Maker.

*Een **if**…*

*Een **else**…*

Bijvoorbeeld…

6. Wat is een **iteratie?** Geef een voorbeeld in Game Maker.

Een iteratie is…

Bijvoorbeeld…

7. Wat is een **functie**? Geef 3 voorbeelden van functies in Game Maker.

Een functie is…

Drie voorbeelden van functies zijn…

8. Wat is **overerving**? Waarvoor gebruik je overerving in Game Maker?

Overerving is…

Je gebruikt overerving voor…

9. Wat is **recursie**? Waarvoor gebruik je recursie in Game Maker?

Recursie is…

Je gebruikt recursie voor…

10. Bekijk onderstaand scripts. Welke waarde wordt geprint op de laatste regel (`print(A)`)?

Script 1

```
A = 2;
B = 1;
A = A + 10;
A = A + B;
print(A);
```

Script 2

```
A = 2;
A = A * 4
als(A > 10) doe:
   A = A – 1;
anders
   A = A+3;
einde als;
print(A);
```

Script 3

```
A = 2;
B = 0;
terwijl(B < 10) doe:
  A = A + 1;
  B = B + 1;
einde terwijl;
print(A);
```

Antwoord script 1:	Antwoord script 2:	Antwoord script 3:

Appendix F

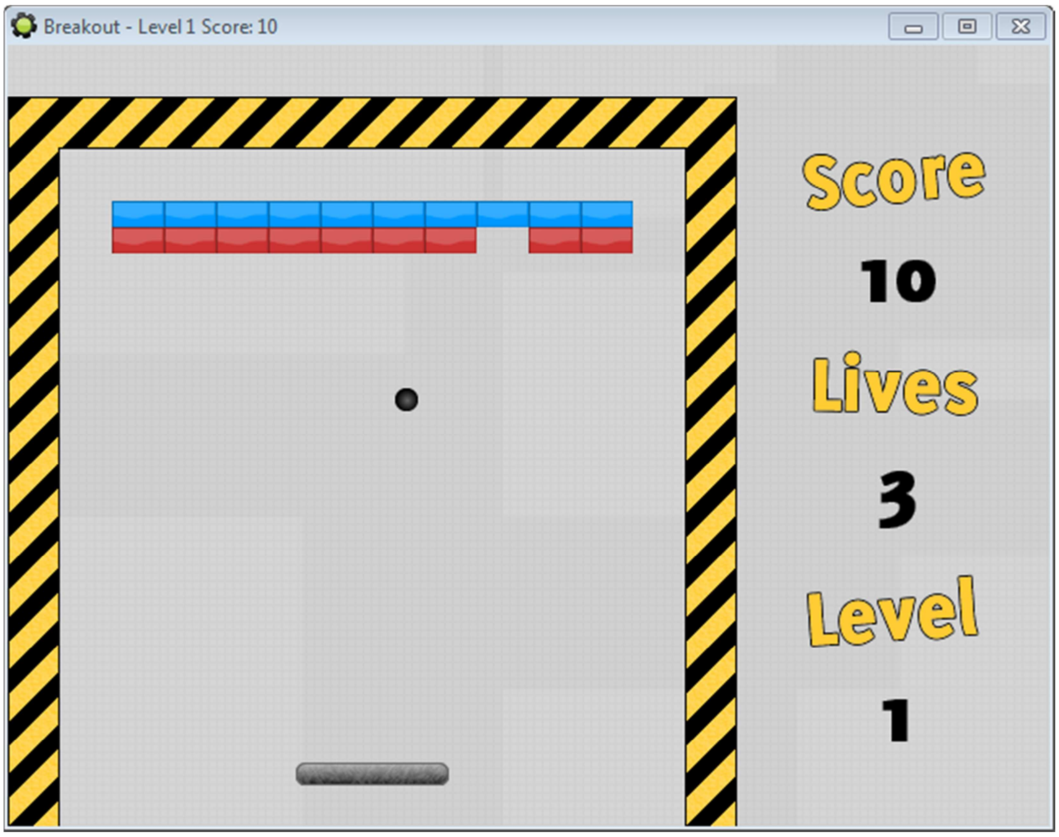
The 14 rated games

THE KRUSTY KRAB

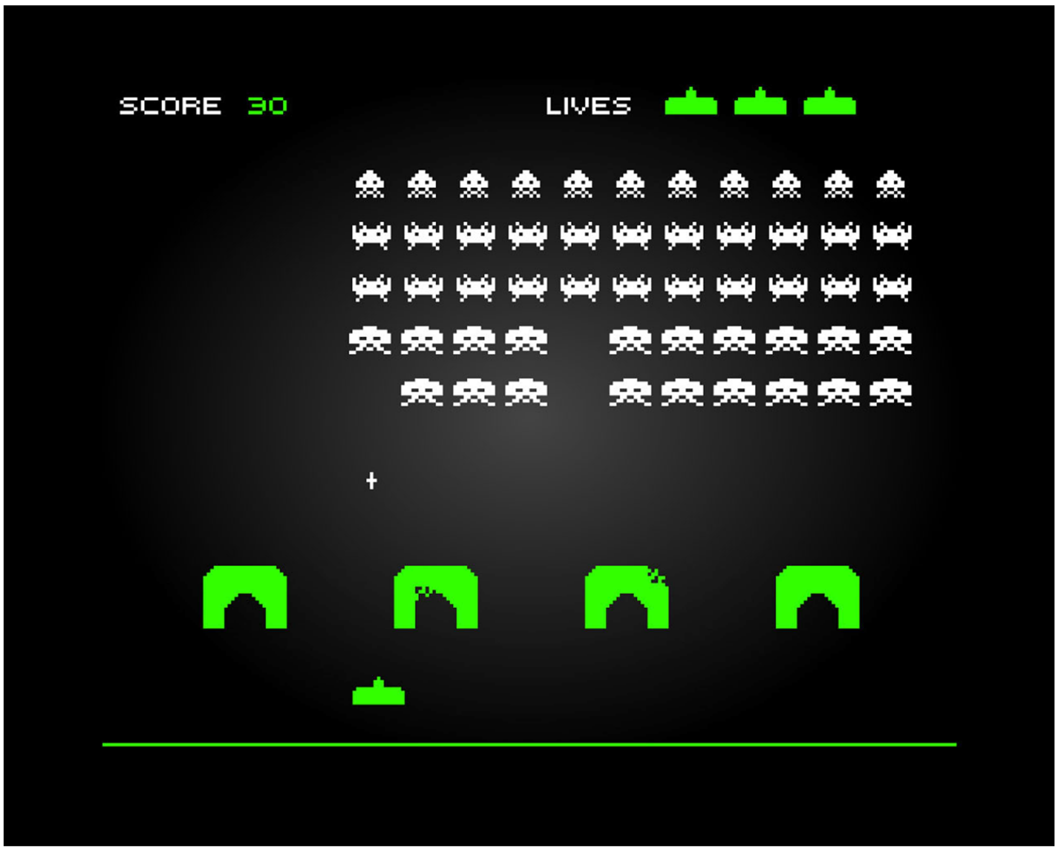
Appendix G

Screen shots used in the interviews

Screen shot used in pre-test



Screen shot used in post-test



Appendix H

Examples of writings during interview

Justus pre-test

object platform

[illegible] botst met bal, bal kaats terug
pijltjestoetsen kan je hem laten bewegen

object bal

botst met de muur kaats die ook terug
botst met het blokje verdwijnt het blokje en gaat de score
naar 10
bots blokje sound

Ga naar de volgende kamer als alle blokjes zijn weggespeeld
dus score 200

Justus post-test

Obj. – platform

	event	actie
pijltjes	links	links
	rechts	rechts
	Spatie	kogel afvuren

Obj. – muur

bots kogel	delete

Obj. – monster

create	Move
~~mous~~	links rechts
bots muur	[illegible] gaat

Muur aan de zijkanten

Tessa pre-test

objects

- plankje
  bewegen naar rechts en links
  pijltjes zorgen daar voor

- balletje
  verdwijnt het blokje
  je krijgt punten erbij
  ♡ misschien levens erbij

Tessa post-test

- Sprites maken
- object ruimteschip
  event: keyboard left   acties: beweeg links
  right   beweeg rechts
  keyboard up   komt lazer naar boven

Punten: als de lazer een monster raakt

object: lazer: collision → monster   acties: Punten
other
self

Kimberley pre-test

De sprite maak je met de mapjes aan de eijkant van het beeldscherm.
Het balletje kaats tegen ~~de~~ het plankje het plankje kaatst het balletje naar een blokje (als je ~~ka~~ goed met het plankje stuurt) als het balletje een blokje raakt verdwijnt dat blokje en komen er punten bij.

Kimberley post-test

Ruimteschip:

event	actie
keyboard left	Rode Pijltjes $\boxed{*}$ $\boxed{\rightarrow}$
keyboard right	Rode Pijltjes $\boxed{*}$ en dan $\boxed{\leftarrow}$
collision laser (monster)	(verliest leven) bij score Set lives -1 laser verdwijnt komt terug bij begin

Monster

collision laser (ruimteschip)	verdwijnt monster + score erbij

Sven pre-test

het balletje begint op het Plankje, na de ? seconden vliegt het balletje weg.

$\boxed{K^{7}}$ als de bal en het blokje elkaar raken gaat het blokje weg.

Sven post-test

Sven post-test

Keyboard ~~lt~~ left move links
event actie

collission van laser met vijand vijand in de Ballenbak en score + 10 relatief

Collission ~~met~~ van laser met tunnel Tunnel word verwijdert

collission laser vijand met Plankje lives -1 relatief