universität wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

Developing computer games in secondary schools - The influence of Game Development on computer science competencies

verfasst von / submitted by

Mag. Oswald Comber

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2016

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 786 880
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Informatik
Betreut von / Supervisor:	Ao.Univ.-Prof. Dipl.-Ing. Dr. Renate Motschnig

Table of contents

Acknowledgements

I heartily thank my colleagues, my students, my friends, and my family, to whom I am thankful in so many ways.

Also, there was one particular person who made these exciting and enlightening years of research possible. Thanks to my thesis adviser ao. Univ.-Prof. Dr. Renate Motschnig, who was the main reason why I started, stuck to, and finished my thesis. Thank you for the constant encouragement and support!

Dedication

To my grandmother, one of my greatest teachers, for helping me learn about those things in life that I could never learn in school!

Abstract

Can the development of video games positively influence the learning of programming? To answer this question in the context of computer science classes in an upper secondary school with 15-year-old teenagers, separate groups of students were accompanied in their programming learning activities in classroom via design based research. The results of three years of research confirm that developing a computer game triggers high motivation and engagement as long as a few key factors are met. These factors include (1) *realistic expectations* regarding feasible goals that must be communicated and recognized by all participants; (2) *reduced complexity* during the development process wherever possible; and (3) *preparedness* of the game development environment and the teacher. To reduce complexity and move beyond programming learning environments, I developed a specialized framework called Game Programming in Schools (GamePinS) and used it for three years in a secondary school.

In addition to motivation and engagement, the essential question posed was, *"Can developing a computer game boost the IT skills of students even more than other software development scenarios?"* To answer this question, students doing game development with GamePinS were compared with students implementing their own selected software projects with the help of a programming skills test in two subsequent school years. While questions focused on basic *programming skills* showed that both groups performed equally, students who did game programming outperformed the other students in the fields of *logic*, *understanding source code*, and *analytical thinking*.

Kurzfassung

Kann die Entwicklung von Computerspielen das Programmieren-lernen verbessern? Um diese Frage im Kontext des Unterrichts von 15jährigen Schülerinnen und Schülern im Realgymnasium zu beantworten, wurden Lernende in unterschiedlichen Gruppen mit Hilfe eines Design-Based-Research-Ansatzes forschend begleitet. Die Resultate aus drei Jahren Feldforschung, lieferten die Bestätigung, dass die Entwicklung von Computerspielen mit hoher Motivation und hohem Engagement einhergeht. Als entscheidend für eine förderliche Wirkung der Spieleentwicklung wurden drei Schlüsselfaktoren identifiziert. Erstens, es müssen realistisch erreichbare Spieleprojektziele gesteckt werden. Dies geschieht durch eine offene Kommunikation, welche Spiele, in welchem Umfang, im Informatikunterricht im Bereich des Möglichen liegen und welche Spiele nicht realisierbar sind. Zweitens, die Komplexität der Spieleentwicklung sollte soweit wie möglich reduziert werden. Zu diesem Zweck wurde das Framework GamePinS entwickelt und über drei Jahre hinweg im Unterricht eingesetzt und begleitend erforscht. Drittens: Eine große Herausforderung liegt in der Vorbereitung. Um erfolgreich Spiele im Informatikunterricht zu entwickeln, müssen die verwendeten Werkzeuge sowie die Lehrperson sehr gut vorbereitet sein.

Neben der Erforschung der Motivation und des Engagements durch Spieleentwicklung wurde noch einer weiteren sehr zentralen Frage nachgegangen: *„Kann Spieleentwicklung die IT-Kompetenzen der Schülerinnen und Schüler besser fördern als vergleichbare andere Softwareentwicklungsprojekte?“.* Um diese Forschungsfragen zu beantworten, wurden über zwei Schuljahre hinweg insgesamt sechs verschiedene Gruppen an Schülerinnen und Schülern, die von zwei unterschiedlichen Lehrern unterrichtet wurden, verglichen. Die Resultate des IT-skills-Tests zeigten, dass bei grundlegendem Programmierwissen keine Unterschiede auftraten. In den Bereichen der Logik, des Verstehens von Quellcode und dem analytischen Denken erzielten jedoch die Schülerinnen und Schüler, die mit der Spieleentwicklung beschäftigt waren deutliche bessere Ergebnisse, als die Lernenden, die sich andere Softwareprojekte realisierten.

1 Introduction

Video games are a widespread form of entertainment and a growing economic factor (Entertainment Software Association, 2015), but most computer games are excitement and fun for young persons[1]. Playing a computer game is an engaging task that nurtures high motivation and, in most cases, an intensive engagement. Serious games utilize this “fascination for games” by delivering educational content in a playful way (Halbeisen, 2011; Michael & Chen, 2005). In addition to serious games, there is another opportunity to utilize games as a catalyst for successful learning, i.e., *developing computer games.*

Developing computer games not only boosts engagement and motivation but also covers a wide variety of computer science challenges. By using game development in a teaching environment, many aspects of computer science can be covered. In the area of programming, countless coding concepts can be delivered to a class via game development. Some, but far from all, important ideas include algorithms (e.g., sorting, path finding), modeling, and basic programming concepts (e.g., data types, loops, object-oriented programming). Beyond programming, numerous other activities are viable scenarios for learning, including graphics design, audio engineering, developing game concepts, logical structures, levels or stories, coordinating teamwork, and leading software projects.

Virtually, nothing stands in the way of developing a computer game in a classroom as long as teachers identify and allocate the correct scenarios, time, hardware equipment, and appropriate tools. This can certainly be more difficult than it may appear to be. Therefore, in this thesis, all the steps from the initial idea to the exploration of tools and possibilities and the implementation and successful use of a game development framework in a school are scientifically explored, explained, and evaluated. The reader is invited to join the ongoing five-year research journey through the creative jungle of game development.

[1] Here I refer to young persons, nevertheless, games are no longer mainly enjoyed by young people, but are being played by people of every age and gender (Entertainment Software Association, 2015).

2 Motivation

When I asked my students, “which words come to mind when you think of games?,” they enthusiastically flooded me with titles of their favorite games and all at once told me about the content and stories of their games. I therefore asked them to fill out a short survey on their gaming behavior and answer the question of why they play computer games. The most common answers were “for fun,” “to release tension,” “otherwise I’d be bored,” and “to feel better.”

Seeing the enormous amount of resources young people spend on *competing with each other*, *building cities*, *exploring dungeons*, *scoring spectacular goals*, and *conquering virtual worlds*, I immediately thought of channeling some of this energy for the good of education.

Another issue that has troubled me is that my colleagues and I are primarily teaching how to use Microsoft Office applications. The daily routine in our computer science classrooms unfortunately consists of writing letters, performing calculations with spreadsheets, entering data into databases, and creating PowerPoint presentations. These topics are certainly very useful, but in terms of computer science topics covered, this approach fails to address even the most basic aspects of what computer science is, i.e., information and its processing, algorithms, structures, and patterns. Further, algorithms, structures, and patterns cannot stand alone; to make them work in the daily routine in school and deliver a real understanding, they must be implemented using a language—in most cases, this is a programming language.[2]

From the very beginning, programming itself has been another challenging issue for my students and me because it was connected with such terms as “complicated,” “complex,” and “boring.” There are numerous approaches and environments for reducing the complexity of learning to program, including Scratch, Scratch derivatives such as Snap! (Mönig & Harvey, 2013), formerly known as BYOB, Logo, NetLogo and Star Logo TNG, Microworlds (Papert, 1987), Sqeakland eToys and AgentSheets, but despite these useful

[2] Or in the case of patterns, preferably a modeling language would be used here.

approaches, it remains a challenge to foster and preserve the attention and motivation of young students while also raising the level of complexity of the lessons to learn. Game programming can be the key to making programming more attractive to young people.

3 Research questions, areas of research, and related research

3.1 Research questions

From initial thoughts regarding games and how to make computer science more attractive, four main research questions are identified, all of which have in common that not primarily *playing*[3] but rather *creating games* is the central activity. These four questions are listed below.

(Q1) Can game programming in computer science increase the engagement of students?

(Q2) Does project-based game programming in computer science foster competence in teamwork?

(Q3) Does game programming promote basic concepts of computer science more effectively than typical[4] software development scenarios?

(Q4) Does the complexity level of the framework have an influence on the depth of understanding the concepts to be learned?

To answer these key questions, it is essential to develop the necessary utilities for programming in a secondary school environment. In the most direct way, this might be realized by deriving utilities from existing game development toolkits. In addition, it seems vital to provide a didactical embedding. Finally, to verify the given research questions, four assumptions are formulated and tested with the help of a multivariate approach, i.e., interviews, questionnaires, and a programming skills test.

3.2 About games

According to *Statistics Austria*, approximately one-quarter of young persons between 10 and 19 years of age play computer games for an average of 1.5 hours per day (Statistik

[3] Of course, during their development, games must be played for testing purposes.

[4] The first question to ask before comparing these methods is, “What are typical software development scenarios?” (see Section 5.2.3.8).

Austria, 2010). Addressing the specific secondary school target audience directly, in a short survey regarding computer usage habits (Comber, 2012b), approximately three-quarters of the participants stated that they play computer games. A more extensive survey with 302 participants in 2015 yielded the results shown in Figure 1.

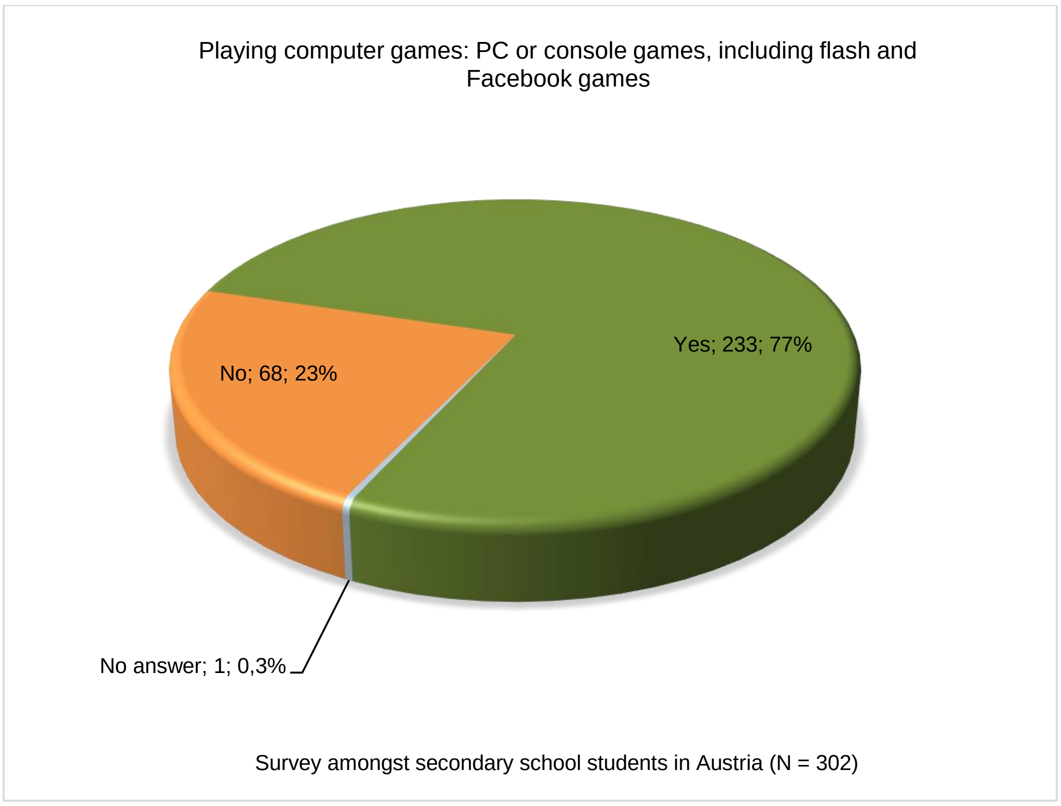


Figure 1 – Results of a survey showing the number of students who play computer games

3.2.1 What constitutes playing a game, and what is a game?

In general, playing can be defined as any "activity that is not required, but is enjoyed" (Strickland, 2001b). Play is, in most cases, autotelic, i.e., *"it is engaged in for its own sake, with the reward inherent in the activity itself"* (Strickland, 2001b).

A game is indeed an interesting combination of numerous aspects. It has various dimensions and differs from related forms of activities in at least one facet. A game distinguishes itself from pure competition, pure entertainment, and a simple challenge by combining all of these aspects. A detailed distinction is expressed by Crawford (2003) in Figure 2.

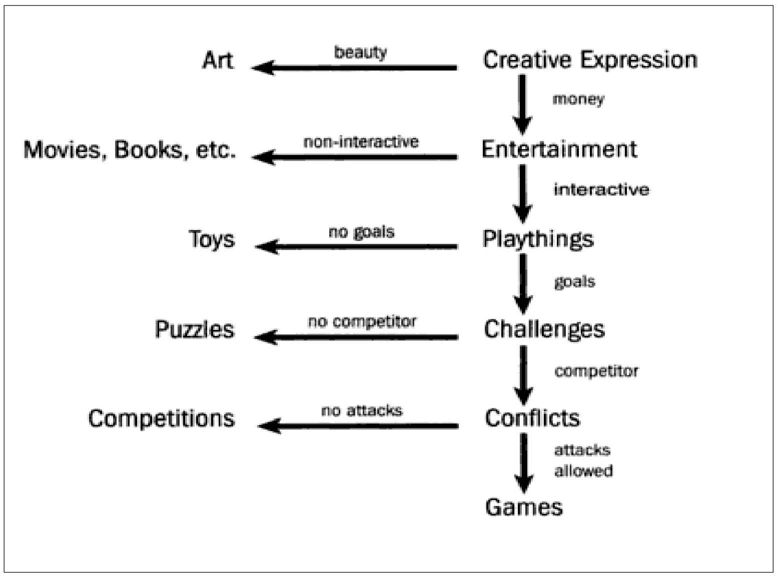


Figure 2 – Chris Crawford on game design (Crawford, 2003, p. 6)

Playing a game might act as a means of venting and releasing stress from our sometimes very demanding daily routine. Although playing might build up some tensions of its own, *"the ultimate result is the reduction of tension and conflict"* (Strickland, 2001b).

A survey conducted in 2012 (Comber) was distributed to secondary school students[5] that fit into the target audience. In this survey, 72% of students answered the question of playing computer or console games positively. The main reasons for playing games were for "fun" (57), out of "boredom" (26), "to release tension" (6), and other reasons (34)[6].

[5] N (survey participants overall) = 140; N_1 (question item age answered) = 139; Age: 93% between 13 and 17 years old, with 42% of students exactly 15 years old.

[6] Multiple answers possible, total answer count: 123.

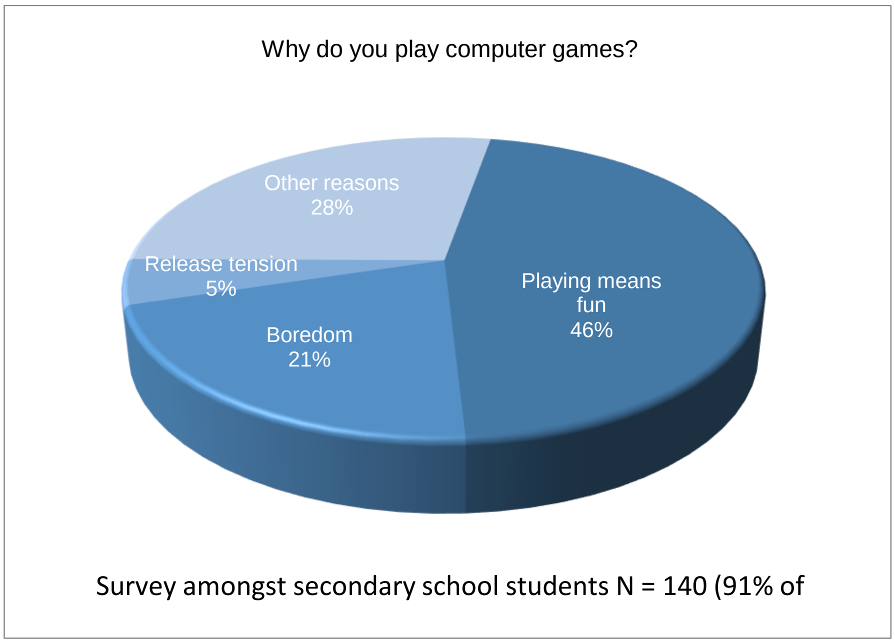


Figure 3 – Survey results showing reasons for playing computer games (Comber)[7]

Games in education can cover a broad variety of different approaches. First, there is the area of serious games (Susi, Johannesson, & Backlund, 2007) that deliver educational content in a playful and frequently entertaining way (Ritterfeld, Cody, & Vorderer, 2010). Second, creating one’s own game offers a promising chance. Game development can be accomplished via high-end game tools, or a game can even be implemented from scratch using an interface like DirectX or OpenGL. While these options are possible, they are not always feasible. In our case, creating a game with existing and adapted tools seemed the best fit, but let us first take a closer look at the different game development toolkits below.

3.3 Game development toolkits

In this section, the overview of game development toolkits starts with learning environments that provide a low threshold, progresses to more complex tools with a medium threshold, and moves on to powerful game engines with a high threshold.

[7] Also compare this with the broader survey described in Section 5.1.

3.3.1 Educational programming languages

Educational programming languages are designed with the intention of being easy to learn, easy to stay with, and easy to use. In the subsections that follow, selected educational programming languages are described.

3.3.1.1 Scratch, Snap!, and BYOB and Squeakland eToys

Scratch is an educational programming language that provides a low threshold to use and is easy to learn. Scratch was developed by the Lifelong Kindergarten Group under the lead of Mitchell Resnick at the Massachusetts Institute of Technology (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). The current version at the time of writing is 2.0. While previous versions of Scratch were implemented in Squeak, a Smalltalk dialect, Scratch 2.0 is implemented using Adobe Flash (see Scratch 2.0 Wiki ") and is shown in Figure 4. Scratch has many aspects in common with Squeakland eToys, which was developed 1996 by Alan Kay (2005).

A key aspect of Scratch is that programming is not done by typing in code, but rather by dragging and dropping code blocks from a scripts area (B in the figure) to the programming area (C in the figure). Scratch 2.0 supports new features, including *Cloning sprites at runtime*, *Cloud Data*, and *Procedures*, a feature for building your own blocks that was originally introduced by BYOB (Mönig & Harvey).[8]

Snap! (Mönig & Harvey, 2013) is a reimplementation of Scratch intended to extend Scratch 1.4 with first-class lists, first-class procedures, and continuations (Mönig & Harvey, 2013). Scratch took some of the most popular ideas from Snap! and BYOB and implemented them in the new release of Scratch noted above. The most influential feature was the ability to build and define one’s own blocks of code.

[8] For details see the Scratch 2.0 webressources in the literature section

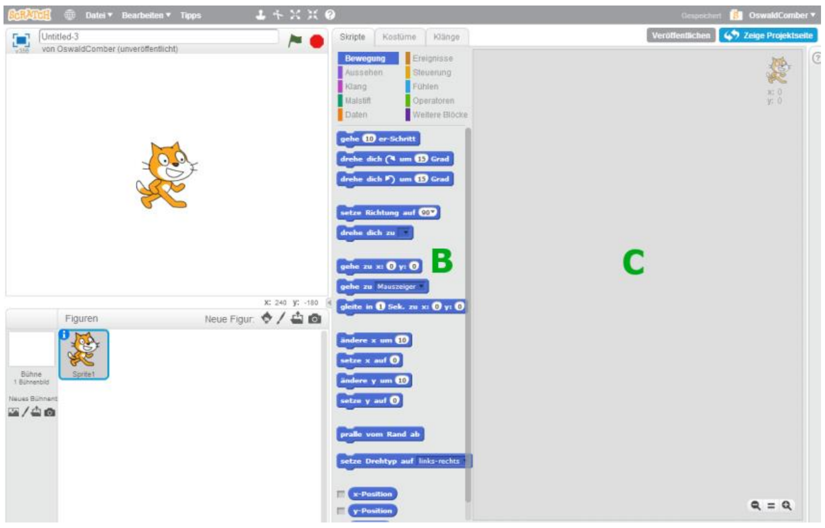


Figure 4 – The graphical user interface (GUI) of Scratch 2.0[9]

As shown in Figure 5, a simple game, e.g., a game called “Zatacka,” also known as “Achtung die Kurve!”, can be implemented very quickly using any of these three languages (using Scratch in the figure).

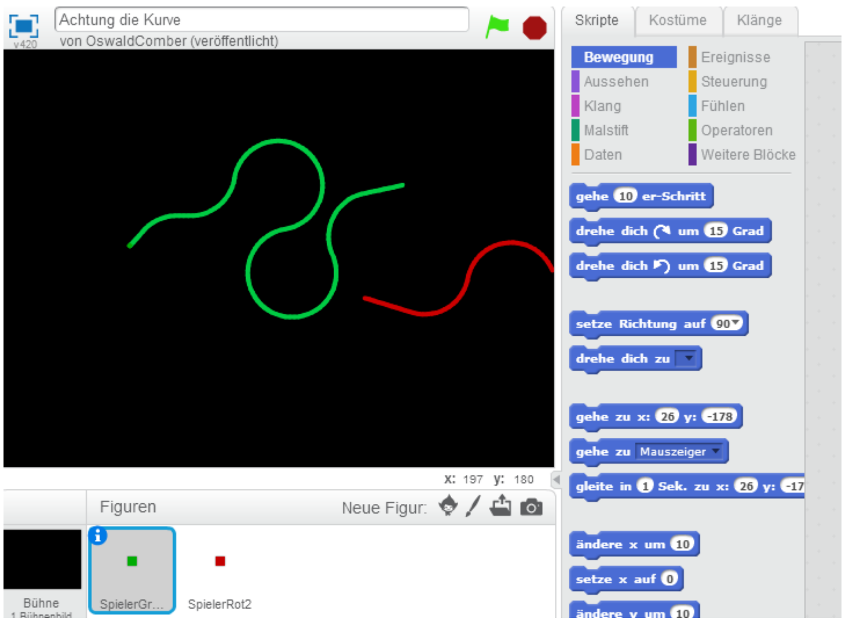
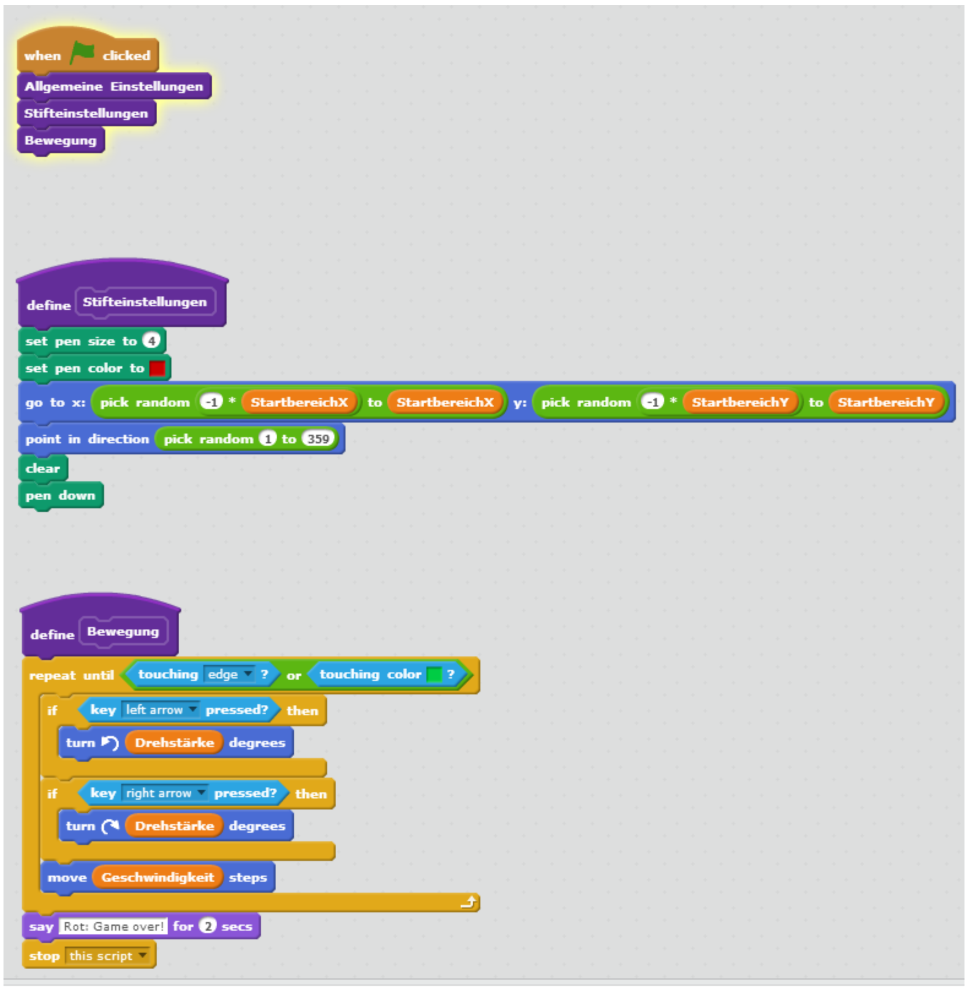


Figure 5 – The “Achtung die Kurve!” game implemented using Scratch

[9] Screenshot captured on August 14, 2013 from http://scratch.mit.edu/.

Figure 6 shows the full source code of a simple version of the “Achtung die Kurve!” game, as used in class.[10]



```
when 🏁 clicked
Allgemeine Einstellungen
Stifteinstellungen
Bewegung

define Stifteinstellungen
set pen size to (4)
set pen color to 🟥
go to x: (pick random ((-1) * (StartbereichX)) to (StartbereichX)) y: (pick random ((-1) * (StartbereichY)) to (StartbereichY))
point in direction (pick random (1) to (359))
clear
pen down

define Bewegung
repeat until <<touching [edge ▾] ?> or <touching color 🟩 ?>>
    if <key [left arrow ▾] pressed?> then
        turn ↺ (Drehstärke) degrees
    if <key [right arrow ▾] pressed?> then
        turn ↻ (Drehstärke) degrees
    move (Geschwindigkeit) steps
say [Rot: Game over!] for (2) secs
stop [this script ▾]
```

Figure 6 – Source code for the “Achtung die Kurve!” game, as used in starter lessons

3.3.1.2 *Agent Sheets and AgentCubes*

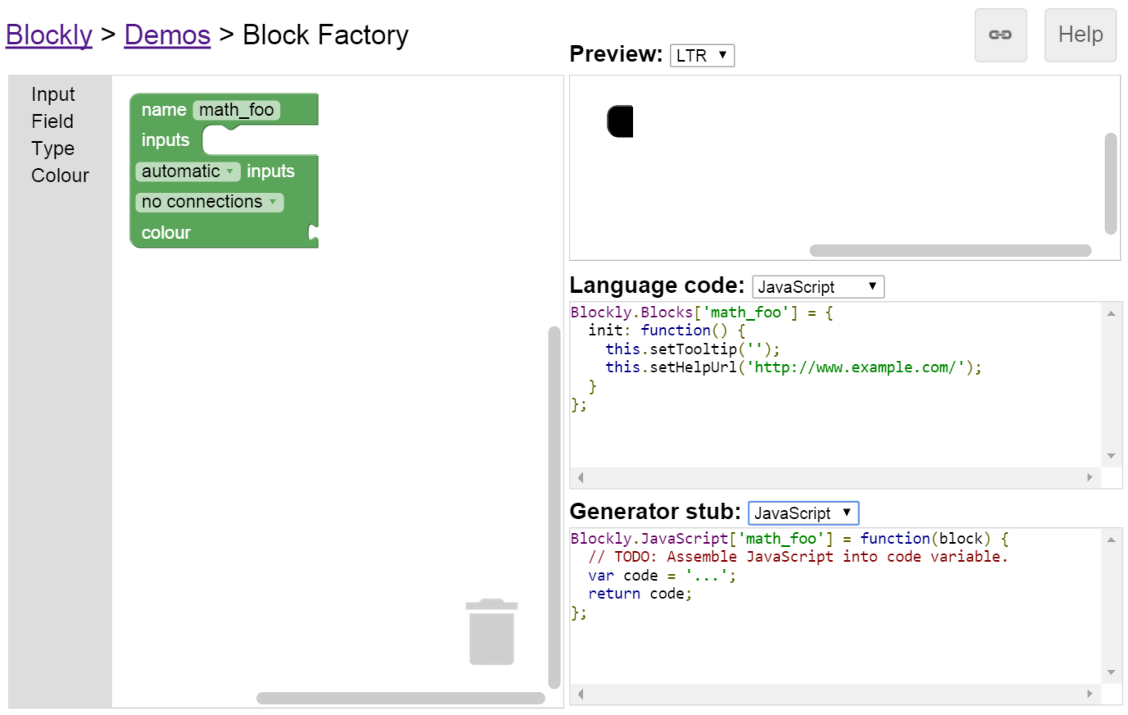
Agent Sheets (Repenning & Sumner, 1995) was the first programming learning environment, which was the first learning IDE to support the idea of dragging and dropping the parts of sourcecode (Repenning, 2014b). In the evolution of Agent Sheets the focus on Scalable Game Design was introduced to boost motivation (Repenning, Webb, &

[10] A short translation of German terms is given here: “Allgemeine Einstellungen“ means “general settings,“ “Stifteinstellungen” means “pen settings,” “Bewegung” means “movement,” “Startbereich” means “starting area,” “Drehstärke” means “rotation,” and “Geschwindigkeit” means “speed.”

Ioannidou, 2010) and the support computational thinking by providing information about the meaning of the program should enable deeper insights and foster computational skills (Repenning, 2011). AgentCubes (Repenning, 2014a) is the 3D-version of AgentSheets and comes with several features to enhanced developing, amongst them there is a possibility to draw 2D-objects and inflate them to 3D . The only downside of the products is, that Agent Sheets Inc. went commercial and therefore AgentCubes and AgendSheets is not completely free like Scratch or many of the other tools.

3.3.1.3 Blockly, Gameblox, Gamefroot, and Roberta

Blockly is an open source library developed by Google that allows developers to build visual programming editors using a block system similar to that of Scratch. Blockly is Web-based and does not require Flash; more specifically, it is a completely client-side application requiring only 150 KB in compressed size. Blockly offers the ability of extension with custom blocks. Blockly also implements a feature to export code as either JavaScript or Python (see the Blockly Website).



Language code: JavaScript

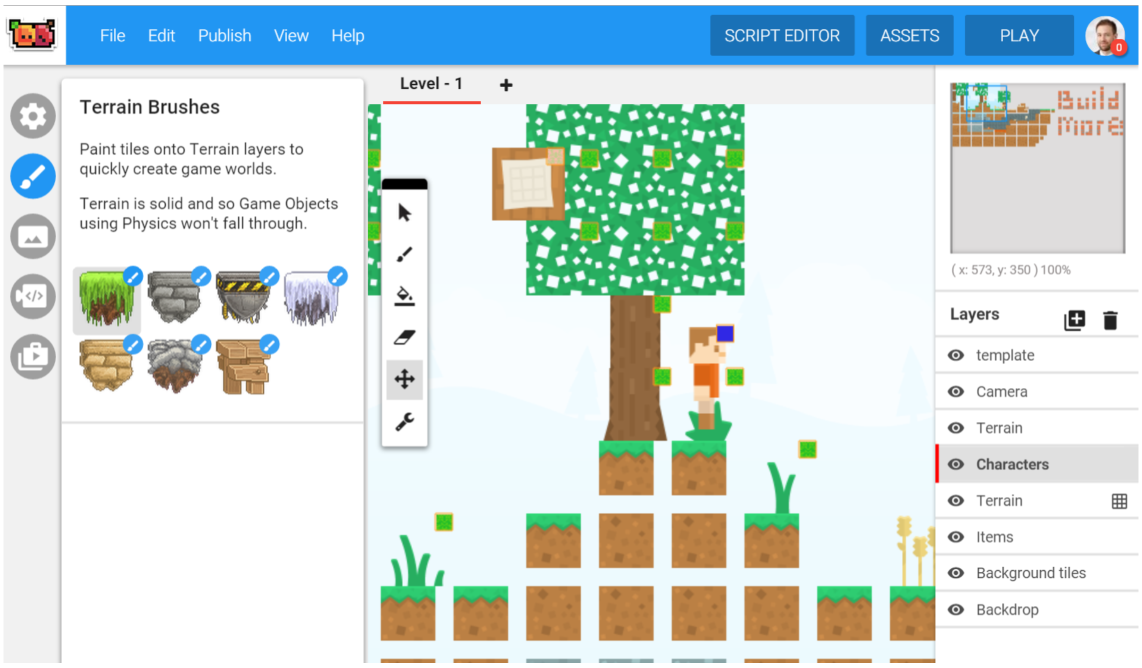
```
Blockly.Blocks['math_foo'] = {
  init: function() {
    this.setTooltip('');
    this.setHelpUrl('http://www.example.com/');
  }
};
```

Generator stub: JavaScript

```
Blockly.JavaScript['math_foo'] = function(block) {
  // TODO: Assemble JavaScript into code variable.
  var code = '...';
  return code;
};
```

Figure 7 – Blockly block factory interface

Based on Blockly, game development tools Gameblox (MIT Scheller Teacher Education Program, 2014) and Gamefroot (2013) are both toolkits focused on simple block-based game development with a low threshold. Gamefroot offers a decent library of game-building resources, such as graphics or tiles, for free, as well as some paid content. In general, Gamefroot may be used for commercially developed games. Gameblox is developed by the MIT Scheller Teacher Education Program and supports learning game programming or even learning programming through game programming.



Paint tiles onto Terrain layers to quickly create game worlds.

Terrain is solid and so Game Objects using Physics won't fall through.

Figure 8 – Gamefroot game development editor

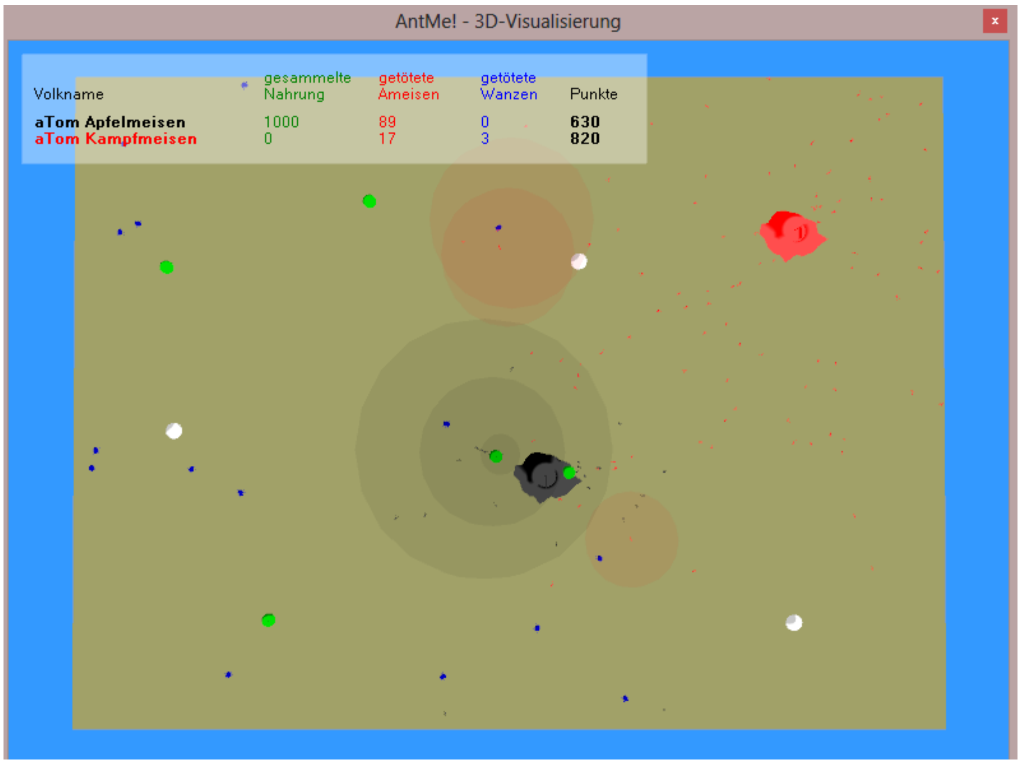
Roberta[11] is a toolkit based on Blockly that allows developers to easily program robots (Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme IAIS, n.d.). Roberta is developed and maintained by Fraunhofer-Institut für Intelligente Analyse und Informationssysteme (IAIS).

3.3.1.4 AntMe!

AntMe! ("Website - AntMe") is both a programming learning simulation and a serious game. It is written in C# and also uses C# to implement or modify the behavior of ants. In

[11] Roberta is also known as Open Roberta or Roberta-Lab.

AntMe!, one or more colonies of ants are in search of food (e.g., sugar or apples) and must defend themselves against bugs.

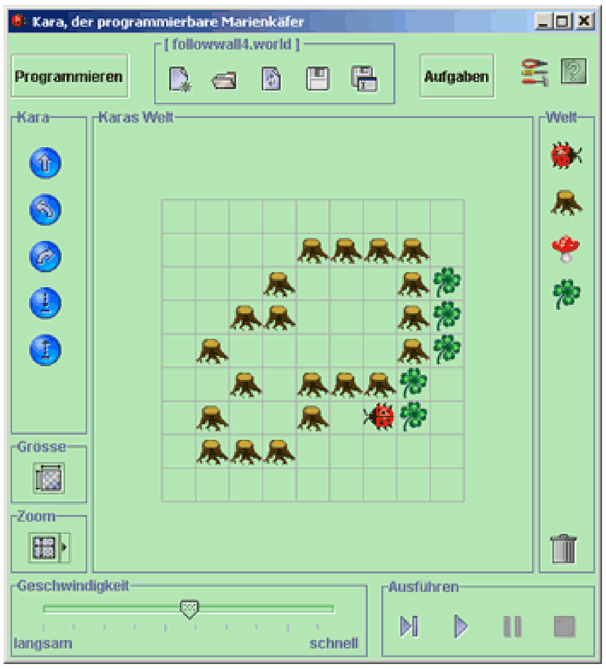


Volkname	gesammelte Nahrung	getötete Ameisen	getötete Wanzen	Punkte
aTom Apfelmeisen	1000	89	0	630
aTom Kampfmeisen	0	17	3	820

Figure 9 – AntMe! simulation with black indicating apple-searching-and-retrieving ants, red indicating warrior ants, green indicating apples, white indicating sugar, and dark blue indicating hostile bugs

3.3.1.5 Kara

Kara (R. Reichert, 2014) is a learning environment that centers around a single entity, i.e., a ladybug named Kara, which can be manipulated using the principles of a finite state machine (National Institute of Standards and Technology). As shown in Figure 10, Kara is situated in a world in which tree stumps, mushrooms, and shamrocks can be placed. In this world, tree stumps are impassable, mushrooms can be moved, but not picked up, and shamrocks can be collected and placed on another field. Programming Kara’s behavior is achieved by dragging, dropping, and configuring elements of a finite state machine’s set into the programming area.



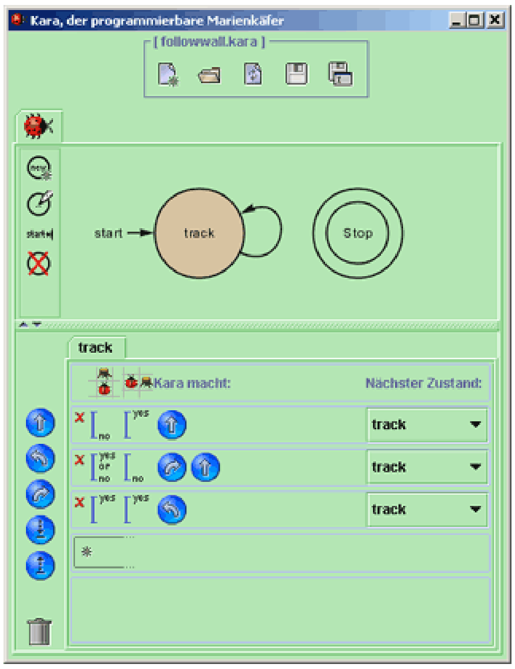


Figure 10 – Kara exists in a simple world[12] Figure 11 – Programming view of Kara[13]

There are also derivative versions of Kara, including Multi Kara, which offers an introduction to *concurrent programming* (Burns & Davies, 1993; Snow, 1992), Lego Kara, which provides a robot based on Lego Mindstorms ("Website - Lego Mindstorms,"), and Turing Kara, which focuses on Turing machines (Barker-Plummer, 1995). Kara also comes with some implementations to ease the learning of Java, JavaScript, Ruby, or Python. For these language-specific versions of Kara, the view shown in Figure 11 is replaced with an interface for entering source code directly.

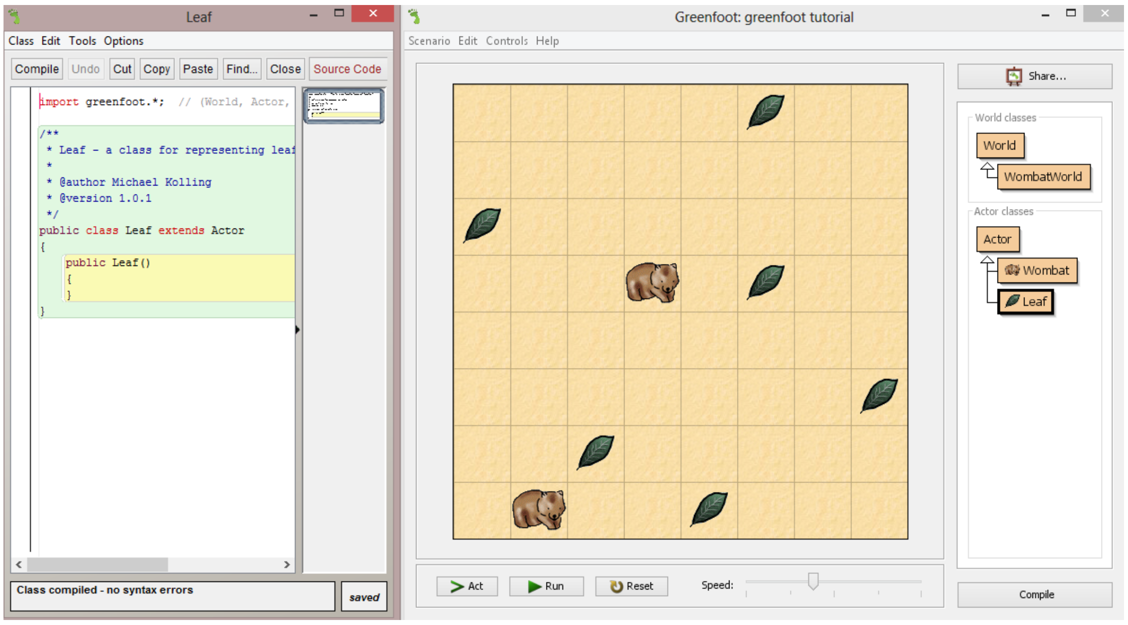
3.3.1.6 Greenfoot

As shown in Figure 12, Greenfoot (Computing Education Group - University of Kent, 2014) has some similarities to Kara. Instead of a ladybug, Greenfoot's basic configuration features wombats and leaves. In general, in Greenfoot, any image can easily be used to make one's own world or game, because the base class is not a wombat, but simply an

[12] Image obtained from (Swiss Educ Team).

[13] Image obtained from (Swiss Educ Team).

actor. Greenfoot offers the possibility to inspect its objects and actors, i.e., the state of its variables. Further, the class diagram is displayed in the main window and updated upon compilation; there is also detailed documentation and tutorials (Computing Education Group - University of Kent).



```
import greenfoot.*;  // (World, Actor,

/**
 * Leaf - a class for representing leaf
 *
 * @author Michael Kolling
 * @version 1.0.1
 */
public class Leaf extends Actor
{
    public Leaf()
    {
    }
}
```

```
Class compiled - no syntax errors
saved
```

Figure 12 – Greenfoot interface with a code view (left), the world (center), and the class diagram (right)

3.3.2 Game programming frameworks

3.3.2.1 Tools for developing browser games

In the World Wide Web, the leading frameworks for creating browser games consist of a combination of Adobe Flash and Action Script, a combination of HTML 5 and JavaScript (addressing the Canvas element), or the XNA-Game framework in connection with Microsoft Silverlight.

The Flash world

In the Flash realm, Adobe's Flash Builder (Software) and, for use as an engine, the Starling framework, both seem interesting. In general, Flash Builder is not free, although educational institutions may apply for free licensing; however, since programming tools in school are supposed to be easily and quickly installed and modified, non-proprietary software better serves our purpose. Therefore, we look to FlashDevelop, an open source integrated development environment (IDE).

Another simple game development framework written in Action Script 3 that might be appropriate for educational use is FlashPunk (Johnston), which is an open source software project designed to work with the Flex application framework (also open source). Another open source framework is Flixel (Saltsman), which is known for its powerful and flexible camera class, FlxCamera.

HTML 5 and JavaScript

In the HTML 5 world, the Canvas element can be addressed via JavaScript and used for game programming. NetBeans (Oracle Corp.) may serve as IDE, supporting code completion, which proves very useful for learning programming with students.

Isogenic Engine (Irrelon Software, n.d.) is a strong but not lightweight game engine. Three reasonably simple and therefore graspable frameworks are CraftyJS (Stowasser), LimeJS (Tiigi et. al.), and melonJS (Biot). All three are compatible with the HTML 5 specification and access the Canvas element. There are also specialties involving these frameworks, e.g., Crafty and melon come with simple collision detection. Melon supports loading maps generated using Tiled Map Editor (Lindeijer). Lime integrates the HTML 5 physics engine Box2D very well. Lime and Crafty are both optimized for mobile devices and implement the Document Object Model (W3C). Crafty further supports an isometric perspective. A summary is shown in Figure 13, which is based on a feature matrix for game engines on GitHub (Vepsäläinen).

Name	2D	3D	Tile-based	Isometric	Sound	Collisions	Physics	AI	Networking	Map Editor	Optimized for Mobile	Canvas	DOM	Web-GL	HTML5
Crafty	X			X	X	X					X	X	X		X
Isogenic Engine	X		X	X	X	X	X	X	X	X	X	X	X		
LimeJS	X				X		X				X	X	X		X
melonJS	X		X		X	X				X		X			X
PropulsionJS	X				X	X	X					X			X
Turbulenz	X	X			X	X	X		X			X		X	X

Figure 13 – Feature matrix of selected game engines[14]

Finally, Turbulenz and PropulsionJS are quite powerful engines but too complex to use compared with Crafty, Lime, and melon.

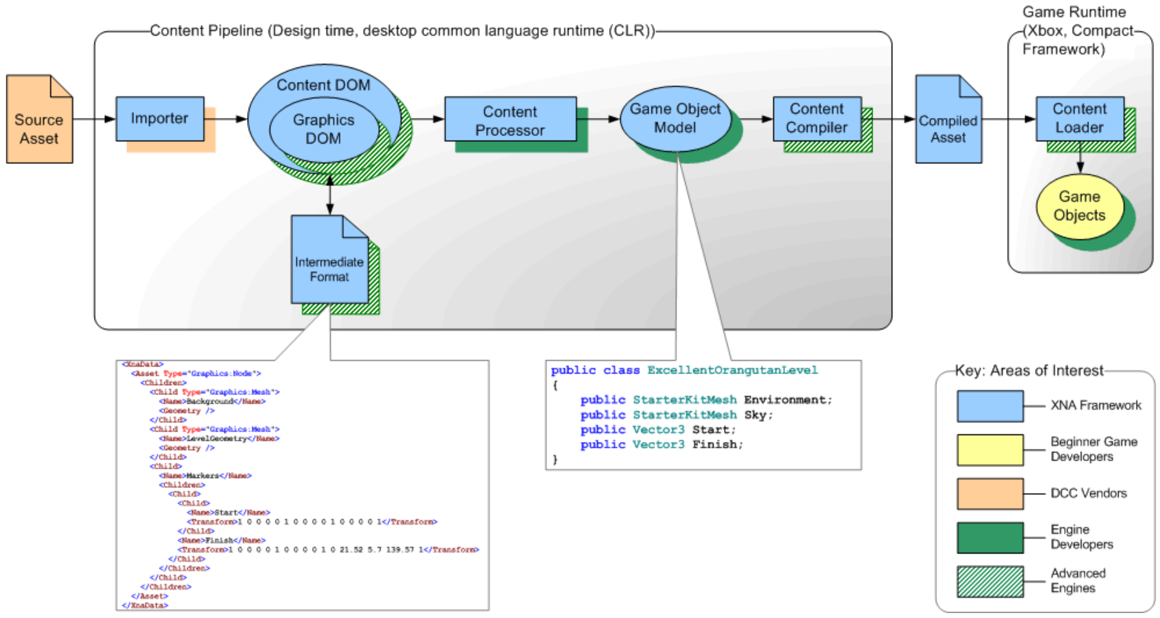
3.3.2.2 XNA Game Studio

XNA (XNA's Not Acronymed) can be used to develop games for PCs with Windows, Windows Phone, and the Xbox. XNA is based on the .NET framework and is essentially free, but publishing Xbox games requires an Xbox LIVE Indie Games account, which is currently $99 per year ("Website - Xbox live - Indie Games,").

XNA Game Studio can be easily integrated into Visual Studio Express ("Website - Microsoft Visual Studio,"), which is also free, but requires registration to obtain a free serial number. On the technical side, XNA is based on the .NET 2.0 Framework. Further, XNA's libraries have the advantage of being *managed libraries* ("Website - XNA Game Studio 4.0," 2011) with good performance. As shown in Figure 14, a content pipeline for importing, loading, managing, and using content is available ("Website - Content Pipeline Architecture,"). The content pipeline provides importers for textures (e.g., as .jpg, .png, .bmp, or .tga files) or three-dimensional models (e.g., Autodesk fbx), DirectX files (.x), or

[14] This is based on (Vepsäläinen) with some additional updates.

DirectX format-compliant effects (.fx) ("Website - Standard Content Importers and Content Processors,"), and works seamlessly with the Common Language Runtime.



```
<XnaData>
  <Asset Type="Graphics:Node">
    <Children>
      <Child Type="Graphics:Mesh">
        <Name>Background</Name>
        <Geometry />
      </Child>
      <Child Type="Graphics:Mesh">
        <Name>LevelGeometry</Name>
        <Geometry />
      </Child>
      <Child>
        <Name>Markers</Name>
        <Children>
          <Child>
            <Child>
              <Name>Start</Name>
              <Transform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</Transform>
            </Child>
            <Name>Finish</Name>
            <Transform>1 0 0 0 0 1 0 0 0 0 1 0 21.52 5.7 139.57 1</Transform>
          </Child>
        </Children>
      </Child>
    </Children>
  </Asset>
</XnaData>
```

```
public class ExcellentOrangutanLevel
{
    public StarterKitMesh Environment;
    public StarterKitMesh Sky;
    public Vector3 Start;
    public Vector3 Finish;
}
```

Figure 14 – Architecture of XNA's content pipeline[15]

Numerous sample projects and basic starter kits (Gamesm) provide the first steps, thus making XNA quite accessible.

3.3.2.3 *MonoGame*

MonoGame (MonoGame Team, 2009) is a free open source implementation of Microsoft's XNA. MonoGame supports Android, iOS, Microsoft Windows and also OpenGL (Kronos Group), Mac OS X, Linux, Windows Phone 8, PlayStation Mobile, and the OUYA (OUYA Inc.) console. The development of Windows games relies on Microsoft .NET; the development of games for Mac OS X, Linux, and Playstation mobile employs the Mono framework ("Website - Mono - an open source implementation of Microsoft's .NET Framework,"),[16] and the creation of Android, iOS, and OUYA games requires Xamarin (Xamarin Inc.).

[15] Image obtained from ("Website - Content Pipeline Architecture,").

[16] Mono is an open source implementation of Microsoft's .NET platform.

When it comes to the choice of an IDE, MonoGame can be integrated with either MonoDevelop ("Website - MonoDevelop "), Visual Studio, or Xamarin Studio (Xamarin Inc.).

3.3.3 Game engines

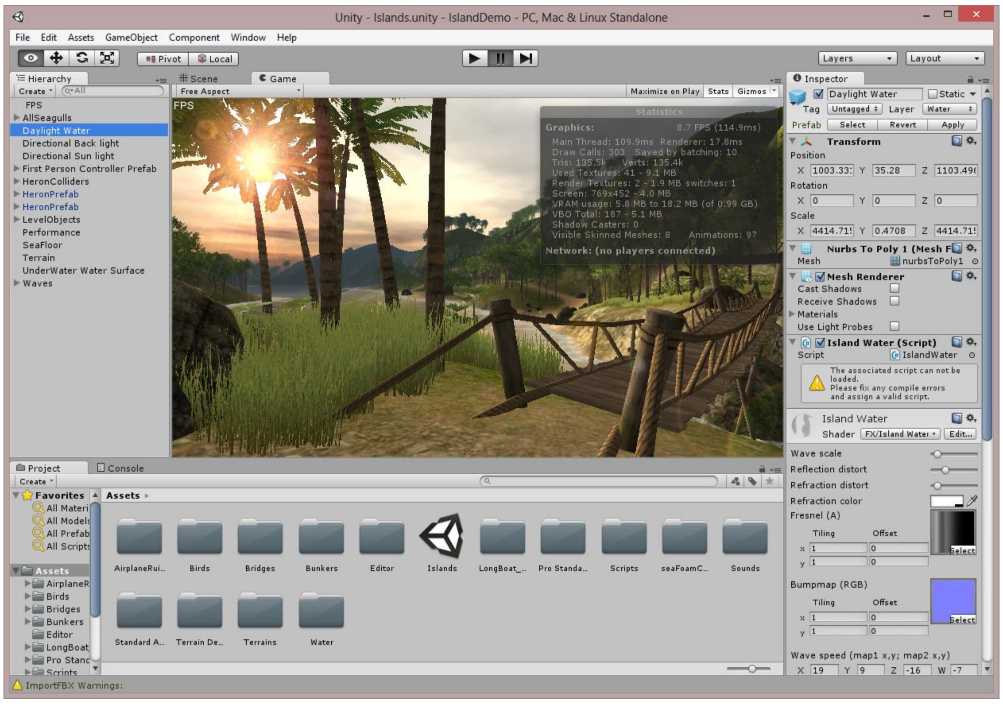
To create games professionally and therefore with a high threshold, there are various game engines. Among the most popular are OGRE, Unity3D (Unity Technologies), the Blender game engine, Unreal UDK (Epic Games), and iodoom3, an Id tech 4 engine (id Software & iodoom3). Extensive lists of game engines can be found at GPWiki and moddb (Reismanis).

3.3.3.1 Unity3D

With an example screenshot shown in Figure 15, the Unity3D game engine is a cross-platform game engine that supports iOS, Android, Windows, Blackberry 10, OS X, Linux, browser games, Flash, PlayStation 3, Xbox 360, Windows Phone 8, and Wii U (Unity Technologies). The aim of Unity3D is to simplify the development of three-dimensional games. However, bigger projects such as Fusion Fall (Cartoon Network, n.d.), one of the games often used to showcase Unity3D's capabilities, still lie out of reach of a one-person game-development team (Creighton, 2010).

Unity3D is based on Mono an open source implementation of Microsoft's .NET framework. Unity3D is free in its basic version, but currently costs[17] $1500 for the pro version (Unity Technologies). Unity3D is very popular among mobile game developers; in a survey of Gamasutra , 53.1% of developers reported using Unity3D. The scripting support of Unity3D covers native *C#*, *Boo* (a language for .NET with a syntax similar to Python) and *UnityScript*, which is a JavaScript-like language developed for use with Unity3D (Unity Technologies).

[17] As of August 24, 2013.



```
Statistics
Graphics:                          8.7 FPS (114.9ms)
  Main Thread: 109.9ms  Renderer: 17.8ms
  Draw Calls: 303   Saved by batching: 10
  Tris: 135.5k      Verts: 135.4k
  Used Textures: 41 - 9.1 MB
  Render Textures: 2 - 1.9 MB switches: 1
  Screen: 769x452 - 4.0 MB
  VRAM usage: 5.8 MB to 18.2 MB (of 0.99 GB)
  VBO Total: 187 - 5.1 MB
  Shadow Casters: 0
  Visible Skinned Meshes: 8      Animations: 97
Network: (no players connected)
```

```
Daylight Water    Static
Tag Untagged   Layer Water
Prefab  Select  Revert  Apply
Transform
Position
  X 1003.33  Y 35.28   Z 1103.49
Rotation
  X 0        Y 0       Z 0
Scale
  X 4414.71  Y 0.4708  Z 4414.71
Nurbs To Poly 1 (Mesh F
  Mesh                 nurbsToPoly1
Mesh Renderer
  Cast Shadows
  Receive Shadows
  Materials
  Use Light Probes
Island Water (Script)
  Script               IslandWater
  The associated script can not be
  loaded.
  Please fix any compile errors
  and assign a valid script.
Island Water
  Shader FX/Island Water  Edit...
Wave scale
Reflection distort
Refraction distort
Refraction color
Fresnel (A)
  Tiling   Offset
  x 1      0
  y 1      0
Bumpmap (RGB)
  Tiling   Offset
  x 1      0
  y 1      0
Wave speed (map1 x,y; map2 x,y)
  X 19  Y 9  Z -16  W -7
```

Figure 15 – Screenshot of a demo project in Unity’s IDE[18]

Unity3D games can also be played in a web browser; all that is required is the Unity Web Player (Unity Technologies).

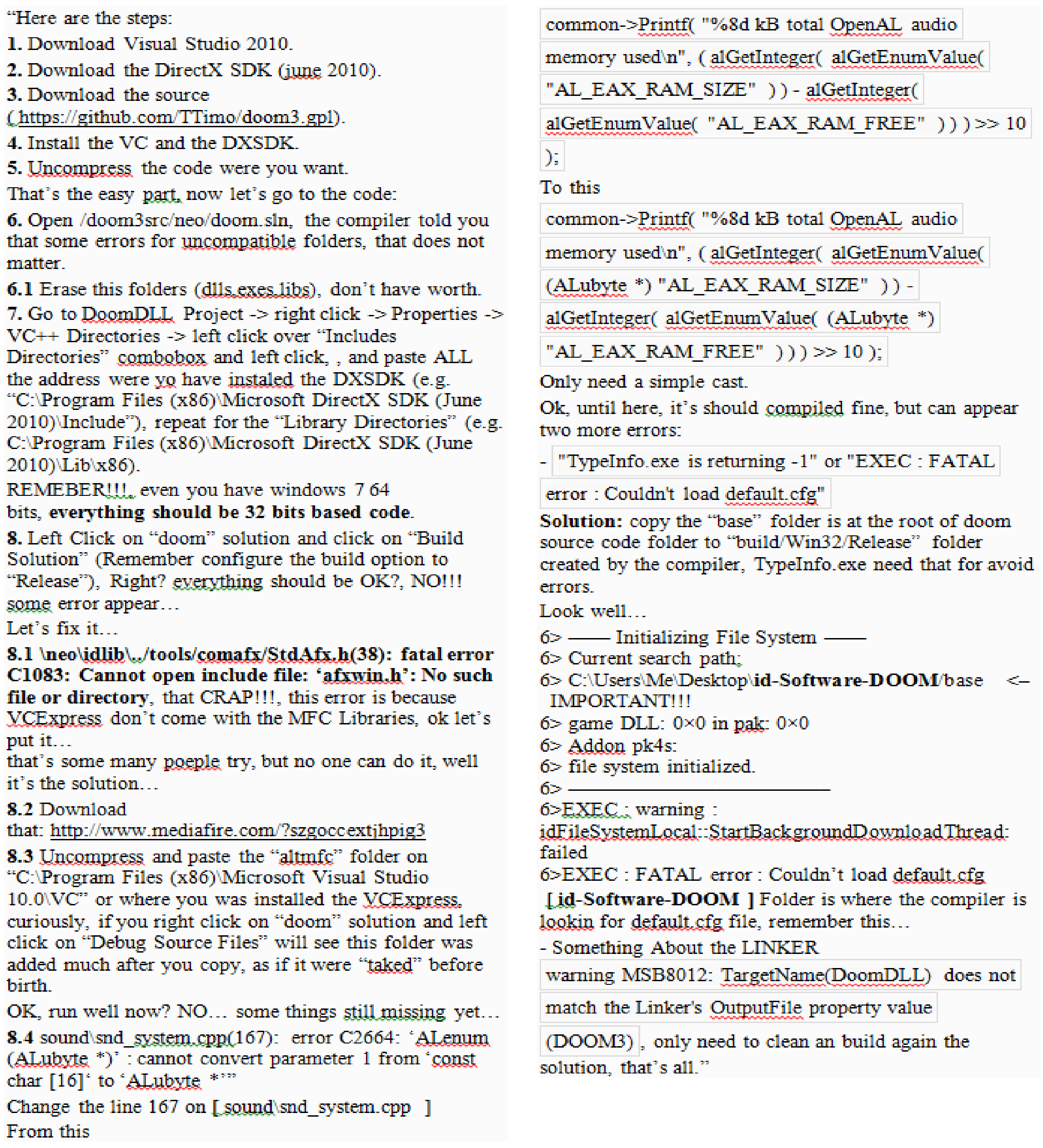
3.3.3.2 iodoom3/Id tech 4

ID Software’s well known Doom Series ("Website - Doom (series),") led to the release of Doom 3 in 2004 using the Id tech 4 engine and was followed by the publishing of the iodoom3 engine in 2011. The engine’s source code is available under GPL v3 from Github.

Scripting in the iodoom3 engine can be accomplished with the Id tech’s own scripting tool, which is syntactically quite similar to C++ (iddevnet).

Although the iodoom3 project offers a Forum and some tutorials, the threshold for beginners is higher than that of Unity3D. For example, to compile source code in Visual Studio 2010 Express, one must follow the procedure shown in Figure 16.

[18] Using Unity3D 4.2.0f2 and the Island Demo downloaded from (Unity Technologies).



"Here are the steps:

1. Download Visual Studio 2010.

2. Download the DirectX SDK (june 2010).

3. Download the source (https://github.com/TTimo/doom3.gpl).

4. Install the VC and the DXSDK.

5. Uncompress the code were you want.

That's the easy part, now let's go to the code:

6. Open /doom3src/neo/doom.sln, the compiler told you that some errors for uncompatible folders, that does not matter.

6.1 Erase this folders (dlls,exes,libs), don't have worth.

7. Go to DoomDLL Project -> right click -> Properties -> VC++ Directories -> left click over "Includes Directories" combobox and left click, , and paste ALL the address were yo have instaled the DXSDK (e.g. "C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Include"), repeat for the "Library Directories" (e.g. C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\Lib\x86).

REMEBER!!!, even you have windows 7 64 bits, **everything should be 32 bits based code**.

8. Left Click on "doom" solution and click on "Build Solution" (Remember configure the build option to "Release"), Right? everything should be OK?, NO!!! some error appear…

Let's fix it…

8.1 \neo\idlib\../tools/comafx/StdAfx.h(38): fatal error C1083: Cannot open include file: 'afxwin.h': No such file or directory, that CRAP!!!, this error is because VCExpress don't come with the MFC Libraries, ok let's put it…
that's some many poeple try, but no one can do it, well it's the solution…

8.2 Download
that: http://www.mediafire.com/?szgoccextjhpig3

8.3 Uncompress and paste the "altmfc" folder on "C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC" or where you was installed the VCExpress, curiously, if you right click on "doom" solution and left click on "Debug Source Files" will see this folder was added much after you copy, as if it were "taked" before birth.

OK, run well now? NO… some things still missing yet…

8.4 sound\snd_system.cpp(167): error C2664: 'ALenum (ALubyte *)' : cannot convert parameter 1 from 'const char [16]' to 'ALubyte *'"

Change the line 167 on [sound\snd_system.cpp]
From this

```
common->Printf( "%8d kB total OpenAL audio
memory used\n", ( alGetInteger( alGetEnumValue(
"AL_EAX_RAM_SIZE" ) ) - alGetInteger(
alGetEnumValue( "AL_EAX_RAM_FREE" ) ) ) >> 10
);
```

To this

```
common->Printf( "%8d kB total OpenAL audio
memory used\n", ( alGetInteger( alGetEnumValue(
(ALubyte *) "AL_EAX_RAM_SIZE" ) ) -
alGetInteger( alGetEnumValue( (ALubyte *)
"AL_EAX_RAM_FREE" ) ) ) >> 10 );
```

Only need a simple cast.

Ok, until here, it's should compiled fine, but can appear two more errors:

- `"TypeInfo.exe is returning -1" or "EXEC : FATAL error : Couldn't load default.cfg"`

Solution: copy the "base" folder is at the root of doom source code folder to "build/Win32/Release" folder created by the compiler, TypeInfo.exe need that for avoid errors.

Look well…

```
6> —— Initializing File System ——
6> Current search path:
6> C:\Users\Me\Desktop\id-Software-DOOM/base   <–
  IMPORTANT!!!
6> game DLL: 0×0 in pak: 0×0
6> Addon pk4s:
6> file system initialized.
6> ————————————————
6>EXEC : warning :
idFileSystemLocal::StartBackgroundDownloadThread:
failed
6>EXEC : FATAL error : Couldn't load default.cfg
```

[**id-Software-DOOM**] Folder is where the compiler is lookin for default.cfg file, remember this…

- Something About the LINKER

`warning MSB8012: TargetName(DoomDLL) does not match the Linker's OutputFile property value (DOOM3)`, only need to clean an build again the solution, that's all."

Figure 16 – How to compile iodoom projects in Visual Studio 2010 Express[19]

3.3.3.3 *Unreal Development Kit (UDK)*

More straightforward to use than iodoom3 is the Unreal Development Kit (Epic Games, n.d.-b), which originated from the third version of the game engine for the Unreal series ("Website - Unreal series,"). UDK is free of charge for education and any non-commercial

[19] Tutorial obtained from (eTiTan, 2012).

projects (Epic Games, n.d.-a). For commercial projects, a license of $99 is required . Further, if any benefit greater than $50,000 is generated, one is required to deliver 25% of the profit over $50,000 to Epic (Epic Games). For example, if you make $60,000 with your game, Epic asks for $2,500.

As shown in Figure 17, UDK comes with a powerful editor, in which it is even possible to run the level directly in the editor. Game logic and behavior can implemented using *UnrealScript* (Epic Games, n.d.-c), which is similar to Java and C++ and was designed to maintain the following goals:

> “To support the major concepts of time, state, properties, and networking which traditional programming languages don't address. [...]
>
> To provide Java-style programming simplicity, object-orientation, and compile-time error checking. [...]
>
> - a pointerless environment with automatic garbage collection;
> - a simple single-inheritance class graph;
> - strong compile-time type checking;
> - a safe client-side execution “sandbox”; [...]
>
> To enable rich, high level programming in terms of game objects and interactions rather than bits and pixels.” (Epic Games, n.d.-c)

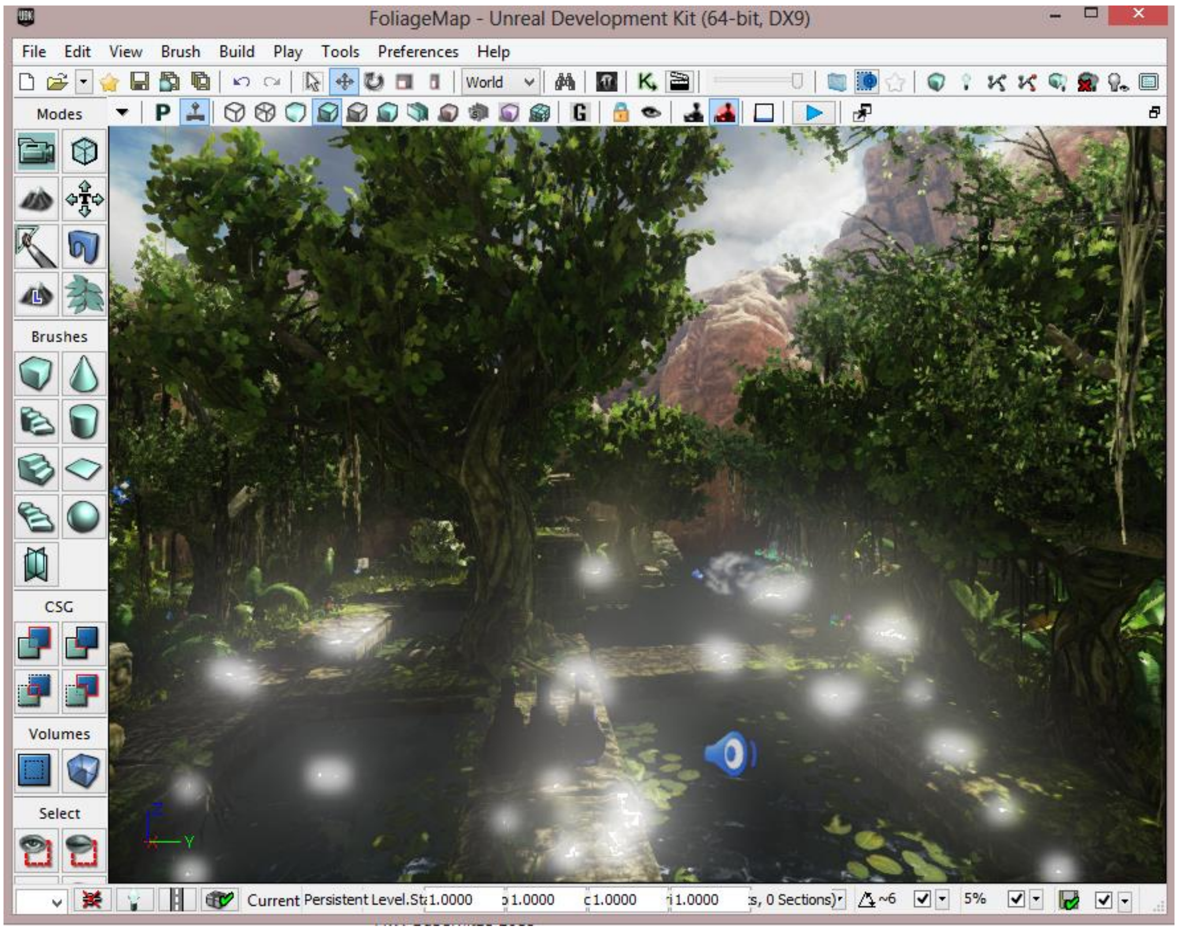

FoliageMap - Unreal Development Kit (64-bit, DX9)
File Edit View Brush Build Play Tools Preferences Help
World
Modes
Brushes
CSG
Volumes
Select
Current Persistent Level.St 1.0000 1.0000 1.0000 1.0000 s, 0 Sections)
5%

Figure 17 – UDK Editor (e.g., Foliage map)[20]

3.3.4 Licensing for educational purposes

The type of license to select eventually plays a role in the evaluation of the best tool for game programming for educational purposes. Among eligible licenses are the GNU General Public License (GPL), the GNU Lesser GPL (Free Software Foundation), the BSD license (University of California Berkeley), the MIT license (Massachusetts Institute of Technology), the Ms-PL (Open Source Initiative; "Website - Microsoft Public License (Ms-PL),"), and the Creative Commons (CC) license (Commons).

[20] Screenshot for UDK downloaded from (Epic Games, n.d.-b).

3.4 Related research

There are various approaches to using game programming in teaching, including game-themed programming assignments (Sung with C# and XNA, as shown in Figure 18), the development of computer games with undergraduate students using 3D-Studio Max and C++ (Taylor & Baskett, 2009), and *"Teaching Object-Oriented Programming Laboratory Computer Game Programming"* using C++ and the Microsoft Foundation Class Library MFC (Woei-Kae & Yu Chin, 2007).

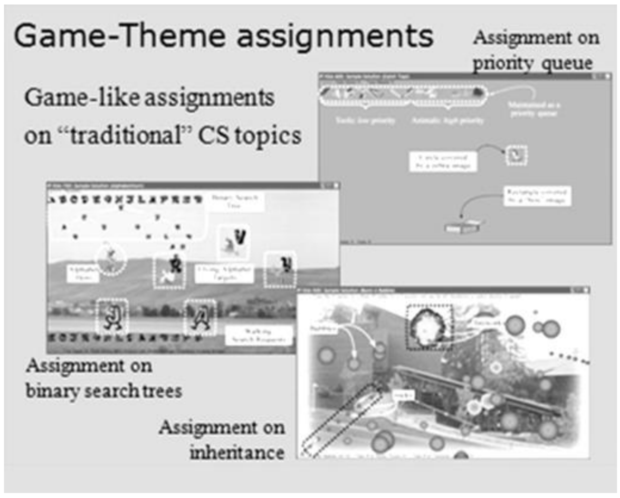

Game-Theme assignments
Assignment on priority queue
Game-like assignments on "traditional" CS topics
Assignment on binary search trees
Assignment on inheritance

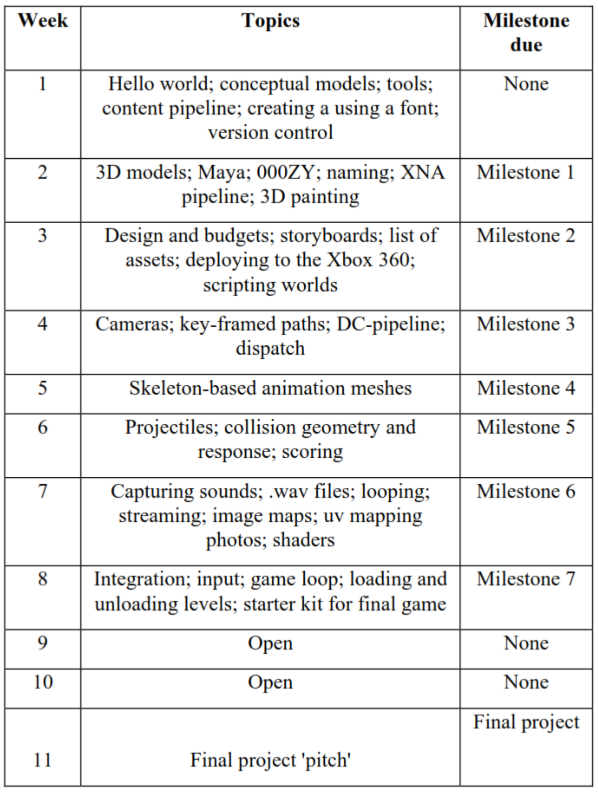
Figure 18 – Game-themed programming assignment (Sung)

Developing in Java also offers additional possibilities, e.g., in connection with the Slick 2D framework, which is based on the Lightweight Java Game Library (LWJGL). This combination was used by Volk and described in *"How to Embed a Game Engineering Course into a Computer Science Curriculum"* (Volk, 2008).

A different approach is to provide developed solutions, including the full code for a game or a segment of a game, then leave out a clear element for implementation by students (Youngblood, n.d.).

Another possibility is to create entire games, which can be done "step by step," either by implementing one game all together or by developing the same game on an individual basis with each of the students. While developing a predefined game in a master-apprentices setting, incorporates pedagogical value – see Situated Learning (Lave & Wenger, 2005) and reduces the risk of failure, it might not motivate learners as much as making their own games. Therefore, it seems promising to let students develop their own individual games. Wang et. al (2011) successfully implemented a concept in a "Software Architecture" course using Microsoft's XNA, extending it with a library called XQUEST (Bian, Wang, Strom, & Kvamme, 2009).

Because the activities described in this thesis start with C# and XNA, it seems relevant to describe additional related research here.Shen provides a brief introduction to XNA and describes its use in a course called *"Introduction to Game Development,"* which was offered at Old Dominion University (Norfolk, Virginia) in "*Teaching Game Development Using Microsoft XNA Game Studio"* (Shen, 2009). Linhoff and Settle (2005) conducted a course in game programming called GAM 380 covering the topics shown in Figure 19.



Week	Topics	Milestone due
1	Hello world; conceptual models; tools; content pipeline; creating a using a font; version control	None
2	3D models; Maya; 000ZY; naming; XNA pipeline; 3D painting	Milestone 1
3	Design and budgets; storyboards; list of assets; deploying to the Xbox 360; scripting worlds	Milestone 2
4	Cameras; key-framed paths; DC-pipeline; dispatch	Milestone 3
5	Skeleton-based animation meshes	Milestone 4
6	Projectiles; collision geometry and response; scoring	Milestone 5
7	Capturing sounds; .wav files; looping; streaming; image maps; uv mapping photos; shaders	Milestone 6
8	Integration; input; game loop; loading and unloading levels; starter kit for final game	Milestone 7
9	Open	None
10	Open	None
11	Final project 'pitch'	Final project

Figure 19 – Topics covered in the GAM 380 course[21]

[21] Screenshot from (Linhoff & Settle, 2008p. 252).

Wendel et. al. (2010) employed Unity3D in a course at the University of Darmstadt and observed that game development had the potential to be very captivating, although students had to handle high workloads.

4 Design Based Research meets Game Programming in Schools

4.1 Methodology and research design

4.1.1 Practical research design

To approach a well-balanced didactical and technical framework, the plan was to conduct research over a period of three years, not only to reach different groups, but also to reach different cohorts. Several concepts developed in the first year were reflected upon and improved over subsequent years. To accomplish this, a combination of questionnaires, student interviews, and regular reflections were applied.

Using different game development environments, the courses were implemented in Moodle (Dougiamas) to enhance traceability and provide eLearning support for students.

4.1.2 Research model: design-based research (DBR)

The setting for our research is defined as follows: *game development in the classroom with students over a period of three years, testing different game development environments.* In the process of this research, theory and practice influence one another. General theory and our derived theory, in conjunction with the corresponding hypothesis, serves as a foundation to providing a starting point for developing the initial research design. Practical intervention is based on this design; after the intervention, results are evaluated, reflected upon, and taken into account as the new design is established for the next year.

The classroom setting makes it very difficult to establish a lab setting in which different variables can be isolated, at least not without disturbing lessons in a way that renders the intended activities useless. Therefore, a research model that allows interaction and intervention without the need for an entirely controlled environment is crucial. To capture as many aspects as possible, a mix of qualitative and quantitative methods were employed.

Meeting those demands might seem difficult at first, but design-based research (DBR) (Anderson & Shattuck, 2012; Barab & Squire, 2004; Brown, 1992; Sandoval & Bell, 2004;

Williams, South, Yanchar, Wilson, & Allen, 2011) is a very convenient approach for meeting these requirements. Key qualities of DBR include *"being situated in a real educational context"* (Anderson & Shattuck, 2012, p.16) and *"focusing on the design and testing of a significant intervention"* (Anderson & Shattuck, 2012, p.16). These qualities are similar to action research (Altrichter & Posch, 2007; Moser, 1977), which actually incorporates some aspects of DBR. Our practical interventions met the idea of being *pragmatic*, a trait Reeves et. al. (2013) identifies as one of the cornerstone points of DBR. Different iterations are mentioned in Anderson and Shattuck (2012). The fact that our research model used a methodological mix is also argued and supported by Maxcy (2003). In short, *"design-based research goes beyond merely designing and testing particular interventions,"* meaning we are "*constructing cumulative design knowledge"* and *"developing contextualized theories of learning and teaching"* (Design-Based Research Collective, 2003, p. 6).

4.2 The concept and goals of Game Programming in Schools

The goal of *Game Programming in Schools* (*GamePinS*) is to provide a technical framework and didactical setting for learning. The GamePinS logo is shown in Figure 20. Here, the technical framework is both a skeleton and a boilerplate, making game programming with XNA easier. Further, the didactical setting is implemented as a Moodle course. Both of these aspects of GamePinS are intended to promote important concepts of computer science effectively and in a motivating way. Further goals are to (1) develop an approach that balances an "only user-side" approach with intense and demanding programming requirements, (2) balance complexity and a deeper understanding with simplicity (perhaps missing the most important ideas underlying computer science), and (3) adjust the workload such that it falls between under-challenging and overstraining.

Figure 20 – The GamePinS Logo

GamePinS is intended to address typical computer science concepts and foster these concepts in Austrian secondary school IT education programs with pupils ages 14 to 18.

4.2.1 Important concepts in computer science

The first step was to identify the fundamental concepts of computer science to include, i.e., my research started with the key question, "what are the important concepts in computer science?"

In (Dagienė, 2011), six significant concepts for computer science education are identified as follows:

> "- Information: the conception of information, its representation (symbolic, numerical, graphical), encoding, encrypting;
>
> Algorithms: action formalization, action description according to certain rules;
>
> Computer systems and their application: interaction of computer components, development, common principles of program functionality, search engines;
>
> Structures and patterns: the components of discrete mathematics, elements of combinatorics and actions with them;
>
> Social effect of technologies: cognitive, legal, ethical, cultural, integral aspects of information and communication technologies;
>
> Informatics and information technology puzzles: logical games, mind maps, used to develop technology-based skills." (Dagienė, 2011, p. 18)

From a student's point of view, learning and adapting to these areas of knowledge will not only foster a deeper understanding of computer science but also contribute to problem solving strategies in school and later life, i.e.,

> "It has been agreed that on some of the main concepts to be taught in general education, e.g., algorithms and programming (as a separate or integral part of algorithm construction) is one of the most important concepts of informatics." (Dagienė, 2011, p. 18)

The main goal behind my research is therefore to address these important concepts of computer science education via the GamePinS project.

4.2.2 The GamePinS framework

The GamePinS framework is conceptualized to support teachers in developing games in a straightforward manner and promote essential concepts of computer science. The technology decisions were based on actual installed software on the school network, i.e., Visual Studio Express 2010, and the possibility to easily install XNA. As such, students actually write code instead of operating with drag-and-drop environments (i.e., Scratch or Blockly). Further, the GamePinS framework should be easily evaluated and tested in accordance with intended learning outcomes.

4.2.2.1 The teacher's role

In general, everyone should be able to use GamePinS, but helpful skills and attitudes for teachers include the following:

- Basic knowledge of how programming languages work.
- A deeper understanding of at least one programming language, preferably C#, JavaScript, or Scratch.
- An awareness of basic concepts of computer science, including object-oriented programming, variables, properties, methods (a.k.a. functions or procedures), arrays, lists, algorithms, searching and sorting algorithms, file I/O, etc.
- The ambition to get involved with game programming and the GamePinS framework.
- The motivation to engage in and lead project-based classes.

4.2.2.2 Computer science and the learning method

The fundamental idea underlying GamePinS is to promote concepts of computer science in a motivating way for students. An important goal here is for the developed learning method to have a direct impact on the understanding of computer science concepts. Students should learn to think logically, methodically, in a structured fashion, on abstract levels, and also at a very detailed level, thus benefitting from these learning outcomes beyond just the computer science classroom.

One of the intents of GamePinS is to impact classroom courses such that they reach their goals, as specified in the official curriculum of the Austrian Federal Ministry for Education, Arts, and Culture (BMBF, p. 2):

> Insights into key concepts and methods of computer science, typical ways of thinking and working in computer science, their historical development, technical and theoretical foundations learn the basic principles of machines, algorithms and programs.[22]

Through GamePinS, a more realistic picture of game programming, computer science, and the work of a computer scientist should be delivered to students. An anticipated outcome here is that students with a more realistic view of *informatics* (as compared to students who solely learn to use Office applications) will enroll in computer science courses and programs at the university level, and if game programming for students indeed turns out to be "fun," that the number of students who enroll in computer science will increase.

4.2.3 Didactical approach

There are various possible approaches to game programming. Game programming in the classroom might be organized as either game-themed assignments (Sung) or as project-based learning (Boss & Krauss, 2007). For our purpose, the project-based idea seems the best fit, because developing a game shows typical traits of a project (Connecticut State University; Lock, 1995; T. Reichert, 2009) and project management is also often considered part of computer science[23]. In our case, game development projects are characterized by the following phases:

- Preparation: Common for all learners, students learn the fundamentals of programming, game programming, computer graphics, and sound editing, regardless of their later roles in game development projects.

[22] Translated from German O.C.

[23] See dedicated courses in the curricula for computer science at the University of Vienna (University of Vienna - Senate) and the Technical University of Vienna (Technical University of Vienna).

- Project start: The start of a project includes an agreement between the teacher and students to work in groups on the project, which requires project groups to be formed, the election of a team leader, and the assignment of different roles (e.g., project manager, engine programmers, sound designers and programmers of sound libraries who reduce the workload of engine programmers, graphics designers, and story authors who are also often assigned as level designers).
- Planning: The planning phase includes initial project definition and early design aspects, answering the question as to what kind of game should be developed (e.g., role playing game, jump and run game, and adventure game).
- Monitoring and control: Crucial to project success is monitoring to ensure that the project and team members are proceeding as intended or, if not, determining how to get help them to make progress.
- Project finish: In this final phase, students present their games, participate in free reflection about the work, and answer a questionnaire regarding the project and the project-based game development process.

In general, the role of the teacher is as a consultant, bringing in expertise on project management and game development. From my own experience, it is not enough to be a coach; one must also be an expert in all areas that are important to game programming. The teacher in this case must be able to fully understand the process of game development and be able to develop such a game by him or herself. Simply coaching and hoping that pupils succeed might work out, but this approach can result in complete failure. After a first survey about game development, many similar statements pointed out the importance of the help of at least one person who has knowledge in the area of game development, for example:

"In the beginning I was highly motivated, [but] when we started, I experienced how difficult it is." (Students comment on game development)

Overall, it is crucial that a framework that includes a curriculum plan is fully developed such that an average student can accomplish the required game programming tasks without being overstressed. Further, the teacher must be able to assume her or his role without being overburdened.

4.2.4 Expected outcome

The expected outcome was that GamePinS would make computer science in school more attractive to students, i.e., increase motivation (Boekaerts, 2002; Reeve, 2009) and engagement (Corno & Mandinach, 1983) in the subject, foster teamwork competencies (Lingard & Barkataki, 2011; Motschnig-Pitrik & Figl, 2007), and promote important concepts of computer science. To verify the expected outcome, the following four assumptions, based on the research questions, were formulated:

(A1) Game development in computer science increases the engagement of students.

(A2) Project-based game programming in computer science fosters teamwork competencies.

(A3) Game programming promotes basic concepts of computer science more effectively than other typical software development scenarios.

(A4) The complexity level of the framework used influences the depth of understanding of the concepts to be learned.

4.2.5 Research design

Research regarding game programming follows a certain scheme, which is applied to all three years of research. Before the school year started, the learning goals, learning materials, course content, and design for the entire year were planned and implemented in Moodle. At the beginning of the first semester, students were asked to fill out a questionnaire regarding their computer usage, attitudes toward gaming and developing games, and expectations with respect to game programming in school. Then, students learned C# programming and how to use Visual Studio Express, how to create sounds in Audacity, and how to create graphics in the Gnu Image Manipulation Program (GIMP). These game programming activities ranged over two semesters, i.e., an entire school year. At the end of the second semester, students were asked to fill out another questionnaire regarding their computer usage, attitudes toward gaming and developing games, and how their view and motivation has changed. The questionnaire also includes a self-assessment regarding the development of their own IT skills, covering *programming*, *understanding* *of algorithms*, *project management*, and *team skills*, as well as the development of motivation.

A chat interview with selected students conducted and evaluated in a qualitative manner was also employed to obtain further data for interpreting the information collected in the questionnaires.

To obtain relevant data regarding computer usage and attitudes toward gaming and developing games, a general survey targeting all pupils not involved in GamePinS activities or any other activities surrounding GamePinS (i.e., a control group) was conducted.

4.2.5.1 General questionnaire regarding gaming and computer habits

A general questionnaire regarding gaming and computer habits was incorporated into all years of game development to gather general data about the target group and determine student attitudes toward game programming, including whether game development is of any interest to young persons at all. The hypothesis here is that game development is more attractive to students than the regular curriculum.

The general questionnaire included a simple set of items or questions. These items were stored in a MySQL Database (Oracle Corp. et. al.). With the help of PHP, the survey was generated and delivered to students as HTML pages within their respective browsers.[24]

4.2.5.2 The school, students, participation, and schedule

All research took place in an AHS[25], i.e., the GRG 16 Maroltingergasse 69-71 in the 16th district of Vienna, Austria. In general, students enter this type of school at the age of 10 and finish it with the “Reifeprüfung” at the age of 18. In total, there are over 1100 students attending the GRG 16 and over 120 teachers working in this school.

As summarized in Figure 21, the research schedule allowed a four-year research span with an initial phase (i.e., pre-research regarding game development), three iterations of

[24] For the source of the questionnaire-tool/the PHP-Code see (Comber, 2008)

[25] “Allgemeinbildende höhere Schule”

each research cycle, and a dedicated development phase after year one of game development. During the student interactive research span from September 2011 to July 2015, 302 students were included in the general research, of which 90 students participated actively in the game development activities.

	Research and practical activities	Starting
	Approval from the Board of Education to conduct research in school	Mar 2011
Year 1	Initial questionnaire about game programming (I1)	Sep 2011
	Game programming year 1 – two groups, both with C# and XNA	Sep 2011
	Final questionnaire and statements about game programming	Jun 2012
	Evaluation of year 1, reflection, improvement of concept	Jul 2012
	Development of the GamePinS framework and online courses	Jan 2013
Year 2	Initial questionnaire about game programming (I2)	Sep 2013
	Game programming year 2 – GamePinS (evolved from year one)	Sep 2013
	Final questionnaire and statements about game programming	Jun 2014
	Evaluation of year 2, reflection, improvement of framework	Jul 2014
Year 3	Initial questionnaire about game programming (I3)	Sep 2014
	Game programming year 3 – working with the improved framework	Sep 2014
	IT skills test with two different teachers and four groups (F3)	Jun 2015
	Evaluation of year 3, reflection, improvement of framework	Jun 2015
	Final processing, analysis, and evaluation of all gathered data	Jul 2015
	Research finished, submission of thesis	Dec 2015

Figure 21 – Schedule of the GamePinS project

4.3 The GamePinS Framework

Game development covers numerous areas of computer science. The development process involves planning, specifying, developing, testing, and distribution. The development process also covers graphic design, level design, sound design, story and game concept design, and engine programming. For all tasks in the development process, there many tools, and the choice of tools is not always easy and is often limited by the lab environment; more specifically, preinstalled software works in favor of specific tools. Note that the technical preconditions for three years of game development in a computer science classroom is covered in the next chapter.

4.3.1 Practical environment conditions for GamePinS

The preconditions in the school in which game development activities took place were Windows-based PCs with the following software, relevant for game development, already installed:

- Windows 7
- Visual Studio Express 2010
- .NET-Framework 4.0
- XNA Game Studio 4.0
- GIMP 2.6 (Kimball et. al., 2014)
- Audacity 2.0.2

The game development process employed many of these preinstalled tools. Programming was done using Visual Studio Express C# 2010 and XNA 4.0 as an interface. Further, audio engineering occurred via Audacity 2.0.2, while game graphics and tiles for maps were designed using GIMP 2.6. Only in the field of *Level Editing* was it necessary to download an additional free tool and run it from the local drive; this tool was the Tiled Map Editor (Lindeijer & Cook, 2013), shown in Figure 22.

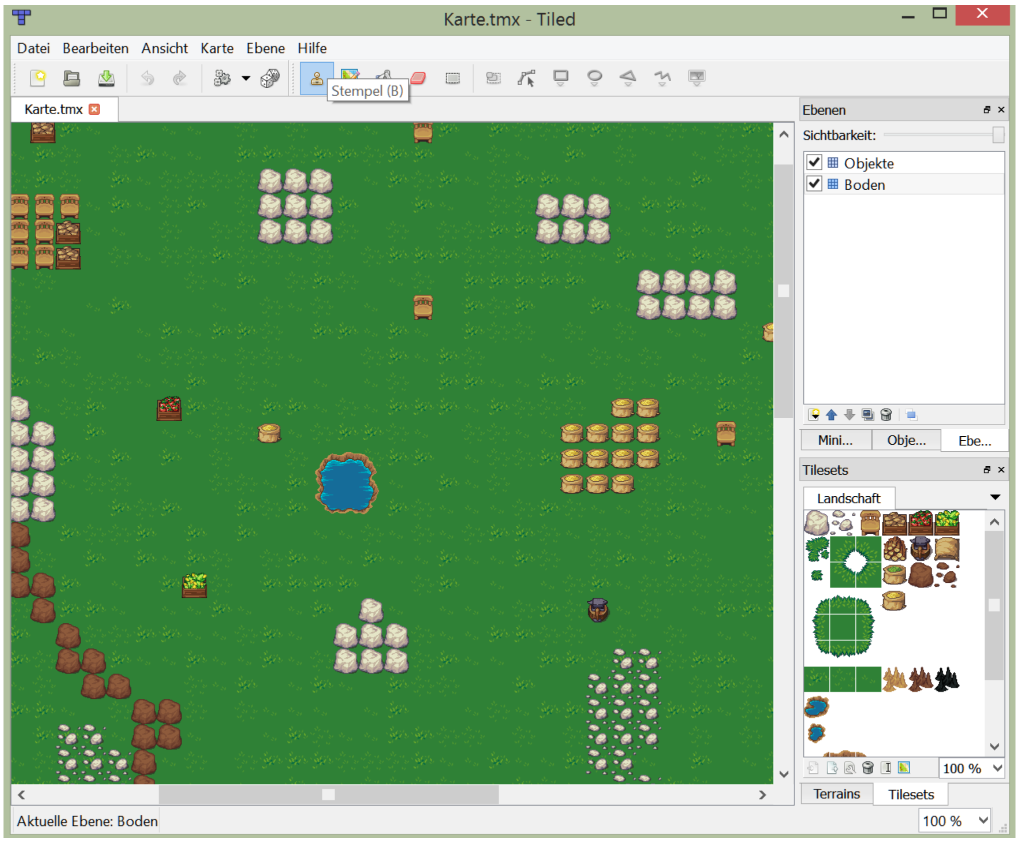


Figure 22 – Tiled Map Editor screenshot with sample tiles from Liberated Pixel Cup (2013)

Other functionality, including the map import and game physics tools, was implemented using libraries that were embedded in the GamePinS samples (see Section 4.3.3 for more details regarding The structure of GamePinS).

4.3.2 The didactical approach

Game development activities were always embedded within an entire school year of computer science education, which did not exclusively include game development. The majority of students already enjoyed one hour of computer science per week in their previous year and two hours in their first year. The curriculum primarily was covered in the classroom, but this was also backed by a Moodle course (Comber, 2013). The template of the Moodle course, MC_Comber, is available online for guests with the correct password[26]. This curriculum included a brief review of Office applications, a glance at

[26] For the password see the reference section of this document

Scratch, including its limitations for game programming, graphics design and creation using GIMP, audio editing with Audacity, creating game levels with Tiled, and learning the basics of C#.

4.3.2.1 Curriculum topics, prior to starting with Game Development

In Microsoft Word, students were required to design a flyer for an event of their own choice. Further, young learners created a professional template for their "prescientific high school graduation thesis[27]." This professional template task included correct formatting, i.e., the importance of using proper styles and formatting, the routine of specifying headers and footers, including automated document information, especially page numbering, the theory and practice of sections and section breaks, the correct definition of captions and cross-references, and the automatic generation of a table of contents and a list of figures.

In PowerPoint, students repeated their presentation techniques and had to plan a trip around the world for their teacher, who fictionally won the lottery, donated a major part of the winnings to charity, and had approximately $150,000 left to travel around the world.

In Microsoft Excel, students were encouraged to understand basic spreadsheet principles and implemented a time table of their weekly activities, including time in school, with friends, homework, sports, watching TV, on a PC, console, or smartphone, sleeping, and travel times to and from such activities. Personal statistics were represented via diagrams to visually identify time wasted and see how much percentage of time each activity consumed per week.

Other topics, including file management, HTML, and the history of computer science were covered in the previous classes, but omitted to provide more time for game development tools and GamePinS itself.

With GIMP, students first learned how to use and configure GIMP to establish a smooth workflow. Next, students integrated the concept of layers and mastered multi-layer techniques with such assignments as creating a movie poster for any movie of their choice

[27] Translated from (Lasselsberger, Gschwendtner, & Bundesministerium für Bildung und Frauen, 2015).

and manipulating and integrating their own appearance, e.g., their face, into the movie poster. After accomplishing these tasks successfully, students designed tiles to build their levels, as shown in Figure 23. Lastly, students designed their game characters using GIMP.

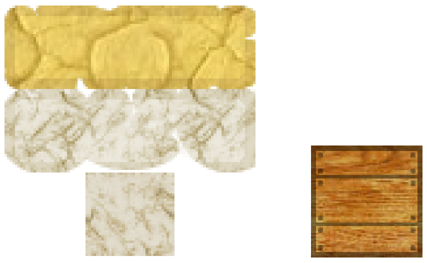


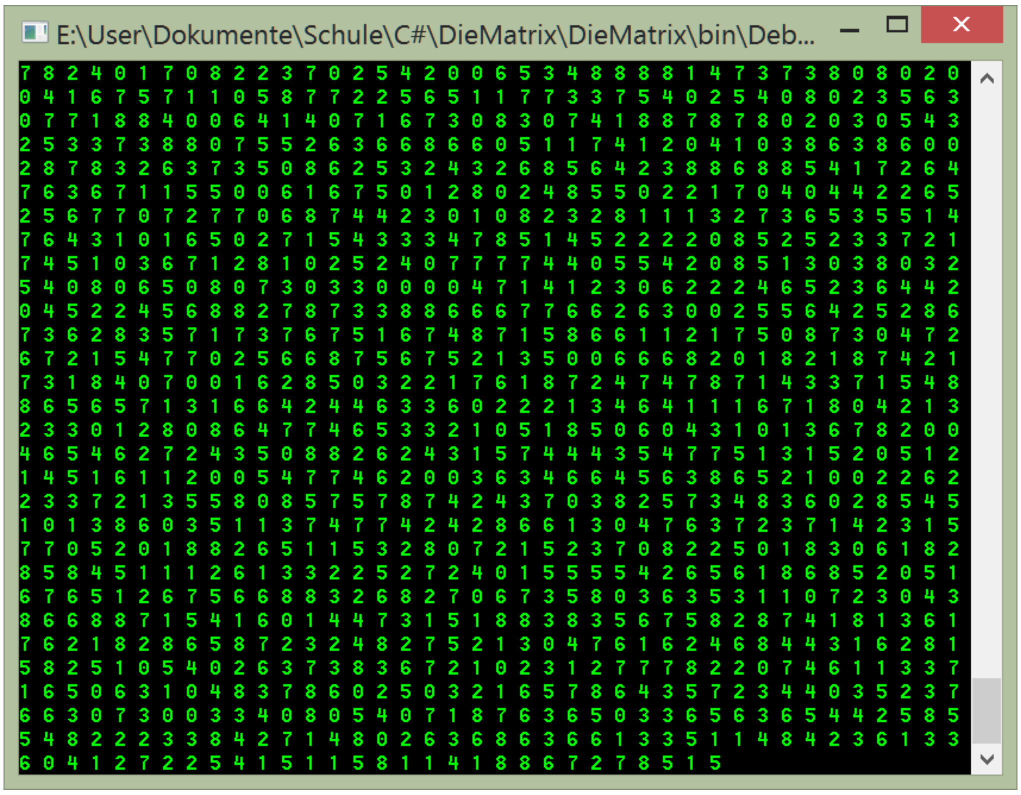
Figure 23 – Sample tiles for a platform-based game

With Tiled Editor, students used tiles created in GIMP to assemble levels for their games. No matter what their choice of game was, students first had to create a platformer/side-scroller and one top-view game level.

With Audacity, students dove directly into recording and engineering their sounds for their games. They learned about the basic options of Audacity, how to copy, cut, and paste, and how to export audio projects in the proper formats.

Students gained fundamental knowledge of programming using C# by writing programs for the console via Visual Studio 2010 Express. Tasks started with basic input/output programs[28], such as greeting the user and asking for his or her name, then storing it in a variable and making nice comments about the user's name. Other programs included arithmetical operations, conditionals, and introducing a countdown program with the help of a "for" loop. After introducing more techniques, including random number generators, students used "while" loops to implement a program called "The Matrix" that displayed random numbers floating down the screen, as shown in Figure 24. Further, students took on the challenge of creating a dice game that included handling of errors and invalid inputs.

[28] First of all with a traditional "Hello World" program.



```
7 8 2 4 0 1 7 0 8 2 2 3 7 0 2 5 4 2 0 0 6 5 3 4 8 8 8 8 1 4 7 3 7 3 8 0 8 0 2 0
0 4 1 6 7 5 7 1 1 0 5 8 7 7 2 2 5 6 5 1 1 7 7 3 3 7 5 4 0 2 5 4 0 8 0 2 3 5 6 3
0 7 7 1 8 8 4 0 0 6 4 1 4 0 7 1 6 7 3 0 8 3 0 7 4 1 8 8 7 8 7 8 0 2 0 3 0 5 4 3
2 5 3 3 7 3 8 8 0 7 5 5 2 6 3 6 6 8 6 6 0 5 1 1 7 4 1 2 0 4 1 0 3 8 6 3 8 6 0 0
2 8 7 8 3 2 6 3 7 3 5 0 8 6 2 5 3 2 4 3 2 6 8 5 6 4 2 3 8 8 6 8 8 5 4 1 7 2 6 4
7 6 3 6 7 1 1 5 5 0 0 6 1 6 7 5 0 1 2 8 0 2 4 8 5 5 0 2 2 1 7 0 4 0 4 4 2 2 6 5
2 5 6 7 7 0 7 2 7 7 0 6 8 7 4 4 2 3 0 1 0 8 2 3 2 8 1 1 1 3 2 7 3 6 5 3 5 5 1 4
7 6 4 3 1 0 1 6 5 0 2 7 1 5 4 3 3 3 4 7 8 5 1 4 5 2 2 2 2 0 8 5 2 5 2 3 3 7 2 1
7 4 5 1 0 3 6 7 1 2 8 1 0 2 5 2 4 0 7 7 7 7 4 4 0 5 5 4 2 0 8 5 1 3 0 3 8 0 3 2
5 4 0 8 0 6 5 0 8 0 7 3 0 3 3 0 0 0 0 4 7 1 4 1 2 3 0 6 2 2 2 4 6 5 2 3 6 4 4 2
0 4 5 2 2 4 5 6 8 8 2 7 8 7 3 3 8 8 6 6 6 7 7 6 6 2 6 3 0 0 2 5 5 6 4 2 5 2 8 6
7 3 6 2 8 3 5 7 1 7 3 7 6 7 5 1 6 7 4 8 7 1 5 8 6 6 1 1 2 1 7 5 0 8 7 3 0 4 7 2
6 7 2 1 5 4 7 7 0 2 5 6 6 8 7 5 6 7 5 2 1 3 5 0 0 6 6 6 8 2 0 1 8 2 1 8 7 4 2 1
7 3 1 8 4 0 7 0 0 1 6 2 8 5 0 3 2 2 1 7 6 1 8 7 2 4 7 4 7 8 7 1 4 3 3 7 1 5 4 8
8 6 5 6 5 7 1 3 1 6 6 4 2 4 4 6 3 3 6 0 2 2 2 1 3 4 6 4 1 1 1 6 7 1 8 0 4 2 1 3
2 3 3 0 1 2 8 0 8 6 4 7 7 4 6 5 3 3 2 1 0 5 1 8 5 0 6 0 4 3 1 0 1 3 6 7 8 2 0 0
4 6 5 4 6 2 7 2 4 3 5 0 8 8 2 6 2 4 3 1 5 7 4 4 4 3 5 4 7 7 5 1 3 1 5 2 0 5 1 2
1 4 5 1 6 1 1 2 0 0 5 4 7 7 4 6 2 0 0 3 6 3 4 6 6 4 5 6 3 8 6 5 2 1 0 0 2 2 6 2
2 3 3 7 2 1 3 5 5 8 0 8 5 7 5 7 8 7 4 2 4 3 7 0 3 8 2 5 7 3 4 8 3 6 0 2 8 5 4 5
1 0 1 3 8 6 0 3 5 1 1 3 7 4 7 7 4 2 4 2 8 6 6 1 3 0 4 7 6 3 7 2 3 7 1 4 2 3 1 5
7 7 0 5 2 0 1 8 8 2 6 5 1 1 5 3 2 8 0 7 2 1 5 2 3 7 0 8 2 2 5 0 1 8 3 0 6 1 8 2
8 5 8 4 5 1 1 1 2 6 1 3 3 2 2 5 2 7 2 4 0 1 5 5 5 5 4 2 6 5 6 1 8 6 8 5 2 0 5 1
6 7 6 5 1 2 6 7 5 6 6 8 8 3 2 6 8 2 7 0 6 7 3 5 8 0 3 6 3 5 3 1 1 0 7 2 3 0 4 3
8 6 6 8 8 7 1 5 4 1 6 0 1 4 4 7 3 1 5 1 8 8 3 8 3 5 6 7 5 8 2 8 7 4 1 8 1 3 6 1
7 6 2 1 8 2 8 6 5 8 7 2 3 2 4 8 2 7 5 2 1 3 0 4 7 6 1 6 2 4 6 8 4 4 3 1 6 2 8 1
5 8 2 5 1 0 5 4 0 2 6 3 7 3 8 3 6 7 2 1 0 2 3 1 2 7 7 7 8 2 2 0 7 4 6 1 1 3 3 7
1 6 5 0 6 3 1 0 4 8 3 7 8 6 0 2 5 0 3 2 1 6 5 7 8 6 4 3 5 7 2 3 4 4 0 3 5 2 3 7
6 6 3 0 7 3 0 0 3 3 4 0 8 0 5 4 0 7 1 8 7 6 3 6 5 0 3 3 6 5 6 3 6 5 4 4 2 5 8 5
5 4 8 2 2 2 3 3 8 4 2 7 1 4 8 0 2 6 3 6 8 6 3 6 6 1 3 3 5 1 1 4 8 4 2 3 6 1 3 3
6 0 4 1 2 7 2 2 5 4 1 5 1 1 5 8 1 1 4 1 8 8 6 7 2 7 8 5 1 5
```

Figure 24 – Output of a program that used randomly generated numbers and loops to create “The Matrix”

4.3.2.2 The didactical principles

The development of computer games can be a fertile learning experience for various school subjects. In particular, it certainly is promising for computer science. Both theoretically and practically, a game development project can span an entire computer science curriculum in Austria, e.g., see the curriculum at the Ministry of Education (BMBF, 2004). When it comes to programming itself, key concepts were successfully employed in the most recent year of my programming routine. Freely nameable terms of source code were named in the local language, i.e., German, while commands and predefined methods and interfaces were in English. What seemed here an awkward mashup at first glance had some didactical advantages. The German naming of a variable, for example, emphasizes the programmer’s choice of the name, thereby making it easier for students to grasp the concept of classes and instances and prevent such mistakes as trying to refer to a variable by typing its class instead of its name, e.g.,

```
string input;
```

```
        string = Console.ReadLine();          /*  common  mistake  */
```

Based on feedback from students and a thorough observation of the research setting, three principles were formulated. These principles help make game development with a class of teenagers more productive and rewarding.

The first principle is to set reachable goals, being firm and clear about exaggerated expectations. Further, it is vital to focus on realistically feasible game projects. Feasible games that qualify include simple jump and run or platformer games, adventure games, certain role playing games, top-view racing games, and puzzle games. At any rate, it is very important to avoid frustration among students and teachers. Unrealistic goals can also lead to losing oneself in the details during planning and wasting time even before anything is accomplished (Comber & Motschnig, 2015).

The second principle is to reduce complexity wherever possible. Students in an early year of game development working only with C# in Visual Studio 2010 Express and XNA stated that their main source of dissatisfaction was the complexity of game programming. The complexity in the first year of game development caused actual student motivation to lag behind their anticipated motivation. The solution to this challenge was to provide a simplified framework, but still have students write source code to embed a physics simulation. Out of this, GamePinS was born and employed successfully for the two years that followed.

The third principle emphasizes being extremely well-prepared. There are two reasons why being well-prepared is critical. First, there is a high learning threshold for computer science. There are approaches to projects that work well in other classes (e.g., history, geography, language, and arts); for example, teachers may set up stimulating environments with plenty of resources, books, colored pens, flipcharts, and fancy objects to tinker with; as a result, students in our school are able to complete excellent project work by themselves. In computer science, unfortunately, this approach fails. When it comes to transforming ideas into real results, the threshold is usually too high to accomplish these goals alone in computer science. Beyond this, technical problems might occur in the PC lab. These problems can be minimized through advance testing. Stating that “yesterday, it did work!” does not count as being extremely well-prepared. Excellent preparation further includes being an expert on what you want the students to understand and achieve. Relying solely on ingenious hackers in class is not a reasonable option. Further, aside from the individual qualities and principles that every lecturer and teacher brings to the classroom, these three principles are vital for successful game development with young learners (Comber & Motschnig, 2015).

4.3.3 The structure of GamePinS

GamePinS itself is a template with two sample projects that integrates useful libraries and importers, such as the Farseer Physics Engine and Tiled Map Importer. GamePinS implements a class hierarchy that is included in the GamePinS solution packages and can be modified directly if students need or wish to do so.

4.3.3.1 Farseer Physics Engine

Among the embedded libraries, one particularly useful library is the Farseer Physics Engine. The “Farseer Physics Engine is a collision detection system with realistic physics responses” (Weber, 2013). In detail, Farseer provides continuous collision detection with a time-of-impact solver, the appropriate callbacks for interaction, convex and concave polygons, and circles and multiple shapes per body. Farseer features collision groups and categories, friction and restitution, several joint types, controllers for gravity and force generators, tools to decompose concave polygons, factories to simplify the creation of bodies, and plenty of other features (Weber, 2013).

4.3.3.2 Map Importer

As a map importer, Squared.Tiled (Gadd, 2009) was employed. The source code was slightly modified to fit the needs of GamePinS and is included in the GamePinS template and samples. Maps are best saved as .tmx file with gzip compression and can be imported by creating a new instance of a map, i.e.,

```
// Load map. Parameters: Content Pipeline, World and Tileset
                  Karte = new cKarte();
```

```
Karte.Laden(this.Content, Welt, "Landschaft");[29]
```

Parameters for loading the map in this example are the Content Pipeline, the world ("Welt") for our physics simulation, and the name of the tileset. The map name must be specified in the cKarte class.

4.3.3.3 The GameLoop

Shown in Figure 25, the game loop was derived from XNA's GameLoop and simplifies the code structure. Students are encouraged to understand the difference between console programming and game programming.

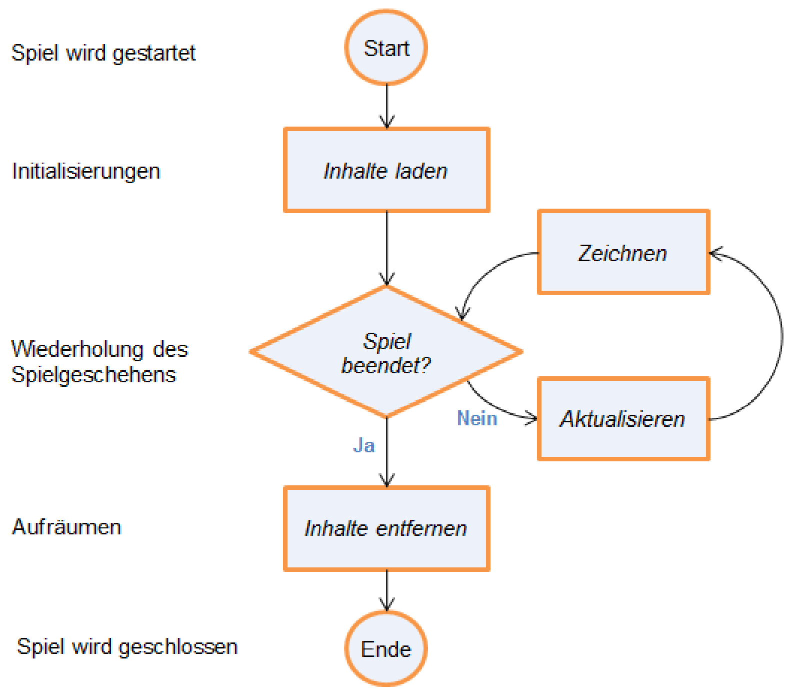


Figure 25 – The Game Loop as described in GamePinS for students

Source code for the game loop includes basic elements, such as a constructor, initialize and load methods, a draw method, an update routine, and a method to unload content[30].

[29] German instance names for didactical reasons (see Section 4.2.3).

[30] Although the UnloadContent routine is rarely used

Some code snippets, such as Draw and UnloadContent, are collapsed. Source code for the Game Loop with translated comments but original class names is as follows:

```
namespace GamePinS
{
    public class Spiel : Microsoft.Xna.Framework.Game
    {
        // Constructor
        public Spiel()
        {
                          // Basic settings
        }

        protected override void Initialize()
        {
                          // Initalize, e.g., Camera, Physics, Map
        }

        void InhalteLaden()
        {
                          // Load the game characters, levels, ...
        }

        protected override void UnloadContent()
        {
                          // Unload the content, if necessary
        }

        protected void Aktualisieren(GameTime paGameTime)
        {
                          // Update, e.g., collision detection
                          // play sounds
        }

        protected void SpielelementeZeichnen(GameTime pGameTime)
          {
           // Draw the players, the level, everything that needs to be drawn
        }
    }
}
```

4.3.3.4 GamePinS classes and hierarchy

For easier orientation of students and to teach object-oriented programming, GamePinS consists of a class hierarchy, as summarized in Figure 26. The class hierarchy offers options that keep game programming simpler in the beginning.

class name	description
cAnimiertessprite	animated sprite
cAudio	audio class
cBaustein	brick - level building
cDebuginfos	debug infos
cEbene	layer of a level/map
cEinstellungen	settings class
cKamera	camera class
cKarte	class for the map
cPhysiksprite	physics sprite
cSpielerDraufsicht	player topview
cSpielerSeite	player sideview
cSprite	sprite class
cText	class for displaying text

Figure 26 – Class names and descriptions of classes used in GamePinS

Nonetheless, students can step into the class source code at any time if they feel the need to make modifications. The techniques for creating basic objects is covered in the lessons, and students learn how to create a sample object, i.e., an elephant, in the classroom; this class is defined as follows:

```
public class cElefant : cSpielerSeite
{
    public void Laden(ContentManager pInhaltsmanager, World pWelt)
    {
        Spielergrafik = "Elefant";
        Geschwindigkeit = 140;
        this.Startposition.X = 500f;
        this.Startposition.Y = -100f;
        float Masse = 12f;
        base.Laden(pInhaltsmanager, Spielergrafik, pWelt, this.Startposition.X,
             this.Startposition.Y, 1, Masse, BodyType.Dynamic);
    }
}
```

Elefant is derived from the player's *cSpielerSeite* class and can be created with very few extra values. Here, the load method gets the *ContentManager* and the world for the physics engine as parameters. The name of the player’s graphic is set using *Spielergrafik*. The word *Geschwindigkeit* means speed and determines how fast the elephant can run. The *Startposition.X* and *Startposition.Y* values are set to coordinates on the top of the screen. XNA uses a right-handed coordinate system in which the center is in the middle of the screen. *Masse* determines the mass of the elephant for the physics simulation, and *base.Laden()* calls the base method with the *ContentManager*, the player sprite, the world, and the starting positions. The next parameter is the parameter for the body form in the physics simulation and is either a circle (i.e., 0) or a rectangle (i.e., 1)[31]. The *BodyType* can either be static (i.e., a level element) that does not move or dynamic. This applies to all elements (i.e., players, balls, boxes, and so on) that are included in the physics simulation.

As shown in Figure 27, the class hierarchy takes two aspects into consideration, i.e., simplicity and performance. Therefore, the topmost parent class is a sprite class (i.e., cSprite), which has three child classes, i.e., an animated sprite, a class for a map element, and a class for the background. The animated sprite class has the physics sprite as a child and the physics sprite leads to the player's class, the (non-player) creature class, and other physics object classes.[32]

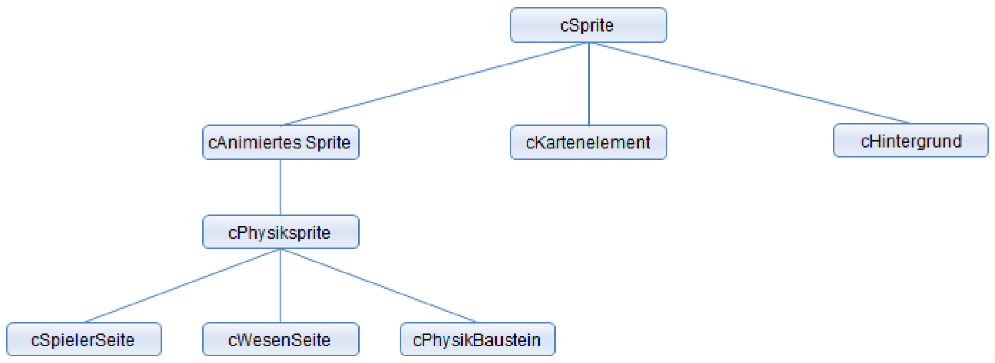


Figure 27 – Excerpt of the class hierarchy of GamePinS

From the class hierarchy, one might wonder why there is another step from the animated sprite class to the physics sprite class. As mentioned above, performance is important and as every physics object needs resources, there is the possibility of making some animated objects, e.g., birds in the background, that do not consume extra resources.

[31] The values 0 and 1 here are relics and will have proper constants defined in the next version.

[32] For the complete list of classes, see Figure 26 – Class names and descriptions of classes used in GamePinS.

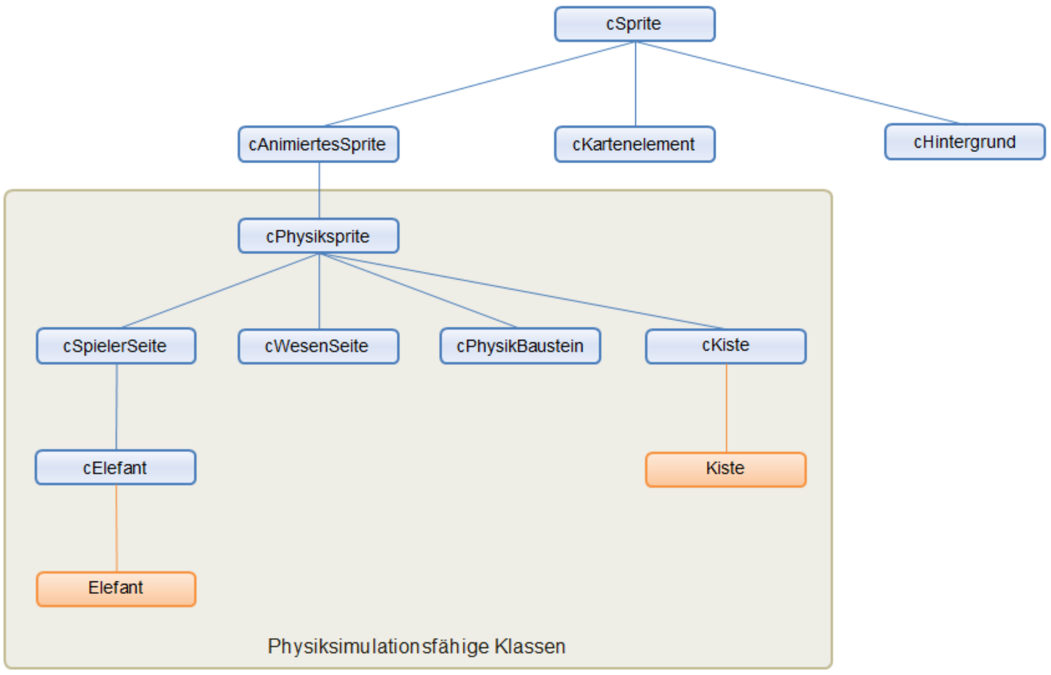


Figure 28 – Example of using the class hierarchy for a two-dimensional platformer game with physics simulation

In other words, there are no restrictions at the bottom level of the hierarchy. In fact, students were encouraged to use whatever class they wanted to reach their goals. If the young programmers do not need physics in their role playing game, they may also derive everything from the animated sprite class and perform their collision detection with the help of the map objects of the map class.

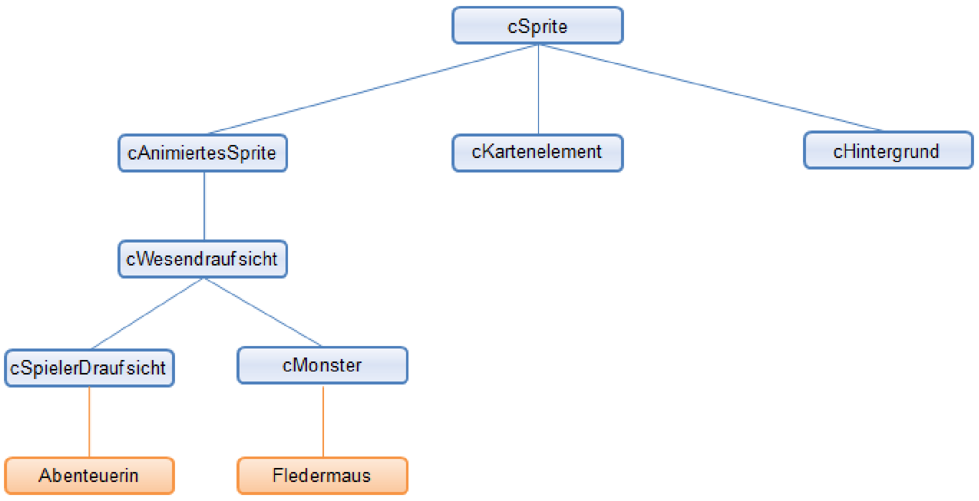


Figure 29 - Example of using the class hierarchy in a top-view role playing game

5 Research and Results

5.1 General attitudes toward games and game development in the classroom

"Developing a computer game? That's what you're doing with students? Aren't they already, enough sitting in front of their computers, wasting their time with digital stuff?" (Anonymous teacher in an Austrian Secondary school on Game Programming, 2013)

This quote is typical when I tell my teacher colleagues or others not in the IT area what I have been doing in my research. Confronted with the issue of spending too much time in front of a computer, my intentions are the following:

"I don't want students to spend more time in front of the computer, but I want to encourage them to use their time differently. I try to excite them, get them to move from being just consumers of games to actual game creators, which can be a source of plenty of fun and development, but also an original and valuable learning experience."

Below, I address the basic ideas of the GamePinS activities in response to the attitude noted above, thus describing how spare time activities of students connect with electronic media. Further, I elaborate on the role of game development and the general attitudes toward game development.[33]

The quote from the colleague at the beginning of this section, aside all other intentions, surely raises the question as to how much time students spend in front of digital media and what they are doing. A study from the Statistics Austria (Statistik Austria, 2010) exists regarding the spare time of juveniles; this survey helps to obtain a picture of the specific target audience and answer such questions as "how much time are students spending with digital media?" and "what are they doing in front of their devices?" Another goal of this survey was to determine the attitudes of young learners toward game development in the classroom.

[33] This part of the research was general research with a larger sample size; to see its illustrated role, refer to Figure 44 – Research approach using questionnaires and .

To address the noted issues, I wanted to state that game development is not intended to add extra hours in front of the PC to the students' spare time, but instead to playfully and energetically foster their IT skills in class. If students are willing to do something at home, the goal is to lead them from simply consuming games and videos for fun to actually creating games during their time in front of the PC. This is still time with digital media, but seems a very valuable addition to their other digital media spare time activities.

Arguing that GamePinS enhances programming skills is one aspect of this, but as an IT teacher, enforcing programming instead of concentrating only on Office applications, I often hear the response that "students surely won't all be programmers, so not all them will need this programming stuff."

Aside from the fact that more diversified education means more chances and, further, that IT skills are perceived as vitally important in most leading countries, e.g. see the recent TechHire initiative from the US-president (Obama), IT skills can be very useful for general problem solving in numerous fields. Analytical thinking, modeling problems and solutions, abstraction, going from theory to implementation of solutions, and logical thinking are just some aspects that are enhanced by programming (Shein, 2014).

Back to the original enquiry, aside from standard demographic data, students answered questions to help determine what they were doing with computers in their spare time and what students were expecting from game development.

5.1.1 Statistical data

5.1.1.1 Gender distribution

As shown in Figure 30, the number of female and male students differed in the target audience, with 105 female participants and 191 male participants.

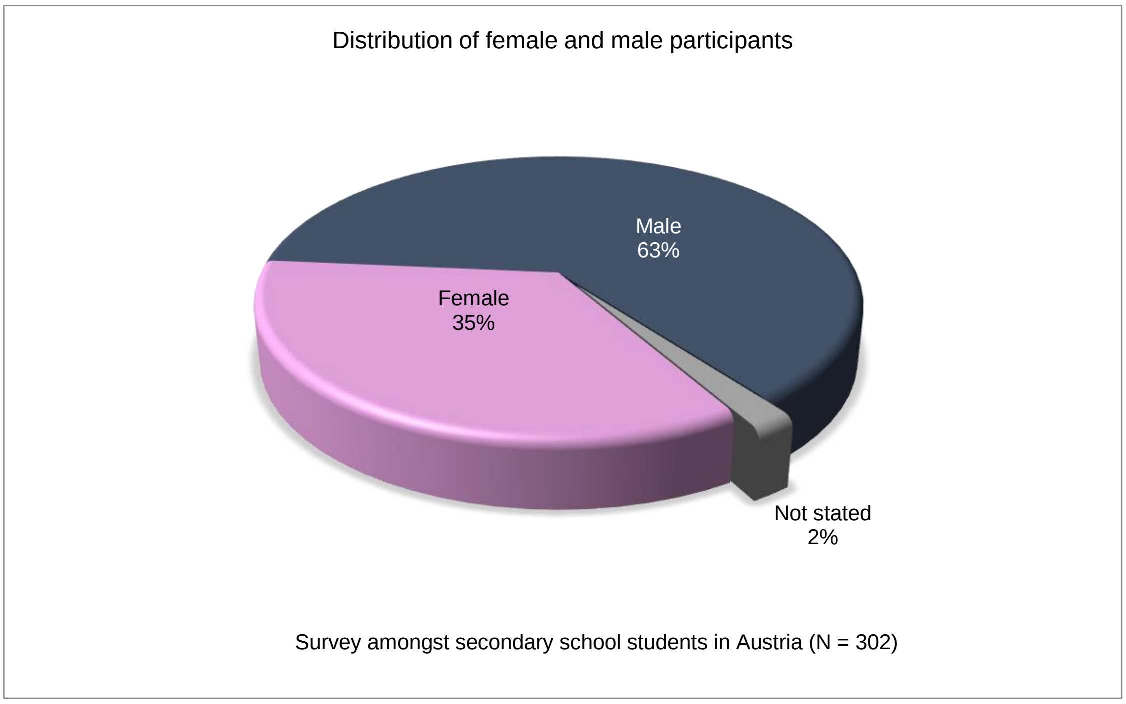


Figure 30 – Distribution of female and male participants in the game development survey

The female/male distribution was not due to participation itself (i.e., that more males chose to participate), because in all classes, almost all students agreed to complete the survey, but the classes happened to be composed of more males than females. To explain the classroom composition, a look at gender roles and subjects might clarify this composition. Boys and girls in secondary school are still slightly influenced by traditional roles. In the target school[34], boys tend to choose natural and technical science-related subjects more often than girls. Girls statistically tend to favor languages over science and technology.

[34] GRGORG 16, Maroltingergasse 69-71, 1160 Wien

In GRGORG16, computer science was part of the natural/technical curriculum branch in third and fourth grades. In the language curriculum branch, students did not have computer science as a subject in third and fourth grades at all. Only in fifth grade did all students from all curriculum branches independently take computer science classes. The above led to an overall breakdown in the game development questionnaire of 105 female and 191 male participants[35].

5.1.1.2 Internet access

As shown in Figure 31, of the 302 participants, 298 or 97.7% had Internet access at home; in one case, it was not stated, and only two participants noted that they did not have Internet access at home. This result correlates well with the values found in Statistics Austria (Statistik Austria, 2014), where 95.5% of households with one adult and children have Internet access, and 97.8% of households with two adults and children have Internet access. On average, in Austria, 81% of households have Internet access.

[35] In six instances, the participant’s gender was not stated.

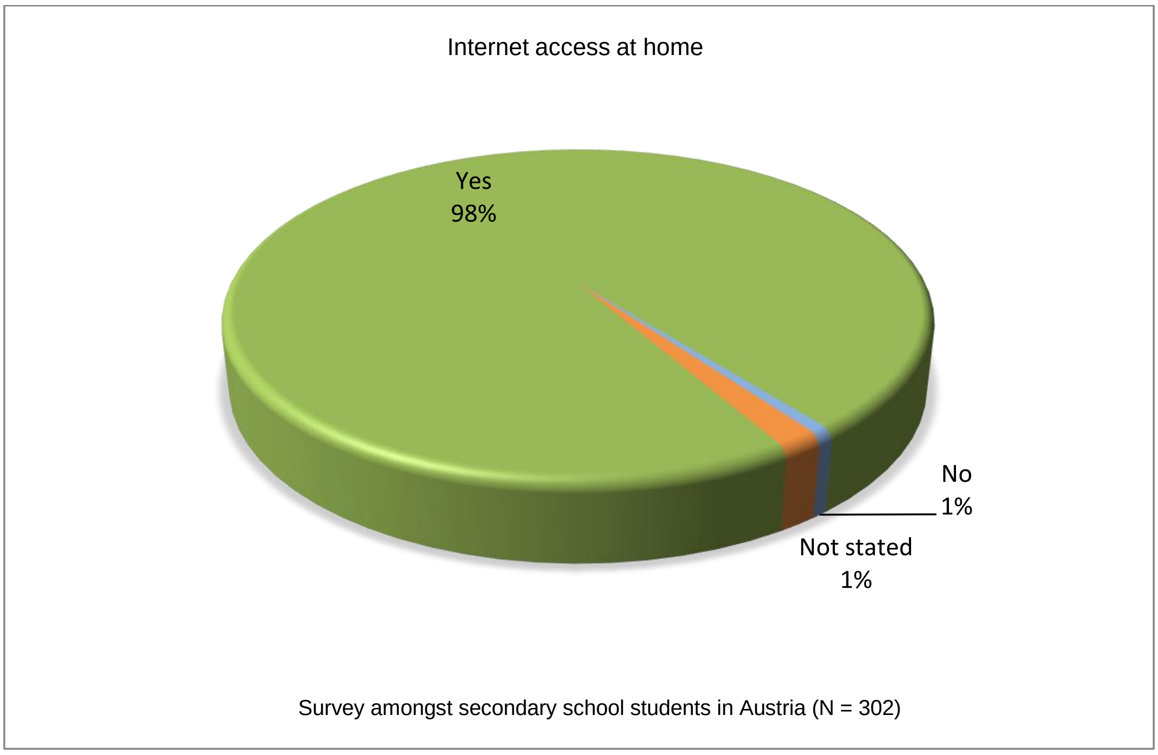


Figure 31 – Internet access of students in the targeted secondary school

5.1.1.3 Age structure

As shown in Figure 32, most students (i.e., 106) who answered the questionnaire were 14 years old. Not surprisingly, that is the age most students are when they reach fifth grade, where every student has a mandatory computer science class at GRGORG 16. Of the rest, 68 students were 13 years old, and 81 students were 15 years old. In general, secondary schools in Austria are required to have two hours per week of computer science at the minimum (BMBF, 2003). If a school has permission for an autonomously designed curriculum, more than two hours of computer science per week are allowed in an entire school year. In this case, those two hours do not have to be in the fifth grade (BMBF, 2003). Otherwise, only the mentioned two hours of computer science education are held in the fifth grade, i.e., the first advanced level grade (BMBF, 2004).

GamePinS is designed for 13-year-old students and upward. The age of participants in the questionnaire and that of the target group of GamePinS matched well; therefore, an estimation as to whether GamePinS would be accepted by young learners was performed using the following questions.

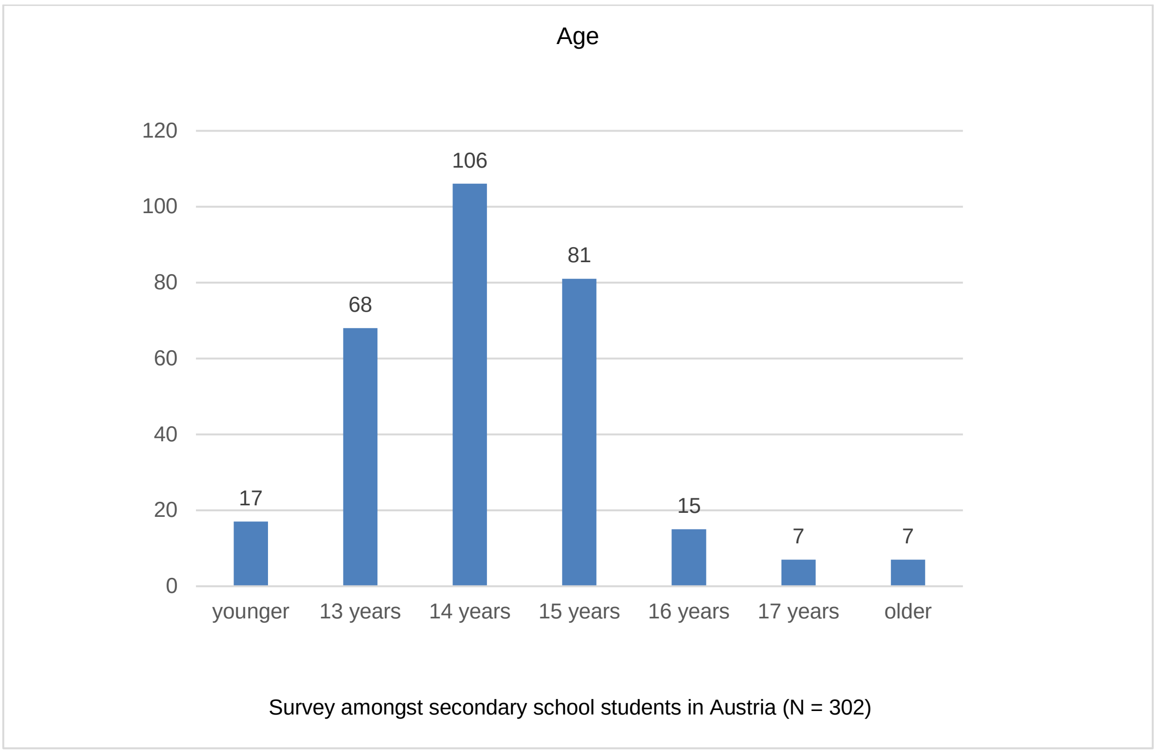


Figure 32 – Age of students from the target audience for game development

5.1.2 Activities of students on the computer in their spare time

As shown in Figure 33, student activities during their spare time on a computer emerged in an interesting pattern. For *communication, surfing, and reading articles on the Internet*, *activities for school*, and *other activities*, the time spent per week was primarily between *not at all* and *less than 2 hours*, then dropped considerably; however, for playing games, the curve did not drop so rapidly, with 46 students playing more than 10 hours per week.

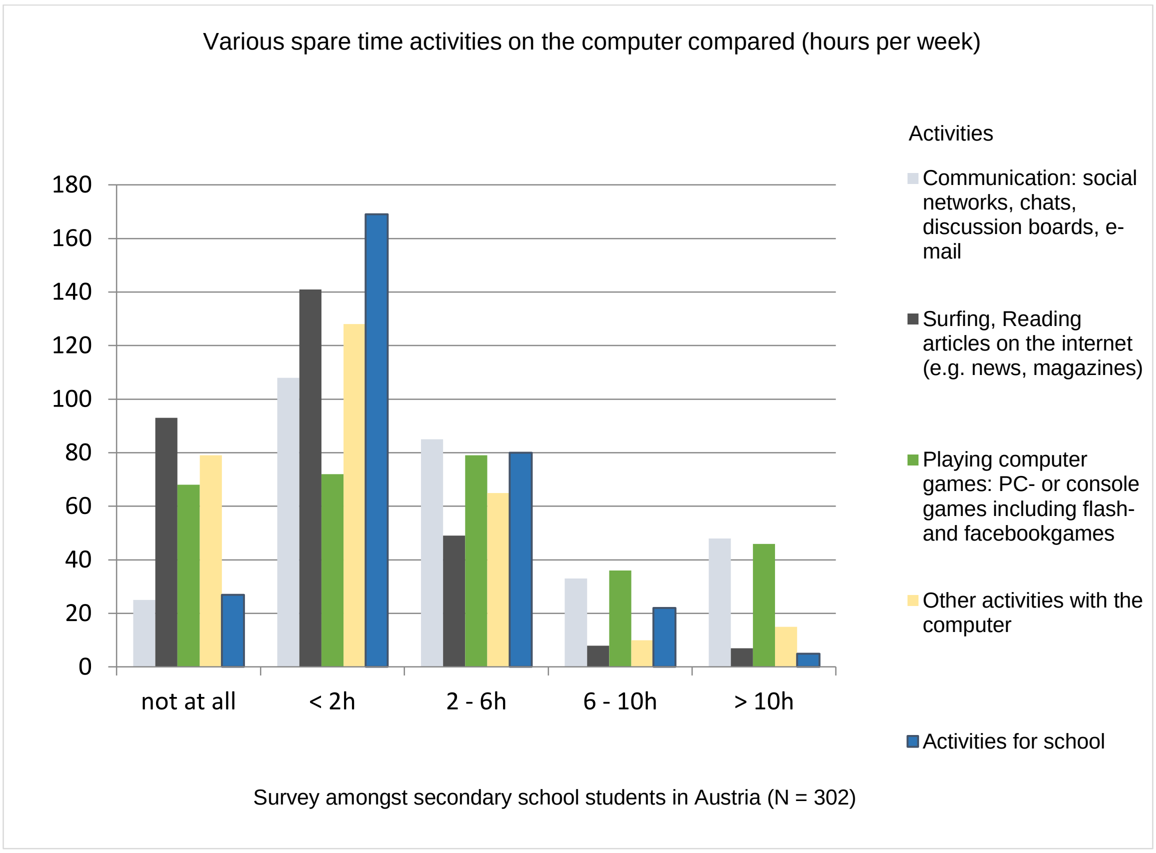


Figure 33 – Activities of students on the computer in their spare time

5.1.2.1 Time spent in front of a computer for school-related activities

As shown in Figure 34, 27 students did not use any of their time in front of a computer for school-related activities. The number of students who spent from a few minutes up to two hours per week was 169. A further 80 students spent between two and six hours of their time per week on the computer for school. Only 22 young learners used six to ten hours per week for school, and five students worked more than ten hours per week for school on a computer.

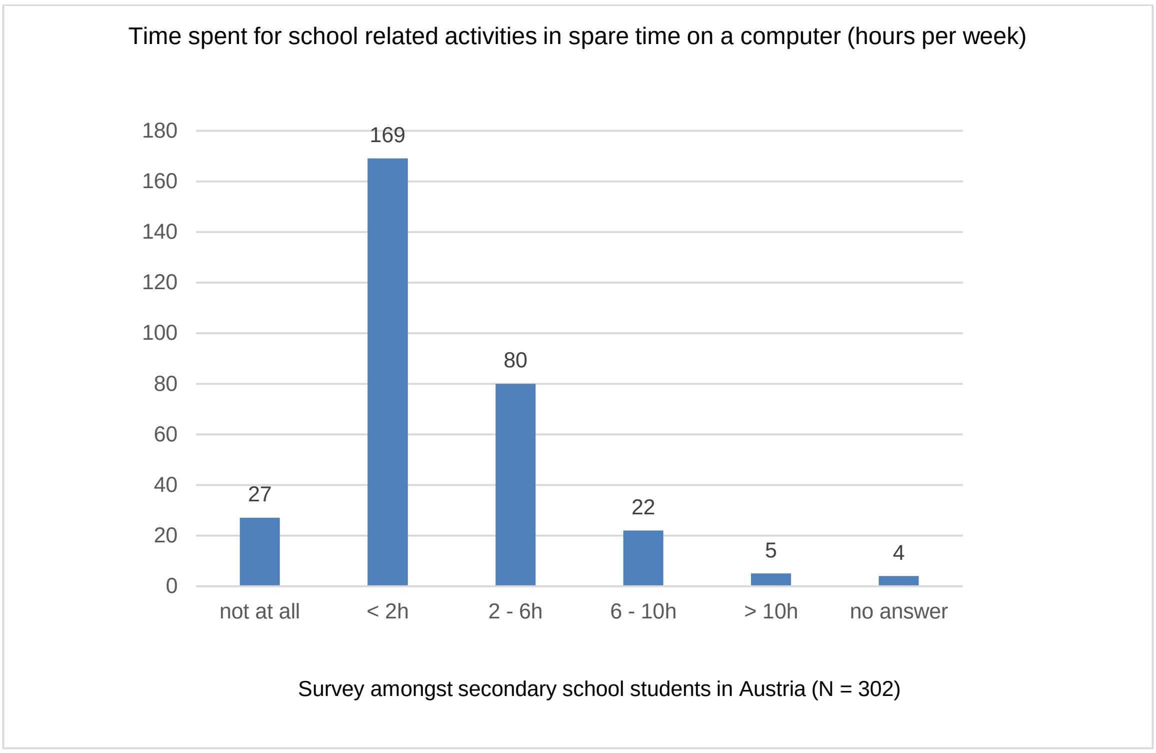


Figure 34 – Spare time spent for school-related activities in front of a computer

5.1.2.2 Spare time spent playing computer games

As shown in Figure 35, playing games was popular among young people. Approximately 77% of students stated that they played computer games regularly or at least occasionally. Within the group of these game-playing students, one-third limited their playing time to less than two hours per week, another third played between two and six hours per week, and the remaining third more than six hours per week, including the 20% who played more than 10 hours per week. The collected values were obtained during the semester with validity for the semester. One student added that during holidays, he played much more than during the semester.

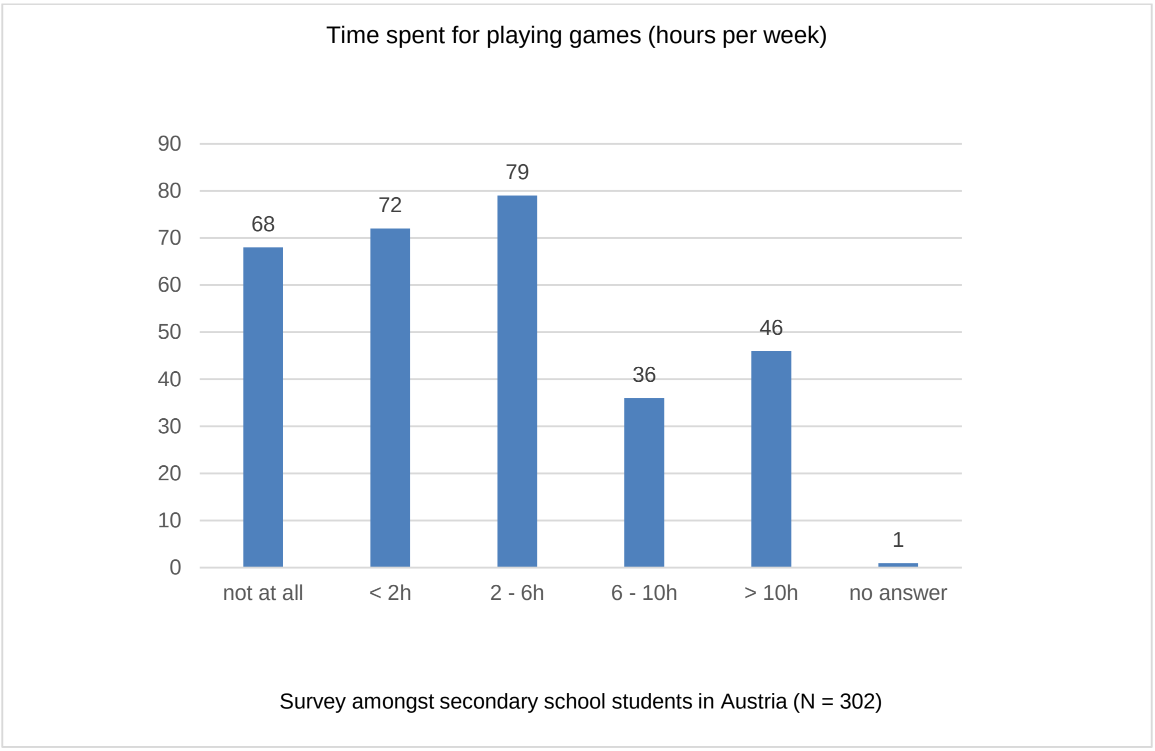


Figure 35 – Spare time spent playing computer games

Compared with the time spent mandatorily for school, gaming has huge potential to involve students both inside and outside of the classroom. The willingness to spend time with games may be due to the effect of intrinsic motivation (Deci & Ryan, 2005; Przybylski, Weinstein, Murayama, Lynch, & Ryan, 2012), perceived autonomy (Ryan, Rigby, & Przybylski, 2006), a principle of amplification and rewards (Yee, 2006) based on simple conditioning (Skinner, 1953; Watson & Rayner, 1920), or most likely a combination of these phenomena.

5.1.2.3 Communication via social networks, chat, discussion boards, and e-mail

Interestingly, 25 students did not spend any time chatting, sending e-mail, using discussion boards, or on social networks[36]. For 108 students, the time spent with social networks, chats, discussion boards, and e-mail was less than two hours per week. A further 85 students communicated from two to six hours per week. There was an interesting gap

[36] The alternatives of calling via phone or sending text messages were not counted in this item; therefore, this does not imply that those students did not communicate via any digital devices.

between spending time communicating moderately and more extensively, i.e., only 33 students spent six to ten hours per week, whereas 48 students used social networks, chats, discussion boards, and e-mail more than ten hours per week.

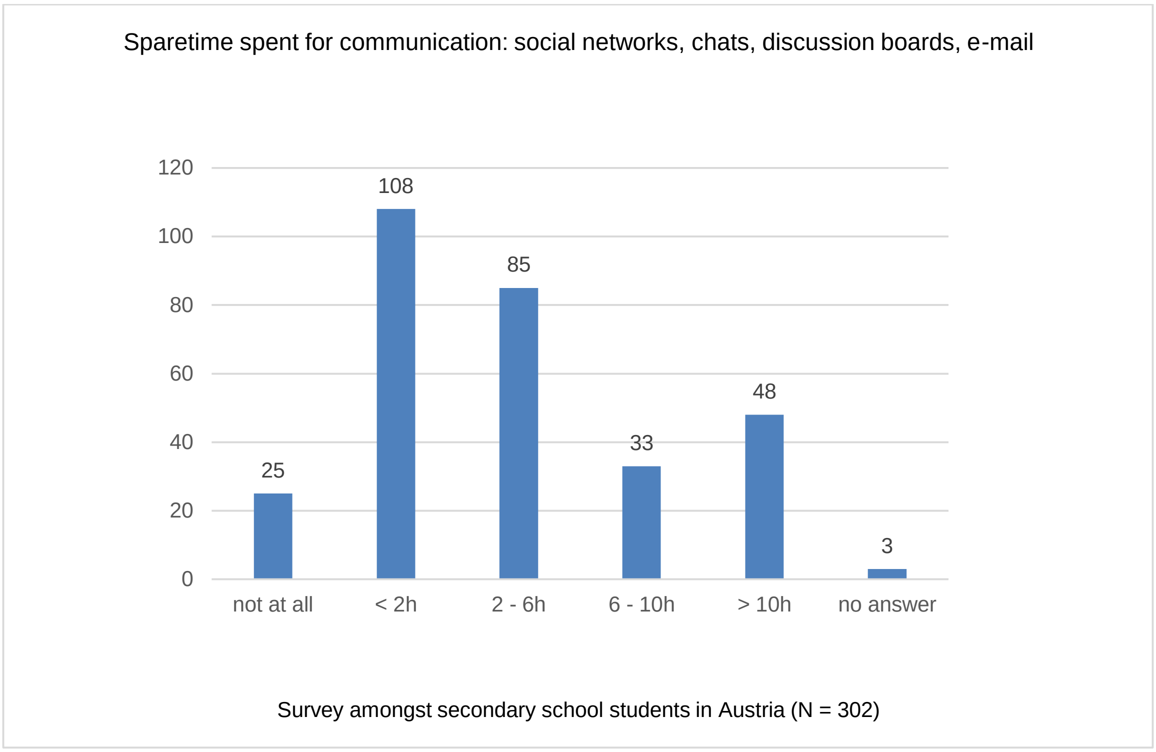


Figure 36 – Time used to communicate via social networks, chats, discussion boards, and e-mail

5.1.2.4 Reading articles

Students spent less time reading articles than reading messages from friends. As shown in Figure 37, 93 students did not read any articles, journals, or news at all. A further 141 spent less than two hours per week, and 49 spent between two and six hours per week. Eight students read between six and ten hours per week, and seven for more than ten hours per week.

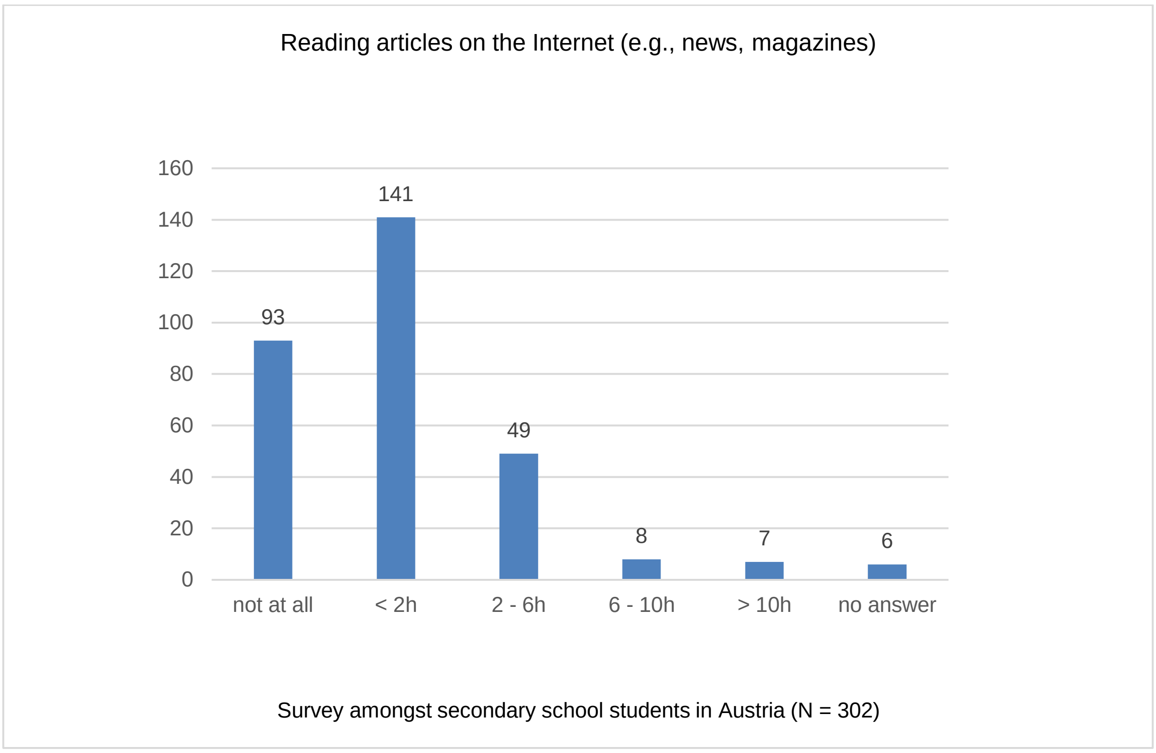


Figure 37 – Time spent reading articles on the Internet

5.1.2.5 Programming in student spare time

While it was no surprise that 70% of students did not code at all in their spare time, it was indeed surprising that 29% of these young learners used their spare time to program[37]. Even more surprising is that programming for school was excluded, i.e., this item asked specifically for programming out of one's own interest and not for school.[38]

[37] Note that 1% provided no answer.

[38] German item text: Programmieren aus eigenem Interesse (also nicht für die Schule)

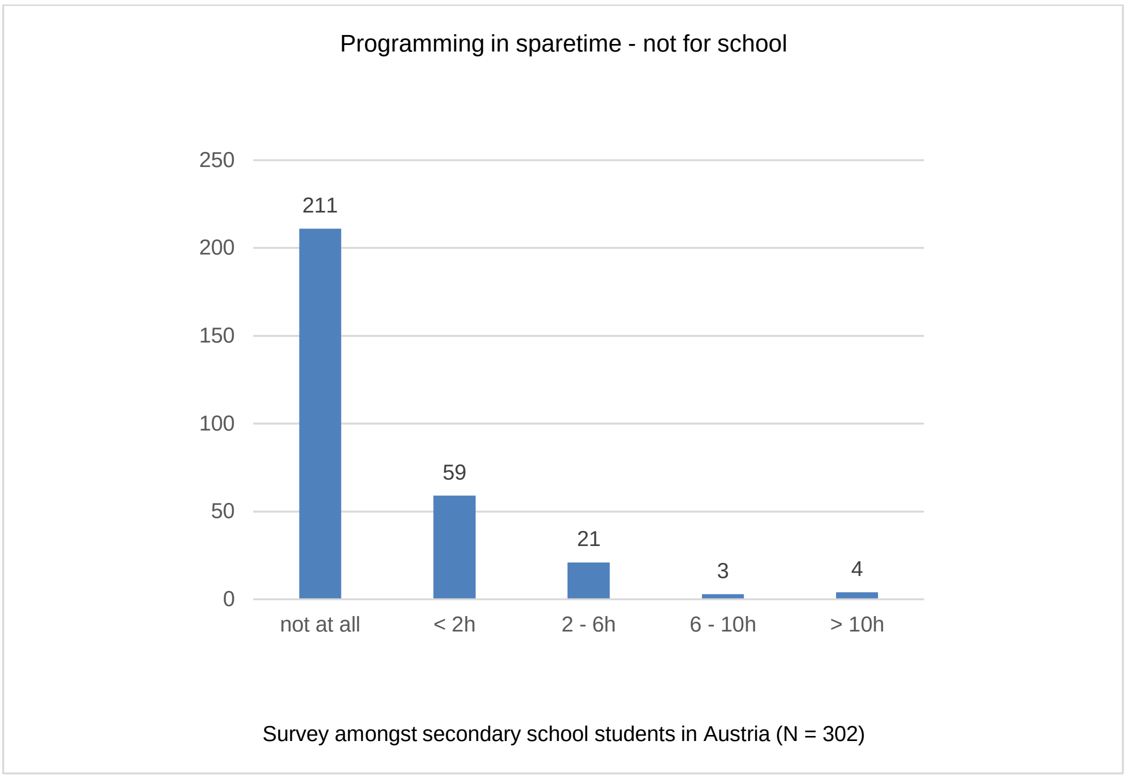


Figure 38 – Programming in spare time out of one’s own interest

5.1.2.6 Spare time spent on various other activities

Aside time spent playing games, on social networks, surfing, reading, activities for school, and a little bit of programming[39], on average, students spent very little time on other activities on the computer. As shown in Figure 39, 42% spent less than two hours per week, 22% spent between two and six hours per week, and 26% did not spend any time on any other activities aside from the aforementioned activities.

[39] See Sections 5.1.2.1 through 5.1.2.5 and Figure 33.

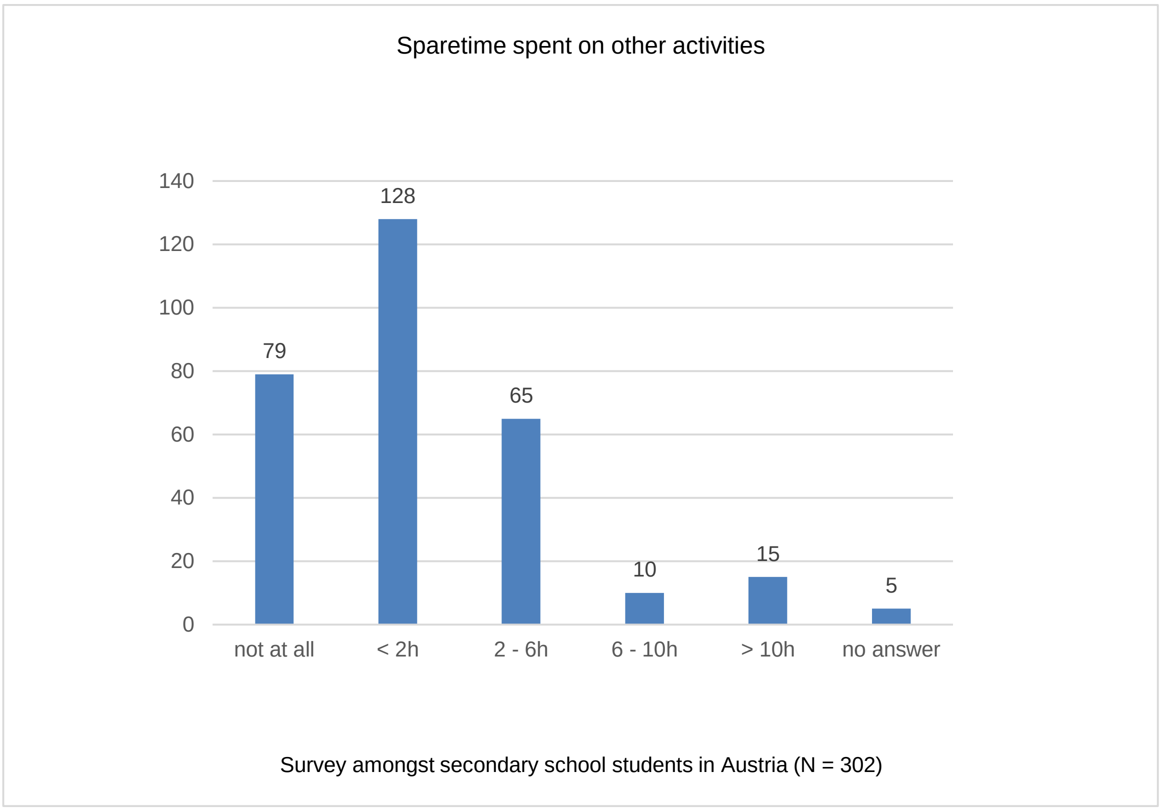


Figure 39 – Amount of spare time spent on various other activities on the computer

Students were also asked to name some of the other activities not covered by the questionnaire. Answers ranged from producing videos for their own YouTube channel and just watching Youtube or other movies to collecting, organizing, and listening to their own music collections to viewing questionable videos on questionable Web sites[40] to clearly stating consuming pornography. The Internet has introduced a new dimension to the topic of pornography, reviving an old discussion (Gernert, 2010; Magdalena Mattebo, Tydén, Häggström-Nordin, Nilsson, & Larsson, 2013; Owens, Behun, Manning, & Reid, 2012; Zillich, 2011).

The numerous answers from the students were collected as free text and clustered into groups afterwards, summarized in Figure 40. The distribution of time spent on other activities displayed an affinity for videos. Approximately one-quarter of students stated

[40] …whatever that means?

that they were “watching or editing YouTube videos,” with 18% watching videos elsewhere. The next activity was listening, collecting, or organizing music, followed by searching for information (e.g., “Google things”).

After music and search, the next activity was online shopping, which seemed small, but is on the rise in Austria (Statistik Austria, 2013) and also an economically (Limayem, Khalifa, & Frini, 2000) and socially (Overby & Lee, 2006) important matter around the globe, especially for teenagers (Alam, Bakar, Ismail, & Ahsan, 2008; Thomson & Laing, 2003).

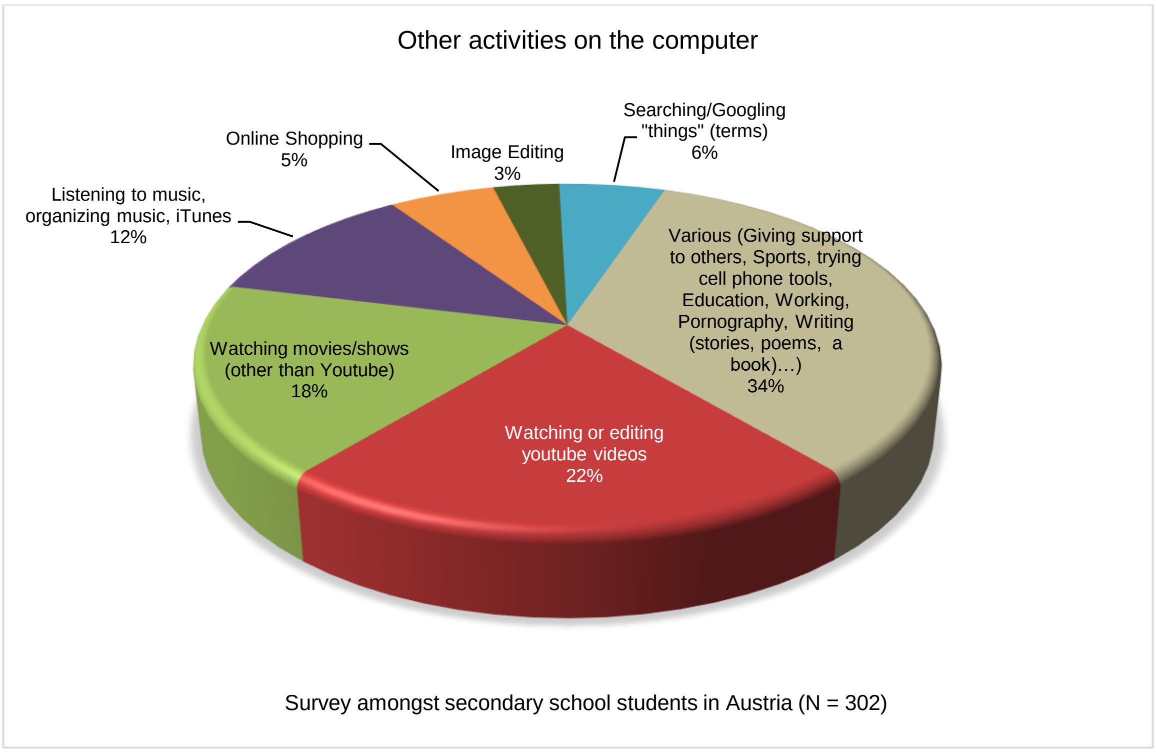


Figure 40 – Distribution of “other” activities on the computer

5.1.2.7 Developing a computer game in computer science education is interesting

To analyze the potential for game development in school, students were asked if they agreed with the statement, “Developing a computer game in computer science education is interesting.” The majority of students (i.e., 80%) agreed that developing a game was interesting. As shown in Figure 41, 51% fully agreed, 29% agreed, 7% showed a neutral attitude, 7% disagreed, and 5% fully disagreed.

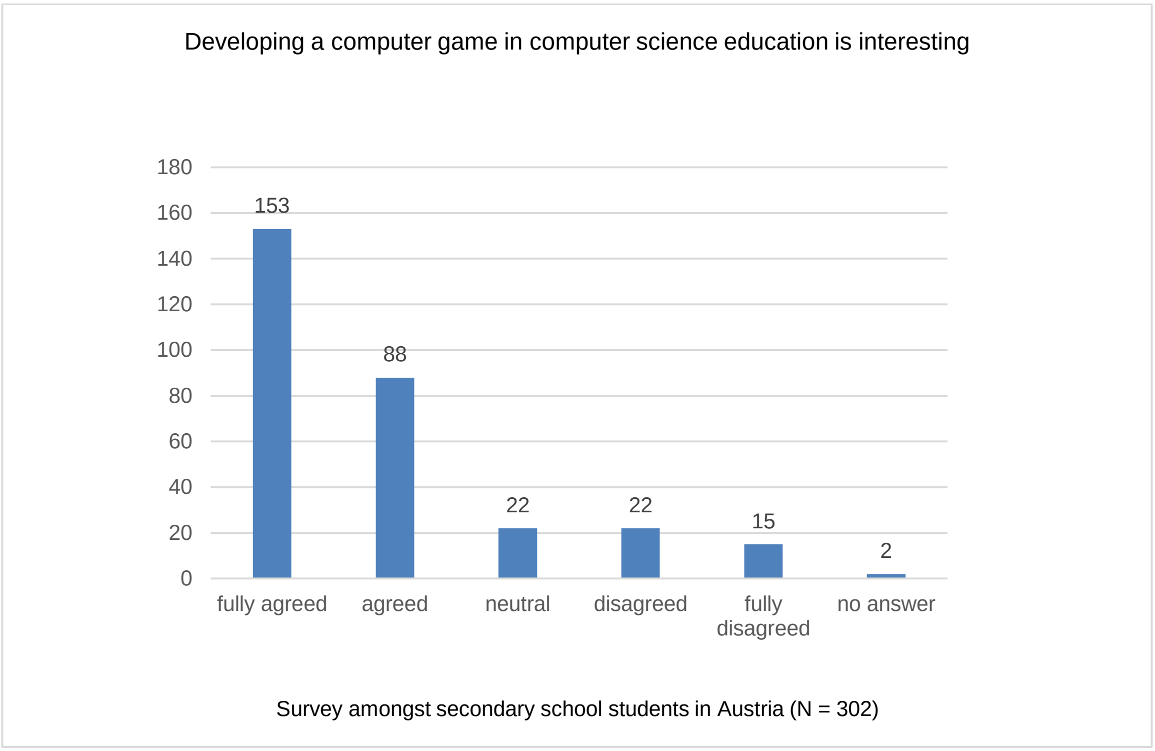


Figure 41 – Survey results regarding whether developing a computer game in computer science education is interesting

5.1.2.8 *Game development in students' spare time*

Game development is a time- and resource-consuming endeavor that is "Harder Than You Think" (Blow, 2004). In contrast, *time* in computer science class seems always to be too short, even if teachers and students have more than two weeks (Rankin, Gooch, & Gooch, 2008). One idea here was to see if students were also ready to use their spare time to further develop a computer game. As described in Section 5.1, it should not be a goal for students to spend even more time with digital devices, but instead to use their time in different ways to become creators rather than merely consumers.

When it comes to spare time, students seem to be more careful before agreeing to a statement in which they are asked whether they would use their spare time for game development. As shown in Figure 42, 45% agreed with the statement, "for developing a computer game, I'm ready to use my spare time outside of time spent in school." A further 40% disagreed, and 15% were neutral.

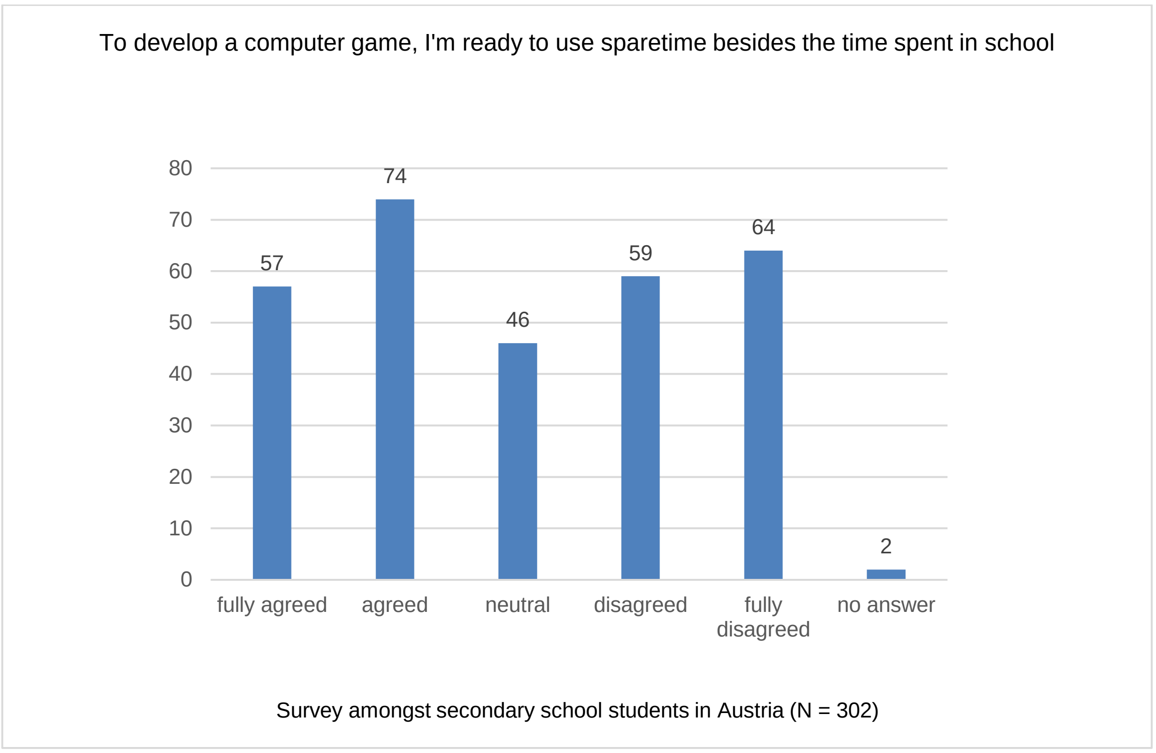


Figure 42 – Survey results in response to, "For developing a computer game, I'm ready to use my spare time outside of time spent in school"

5.1.2.9 *Reasons for playing computer games*

I have always been fascinated by video games and am aware of several reasons for this strange but familiar force of attraction. For single-player games, in my case, the aspect of diving into a different world was always the key reason for enjoying a game. For multiplayer games, the leading reason was the challenge of competing, either alone or even better as a part of a team, with friends or other people I did not even know. Both aspects were connected with a pleasant satisfying feeling, with an overall connotation of a great deal of fun. The question then is: *is this also true for the average young player?*

To find out what attracts students to playing computer games, their free text comments were evaluated, counted, and clustered; the results are shown in Figure 43.

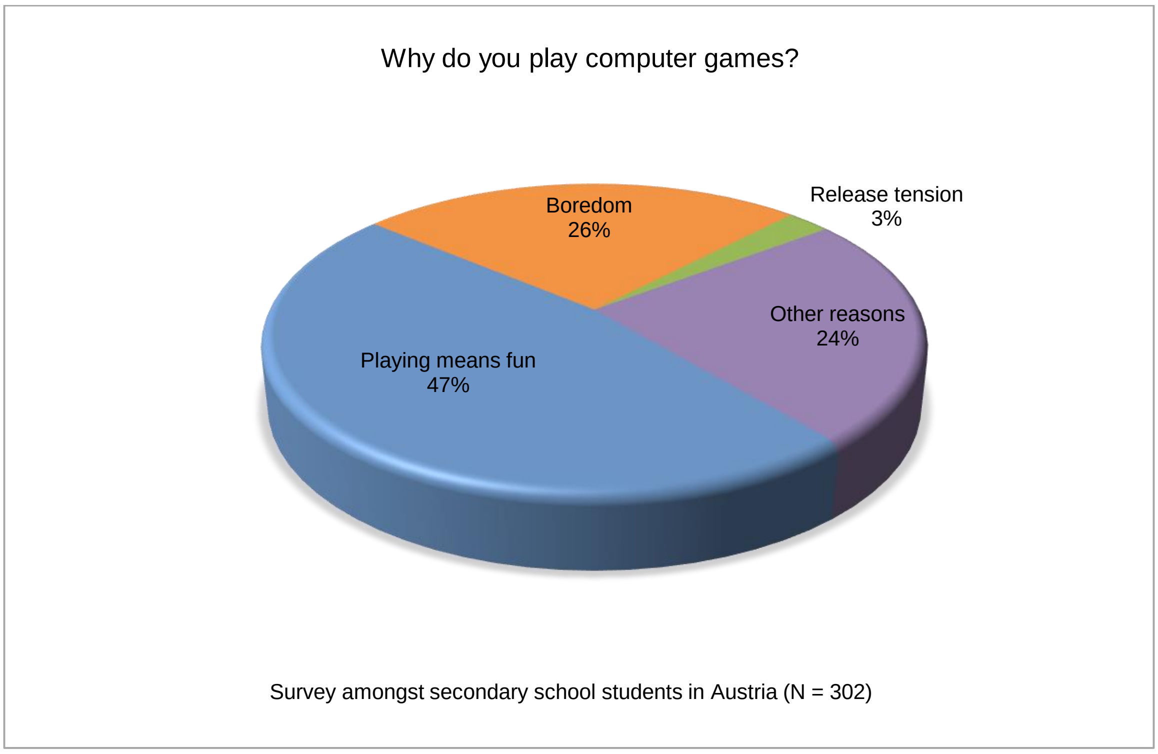


Figure 43 – Survey results regarding reasons for playing computer games

Primarily, playing computer games translates simply to fun. Approximately one-quarter play out of boredom and another quarter for various reasons. Interestingly, only 3% noted they play computer games to release tension. The fun young gamer's experience might come from the group experience (e.g., *“ich dabei abschalten kann und es Spaß macht allein oder vorallem mit Freunden.”* - *“I can switch off and have fun alone or with friends.”*) and from the escapism in traveling to other worlds and being able to be a different character (e.g., *“Ich spiele oft Compuerspiele (meistens Online Spiele), weil man selbst einfach auch in eine andere Rolle schlüpfen kann und so sein kann wie man will.”* - *“I often play computer games (mostly online games), because you can dive into another role and be whoever you like.”).*

5.2 First year of game programming with XNA

In the first year[41] of game programming, there was no GamePinS framework, only the intention to use game programming in schools to increase learning by providing lessons in the interesting and motivating field of computer science. In that first year, students used the following tools to create their games:

- Microsoft Visual Studio C# 2010 Express
- XNA Game Studio 4.0
- GIMP 2.4
- Audacity

5.2.1 Course and research design

The initial research design consisted of a general questionnaire and a post-GamePinS questionnaire, as illustrated in Figure 44. First, groups A and B had to complete the general questionnaire before beginning the game programming activities. Next, GamePinS activities followed. At this time, GamePinS was more of a didactical framework describing the structure of two semesters of computer science with game programming. The game programming itself was realized using Visual Studio C# 2010 Express and "plain" XNA. At the end of the second semester, students in both groups filled out the given survey again. In addition, the general questionnaire targeted a group of students (i.e., group C) who were not involved in game development, thus helping to determine attitudes toward game development.

[41] The first year started in September 2011 and ended in June 2012.

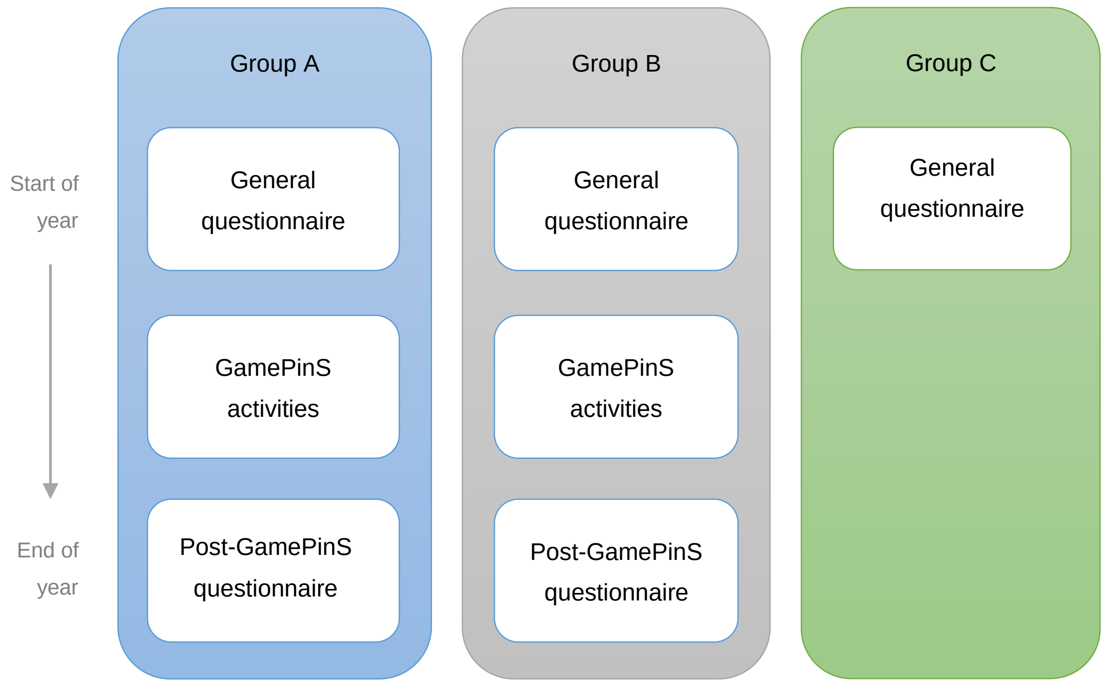


Figure 44 – Research approach using questionnaires and three groups in the first year

The general questionnaire contained the same questions for groups A, B, and C. Nevertheless, groups A, B, and C were evaluated separately. For groups A and B, it was important to compare expectations about game programming with the experiences students had after one year of game programming. Group C was included to obtain data to generate a bigger sample for determining attitudes toward game development.

5.2.2 Activities

During the first semester[42], students started with programming lessons in C#, which covered basic input/output operations (e.g., *Console.ReadLine()*, *Console.WriteLine()*), conversion operations (e.g., *int.Parse()*), conditionals, SWITCH statements, and loops (i.e., *FOR*, *WHILE*). Further, the necessary tasks for creating game graphics using GIMP were discussed and practiced, as was audio editing with Audacity.

[42] September 2011 through February 2012.

In the second semester[43], students formed two game programming groups. Groups consisted of three or four core engine programmers, one or two graphics artists, one or two sound engineers, and one project manager. The two teams worked on two separate game projects, one was a jump-and-run project in which a paperclip must navigate through a hostile office environment, the second was centered on a *robo-hobo*[44] in a futuristic dystopian scenario on a wasted earth seeking revenge and justice. Game programming was implemented only using XNA, which was more flexible than DirectX programming. Finally, a sample XNA project was introduced to provide an easier start for the students.

5.2.3 Questionnaire and results

The questionnaire for the game programming activities consisted of various questions using ordinal scaled answer types and *free text* answers. Each question was equipped with certain answer possibilities. In addition, the design of the questionnaire provided the possibility to skip certain questions, which resulted in a “not stated” entry.

The group of items using nominal and ordinal scales comprised the following answer types[45]:

- Yes/No question: “Yes”; “No”
- Gender: “Female”; “Male”
- Levels of Agreement: “completely agree”; “agree“; “undecided”; “disagree”; “completely disagree”
- Time per week A: “no time“; “up to 2h“; “2+ to 6h“; “6+ to 10h“; “10h+”
- Time per week B: “up to 5h“; “5+ to 10h“; “10+ to 15h“; “15+ to 20h”; “20h+”
- Changes: “strongly reduced“; “reduced”; “stayed the same”; “increased”; “strongly increased”
- IT skills level: “Beginner”; “Intermediate”; “Adept”; “Power user”; “Expert”
- Age: “younger”; “13”; “14”; “15”; “16”; “17”; “older”

[43] Middle of February 2012 through the end of June 2012.

[44] This term refers according to the students to a decommissioned robot who roams around the envisioned world.

[45] Only answer types listed were used in the survey.

- Activities on Computer: “Communication (Facebook[46], Chats, e-mail)”; “Browsing the Internet”; “Playing games”[47;] “Activities for school”; “Other activities”[48]

5.2.3.1 Statistical data

Statistical data was calculated to look into further details in cases of irregularities within answers or as baseline data in support of new findings that might not be relevant to our current purpose here (i.e., possibly leading to new findings in different contexts, e.g., “gender and computer games”).

Gender

Question text: *Geschlecht*
Translation: *Gender*
Possible answers: *“Female”; “Male”*

As shown in Figure 45, the female/male distribution of participants was not even (i.e., three females and 22 male students). Gender studies in the context of information technology (Demetrulias, 1985; Shashaani, 1994) is also an interesting research area[49], but gender questions are not covered in this thesis, since the answers did not show significant differences, although there might be differences in the way problems were solved by girls versus boys.

[46] “Social” activities on Facebook, without Facebook games.

[47] Games, including Facebook games.

[48] Activities not mentioned in the previous categories.

[49] This applies to both the gaming context and for me personally.

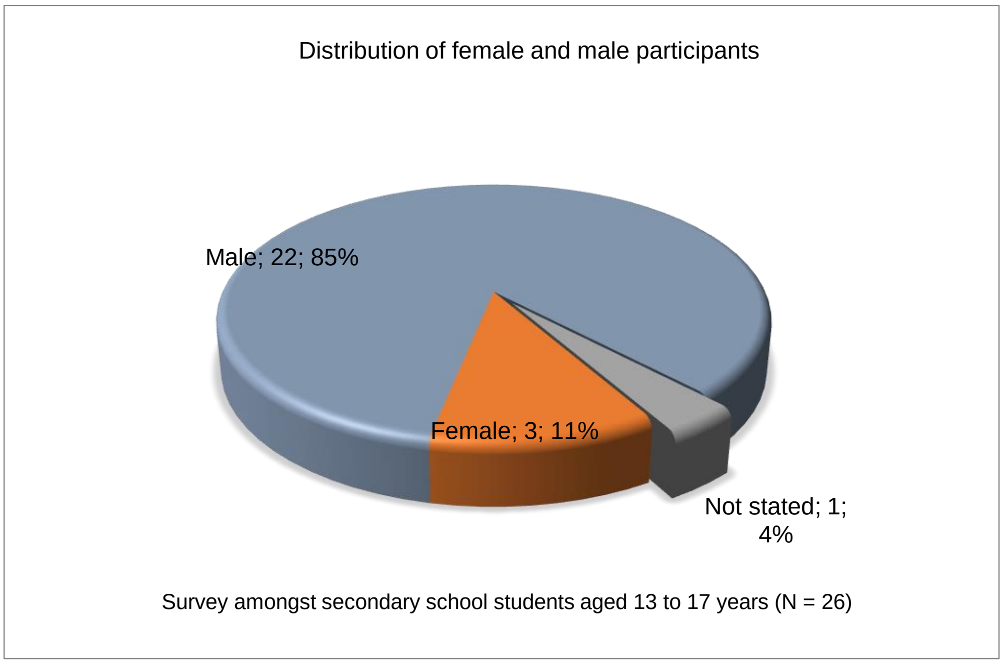


Figure 45 – Distribution of female and male participants involved in GamePinS year 1 (N = 26)

Internet Access

Question text: *Ich habe zu Hause einen Internetzugang.*
Translation: *I have Internet access at home.*
Possible answers: *"Yes"; "No"*

As shown in Figure 46, the question regarding Internet access showed that most students (i.e., 24 participants) had Internet access. There were no negative answers, but two students did not state anything in response to this question.

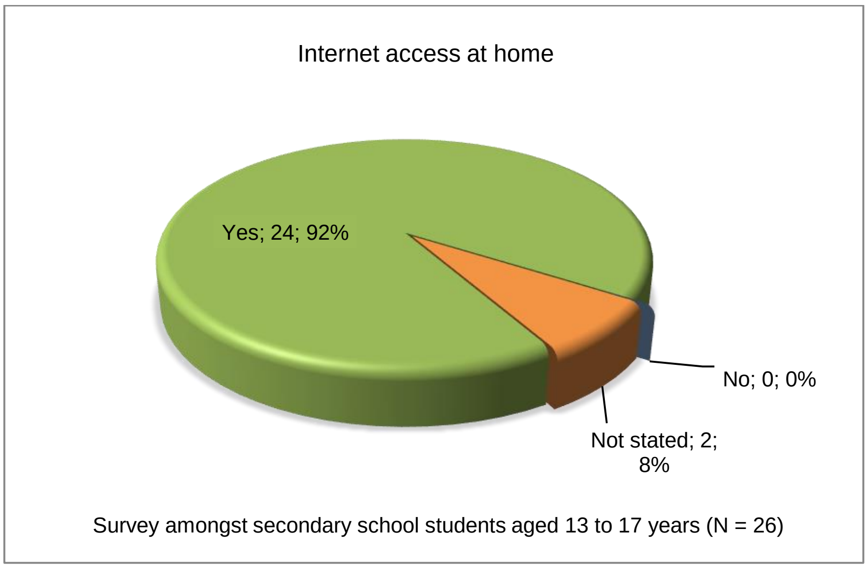


Figure 46 – Internet access at home (N = 26)

Age

Question text: *Alter*
Translation: *Age*
Possible answers: *“younger”; “13”; “14”; “15”; “16”; “17”; “older”*

As shown in Figure 47, the student ages ranged from 13 to 16 years old, which might be somewhat surprising since the survey targeted one cohort, i.e., the fifth grade of AHS-secondary school (i.e., which equals the ninth level counted from the first grade in primary school).[50]

Students are typically 14 years old when they enter the fifth grade, turning 15 at some point during the academic year. Younger students (i.e., 13-year-olds) started primary school at the age of five instead of six, in most cases because they were born in autumn and were ready for school (BMUKK, n.d.-a) as determined by the headmaster of primary schools. The students aged 15 and 16 repeated any grade once or twice during their school-career.[51]

[50] For more information about the Austrian school system, see (BMUKK, n.d.-c) and (BMUKK, n.d.-b).

[51] In the sample, there was also one 17-year-old student, but he did not fill out the survey due to his being absent from class that day.

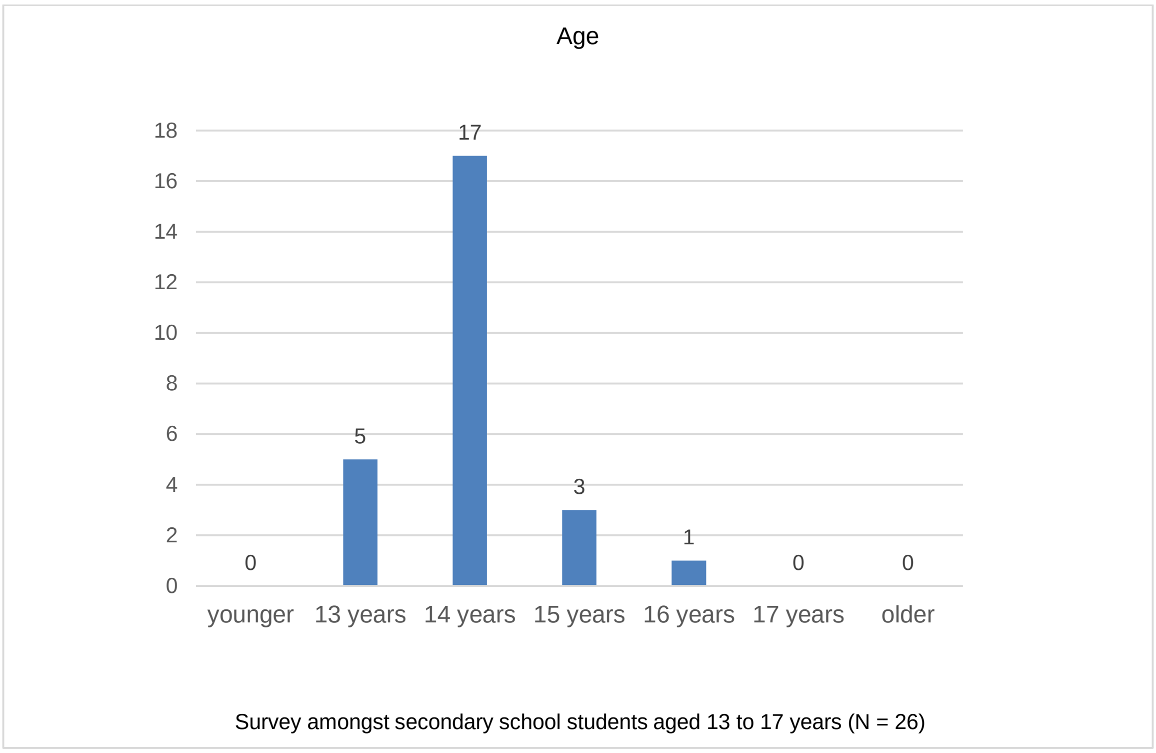


Figure 47 – Age distribution of students participating in GamePinS

5.2.3.2 *Activities with the personal computer or smartphone and IT skills*

Question text: *Meine Computerkenntnisse bewegen sich auf folgendem Niveau*
Translation: *My computer-skills match the following level*
Possible answers: *“Beginner”; “Intermediate”; “Adept”; “Power user”; “Expert”*

As shown in Figure 48, there were no students who classified themselves as a “Beginner.” Interestingly, most students (15 out of 26) counted themselves as “Intermediate” users, raising the question as to why students saw themselves at a lower level than I originally expected[52].

Since there can be many reasons (e.g., recent experiences with complicated software, a weak classification system, low self-esteem) why one sees him or herself in the middle of the skill scale, students were asked to discuss the meaning of the classification in groups, presenting their results later. This approach should enforce anonymity as compared to

[52] The expectation here was “Adept.”

asking single students for their self-classification, then asking them for their reasons. Results showed that students agreed that adepts, power users, and experts had to be able to use every software package that they worked with "fluently" (i.e., much like speaking a language fluently) without the help of the Internet, reading manuals, or trial-and-error techniques.

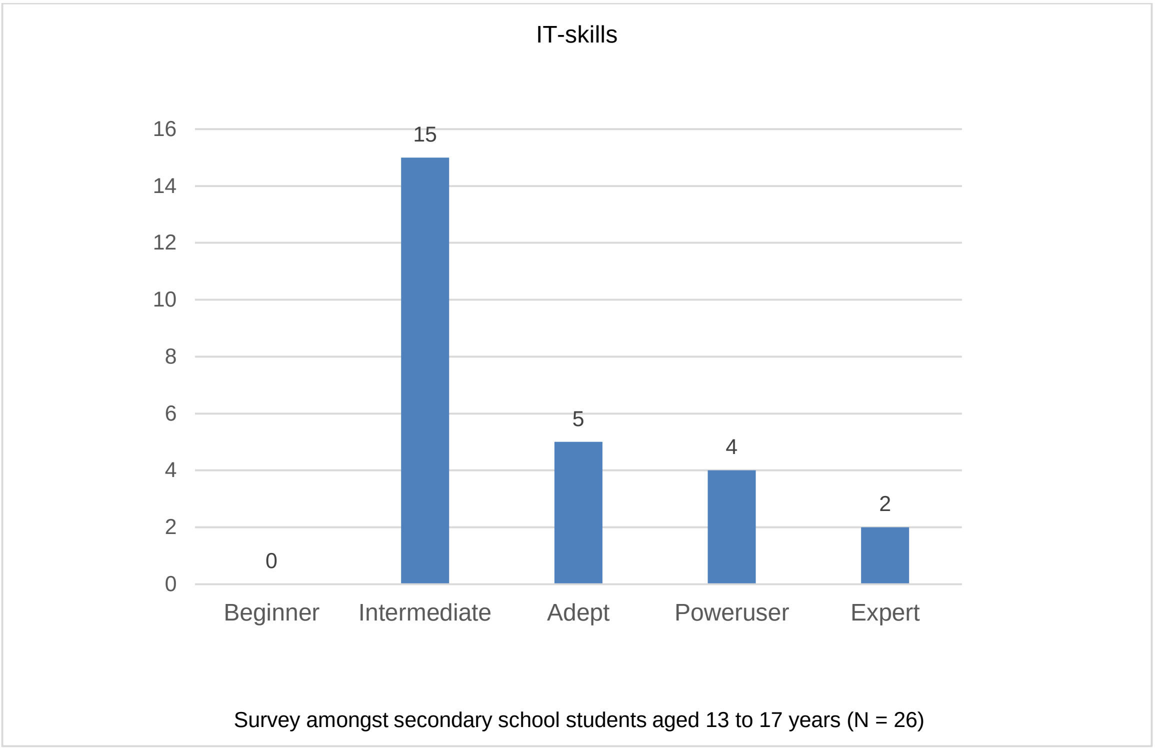


Figure 48 – IT skills of students involved in GamePinS

Remarkably, although students agreed to define "Adept" and above as "operating a program fluently," there was also a consensus that even experts cannot know everything in the software world and are allowed to "Google" information without losing their expert status.

5.2.3.3 *Computer usage: time spent in front of computer*

Question text: *Wie viele Stunden verbringst du pro Woche vor dem Computer?*
Translation: *How many hours do you spent in front of a computer per week?*
Additional remark: *Only the time when not in school*
Possible answers: "up to 5h"; "5+ to 10h"; "10+ to 15h"; "15+ to 20h"; "20h+"

As shown in Figure 49, most students (i.e., 85%) involved in the GamePinS project spent more than five hours per week in front of a computer. The remark *"Only the time when not in school"* should ensure that only the students' spare time was counted.

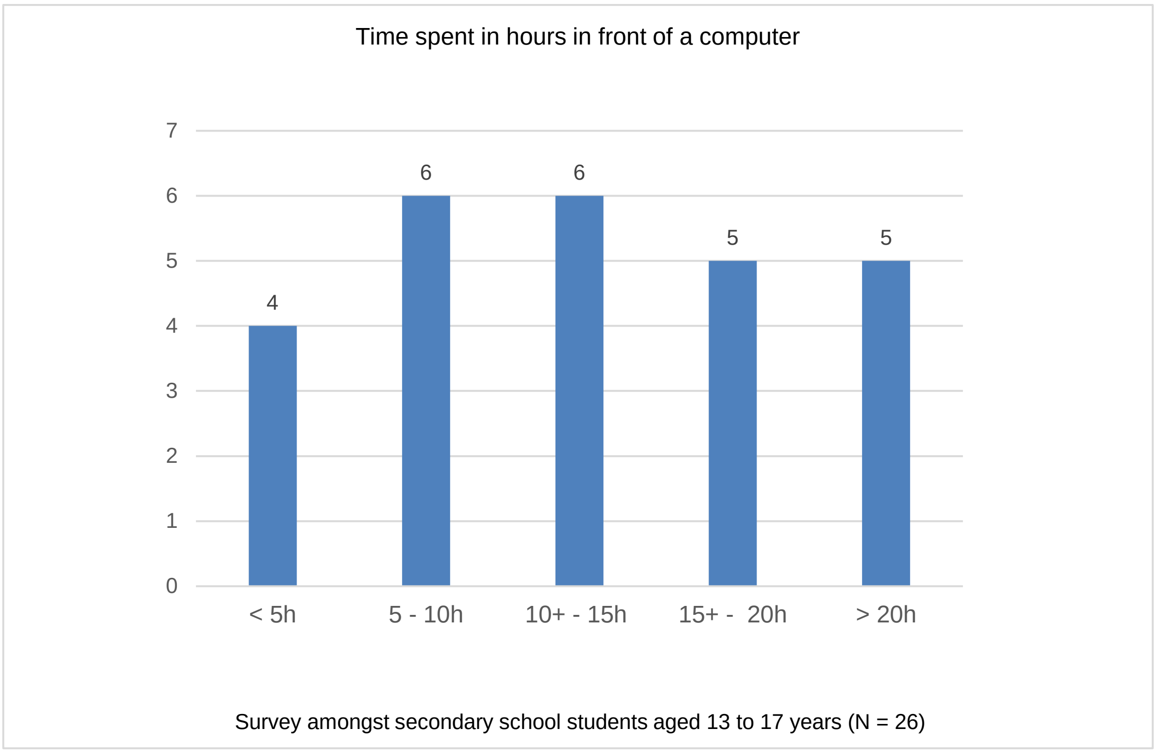


Figure 49 – Time spent (hours per week) using a computer

When the same students were asked how much of the time spent in front of the computer was for "activities for school," three students answered "not at all" and 14 answered "less than 2 hours"[53] (Comber, 2012c). A detailed comparison, including four common activities, is shown in Figure 50.

[53] Note that 10 students answered "2 to 6 hours" and one answered "6 to 10 hours."

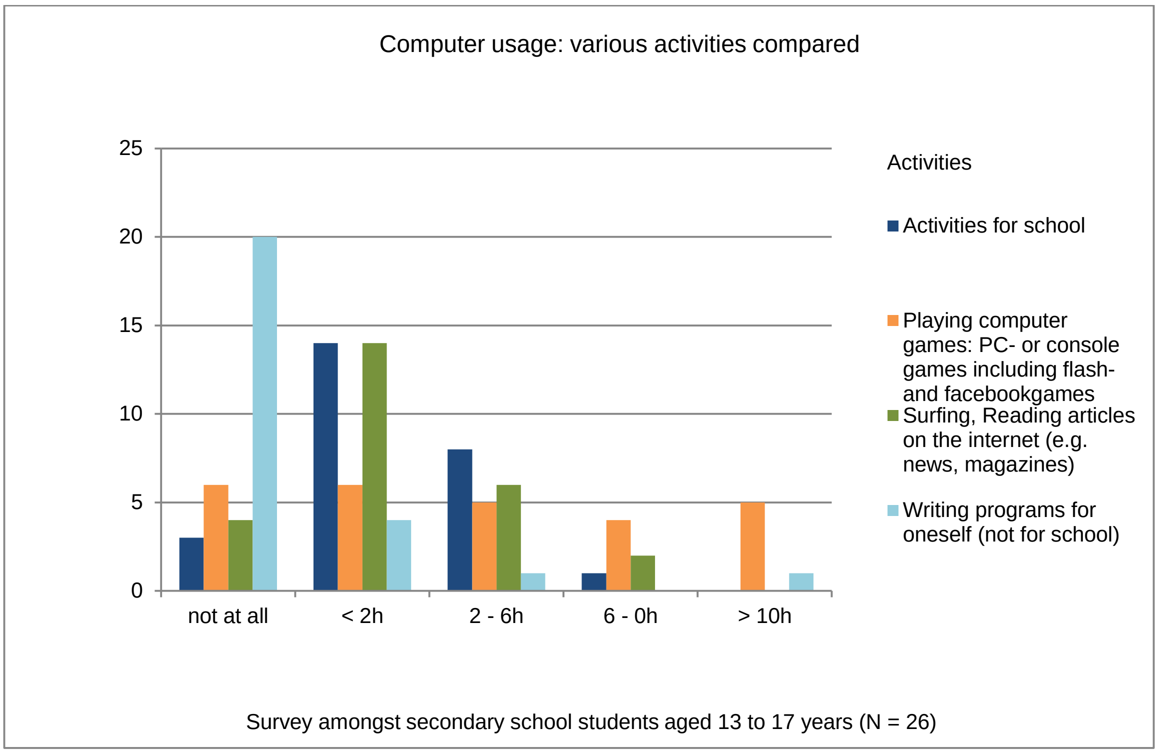


Figure 50 – Time spent for different activities in front of a computer

Eleven students spent less than two hours per week of their time communicating on the computer, nine students spent from two to six hours, four students spent six to ten hours, and two students spent more than ten hours (Comber, 2012c).

The question regarding other activities with the computer revealed the following results:

- not at all: 5
- <2 h: 11
- 2–6 h: 7
- 6–10 h: 1
- >10 h: 1
- no answer: 1

If participants checked "other activities," they were asked to describe these other activities. Clustering was performed on the given answers, with results summarized in Figure 51.

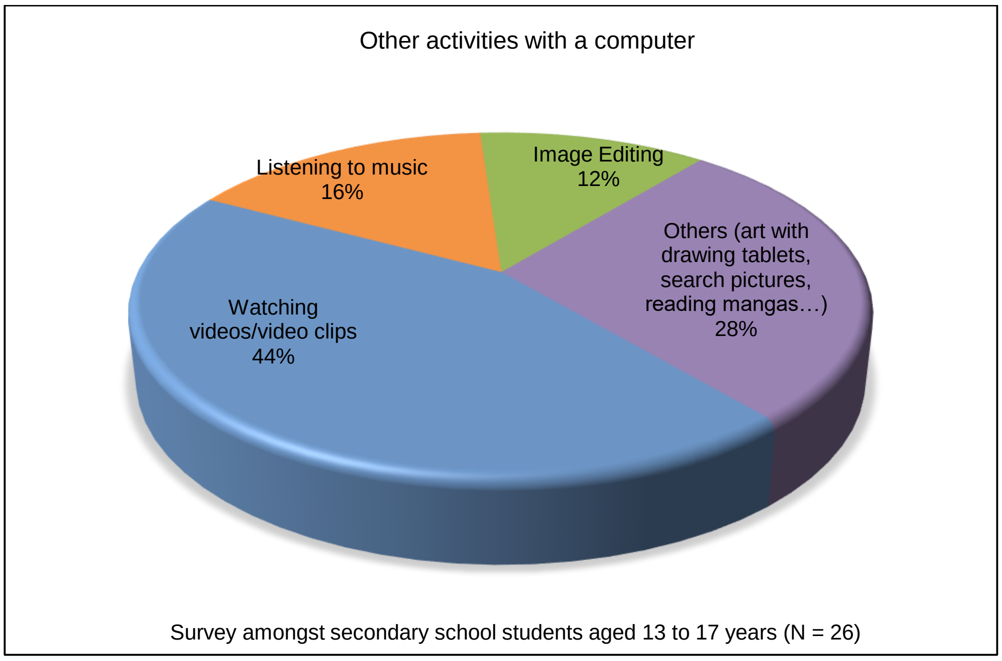


Figure 51 – Other activities in front of the computer

5.2.3.4 *Game development and IT Skills*

The possible answers for the following items were: *"completely agree"; "agree"; "neutral"; "disagree"; and "completely disagree."*

Question text: *Durch die Spieleprogrammierung habe ich gelernt besser zu programmieren!*
Translation: *Through game programming, I've improved at writing programs.*

This item specifically targeted skills for writing programs; the results are shown in Figure 52. On the one hand, many students perceived an improvement in their programming skills, but seven students felt their programming skills decreased. At first glance, that seems to be a paradox, but this perceived decrease is not a paradox, but rather a result of a sharper self-awareness. Programming skills did not actually decrease, but the knowledge of what students did not know before increased, and this fact led to a seemingly paradoxical result here.

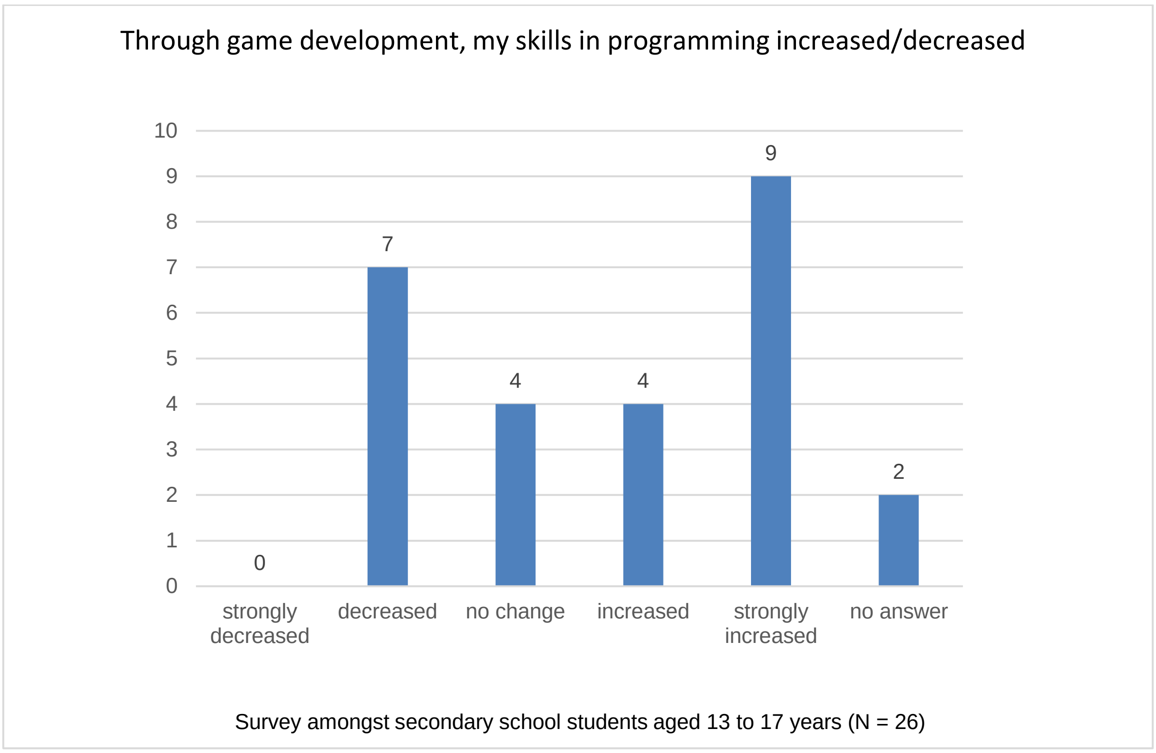


Figure 52 – Perceived improvement in programming skills

Question text: *Durch die Spieleprogrammierung verstehe ich besser wie Programme ablaufen!* (IT2)
Translation: Through game programming, I better understand how programs work.

As shown in Figure 53, similar results were evident for this question, where the logic of programs was tested. As with writing programs, a greater knowledge of how programs work went hand in hand with an enhanced knowledge of all the things the students did not know about this topic.

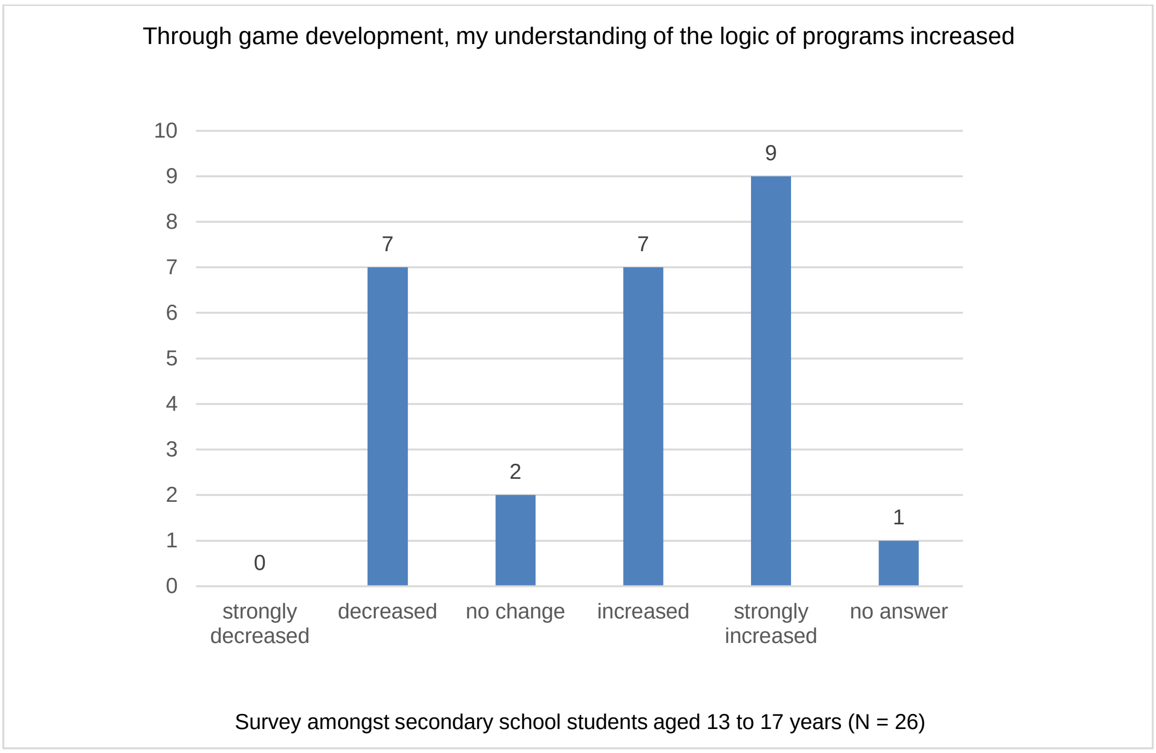


Figure 53 – Changes in the understanding of the logic of programs

5.2.3.5 Algorithms and software development

Items (IT1-3) addressed the understanding of algorithms and software development. Again the categories here were: *"completely agree"; "agree"; "neutral"; "disagree";* and *"completely disagree."* Figure 54 summarizes the results.

Question text: *Durch die Spieleprogrammierung verstehe ich besser was ein Algorithmus ist!* (IT1)
Translation: *Through game programming, I better understand what an algorithm does.*
Question text: *Durch die Spieleprogrammierung verstehe ich besser wie Softwareprojekte ablaufen!* (IT2)
Translation: Through game programming, I better understand how software development projects work.
Question text: *Durch die Spieleprogrammierung welche Probleme beim Entwickeln von Software auftreten!* (IT3)
Translation: *Through game programming, I better understand the problems and challenges that occur during the development process.*

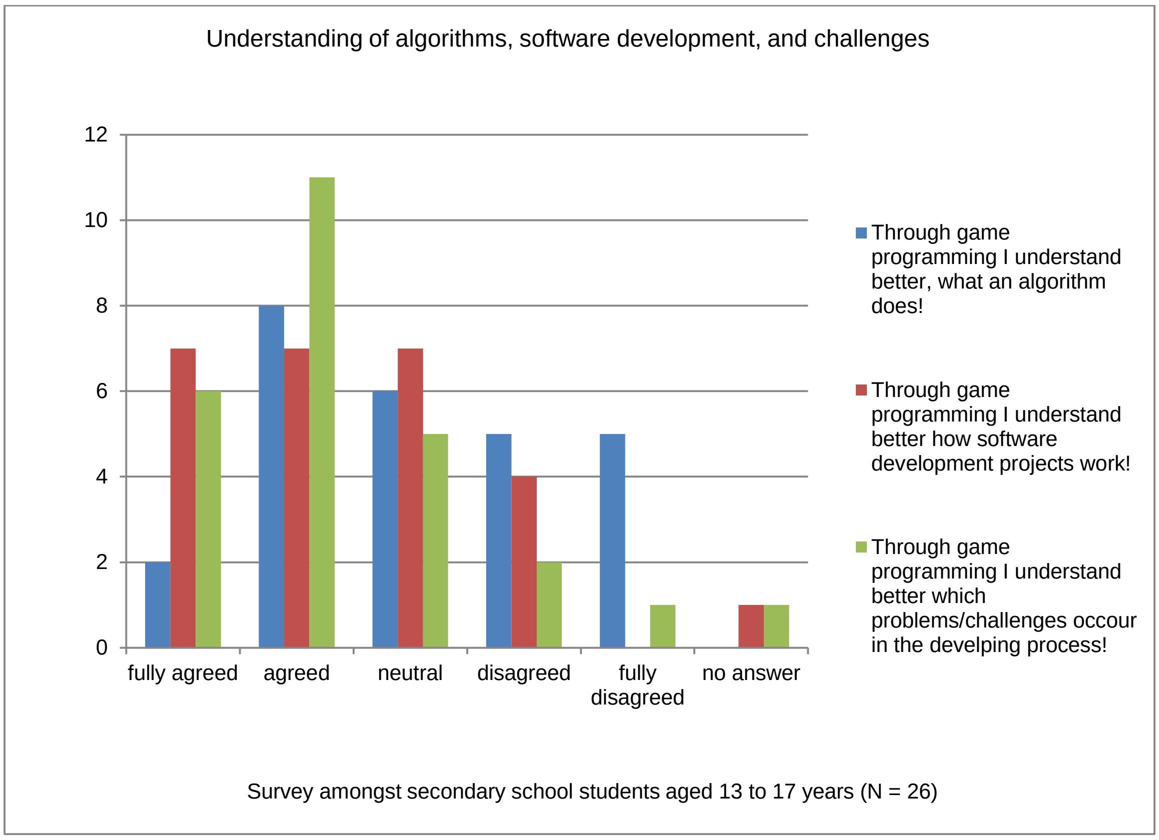


Figure 54 – Understanding of algorithms and software development

5.2.3.6 *Engagement*

(A1) Game programming in computer science increases the engagement of students.

While there are many studies focused on engagement in computer games—e.g., Boyle et al. conducted a meta-study that covered engagement in *playing* digital entertainment games (Boyle, Connolly, Hainey, & Boyle, 2012)—there have been far fewer reported studies on engagement in the development of computer games by students, and there is a scientifically unexplored area in the scope of secondary school computer science education aimed at fostering the understanding of programming, modeling, and algorithms.

The attitude of students toward game programming seemed promising (see Figure 55), but did not meet student expectations following the completion of their game development projects (see Figure 56).

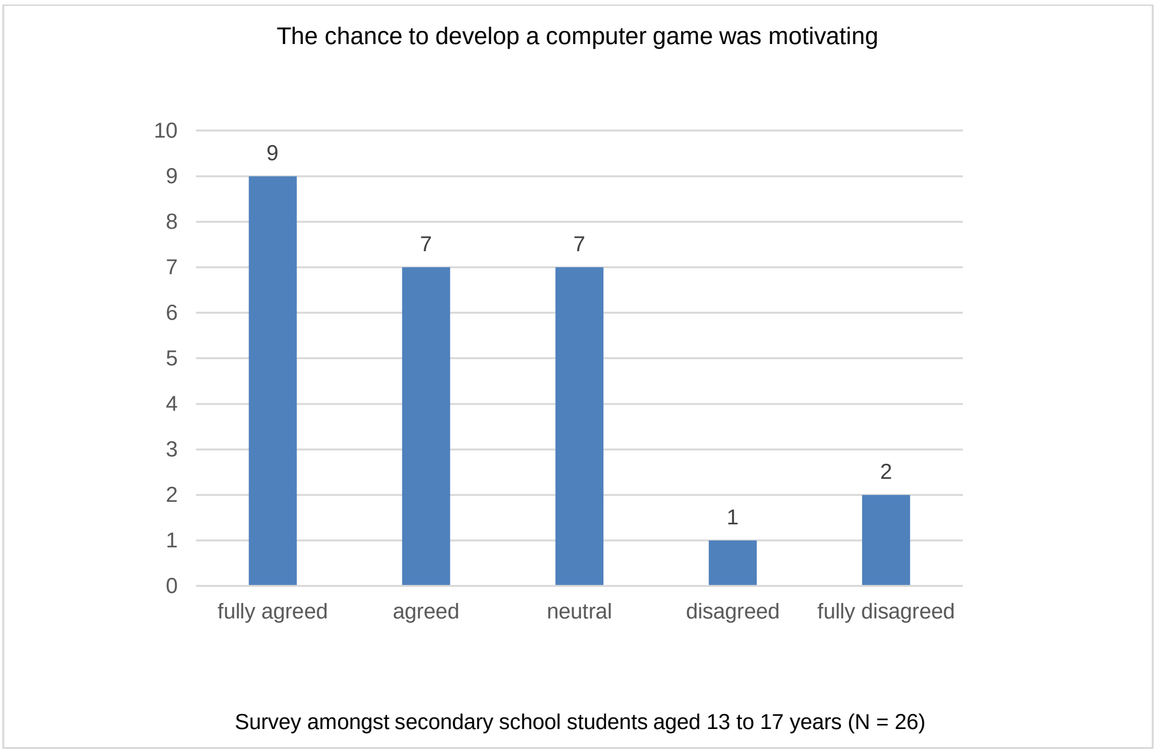


Figure 55 – Attitude of students to the development of a computer game in class

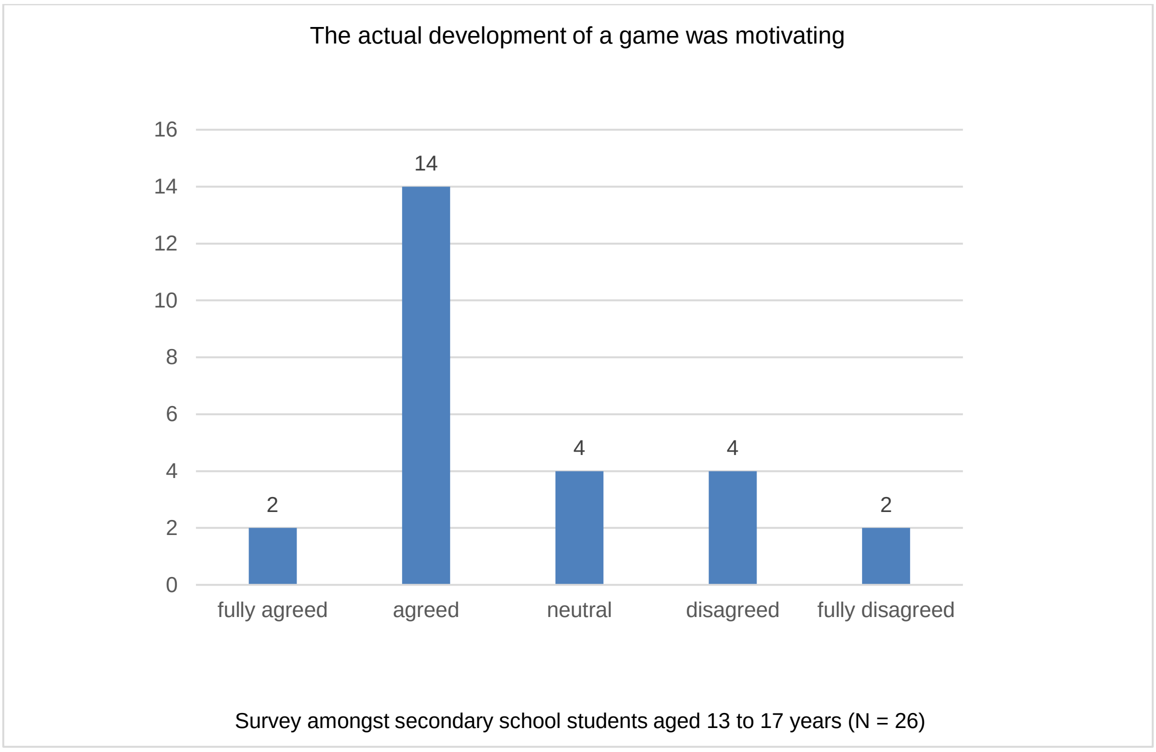


Figure 56 – Change of motivation of students through game development project

5.2.3.7 Teamwork

(A2) Project-based game programming in computer science fosters teamwork competencies.

Project-based learning can improve the teamwork skills of students (Huang, 2010). For our particular case, we investigated whether teamwork skills improved due to GamePinS activities. As shown in Figure 57, especially in secluded afternoon lessons, where there was a tendency among some students to miss class (which was criticized by other students[54]) and where some students did not appropriately participate in the projects[55], it seems interesting to determine if teamwork competencies can still be improved.

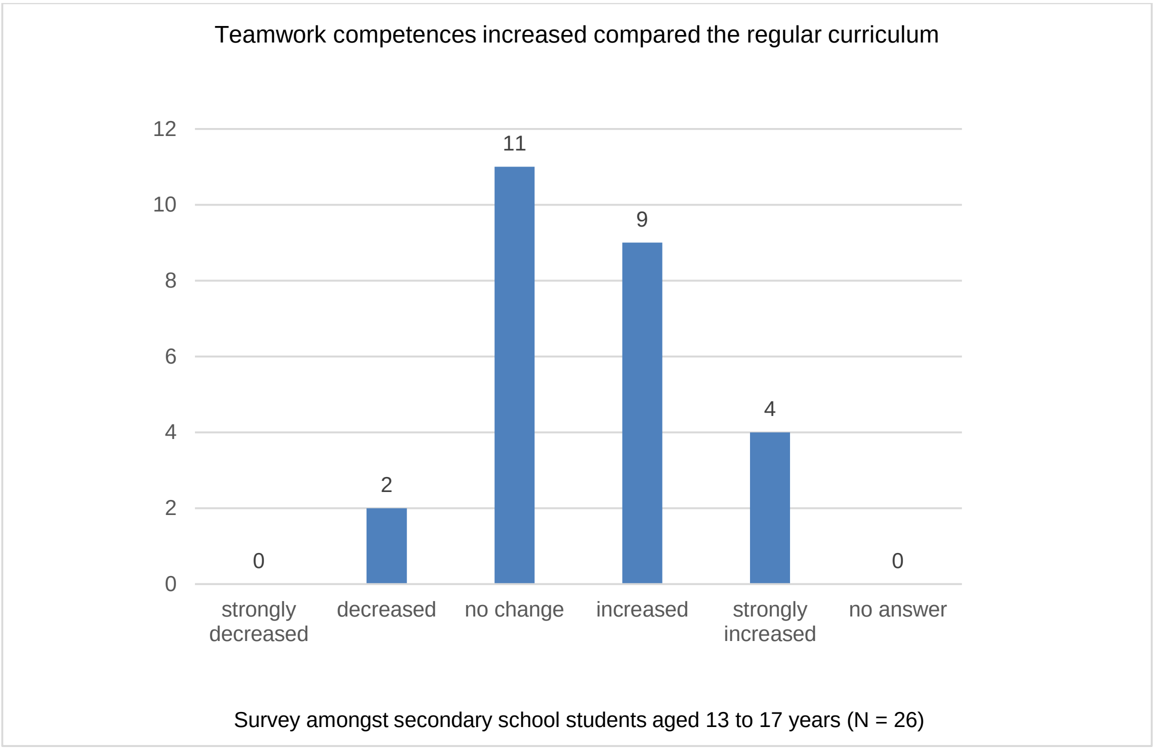


Figure 57 – Changes of teamwork skills through game development

[54] *"I did not like the fact that most students did not appear"; "Nearly no teamwork occurred, because the half [half of the pupils, O.C.] missed [the lessons in the afternoon, O.C.]"* (Comber, 2012a).

[55] "*The others mostly listened to music and did not contribute much to the work"* (Comber, 2012a).

5.2.3.8 Promoting basic concepts of computer science

(A3) Game programming promotes basic concepts of computer science more effectively than other typical software development scenarios.

After formulating assumption (A3), the first question was, *"what are typical software development scenarios?"* Certainly, not all possible alternative scenarios can be tested against game programming. Implementing an example scenario in the style of "Mister Smith owns a small company and he wants an employee to implement a sorting algorithm for customer orders by priority, which is represented by a whole number" seemed to produce too much bias, because such problems appeared boring to students. Therefore, it made sense to specify together with students similar software projects in terms of similar dimensions, workload, and structure.

How did the first round of game development take place? In the first weeks of the school year, students learned about the basics of image processing and programming in guided lessons as a group, i.e., every student listened to the same lecture and performed the same task, such as writing the aforementioned program called "The Matrix." Later, the first steps with C# and XNA were undertaken. In these lessons, loading and displaying our created images (i.e., created in GIMP), resizing, and moving them were covered.

While motivation remained high (Comber), the understanding of algorithms did not significantly change, as shown in Figure 58.

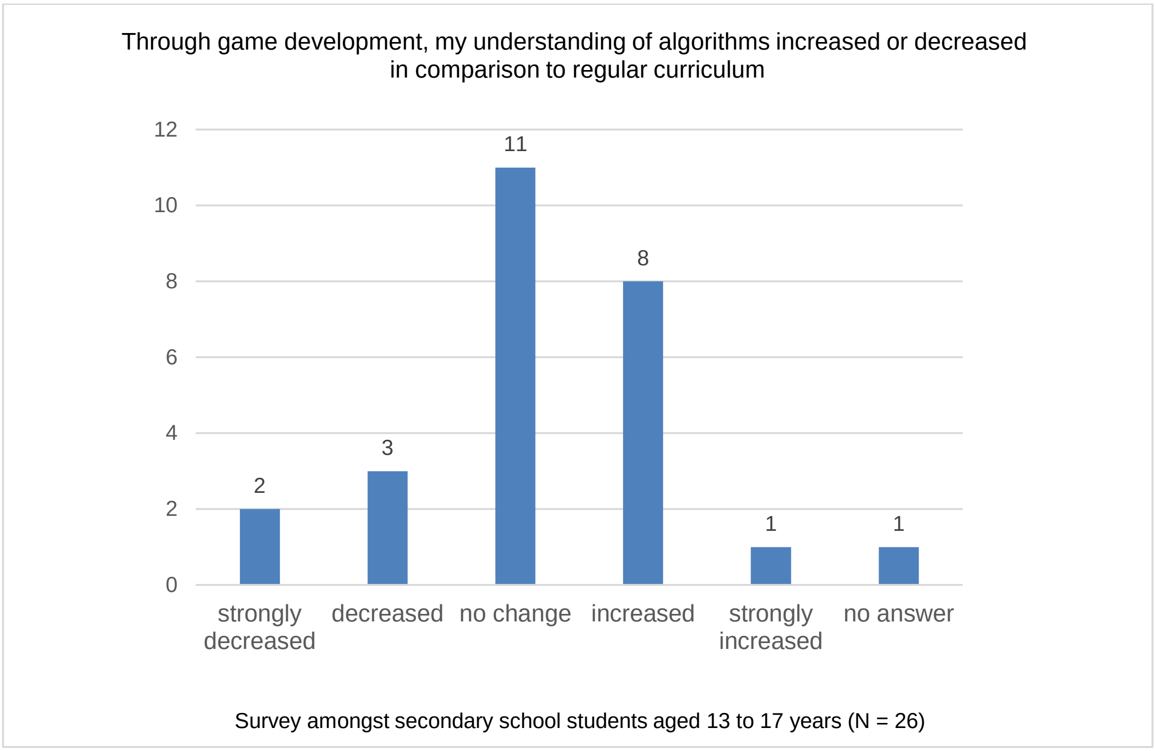


Figure 58 – Changes in the understanding of algorithms in comparison to regular curriculum

As shown in Figure 59, the understanding of the logic of programming increased, though which aspects students referred to here is a subject for further research. In verbal feedback, “object-oriented concepts” and “how to properly address properties and use methods of objects” was mentioned.

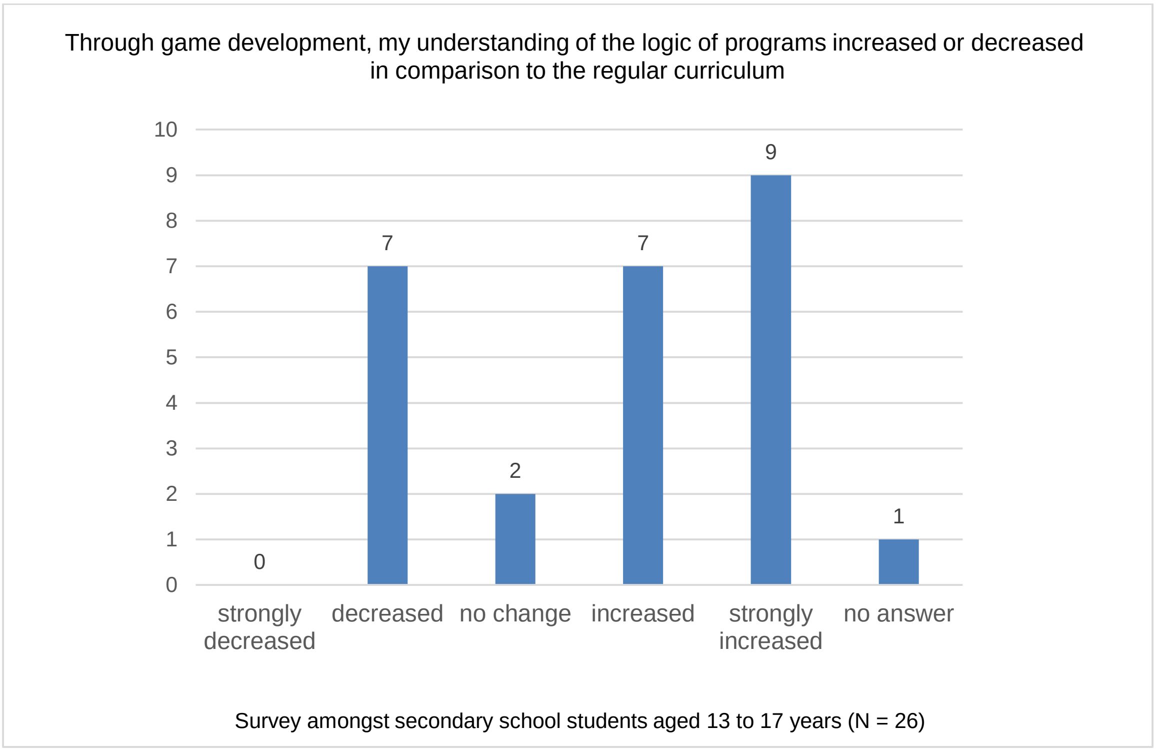


Figure 59 – Changes in the understanding of the logic of programs in comparison to the regular curriculum

5.2.3.9 Complexity of development framework

(A4) The complexity level of the framework used influences the depth of understanding of the concepts to be learned.

At first glance, it might seem apparent that simple frameworks, such as Scratch (Mitchel Resnick et. al.), Logo (Logo Foundation, 2011), AntMe ("Website - AntMe "), Greenfoot ("Website - Greenfoot,"), and KidsProgrammingLanguage (KPL) ("Website - KidsProgrammingLanguage KPL ") (now Phrogram (Phrogsoft, 2012)), are easier to learn and leave more time to attend to concepts of computer science, but on the other hand, the simplicity might hide pathways to deeper understanding. More complex frameworks, such as C# and XNA or JavaScript and the HTML 5 Canvas element, not only require a deeper understanding as to how to properly use them, but also offer more flexibility. The price for this is a steeper learning curve (p. 46). The positively stated assumption is tested to prevent double negation. Both results lead to helpful and interesting conclusions that can serve as a starting point for further investigation.

5.2.4 Summary of the first year

The first year of game programming was conducted using Microsoft Visual Studio Express 2010, the programming language C#, and XNA 4.0 Game Creators Studio. The challenge here was the complicated code necessary just to create a sprite that can then be moved around using the arrow keys. These circumstances caused the motivation in the actual programming to be lower than the anticipated motivation.

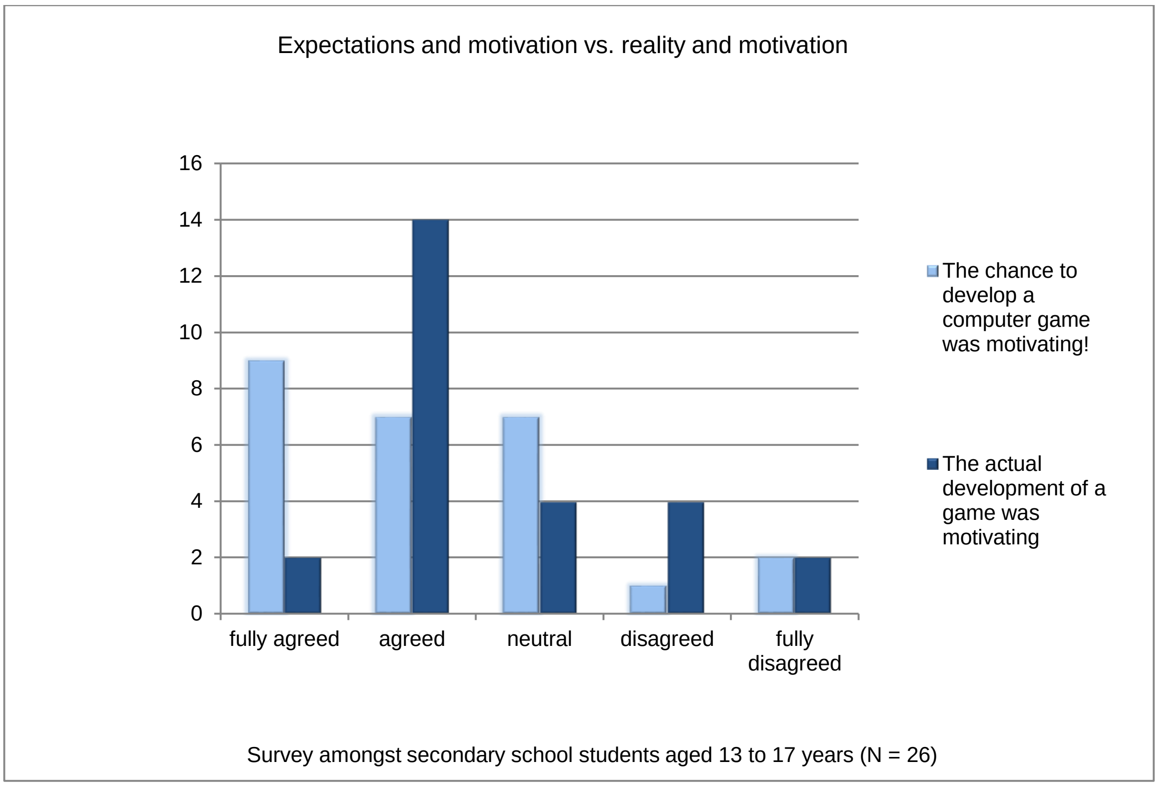


Figure 60 – Expectations and motivation versus actual programming in practice and motivation

As shown in Figure 60, from this unsatisfying situation and specific feedback from students, the seemingly high complexity had a negative impact on actual motivation, which led to the decision to simplify the framework by implementing another layer of methods that would allow the loading and controlling of sprites to be much easier. Thus, the GamePinS framework was born.

5.3 Second year of game programming

In the second year, the GamePinS framework was introduced to students. GamePinS and corresponding preparation activities started in January 2014 and ended in June 2014. From September 2013 through December 2013, other substitute teachers held non-game programming lessons, which varied from how to use Office applications to programming with Visual Basic.

5.3.1 Research design updated to include GamePinS versus other software projects

In the second year, not only was the GamePinS framework introduced, but also an evolution in the research design occurred. Changes were based on the following considerations. In the first year, students involved in game development were asked what differences they experienced between game development and the regular curriculum. This approach had the advantage that the same persons could provide data, and thus it was possible to avoid any bias that arises out of different group compositions. In contrast, there was no means to compare learning outcomes concerning IT skills, because students were going through the regular curriculum[56] in the first place, then were introduced to game development. Thus, every benefit that might stem from game development could also have originated from previously working on the other programming activities.

In recognition of the fact that it would be impossible to obtain useful information about the effect of the program on the students' IT skills if all of the students were in the same group, it was decided to form two groups to allow for a comparison to be made. Both groups learned the same basics in the same way and were only divided after acquiring such

[56] Such content included programming tutorials, assignments, and everyday problems in IT, i.e., console programming, input/output, calculations, randomizing, Windows programming, and projects chosen by students that ranged from an outfit advising program (e.g., answering "which part of clothing (e.g.,., a dress) matches other parts of clothing well (e.g.,., shoes, bag)?") to how to calculate the correct value for Alpine ski binding based on physical data and ski drivers skills.

basic skills into one GamePinS group doing game programming and another group focused on software development.

5.3.2 Getting results with small groups

Group A consisted of six participants, while group B had seven participants. Therefore, no elaborate statistical testing[57] was planned. The data (*see Section 5.3.3*) was instead used to identify tendencies and perform a triangulation together with written statements from students to obtain a detailed picture.

Feedback from the questionnaire and statements from students were very helpful, but to truly answer the question of whether game development can boost IT skills, an IT skills test was conducted with the students (*see Section 5.3.5*).

5.3.3 Quantitative data from the second year

In the following subsections, selected data is presented, with complete datasets available online for 2012 (Comber, 2012b) and 2014 (Comber, 2014).

5.3.3.1 Statistical data

In the second year, there were two groups of seven students that were supposed to participate in the evaluation, but not all were present at the final questionnaire session. The GamePinS group (i.e., group A) was represented by six students, while the Software Projects Development group (i.e., group B) was represented by seven students.

In the GamePinS group, there were six participants between 14 and 15 years old and one 17-year-old student (five male, one not stated) in the survey and the IT skills test. In the other software projects group, seven students participated, with six between 14 and 15 years old and one 13-year-old student (three males, two females, and two not stated) in the survey and IT skills test. All students in both groups stated that they had Internet access at home – see the figures online (Comber, 2014).

[57] As described in numerous statistical manuals, e.g. (Field, 2009).

5.3.3.2 *IT skills self-estimation*

As shown in Figure 61, the IT skills self-estimation results were quite evenly distributed, with most students estimating themselves as intermediate or adept.

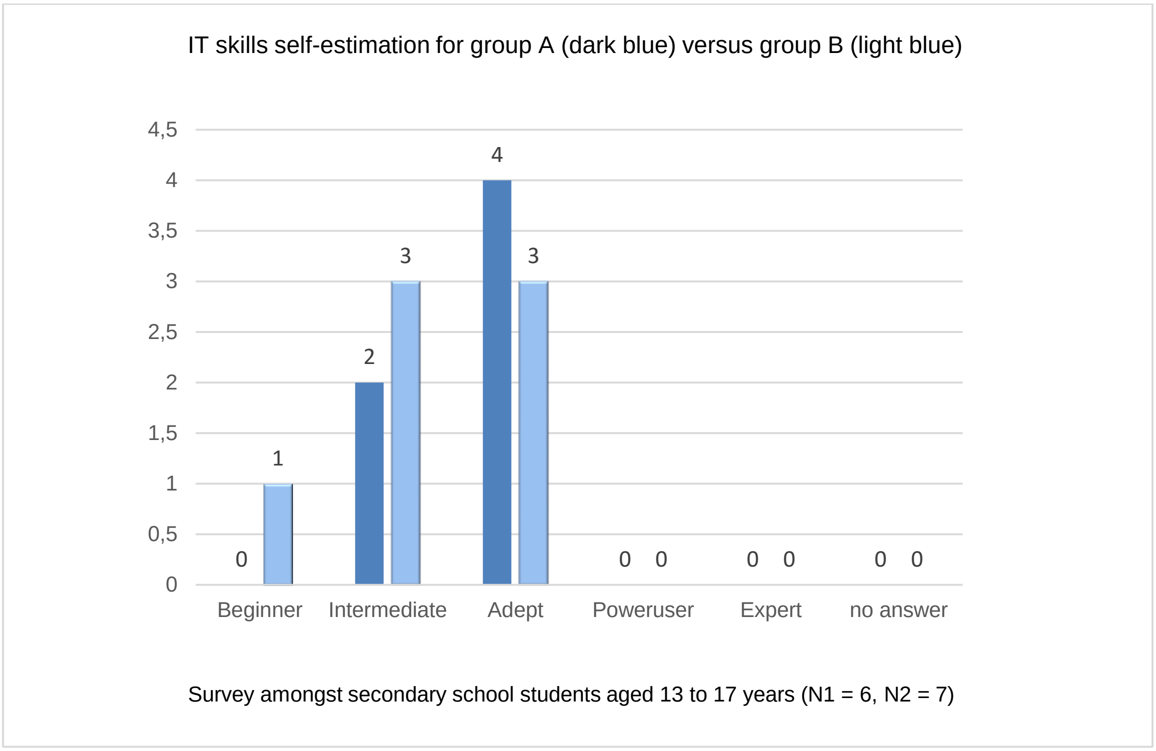


Figure 61 - Distribution of computer IT skills for groups A and B (2014)

5.3.3.3 *Activities and computer usage*

Regarding time spent in front of the computer, with answers of *“communicating via social networks, chats, discussion boards, e-mail,” “playing computer games via PC or console games, including Flash, and Facebook games,” “activities for school,”* and *“surfing, reading articles on the Internet (e.g., news, magazines),”* the results did not reveal any significant deviations from the larger sample of all collected attitudes.

5.3.4 Comparing the GamePinS and software development groups

As shown in Figure 62, improvements to understanding “how programming works” were measured via self-estimation and showed similar results for programming and other software projects.

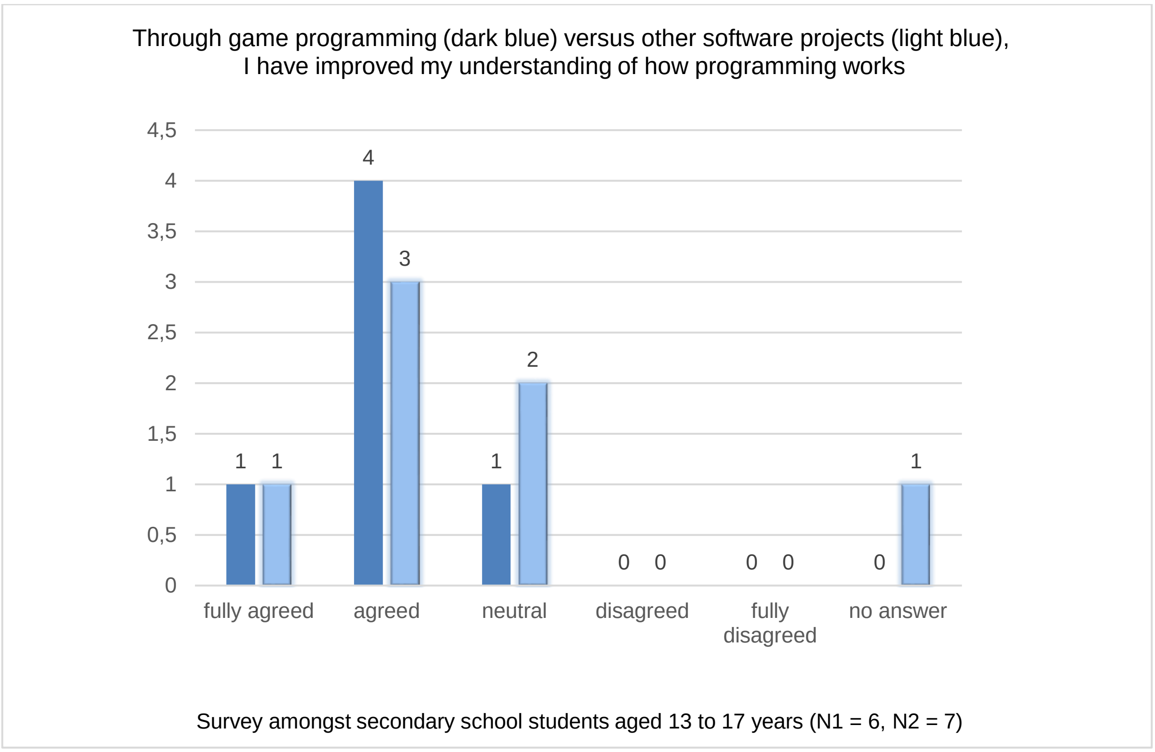


Figure 62 - Improvements to understanding how programs work

5.3.4.1 Actual changes in writing programs

As shown in Figure 63, actual improvements in writing programs were expected to be in correlation with the improvements of understanding programs (i.e., Figure 62), but they were not. The GamePinS group perceived a significant improvement, whereas the other project group that recreated the “2048” puzzle based on a Windows forms project experienced a decrease in actual programming skills.

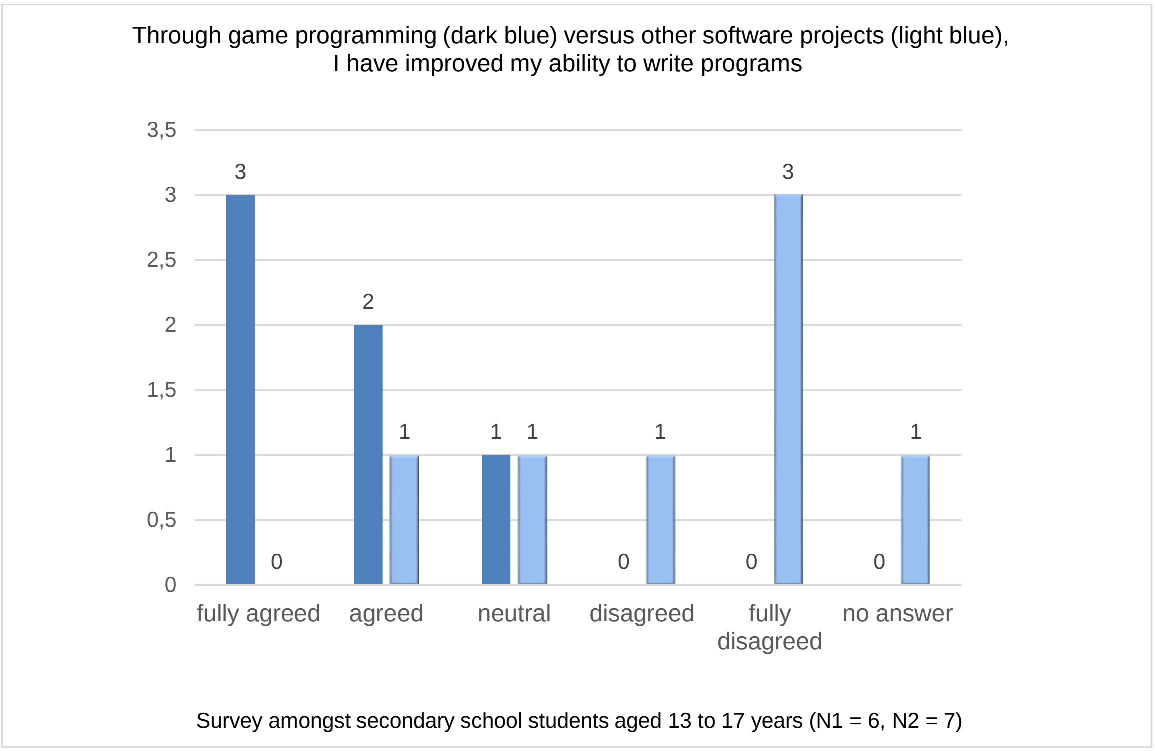


Figure 63 – Actual perceived changes of programming skills

At first glance, an explanation here might be that students from the second group knew more about the things they actually did not know after programming (i.e., recognizing how much more there is to learn). This would also be true for the GamePinS group, because knowledge also increased in this group, but this was not the case. A straightforward explanation for the fact that students working on the other software projects experienced a decrease in understanding of programing skills is that they did not perform as efficiently as they expected to. This might be caused by several reasons, one being that game programming was more interesting and students did not give up so easily given a variety of challenges. For this theory, though, there is only weak support because the motivation in the GamePinS group was only slightly higher, as shown in Figure 64.

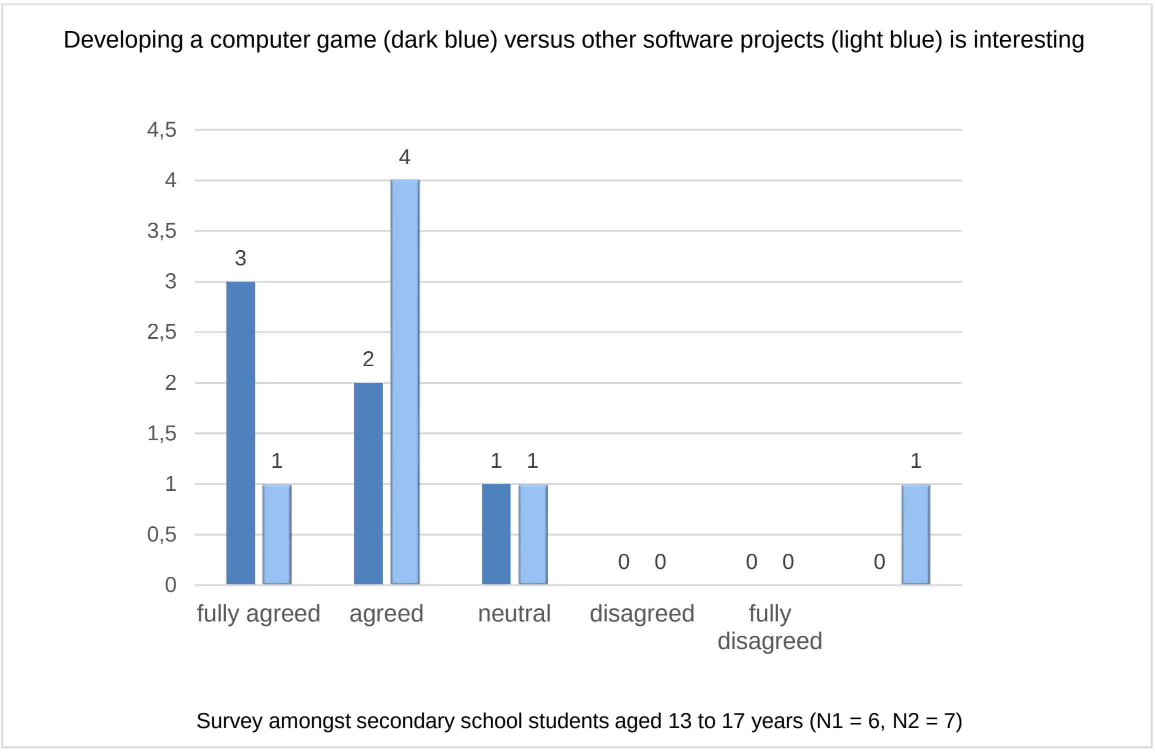


Figure 64 – Results showing how interesting game development was for the GamePinS group versus that experienced by the other software projects group

Another possibility here is that without a supporting structure or framework, jumping into the Windows forms programming (even with a thoughtfully chosen project) is too complex and difficult.

Actual changes in writing programs were measured via self-estimation, but this self-estimation showed a correlation with the IT skills test in which the students from the GamePinS group also performed significantly better than the other software projects group (see Figure 68 and Figure 74); however, the gap in the improvement in programming is very interesting, and it was decided that the results from year two would be cross-checked with the results from year three (once they were obtained).

5.3.4.2 Understanding of algorithms

As shown in Figure 65, students from the GamePinS group perceived a slightly better improvement in the understanding of algorithms than students in the *other software projects* group.

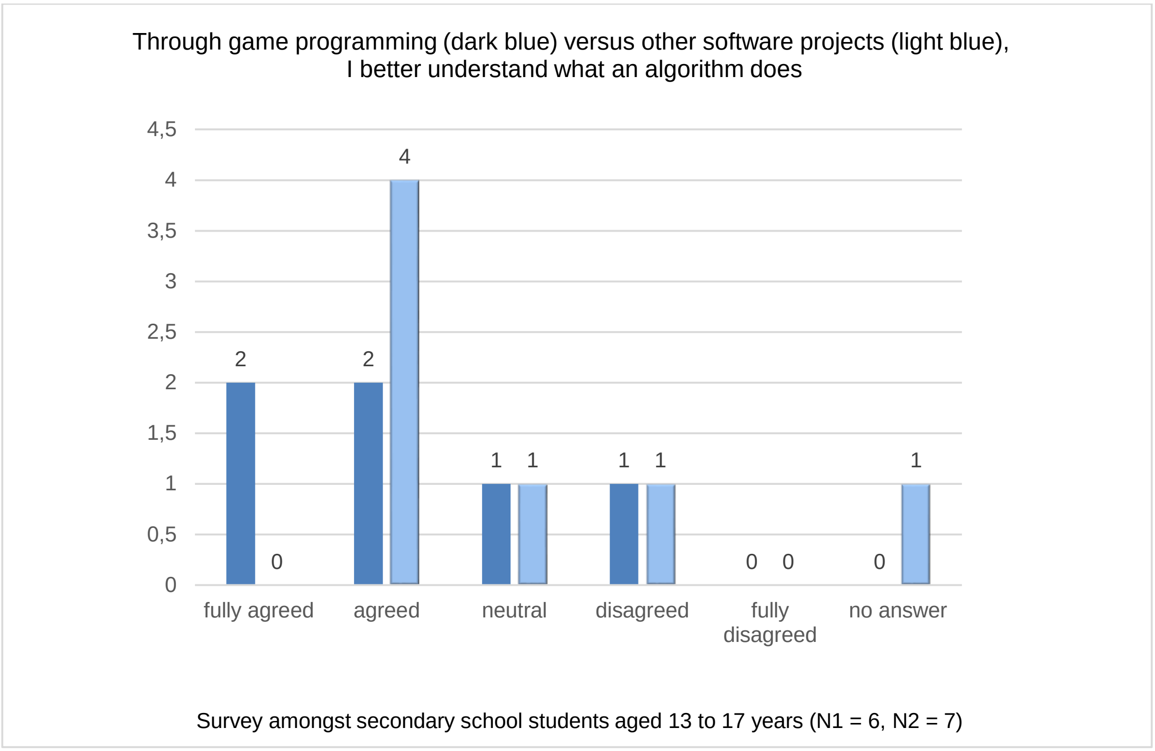


Figure 65 – Results regarding student understanding of algorithms

5.3.4.3 *Understanding software development and challenges in the development process*

Students from both groups, i.e., the GamePinS group and the group working on other software projects, stated that their understanding of how software development projects work increased, but no significant difference between the groups was identified, as shown in Figure 66.

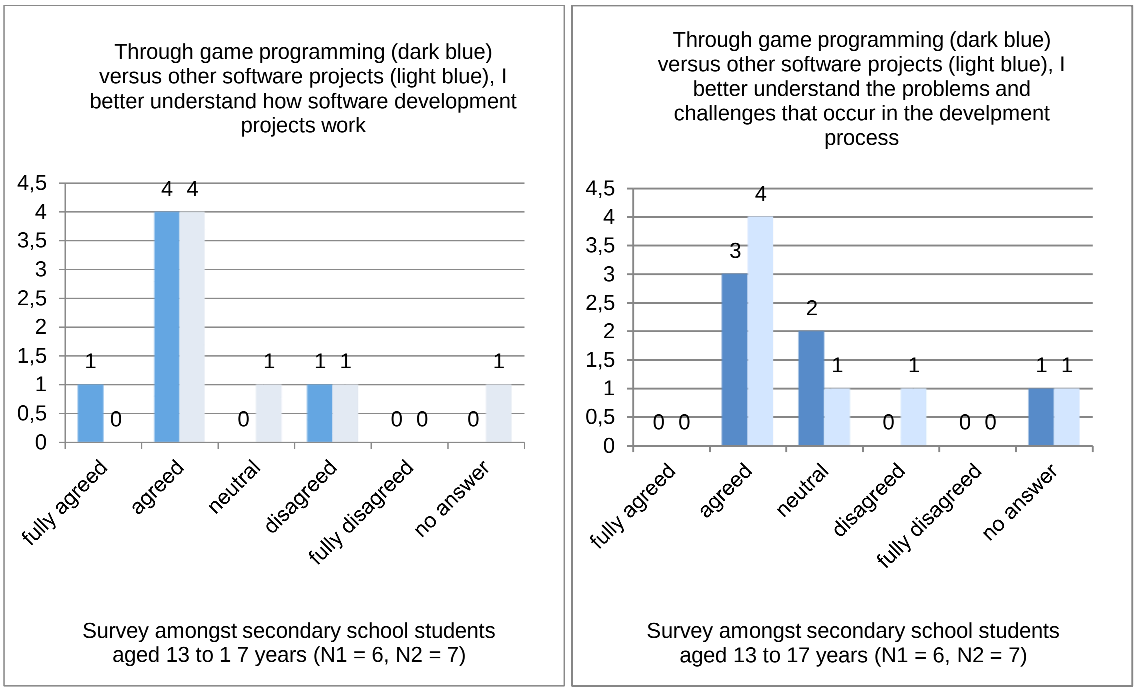


Figure 66 – Results regarding student understanding of software development projects and the challenges of the development process

5.3.4.4 *Teamwork*

As shown in Figure 67, teamwork skills increased in both groups. Compared to the *regular curriculum*, game development showed better teamwork performance, as expected.

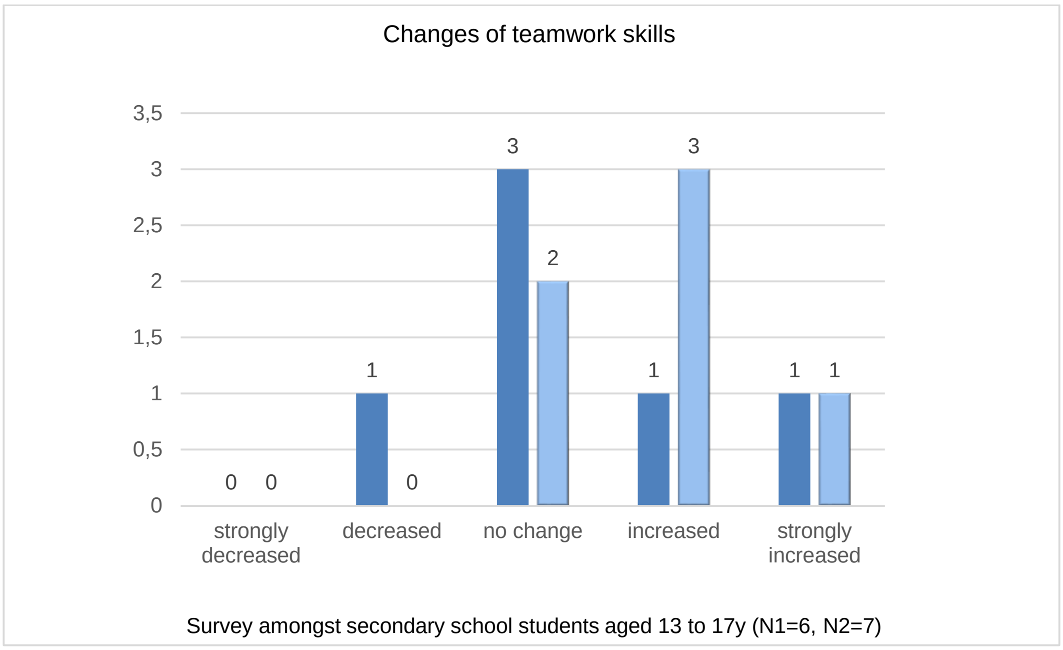


Figure 67 – Changes in teamwork skills

Comparing GamePinS and the other software projects group, teamwork improved more with the latter group than with GamePins. Better teamwork is clearly an attribute of how the learning was organized, namely project-based learning, and is not a specific characteristic of game development.

5.3.5 Achievements in the IT skills test

The IT skills test was implemented as a test in Moodle with the help of the Moodle Quiz tool (GAMING, 2005). The test relied on C#, since this was the language used to implement all game programming for group A and all software development project work for group B. Each quiz item had a certain amount of points assigned, with the total amount of points set to 34.

As shown in Figure 68, the overall IT skills test results average was 25.47 points for the *GamePinS* group and 10.86 points for the *other software projects* group. Details of these results are discussed in Section 5.3.5.1.

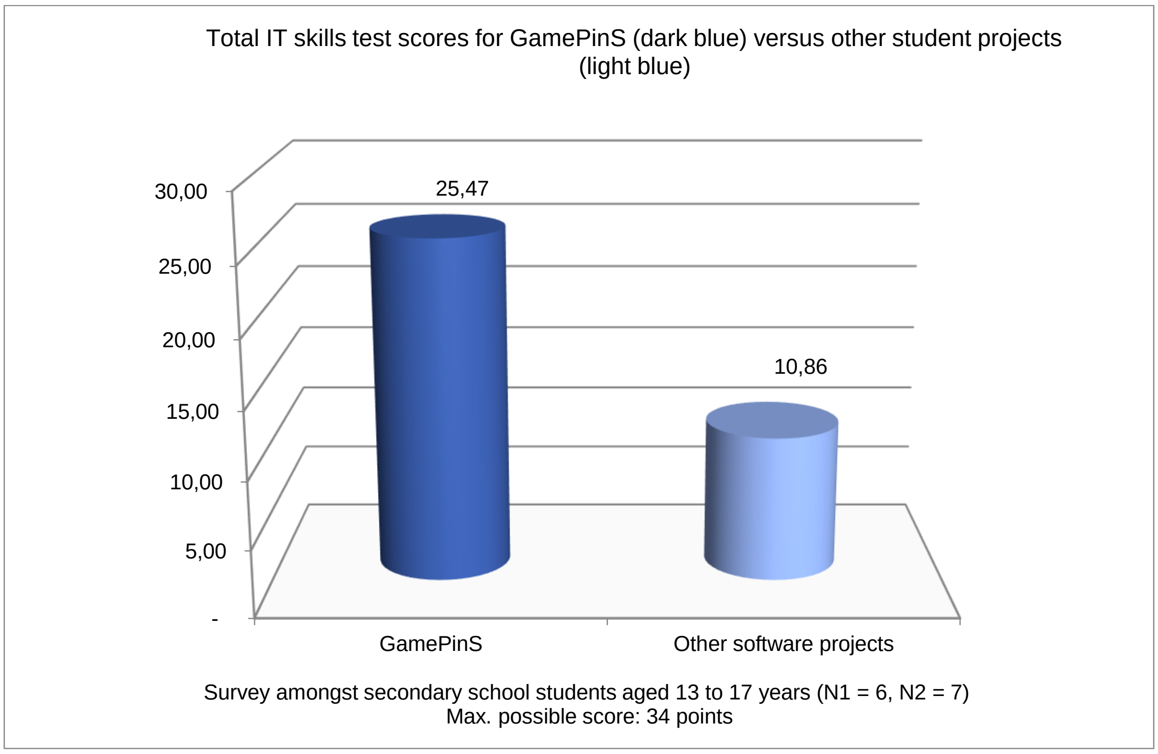
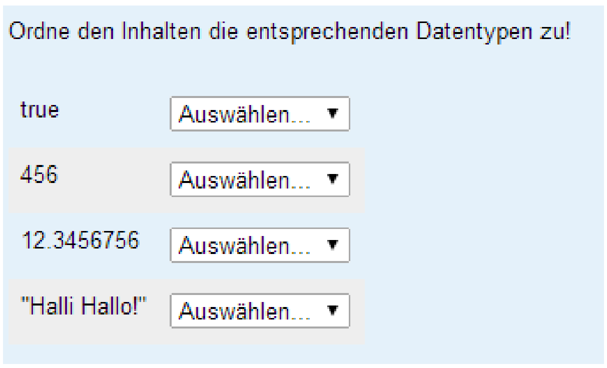


Figure 68 – Total IT skills test scores for the GamePinS and other software projects groups

Q1 – Data types (4 points)

Shown in Figure 69, Question 1 tested basic knowledge of data types. The exercise here was to assign the appropriate data type to the corresponding values.



Ordne den Inhalten die entsprechenden Datentypen zu!

true	Auswählen... ▾
456	Auswählen... ▾
12.3456756	Auswählen... ▾
"Halli Hallo!"	Auswählen... ▾

Figure 69 – Match the data types to the corresponding values

(Translation) Assign the corresponding data types to the following values.

Feedback after the test was finished included the correct and incorrect answers, as well as information regarding the correct answer.

The correct answers are: true ↔ bool

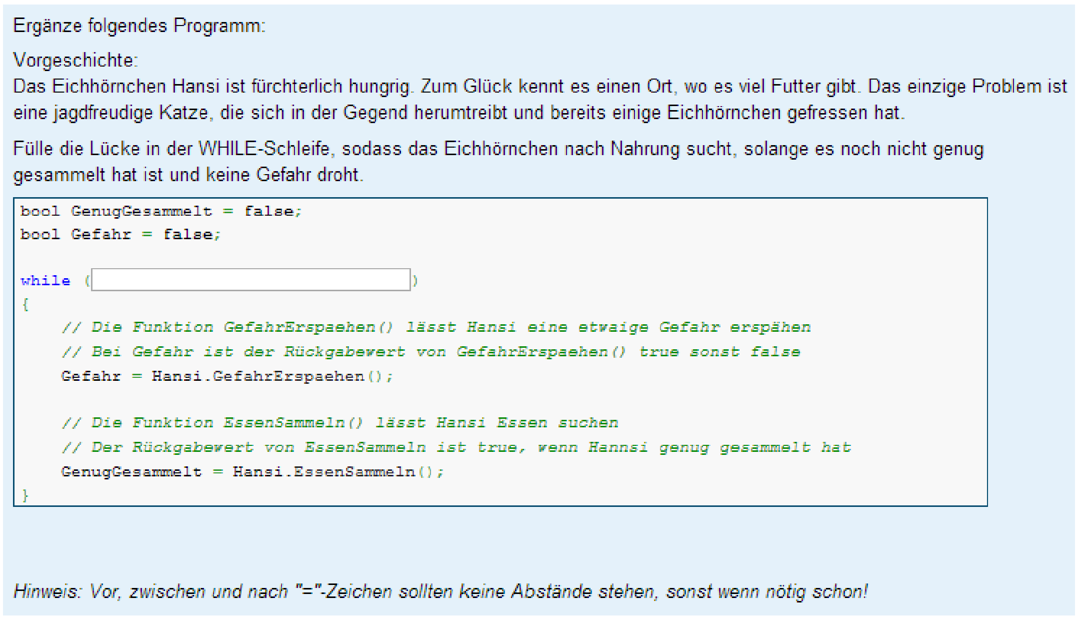
456 ↔ int

12.3456756 ↔ double

“Halli Hallo!” ↔ string

Q2 – Logical conditions (8 points)

Shown in Figure 70, this question determines if students have learned to think logically and transform their logical solutions into correct source code. The task was to formulate a logical condition, which was embedded in a while loop, under which the squirrel continues to search for food.


Ergänze folgendes Programm:

Vorgeschichte:
Das Eichhörnchen Hansi ist fürchterlich hungrig. Zum Glück kennt es einen Ort, wo es viel Futter gibt. Das einzige Problem ist eine jagdfreudige Katze, die sich in der Gegend herumtreibt und bereits einige Eichhörnchen gefressen hat.

Fülle die Lücke in der WHILE-Schleife, sodass das Eichhörnchen nach Nahrung sucht, solange es noch nicht genug gesammelt hat ist und keine Gefahr droht.

```
bool GenugGesammelt = false;
bool Gefahr = false;

while (                    )
{
    // Die Funktion GefahrErspaehen() lässt Hansi eine etwaige Gefahr erspähen
    // Bei Gefahr ist der Rückgabewert von GefahrErspaehen() true sonst false
    Gefahr = Hansi.GefahrErspaehen();

    // Die Funktion EssenSammeln() lässt Hansi Essen suchen
    // Der Rückgabewert von EssenSammeln ist true, wenn Hannsi genug gesammelt hat
    GenugGesammelt = Hansi.EssenSammeln();
}
```

Hinweis: Vor, zwischen und nach "="-Zeichen sollten keine Abstände stehen, sonst wenn nötig schon!

Figure 70 – Testing logical thinking and the ability to transform results into correctly written source code

The story for this test item was centered on a squirrel that was introduced earlier in different examples during lessons, thus students were familiar with the squirrel’s sample character, so they did not have to wonder about biological, or any other, circumstances and could therefore concentrate solely on the solution.

(Translation) The squirrel Hansi is terribly hungry. Luckily, Hansi knows a place where plenty of food is available. The problem is a hunting cat that lurks around the food and has already devoured some squirrels.

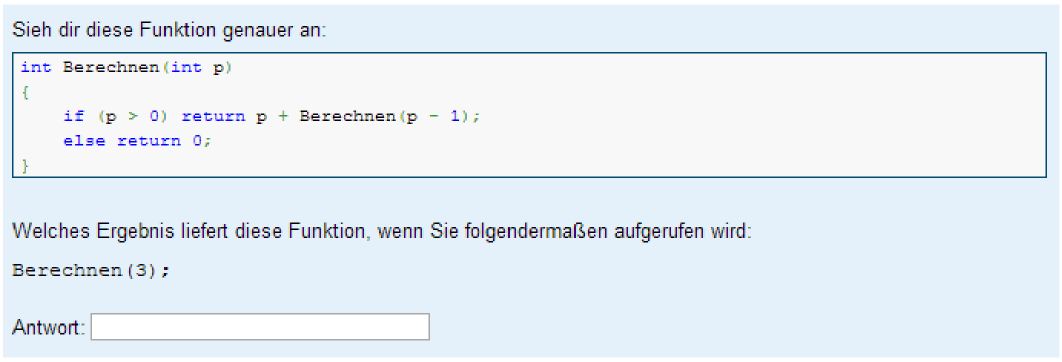
Fill in the blank space in the WHILE-loop such that the squirrel collects food as long as it has not collected enough already and also as long as there is no danger.

```
bool collectedEnough = false;
bool Danger = false;

while (                              )
{
    // The function LookoutforDanger() returns true if danger is detected
    Danger = Hansi.LookoutforDanger();

    // The function CollectFood() lets Hansi collect food
    // it retuns true if enough food was collected
    collectedEnough = Hansi.CollectFood();
}
```

Possible correct answers were: `while(!collectedEnough && !Danger)` and `while(collectedEnough==false && Danger==false)`. Any similar permutations, such as `while(!Danger && !collectedEnough)`, also counted as correct answers.

Q3 - Understanding what an algorithm does (8 points)

For Question 3, the task was to analyze what the algorithm shown in Figure 71 does.


Sieh dir diese Funktion genauer an:

```
int Berechnen(int p)
{
    if (p > 0) return p + Berechnen(p - 1);
    else return 0;
}
```

Welches Ergebnis liefert diese Funktion, wenn Sie folgendermaßen aufgerufen wird:

`Berechnen(3);`

Antwort:

Figure 71 – Test item of understanding what an algorithm does

(Translation) Take a closer look at this function:

```
int Calculate(int p)
{
    if (p > 0) return p + Calculate(p - 1);
    else return 0;
}
```

Which result does this function produce, if it is called like this:

```
Calculate(4);
```

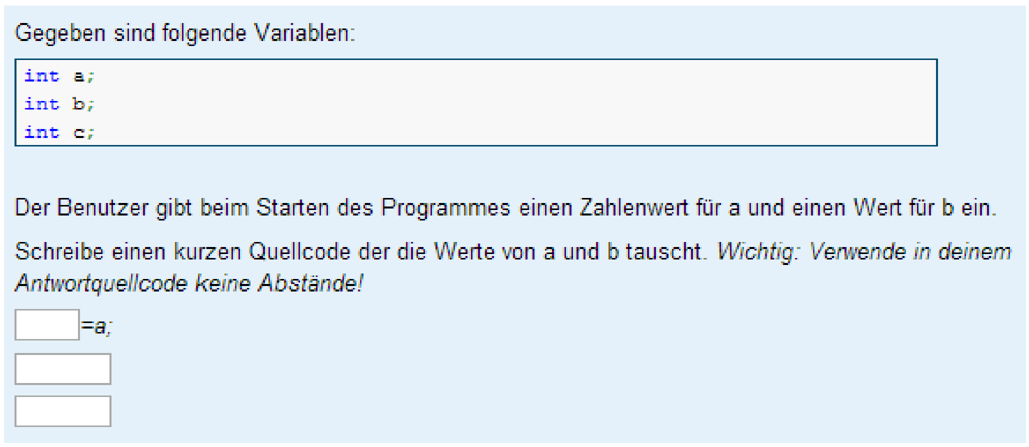
Given four as an example value, during the IT skills test, random values between two and five were used, with the correct result automatically calculated by the quiz tool using the formula $\frac{n\,(n+1)}{2}$. This formula is reminiscent of Carl Friedrich Gauß (GAMING, 2005). Indeed, the formula itself is of even older origin, going back to the Pythagoreans (Thomas W Malone & Lepper, 1987); however, in the context of solving this problem from the student’s perspective, the student’s level of knowledge with respect to the formula did not have any effect.

After the student answered this question, feedback here was an explanation of what the function calculates:

(Translation) This function calculates the sum from p down to 1 in a recursive way.
Example: p = 4;
The function calculates: 4 + 3 + 2 + 1
Result: 10

Q4 – Variables (6 points)

Shown in Figure 72, the task of Question 4 was to switch the values of two variables. The use of a temporary variable was not only allowed, but requested.


Gegeben sind folgende Variablen:

```
int a;
int b;
int c;
```

Der Benutzer gibt beim Starten des Programmes einen Zahlenwert für a und einen Wert für b ein.

Schreibe einen kurzen Quellcode der die Werte von a und b tauscht. *Wichtig: Verwende in deinem Antwortquellcode keine Abstände!*

```
[    ]=a;
[      ]
[      ]
```

Figure 72 – Switching the values of two variables with the help of a temporary variable

(Translation) Given the following variables:

```
int a;
int b;
int c;
```

On program start, the user enters the value for a and another value for b.

Type in the necessary source code to switch the values of a and b. Important: Please do not use spaces in your source code[58]*.*

The solution is:

```
c=a;
a=b;
b=c;
```

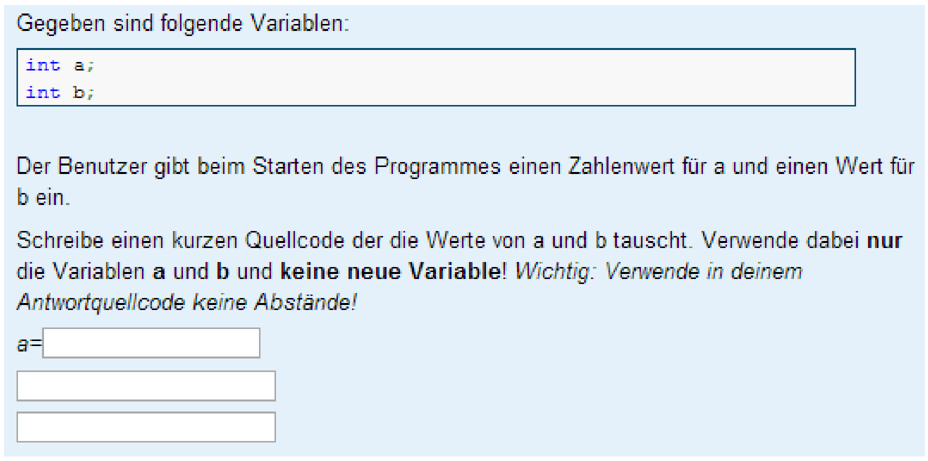
The recommendation not to use the extra spaces was stated, because the automatic processing of the test and automatically generated feedback was easier to specify without any deliberate number of spaces. After the test, together with the students, a review to correct the points for correct answers marked as incorrect by the Moodle quiz system took place.

Q5 – Mathematical/analytical thinking (8 points)

For Question 5, shown in Figure 73, students had to switch two variables without the help of a temporary variable. To achieve this, an arithmetical solution is the best approach; however, a hint that a mathematical approach leads to the solution was not given to

58

students, so students had to think analytically to find the approach, and then perhaps do an exemplary calculation in their heads to verify the concept. To grasp the concept in full detail, students also had to think analytically to answer the question, “which variable has which value at which time?” Further, a basic understanding of an algorithm was necessary to succeed. Those numerous demands on various levels are the reason why this item was worth eight points, whereas Question 4 was only worth six points.


Gegeben sind folgende Variablen:

```
int a;
int b;
```

Der Benutzer gibt beim Starten des Programmes einen Zahlenwert für a und einen Wert für b ein.

Schreibe einen kurzen Quellcode der die Werte von a und b tauscht. Verwende dabei **nur** die Variablen **a** und **b** und **keine neue Variable**! *Wichtig: Verwende in deinem Antwortquellcode keine Abstände!*

a=

Figure 73 – Switching the values of two integer variables without using a temporary variable

(Translation) Given are the following variables:

```
int a;
int b;
```

On program start, the user enters the value for a and another value for b.

Type in the necessary source code to switch the values of a and b without the use of a temporary variable. Important: Please do not use spaces in your source code.

A solution here is

```
a=a+b;
b=a-b;
a=a-b;
```

5.3.5.1 *Detailed results of the IT skills test*

Shown in Figure 74, the results of the IT skills test were rather surprising. Although scientifically, various different outcomes were anticipated, such a strongly significant difference between the GamePinS group (i.e., group A) and the other software projects group (i.e., group B) was not expected. While for the data type question (i.e., Question 1), groups A and B scored nearly the same, all other questions showed a huge gap. For example, for Question 2, regarding logical conditions, students undergoing the GamePinS curriculum earned nearly twice as many points as students in the other group. Similarly for Question 3, regarding the recursive function, students from the GamePinS group performed twice as well. For Question 4, regarding the switching of variables with a temporary variable, the GamePinS group showed significantly better results. To successfully complete the last question (i.e., Question 5), regarding the switching of variables without the use of a temporary variable, analytical and mathematical thinking were required; the GamePinS group earned an average score of 5.67, which was more than 10 times higher than that of the other software projects group, who had an average score of 0.46.

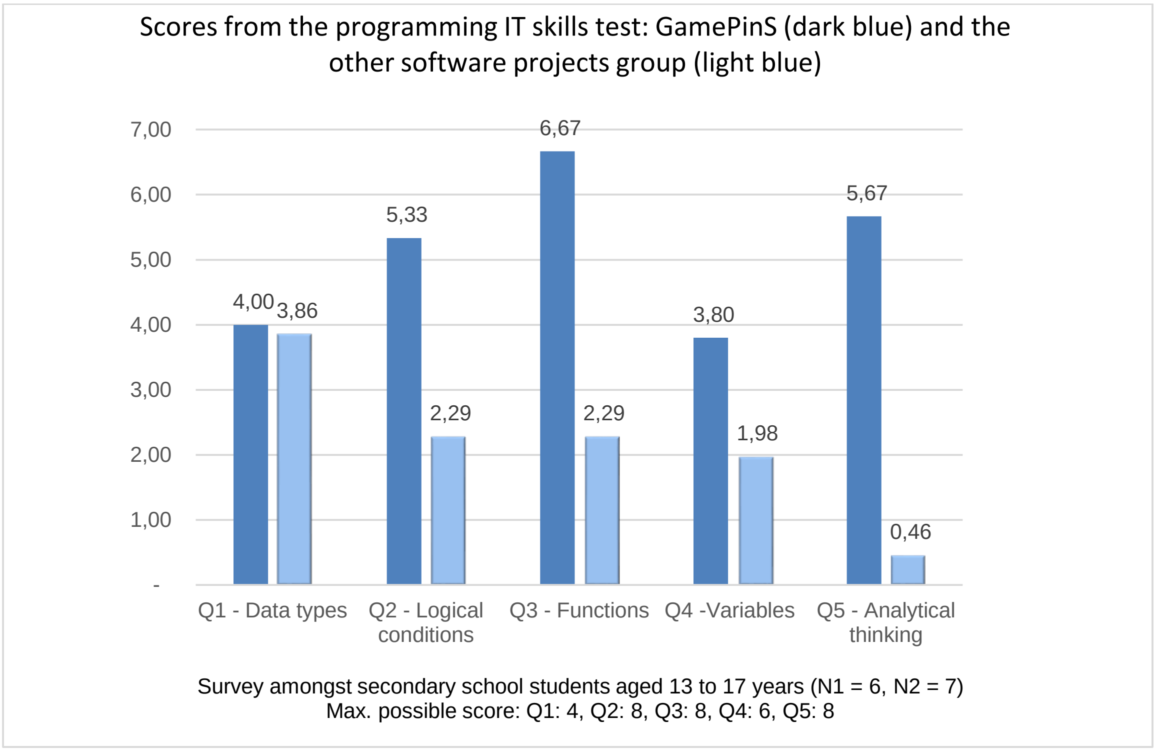


Figure 74 – Scores from the IT skills test: GamePinS versus other software projects

5.3.6 Discussion of the IT skills test

The IT skills test was designed to determine the skills of students in selected areas of computer science; however, the design of the IT skills test had some limitations. One of these limitations was the amount of time allocated to the test in school, given a maximum of 50 minutes per lesson without a break. This had to take into account that students needed to enter the lab, log in to their PCs, then start the test (i.e., approximately five minutes). Explanation of the test took another five minutes. The buffer for technical difficulties was also five minutes. This meant that the net test time was limited to only 35 minutes.

Except for the *data types* question, the underlying goal of the test was not to determine how students reproduce previously learned material, but rather to let them approach and solve new test items that they have never specifically seen before, an approach I call the "riddle approach." Compared with simple regurgitation of material, solving the given riddles requires more thinking, creativity, and time. To provide the necessary time to let students tinker with and think about solutions, the number of items had to be limited.

The necessary time/workload balance was estimated with the help of a student that was not part of GamePinS or the other software projects groups; this student completed the test separately. I was aware that too few items on the test would influence the validity of the skills to be tested in a problematic way, so more research here to support the results is a possible next step; however, through the limited item approach, nonetheless, comparative results showing how each student group performed was determined.

5.3.7 Student feedback

How did you experience the game programming?

Overall, game programming was noted as being fun (e.g., *"It was fun and I also learned something," "It was fun, but we didn't get so far," "In the beginning, I missed many lessons, but when I attended regularly, the lessons were fun"*). Game programming was also described as interesting, astonishing, and a great experience (e.g., *"It was interesting to create a game,"* "*It was interesting to see how a simple game was created, but we also talked too much, which was on the other hand motivating*," *"Programming was an astonishing thing, it was a great experience"*). All feedback from students was positive and had "fun" and/or "interesting" in their statements. Since learning is very effective when it is fun (GAMING, 2005; Lonati, Monga, Morpurgo, & Torelli, 2011; Thomas W Malone & Lepper, 1987; Thomas W. Malone, 1980) and/or interesting (Hidi, 1990; Renninger, Hidi, & Krapp, 2014), game programming is clearly a promising means to making learning more effective.

Can game development increase motivation?

All students agreed that game development increases the motivation to learn programming. One student stated the importance of motivation when doing things from one's own will (i.e., *"Yes, but you have to be motivated if you want to do it yourself out of your own will").* Another student pointed out that one source of motivation also stems from the improved understanding of programming itself (i.e., *"I think 'Yes,' because you understand better how to program").* A third student pointed out that *"…it depends on what you program."*

One student emphasized that the practical work compelled him to think more independently (i.e., *"Yes, I think so, because you work more practically and because of this, you think more by yourself").*

Another student stated that *"game programming is not that interesting for me."* This statement interestingly was from the student who answered the first item with *"In the beginning, I missed many lessons, but when I attended regularly, the lessons were fun."* Either there is a contradiction between these two statements or the student had game programming in mind when stating that "*game programming is not that interesting"* and was thinking of other aspects of the lessons when stating *"…when I attended regularly, the lessons were fun."* Certainly, not everyone has to like games or game development, but if the outcome of the lessons is sufficient, it might be seen a success.

(Can game development increase the motivation?) Was this the case in this class? Why? Why not?

One student found the aspect of teamwork motivating, stating, *"Through working in teams, Yes!"* Another student was generally interested in learning how to write programs (i.e., *"Yes, because I'm interested in learning how to write programs."* A third stated *"Yes, in any case and because I and nearly all others play games on the PC."* Another student mentioned that he wanted to do some game programing anyway and pointed out, *"Yes,* *because, when you want to do something by yourself and accomplish it, it is satisfying."* Another student declared, *"I was mostly (except when I was tired) very motivated, because the lessons were designed to be rich in variety."* Only the student that stated that game programming was not interesting in the previous item left this field blank.

Would you prefer a more detailed structure for the project-based approach?

Aside from wondering if more examinations[59] would be more effective than talking through material from previous lessons, there was no call for a more detailed project plan (e.g., *"Does not have to," "No, and it was more interesting than in other subjects," "I think you (the teacher) had a specific plan"*).

What I want added

For this question, students primarily pointed out again that the lessons equated to fun and that they also gained some experience in programming (e.g., *"The lessons meant a lot of fun and one learns from it," "I thought the lessons were fun," "I want to learn more about programming," "The lessons were the funniest and most interesting of all subjects. Please stay like that so that other students might have fun in your lessons").*

Another student pointed out that although he likes computer science, other students might prefer other subjects; he stated, *"I like computer science. Another one likes Spanish. Eaten was a peafowl.[60] You are a good teacher and fun!"* Also, one request was made to do more with robots, especially Lego Mindstorms ("Website - Lego Mindstorms,") (e.g., *"but I also wanted to work with Lego Mindstorm robots"),* which is also well-known for being used to motivate students in computer science lessons.

5.3.8 Summary of the second year

Activities in the second year took place from January 2014 through June 2014. This was a shorter period than planned. The school year ranged from September 2013 to June 2014, but as the responsible teacher, I returned to school after paternity leave in January 2014

[59] A student stated that *"perhaps some more examinations, but continuing talking was motivating"*.

[60] Not during the IT lessons, but rather by an ice bear in the zoo.

and started the curriculum anew. Although the overall time was less than in the previous year of game development, students were more satisfied with the game programming activities in the first year without GamePinS (compare to Section 5.2.4).

The GamePinS group estimated their improvement in programming to be considerably higher than that of the other software projects group. Students working with GamePinS performed more than twice as well as the other group in the IT skills test. In general, the GamePinS group was as motivated as the other group, but learned more than the other group.

A possible bias here could have been the aforementioned previous activities in computer science classes. In particular, one group worked only with end-user Office applications, and the other group had some programming lessons with Visual Studio 2010 and Visual Basic, so they had an advantage in already knowing the Visual Studio development environment. Surprisingly, the group with the Visual Studio/Visual Basic advantage, which was group B (i.e., the other software projects group), performed weaker on the IT skills test. Group A (i.e., the later GamePinS group) were working only with Office applications, but performed much better on the IT skills test.

To summarize, the GamePinS activities led to higher scores on the tested IT skills than in the other scenario, while overall, students were highly motivated.

5.4 The third and final year of game programming

Results from the previous year led to some interesting questions. In particular, one question arose in several academic and research discussions, i.e., "are the better results of the game development group really caused by a process within game development itself, or are the outcomes caused by consequences of experimenter bias?" The question regarding the causality of the intervention and outcomes is a legitimate and very delicate question, because in our setting, I served as both the teacher who introduced GamePinS and also the one who developed and employed it. Further, the experimenter was also the teacher and had expectations[61] regarding the outcome. To summarize, the question as to the real cause of the observed improved performance of the GamePinS group had to be explored.

The question of the cause of the performance improvement of the GamePinS is certainly reminiscent of a standard problem of experimental design (Strickland, 2001a); there were independent variables (i.e., the GamePinS curriculum versus the other software projects curriculum), corresponding dependent variables (i.e., the performance of the students in both groups), and many confounding variables (i.e., learning history or experience, skills in the field of programming, the general intelligence and capabilities of students). While such factors as experience, skills, and intelligence were estimated to be neutralized or at least toned down within the sample, it seemed critical to extract the influence of the teacher.

The influence of an experimenter can have a significant confounding impact. In addition, if the experimenter is also the teacher, a possible bias must not be ignored and has to be examined. To test the teacher's influence, the classical control group design seemed appropriate. Therefore, the focus of the last year was on researching how the

[61] Expectations that I was consciously aware of and tried to take into account, but there is always a chance to cause bias.

students of a comparable teacher performed with and without GamePinS. Out of the control group design, as shown in Figure 75, a setting with four groups of students and two teachers was developed, i.e., Teacher X – GamePinS (XA), Teacher X – OSCP[62](XB), Teacher Y – GamePinS (YA), and Teacher Y – OSCP (YB).

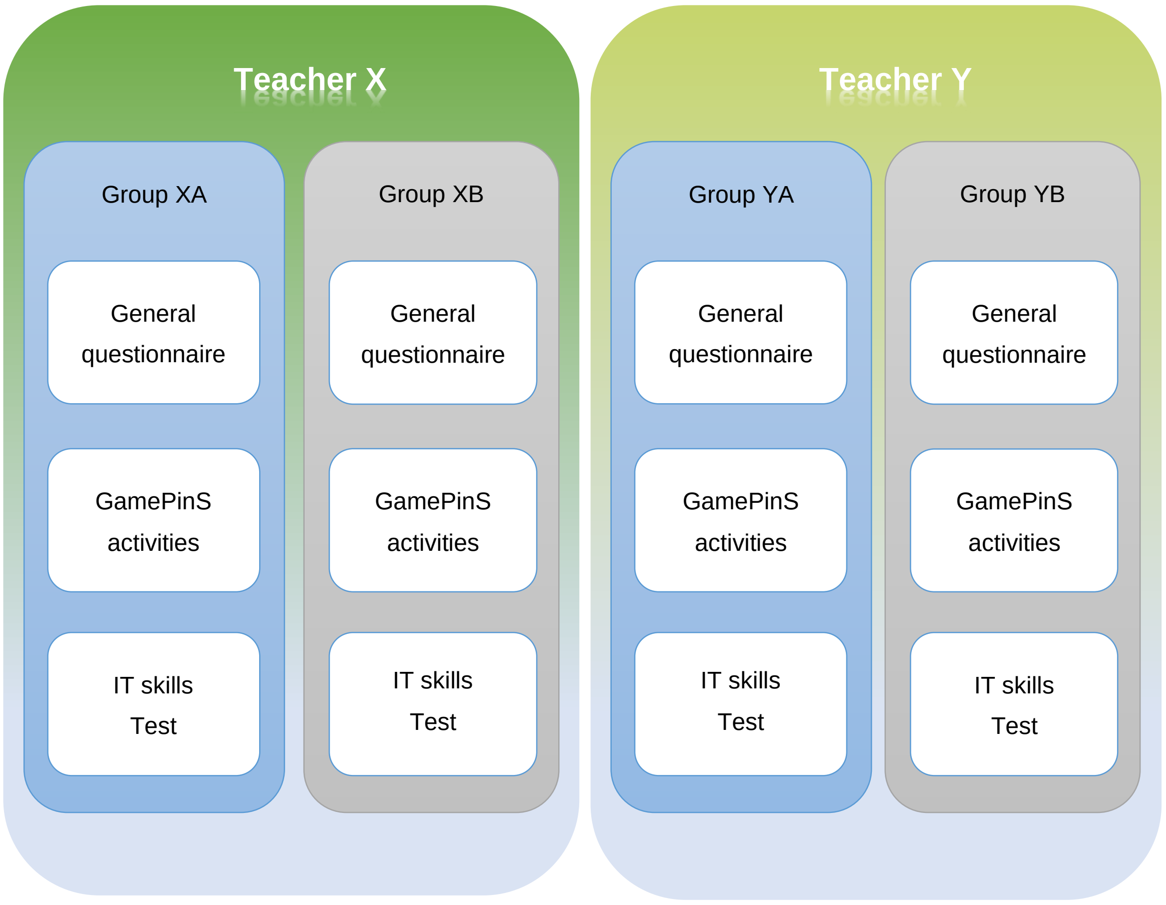


Figure 75 - Experimental design of the GamePinS versus other student chosen activities with control groups

Each teacher coached one group doing GamePinS and another group doing other programming projects. The initial questionnaire was used to determine student attitudes toward game development. Results are discussed in Section 5.1.

[62] Other Student Chosen Projects

5.4.1 Achievements in IT skills test

Achievements in the field of IT skills were again determined by five different assessments in an IT skills test. To check for bias, teacher X analyzed midterm grades and assigned the group with the better grades to other student software projects, while the group with the lower grades were assigned to GamePinS. Thus, this year, the GamePinS group was also tested against the other group, with better-performing students in computer science class in the previous midterm. For teacher Y, the average of the grades in two groups were the same, so students were assigned randomly to either the GamePinS group or the other group.

As shown in Figure 76, students of teacher X, i.e., the teacher who conducted GamePinS the previous year, achieved better results for GamePinS (i.e., a total average of 22.04) than for other student software projects (i.e., a total average of 14.25). The groups taught by the teacher who joined the GamePinS activities in the actual school year yielded the following results: GamePinS (21.60) and other student software projects (5.46). The maximum number of possible points on the IT skills test was 36. For the total average, the total scores of the individual students were added and divided by the number of students.

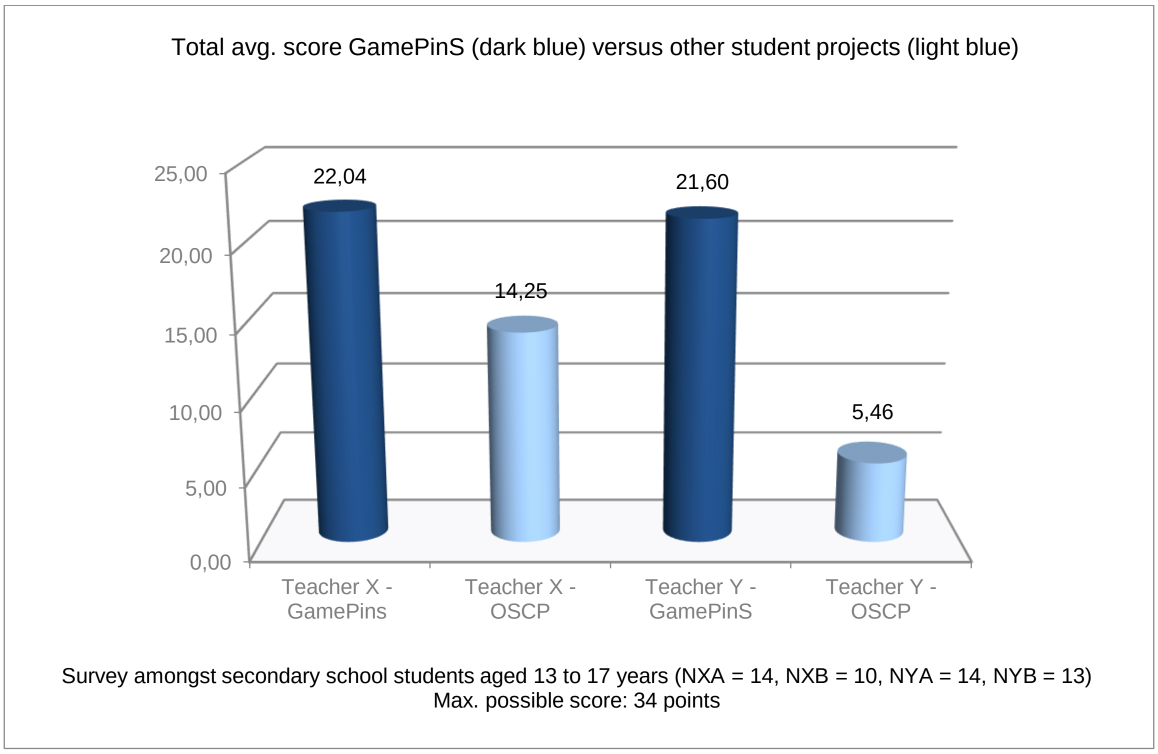


Figure 76 – Achievements in the IT skills test, with two teachers and two groups per teacher

5.4.1.1 *Detailed results of the IT skills test*

The single items of the IT skills test were also analyzed. Students had to answer questions regarding data types, logical conditions, algorithms, variables, and analytical thinking. The outcomes for the single items contributed to a total score, though each question was not equally weighted. On some test tasks, students of both groups from teacher X performed similarly (e.g., data types and variables), while on other items, students performed quite differently (e.g., logical conditions, algorithms, and analytical thinking). For newly joined teacher Y, the differences were significant for all five parts.

Q1 – Data types

Question 1 regarding data types was again a question that required students to map content to the most appropriate data types (see Section 5.4.1.1).

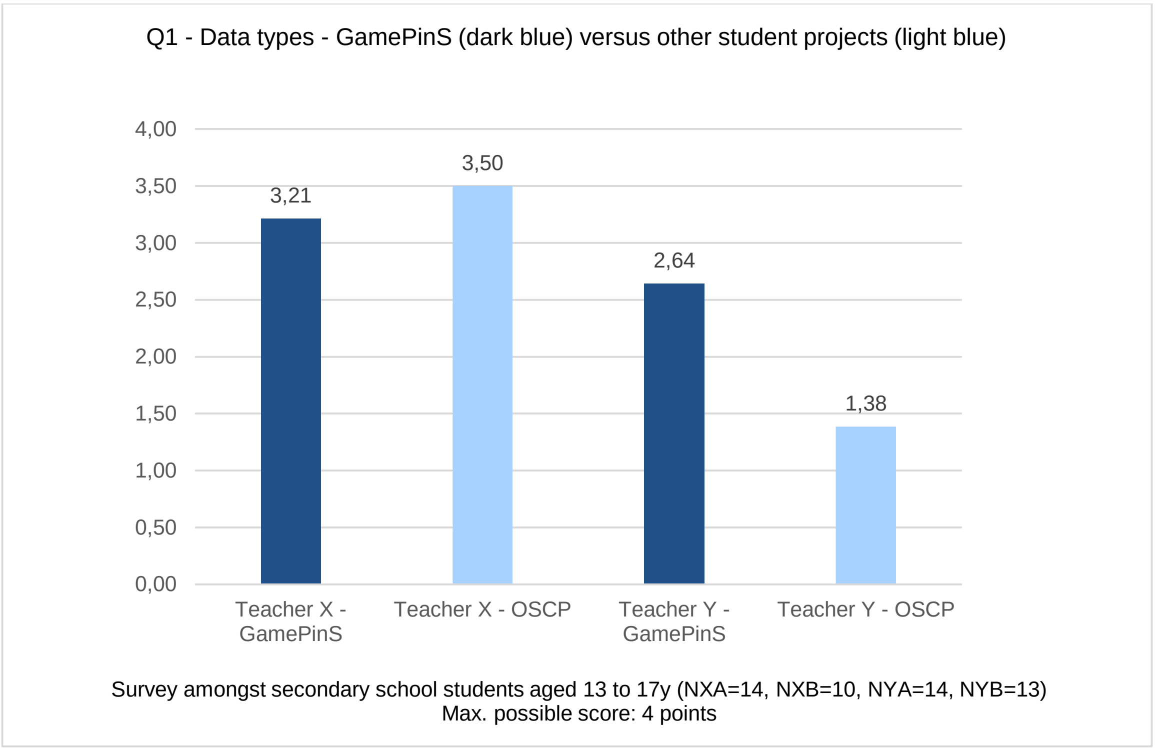


Figure 77 – Scores of student groups on Question 1 regarding data types

As shown in Figure 77, students of teacher X performed nearly equally, with the interesting fact that the non-GamePinS group performed slightly better, which was likely caused by the generally better IT performance of the other group from teacher Y.[63] For teacher Y, the GamePinS students performed slightly better than the other student projects group. The 2015 results confirmed the findings of the previous year. In the particular field of data types, there were no large differences between GamePinS and other student projects groups.

Q2 – Logical conditions

As shown in Figure 78, the question to examine student knowledge and skills with loops and logical conditions showed a significant advantage for students working with GamePinS (i.e., with a score of 4.57) with teacher X as compared with the other group (i.e., with a score of 1.20) from teacher X. For teacher Y, this question yielded lower results for this specific item, but the GamePinS group (i.e., with a score of 1.71) also performed significantly better than the other group (i.e., with a score of 0.62).

[63] *For an explanation of the fact that the second group from teacher X had better results than last year, see Section 5.4.1.1.*

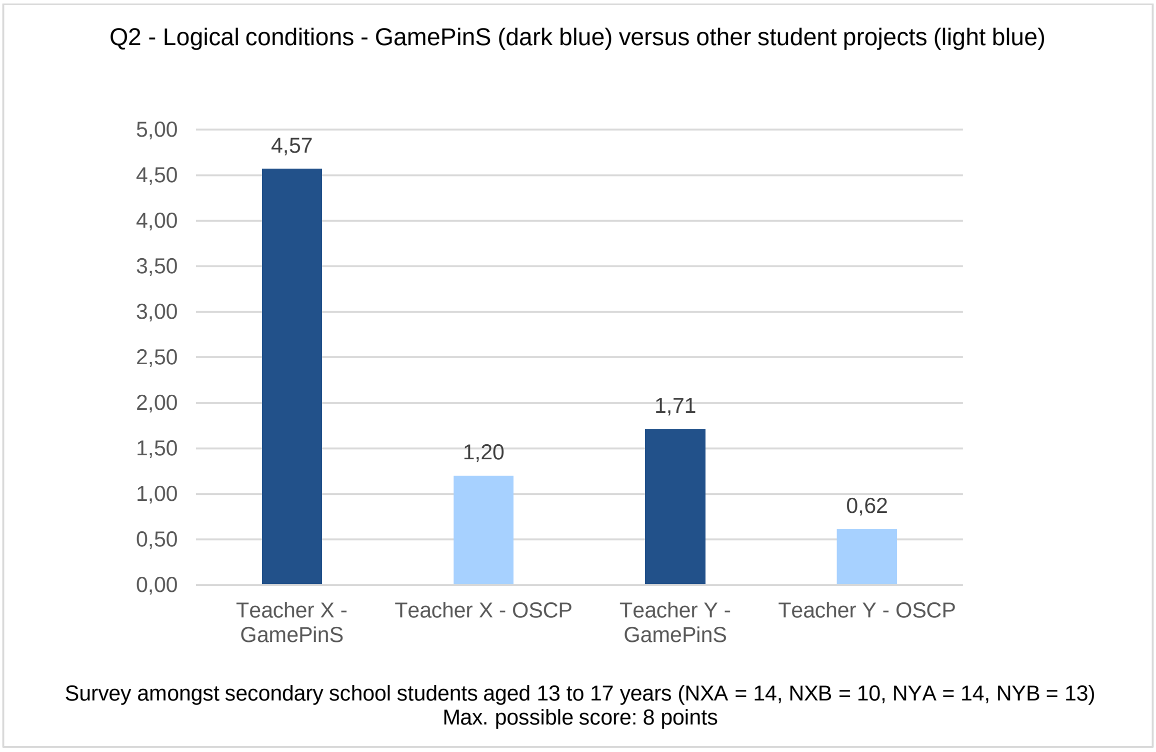


Figure 78 – Scores of student groups on Question 2 regarding logical conditions

Q3 - Understanding what an algorithm does

As shown in Figure 79, understanding what an algorithm does did not show huge differences in the groups of teacher X, although the GamePinS group did perform better once again, with 4.57 points, whereas the other group earned 3.20 points. Again, the interpretation of the lower difference here seems to be due to the generally better-performing other projects group. For teacher Y, with randomly assigned groups, the GamePinS group, with a score of 6.86, clearly outperformed the other projects group with a score of 2.46.

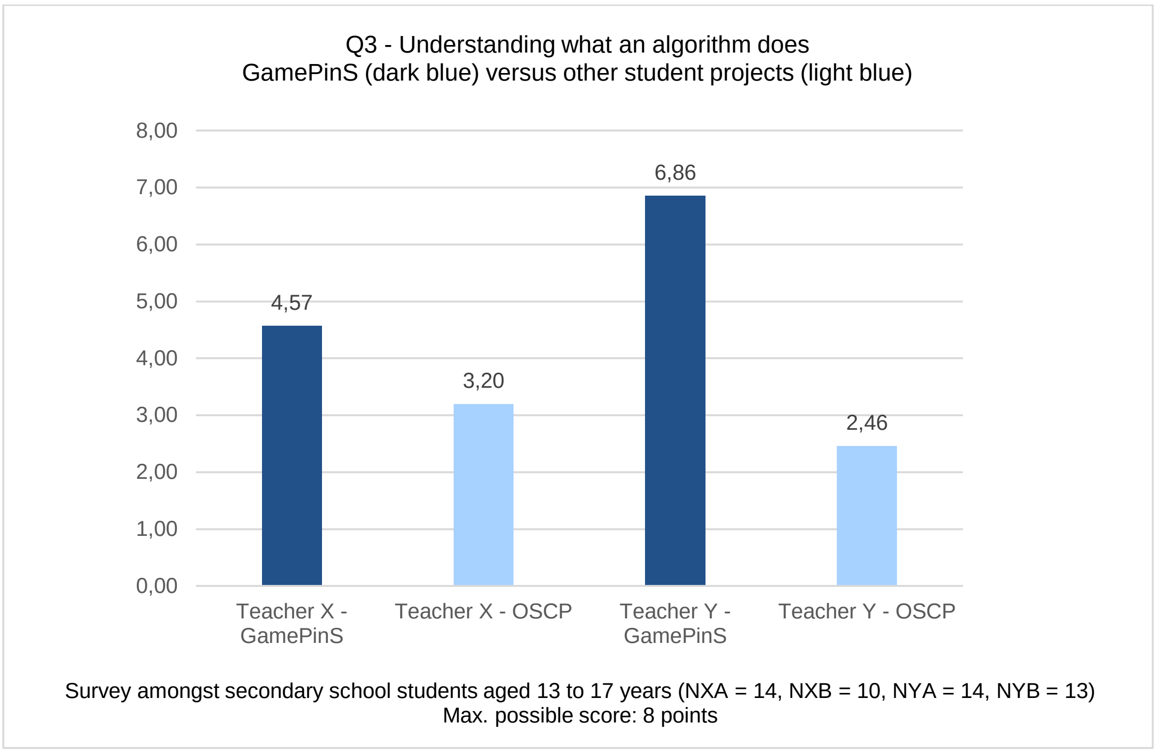


Figure 79 – Scores for student groups on Question 3 regarding understanding what an algorithm does

Q4 – Working with variables

For solving Question 4, students had to work with variables and switch the values of two variables. The use of a third variable as a temporary variable was allowed and also encouraged by a hint in the test item. As shown in Figure 80, results for both groups of teacher X were almost the same, whereas the groups of teacher Y showed a large difference.

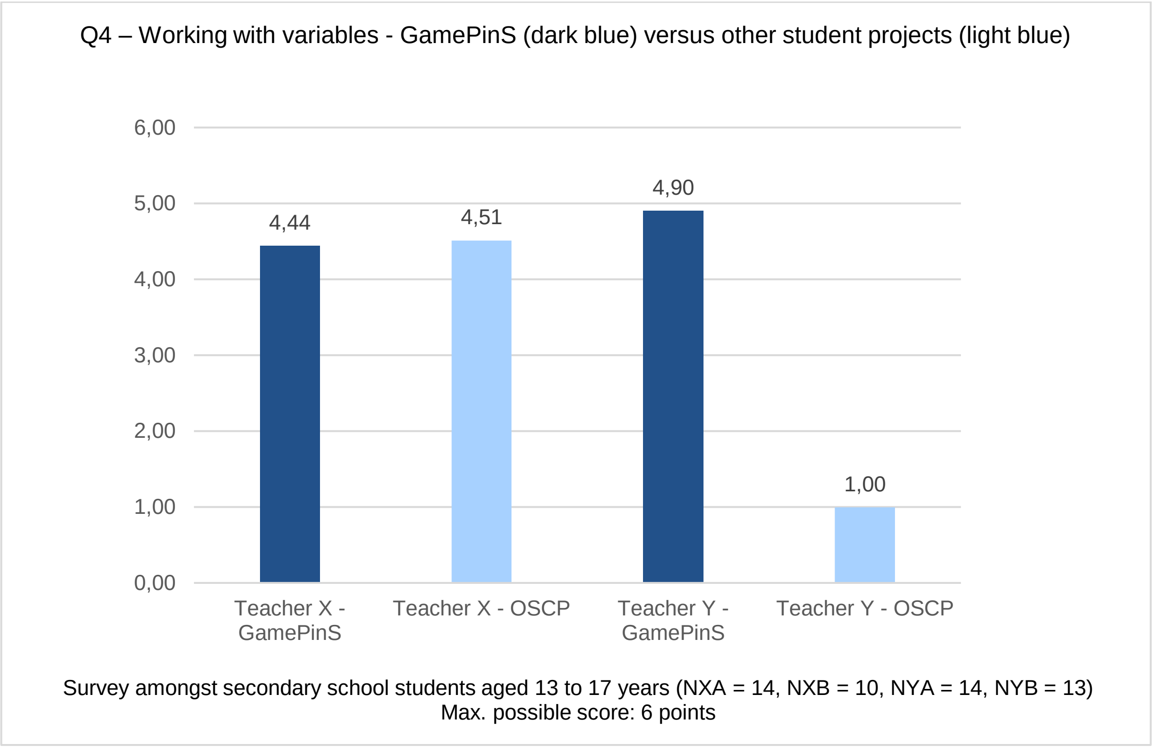


Figure 80 – Scores for student groups on Question 4 regarding variable swapping with a temporary variable

Q5 – Variables and mathematical/analytical thinking

To correctly solve Question 5, students needed knowledge of variables, but also a certain amount of mathematical, analytical, and algorithmic thinking. Here, two integer variables had to be switched without the use of a temporary variable (see Section 5.4.1.1). For teacher X, both groups performed as expected, whereas for teacher Y, all students in the other projects group failed the test item, as shown in Figure 81 – Scores of student groups for the mathematical/analytical thinking.

Failing this test item required further inspection. The first theory was that since Question 5 was the last item, the lesson ended and the item was forfeited, but this was not the case, since logs showed that the test was started at the beginning of class and ended well before class ended. The available time was 30 minutes. The second theory, i.e., technical problems, could also be dismissed. Therefore, two question remained, i.e., *“did these students understand the question?”* and *“did the students understand the process and states of switching variables?”*

Before we look for possible mistakes, let us first look at the solution process and the knowledge required to solve this problem. The short yet elegant solution $a = a + b; b = a - b; a = a - b;$ involves the following knowledge[64]:

- Basic knowledge of variables: *What is a variable? How are its values assigned?*
- Very basic math skills
- Importantly, knowledge of when values are used in an operation:
  - One appropriate representation of what happens with the values of the variables is: $\begin{array}{l} 5 <== 5+10 \\ a \ = \ a+b \end{array}$
  - It is important to understand that the value 5 in a is overwritten by the sum $a + b$, which is 15.

To answer the matter of understanding the question and the process, students were asked if or how they understood the question and how they were trying to solve it. The outcomes of this questioning was that students very well understood the goal to be achieved in the item, but the process of the simple algorithm to solve it remained unclear. For illustration purposes, two common mistakes are as follows:

1. Some students did not understand the direction for an assignment process correctly. They thought, for example, that $a = b$ and inserted the value from a into b. So, if a is 5 and b is 10, $a = b$, then $a = 5$ and $b = 5$.
2. Other students tried to solve it as in math class, i.e., when asked, "What does $a = a + b$ do?"
   - $a = a + b \mid -a$
   - $\cancel{a} = \cancel{a} + b$
   - Their answer: a stays 5 and $b = 0$

The mistakes of having the wrong idea of value assignment in variables and/or trying to apply a pure mathematical approach were discovered in connection with the inquiry of the group in which all members failed to answer the question correctly. Interestingly, those mistakes where not a single trait of teacher Y, but were also made by students in the better-

[64] The necessary skills are described in Section 4.10.1.1.

performing group. To summarize, the reasons for the zero-score of individuals was discovered, but the performance of the entire group was a statistical outlier of the Y-OSCP group, which opens another interesting path to follow in further research.

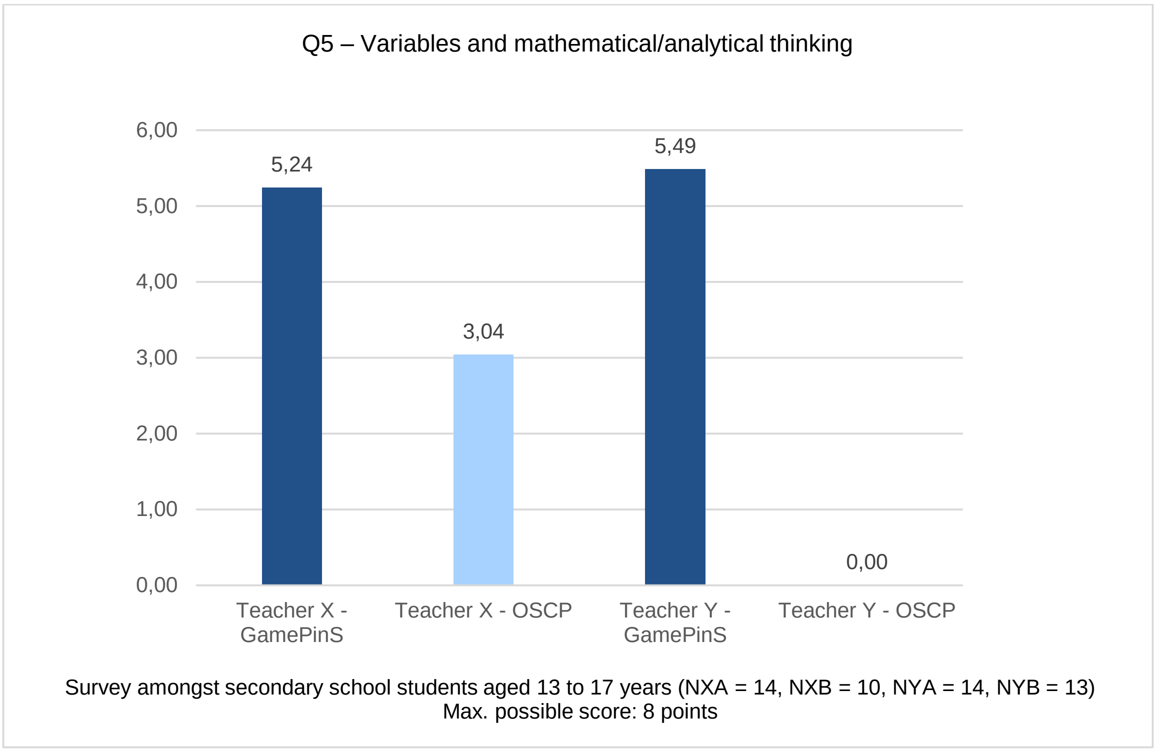


Figure 81 – Scores of student groups for the mathematical/analytical thinking question

5.4.2 Summary of the third year

The third and final year confirmed that GamePinS is boosting IT skills in the covered area of computer science better than other student software projects. As shown in Figure 82, the GamePinS activities outcompeted the circumstances that the projects of the other group were self-selected and therefore increased student motivation. GamePinS led to better average and total results in the IT skills test. The two teacher and four group scenario showed that it was not primarily the teacher who boosts performance of students, but rather the game development. Even the highly skilled other software group, with excellent grades in the previous semester, was outperformed by the GamePinS group in the IT skills test.[65]

[65] OSCP: 14.25 points, GamePinS: 22.04

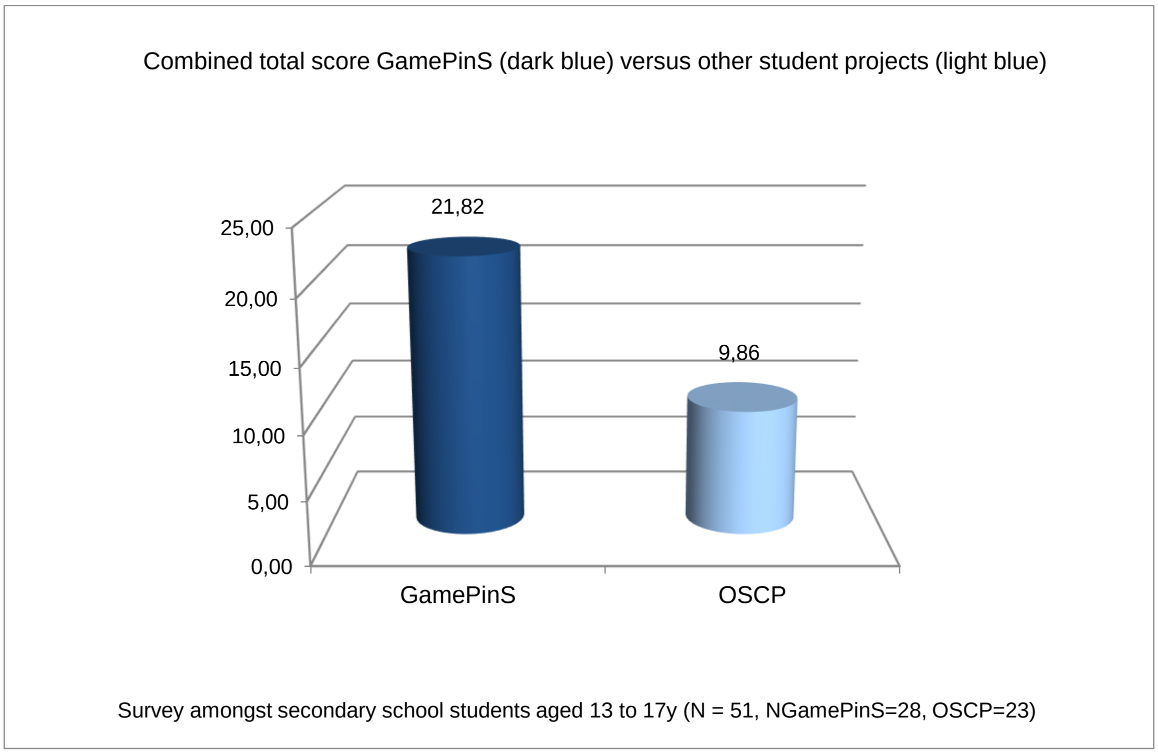


Figure 82 –Total score of the GamePinS and OSCP groups of both teachers combined

6 Conclusions

In this study, students across three different school years developed computer games in the classroom. The results of the game development research supported our basic assumptions and also revealed some additional phenomena. Assumption (A1), i.e., that game development in computer science increases engagement and motivation of students, was proved to be accurate through student feedback and questionnaires; however, an interesting gap between each student's anticipated motivational boost and actual motivational level was discovered. In fact, the reality of game programming was harder than most students anticipated. This gap was especially clear in the first year in which no special set of tools was used (i.e., only "plain" XNA was employed).

The observation of motivational disparities went hand in hand with the complexity of the framework. The corresponding Assumption (A4), i.e., "*the complexity level of the framework influences the depth of understanding of the concepts to be learned,"* was vindicated by the observation that motivation increased with reduced complexity by employing GamePinS and easy-to-use tools. Interestingly, the concept of *more complexity forcing students to generate a deeper understanding* only proved true at the very beginner levels.

When it came to game programming, exactly the opposite effect was observed, i.e., above the total beginner level, higher complexity led most students not to *better understanding*, but rather to *not understanding* the concepts at all. With the less complex, but still complex enough, GamePinS framework, students showed a deeper understanding than with plain XNA. Another dimension of complexity or maybe just an option overload could be determined when comparing the GamePinS group to the other student projects group. Students working with GamePinS clearly performed better at the IT skills test than the other group.

Teamwork did increase, as compared to the regular curriculum; however, the increase was not specifically characteristic of game development, but rather a mechanism caused by project-based learning. Accordingly, Assumption (A2), i.e., *"project-based game* *programming in computer science fosters teamwork competencies,"* was true for game development, but also for the other student software projects.

Assumption (A3), i.e., *"game programming promotes basic concepts of computer science more effectively than other typical software development scenarios,"* was positively verified for understanding of algorithms, logical conditions, variables, and mathematical/analytical thinking, but not for understanding of data structures, where both groups performed equally well. This verification was done by a systematically repeated IT skills test with six different groups doing GamePinS and other student software projects. The groups were under the supervision of two different teachers to identify and manage any possible experimenter bias. Further, the assumption that game programming promotes basic concepts of computer science more effectively was also confirmed by the self-estimation of the students.

In addition to these distinct research results, some useful principles for making game development more productive were identified. These principles included setting attainable goals, reducing complexity, and being extremely well prepared. In our case, the goals were set by talking about the resources necessary for big triple titles, such as having 1000 times more time and at least four times the core crew with plenty of experience in game development, as well as millions of dollars for outsourced tasks. Still, there are enough relatively small but absorbing games, such as jump and run games, adventure games, and small racing games to work on.

Reducing complexity was the second principle, and indeed the complexity of game development with decent physics embedding was a huge challenge for our case at first, but the complexity was successfully tamed via GamePinS. Finally, the attitude of being extremely well prepared is particularly vital in computer science, because we have a high threshold, as well as a good change of technical difficulties, and there should be at least one expert on the scene who knows how to deal with both this high threshold and the technical challenges.

Game development is an engaging and motivating means to enhance effective learning of programming and to boost IT skills, and goes far beyond merely writing source code.

The employment of a specialized framework, such as GamePinS, is helpful, but it is not mandatory to developing one’s own customized solution. It is more important to employ tools that make the development process straightforward, lucid, and ensure a low threshold for beginners. Therefore, the game development tools and course concepts should aim to reduce complexity and support attaining realistically set goals. An inspiring and satisfying game development process supported by a well-prepared teacher and optional existing features, such as a physics simulation, has the potential to unleash a burst of motivation and lead to a rapid improvement in student IT skills.

Figures

References

Alam, Syed Shah, Bakar, Zaharah, Ismail, Hishamuddin Bin, & Ahsan, MN. (2008). Young consumers online shopping: an empirical study. *Journal of Internet Business, 5*(1), 81-98.

Altrichter, Herbert, & Posch, Peter. (2007). *Lehrerinnen und Lehrer erforschen ihren Unterricht. Unterrichtsentwicklung und Unterrichtsevaluation durch Aktionsforschung* (4., überarb. u. erw. Aufl. ed.). Bad Heilbrunn: Klinkhardt.

Anderson, Terry, & Shattuck, Julie. (2012). Design-Based Research: A Decade of Progress in Education Research? *Educational Researcher, 41*(1), 16-25. doi: 10.3102/0013189x11428813

Autodesk 3ds Max. Retrieved 2012, September 2, from http://usa.autodesk.com/3ds-max/

Barab, Sasha, & Squire, Kurt. (2004). Design-Based Research: Putting a Stake in the Ground. *Journal of the Learning Sciences, 13*(1), 1-14. doi: 10.1207/s15327809jls1301_1

Barker-Plummer, David (1995). Turing Machines. Retrieved 2013, August 21, from http://plato.stanford.edu/entries/turing-machine/

Bian, Wu, Wang, A. I., Strom, J. E., & Kvamme, T. B. (2009, 25-28 Aug. 2009). *XQUEST used in software architecture education.* Paper presented at the Games Innovations Conference, 2009. ICE-GIC 2009. International IEEE Consumer Electronics Society's.

Biot, Olivier Website - melonJS. Retrieved 2012, October 7, from http://www.melonjs.org/

Blender (2012). Blender game engine. Retrieved 2012, June 24, from http://www.blender.org/

Blow, Jonathan. (2004). Game Development: Harder Than You Think. *Queue, 1*(10), 28-37. doi: 10.1145/971564.971590

BMBF (2003). Änderung der Verordnung über die Lehrpläne der allgemein bildenden höheren Schulen Retrieved 2015, August 24, from https://www.bmbf.gv.at/schulen/lehrdr/gesetze_verordnungen/VO_LP_AHS03_9431.pdf?4dzi3h

BMBF (2004). Austrian Federal Ministry of Education and Women's Affairs. Curriculum of Computer Sciene in Austria - Lehrplan Informatik für die AHS Oberstufe. Retrieved 2014, August 3, from https://www.bmbf.gv.at/schulen/unterricht/lp/lp_neu_ahs_14_11866.pdf?4dzgm2

BMUKK (n.d.-a). Aufnahme in die Volksschule. Retrieved 2013, September 2, from http://www.bmukk.gv.at/schulen/service/schulinfo/aufnahme_vs.xml

BMUKK (n.d.-b). The Austrian Education System. Retrieved 2013, August 31, from http://www.bmukk.gv.at/medienpool/17684/bw2013_e_grafik.pdf

BMUKK (n.d.-c). Schools and Education. Retrieved 2013, August 31, from http://www.bmukk.gv.at/enfr/school/schools.xml

Boekaerts, Monique. (2002). *Motivation to learn*. Brussels: International Academy of Education.

Boss, Suzie, & Krauss, Jane. (2007). *Reinventing project-based learning : your field guide to real-world projects in the digital age / Suzie Boss, Jane Krauss* (1st ed.). Eugene, Or.: International Society for Technology in Education.

Boyle, Elizabeth A., Connolly, Thomas M., Hainey, Thomas, & Boyle, James M. (2012). Engagement in digital entertainment games: A systematic review. *Computers in Human Behavior, 28*(3), 771-780. doi: 10.1016/j.chb.2011.11.020

Brown, Ann L. (1992). Design Experiments: Theoretical and Methodological Challenges in Creating Complex Interventions in Classroom Settings. *Journal of the Learning Sciences, 2*(2), 141-178. doi: 10.1207/s15327809jls0202_2

Burns, A., & Davies, G. (1993). *Concurrent programming*: Addison Wesley Longman Publishing Co., Inc.

Cartoon Network (n.d.). Fusion Fall. Retrieved 2013, August 29, from http://fusionfall.cartoonnetwork.com/splashpage.php

Comber, Oswald. (2008). *eLearning an Schulen. Master Thesis.* University of Vienna, Vienna.

Comber, Oswald (2012a). SSurvey Results: Student Feedback - Game Programing in School. One schoolyear of game programming in Secondary school. (username: gameprog, password: 2012GamePin$) Retrieved 2012, August 8, from www.comber.at/research/results2012/statements.html

Comber, Oswald (2012b). Survey Results: Attitudes towards "Game Programing in School" - A survey amongst Secondary School students (username: gameprog, password: 2012GamePin$) Retrieved 2012, August 8, from http://www.comber.at/research/results2012/SurveyA_2012.htm

Comber, Oswald (2012c). Survey Results: Game Programing in School - One schoolyear of game programming in Secondary school. A survey amongst students (username: gameprog, password: 2012GamePin$) Retrieved 2012, August 8, from http://www.comber.at/research/results2012/SurveyB_2012.htm

Comber, Oswald (2013). Ressource: MC Comber - a Moodle course for Computer Science and Game Programming. Guest access key: CS&GamePinS*Comber. Retrieved 2015, September 09, from http://www4.edumoodle.at/g16slsz/course/view.php?id=19

Comber, Oswald (2014). Game Programing in School - The second schoolyear of game programming in Secondary school. A survey amongst students (username:

gameprog, password: 2012GamePin$) Retrieved 2014, June 19, from http://www.comber.at/research/results2014/SurveyAandB_2014.htm

Comber, Oswald, & Motschnig, Renate. (2015). *Challenges and opportunities in employing game development in computer science classes*. Paper presented at the EdMedia: World Conference on Educational Media and Technology 2015, Montreal, Quebec, Canada. http://www.editlib.org/p/151464

Commons, Creative Creative Commons Licenses. Retrieved 2012, October 28, from http://creativecommons.org/

Computing Education Group - University of Kent Greenfoot documentation and tutorials. Retrieved 2013, August 21, from http://www.greenfoot.org/doc

Computing Education Group - University of Kent (2014). Greenfoot. Retrieved 2014, August 8, from http://www.greenfoot.org/

Connecticut State University Project Management. Retrieved 2012, August 8, from http://www.ct.edu/it/project/

Corno, Lyn, & Mandinach, Ellen B. (1983). The role of cognitive engagement in classroom learning and motivation. *Educational psychologist, 18*(2), 88-108.

Crawford, Chris. (2003). *Chris Crawford on game design*. Indianapolis, Ind.: New Riders.

Creighton, Ryan Henson. (2010). *Unity 3D Game Development by Example: Beginner's Guide*: Packt Publishing.

Cup, Liberated Pixel (2013). Retrieved 2013, March 22, from http://lpc.opengameart.org/

Dagienė, Valentina. (2011). Informatics Education for New Millennium Learners. Informatics in Schools. Contributing to 21st Century Education. In I. Kalaš & R. Mittermeir (Eds.), (Vol. 7013, pp. 9-20): Springer Berlin / Heidelberg.

Dannenberg, Roger, & Mazzoni, Dominic (2014). Audacity, a free audio editor. Retrieved 2014, August 4, from http://audacity.sourceforge.net/

Deci, EL, & Ryan, RM. (2005). Intrinsic motivation inventory (IMI). *Retrieved July, 23*, 2006.

Demetrulias, Diana Mayer. (1985). Gender Differences and Computer Use. *Educational Horizons, 63*(3), 133-135.

Design-Based Research Collective. (2003). Design-Based Research: An Emerging Paradigm for Educational Inquiry. *Educational Researcher, 32*(1), 5-8. doi: 10.3102/0013189x032001005

Dougiamas, Martin Moodle - Modular Object-Oriented Dynamic Learning Environment. Retrieved 2013, January 8, from https://moodle.org/

Entertainment Software Association (2015). Essential Facts About the Computer and Video Game Industry. Retrieved 2015, August 28, from http://www.theesa.com/wp-content/uploads/2015/04/ESA-Essential-Facts-2015.pdf

Epic Games UDK Commercial Licensing. from http://www.unrealengine.com/udk/licensing/

Epic Games (n.d.-a). UDK Licensing. from http://www.unrealengine.com/udk/licensing/

Epic Games (n.d.-b). Unreal Development Kit. Retrieved 2013, August 25, from http://www.udk.com/

Epic Games (n.d.-c). Unreal Scripting. Retrieved 2013, August 25, from http://udn.epicgames.com/Three/UnrealScriptReference.html

eTiTan (2012). iodoom3 Forum - [Tutorial] Compile in Visual Studio 2010 Express. Retrieved 2013, August 24, from http://www.iodoom3.org/forums/topic/tutorial-compile-in-visual-studio-2010-express/

Field, Andy. (2009). *Discovering statistics using SPSS*: Sage publications.

Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme IAIS (n.d.). Website - Roberta – Lernen mit Robotern. Retrieved 2015, September 9, from http://www.open-roberta.org/

Free Software Foundation GNU Lesser GPL. Retrieved 2012, October 28, from http://www.gnu.org/copyleft/lesser.html

Free Software Foundation (n.d.). GNU General Public License. Retrieved 2012, October 28, from http://www.gnu.org/licenses/gpl.html

Gadd, Kevin (2009). Squared.Tiled. Retrieved 2015, September 9, from http://luminance.org/

Gamasutra Mobile game developer survey leans heavily toward iOS, Unity. Retrieved 2013, August 24, from http://www.gamasutra.com/view/news/169846/Mobile_game_developer_survey_leans_heavily_toward_iOS_Unity.php#.UFze0I1lTAE

Gamefroot (2013). Gamefroot. Retrieved 2015, August 30, from http://gamefroot.com/

Gamesm, Xbox Live Indie Website - XNA Starterkits. Retrieved 2013, August 22, from http://xbox.create.msdn.com/en-US/education/starterkits/

GAMING, BACKGROUND OF. (2005). Games: Making learning fun. *Annual Review of Nursing Education Volume 3, 2005: Strategies for Teaching, Assessment, and Program Planning*, 165.

Gernert, Johannes. (2010). *Generation Porno: Jugend, Sex, Internet*: Fackelträger-Verlag.

Glass et. al., Kevin (n.d.). Slick 2D Game Library. Retrieved 2012, September 2, from http://slick.cokeandcode.com/index.php

Google Inc. Website - Blockly. from https://blockly-games.appspot.com

GPWiki (n.d.). List of game engines. Retrieved 2012, October 28, from http://en.wikipedia.org/wiki/List_of_game_engines

GRGORG 16 (n.d.). Stundentafel des GRGORG 16, Maroltingergasse 69-71. Retrieved 2015, August 24, from http://www.g16.at/node/9

Halbeisen, Sarah. (2011). *Ernste Spiele? Eine Systematisierung von Serious Games*.

Hidi, Suzanne. (1990). Interest and its contribution as a mental resource for learning. *Review of Educational research, 60*(4), 549-571.

Huang, Jun. (2010, 24-27 Aug. 2010). *Improving undergraduates' teamwork skills by adapting project-based learning methodology.* Paper presented at the Computer Science and Education (ICCSE), 2010 5th International Conference on.

id Software, & iodoom3 (2013). iodoom3 - Id Tech 4 engine (Doom 3 Engine). Retrieved 2014, August 3, from http://www.iodoom3.org/

iddevnet Making Doom3 Mods - Scripting. Retrieved 2013, August 24, from http://www.iddevnet.com/doom3/script.php

Irrelon Software (n.d.). Isogenic Engine. Retrieved 2012, October 7, from http://www.isogenicengine.com/

Johnston, Chevy Ray FlashPunk. Retrieved 2012, October 28, from http://flashpunk.net/

Kay, Alan. (2005). Squeak etoys, children & learning. *online article, 2006*.

Kimball et. al., Spencer (2014). GIMP - The GNU Image Manipulation Program. Retrieved 2012, August 9, from http://www.gimp.org

Kölbl, Doris Austrian Federal Ministry for Education, Arts and Culture - Ordinance governing the principles of project centered teaching. Retrieved 2012, July 3, from http://www.bmukk.gv.at/medienpool/6788/pu_erl_engl.pdf

Kronos Group OpenGL. Retrieved 2013, August 28, from 2013, August 28

Lasselsberger, Anna, Gschwendtner, Ferdinand, & Bundesministerium für Bildung und Frauen (2015). VWA - Vorwissenschaftliche Arbeit. Retrieved 2016, Jan 1, from http://www.ahs-vwa.at/

Lave, Jean, & Wenger, Etienne. (2005). *Situated learning legitimate peripheral participation* (1. publ., reprint. ed.). Cambridge [u.a.]: Cambridge Univ. Press.

Limayem, Moez, Khalifa, Mohamed, & Frini, Anissa. (2000). What makes consumers buy from Internet? A longitudinal study of online shopping. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, 30*(4), 421-432.

Lindeijer, Thorbjørn Tiled Map Editor. Retrieved 2012, October 7, from http://www.mapeditor.org/

Lindeijer, Thorbjørn, & Cook, Daniel (2013). Tiled map editor. Retrieved 2013, March 19, 2013, from http://www.mapeditor.org/

Lingard, R., & Barkataki, S. (2011, 12-15 Oct. 2011). *Teaching teamwork in engineering and computer science.* Paper presented at the Frontiers in Education Conference (FIE), 2011.

Linhoff, Joe, & Settle, Amber. (2008). Teaching game programming using XNA. *ACM SIGCSE Bulletin, 40*(3), 250-254.

Lock, Dennis. (1995). *Project management* (5. ed., reprint. ed.). Aldershot, Hants. [u.a.]: Gower.

Logo Foundation (2011). Logo. Retrieved 2014, June 23, from http://el.media.mit.edu/logo-foundation/logo/index.html

Lonati, Violetta, Monga, Mattia, Morpurgo, Anna, & Torelli, Mauro. (2011). What's the Fun in Informatics? Working to Capture Children and Teachers into the Pleasure of Computing. Informatics in Schools. Contributing to 21st Century Education. In I. Kalaš & R. Mittermeir (Eds.), (Vol. 7013, pp. 213-224): Springer Berlin / Heidelberg.

Magdalena Mattebo, RNM, Tydén, Tanja, Häggström-Nordin, Elisabet, Nilsson, Kent W, & Larsson, Margareta. (2013). Pornography Consumption, Sexual Experiences, Lifestyles, and Self-rated Health Among Male Adolescents in Sweden.

Malone, Thomas W, & Lepper, Mark R. (1987). Making learning fun: A taxonomy of intrinsic motivations for learning. *Aptitude, learning, and instruction, 3*(1987), 223-253.

Malone, Thomas W. (1980). *What makes things fun to learn? heuristics for designing instructional computer games*. Paper presented at the Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems, Palo Alto, California, USA.

Maloney, John, Resnick, Mitchel, Rusk, Natalie, Silverman, Brian, & Eastmond, Evelyn. (2010). The Scratch Programming Language and Environment. *Trans. Comput. Educ., 10*(4), 1-15. doi: 10.1145/1868358.1868363

Massachusetts Institute of Technology The MIT License Retrieved 2012, November 11, from http://opensource.org/licenses/MIT

Maxcy, S.J. (2003). Pragmatic threads in mixed methods research in the social sciences: The search for multiple modes of inquiry and the end of the philosophy of formalism. *Handbook of mixed methods in social and behavioral research*, 51-89.

Michael, David R, & Chen, Sandra L. (2005). *Serious games: Games that educate, train, and inform*: Muska & Lipman/Premier-Trade.

MIT Scheller Teacher Education Program (2014). Gameblox. Retrieved 2015, September 9, from https://gameblox.org/

Mönig, Jens, & Harvey, Brian BYOB - Build Your Own Blocks. Retrieved 2013, April 1, from http://byob.berkeley.edu/

Mönig, Jens, & Harvey, Brian (2013). Website - Snap! - formerly known as BYOB. Retrieved 2013, March 30, 2013, from http://snap.berkeley.edu/

MonoGame Team (2009). MonoGame - An open source implementation of Microsoft's XNA. Retrieved 2013, April 1, from http://monogame.codeplex.com/

Moser, Heinz. (1977). *Praxis der Aktionsforschung ein Arbeitsbuch*. München: Kösel.

Motschnig-Pitrik, R., & Figl, K. (2007, 10-13 Oct. 2007). *Developing team competence as part of a person centered learning course on communication and soft skills in project management.* Paper presented at the Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE '07. 37th Annual.

National Institute of Standards and Technology Finite State Machine. Retrieved 2013, August 21, from http://xlinux.nist.gov/dads/HTML/finiteStateMachine.html

Nischewitzer, Alexander, Reimers, Christian, Smith, David, Zistler, Elisabeth, & Zistler, Saphir Programming Skills Development in Secondary Education by means of Modern Educational Programming Languages. Retrieved 2013, January 2, from http://virtuelleschule.bmukk.gv.at/fileadmin/pskills/pSkills-Scratch.pdf

Obama, Barack TechHire Initiative. Retrieved 2015, August 14, from https://www.whitehouse.gov/issues/technology/techhire

OGRE Object-Oriented Graphics Rendering Engine. Retrieved 2012, June 24, from http://www.ogre3d.org/

Open Source Initiative Microsoft Public License (Ms-PL). Retrieved 2013, August 24, from http://opensource.org/licenses/MS-PL

Oracle Corp. Netbeans Integrated Development Environment. Retrieved 2012, October 28, from http://netbeans.org/

Oracle Corp. et. al. MySQL - The world's most popular open source database. Retrieved 2013, August 30, from http://www.mysql.com/

OUYA Inc. Website - OUYA console. Retrieved 2013, August 28, from http://www.ouya.tv/

Overby, Jeffrey W, & Lee, Eun-Ju. (2006). The effects of utilitarian and hedonic online shopping value on consumer preference and intentions. *Journal of Business Research, 59*(10), 1160-1166.

Owens, Eric W, Behun, Richard J, Manning, Jill C, & Reid, Rory C. (2012). The impact of internet pornography on adolescents: a review of the research. *Sexual Addiction & Compulsivity, 19*(1-2), 99-122.

Papert, Seymour. (1987). *Microworlds: transforming education.* Paper presented at the Artificial intelligence and education.

Phrogsoft (2012). Phrogram. Retrieved 2013, August 13, from http://phrogram.com/

Przybylski, A. K., Weinstein, N., Murayama, K., Lynch, M. F., & Ryan, R. M. (2012). The Ideal Self at Play: The Appeal of Video Games That Let You Be All You Can Be. *Psychological Science, 23*(1), 69-76. doi: 10.1177/0956797611418676

Rankin, Yolanda, Gooch, Amy, & Gooch, Bruce. (2008). *The impact of game design on students' interest in CS*. Paper presented at the Proceedings of the 3rd international conference on Game development in computer science education, Miami, Florida.

Reeve, Johnmarshall. (2009). *Understanding motivation and emotion* (5. ed. ed.). Hoboken, N.J.: Wiley.

Reeves et. al., Thomas C. What is Design-based Research? Retrieved 2013, January 6, from What is Design-based Research?

Reichert, Raimond (2014). Website - Kara. Retrieved 2014, August 8, from http://www.swisseduc.ch/informatik/karatojava/index.html

Reichert, Thorsten. (2009). *Projektmanagement. Die häufigsten Fehler, die wichtigsten Erfolgsfaktoren*. Freiburg: Haufe Verlag.

Reismanis, Scott moddb. Retrieved 2012, October 28, from http://www.moddb.com/engines

Renninger, Ann, Hidi, Suzanne, & Krapp, Andreas. (2014). *The role of interest in learning and development*: Psychology Press.

Repenning, Alexander. (2011). *Making programming more conversational.* Paper presented at the VL/HCC.

Repenning, Alexander, & Agent Sheets Inc. (2014a). AgentCubes. Retrieved 2015, April 18, from http://www.agentsheets.com/agentcubes/index.html

Repenning, Alexander, & Agent Sheets Inc. (2014b). AgentSheets. Retrieved 2015, April 18, from http://www.agentsheets.com/

Repenning, Alexander, & Sumner, Tamara. (1995). Agentsheets: A medium for creating domain-oriented visual languages. *Computer, 28*(3), 17-25.

Repenning, Alexander, Webb, David, & Ioannidou, Andri. (2010). *Scalable game design and the development of a checklist for getting computational thinking into public schools.* Paper presented at the Proceedings of the 41st ACM technical symposium on Computer science education.

Resnick et. al., M. Research | Scratch Documentation Site - Academic publications & presentations. Retrieved 2012, February 18, from http://info.scratch.mit.edu/Research

Resnick et. al., Mitchel Scratch. Retrieved 2012, June 23, from http://scratch.mit.edu/

Resnick et. al., Mitchel. (2009). Scratch: programming for all. *Commun. ACM, 52*(11), 60-67. doi: 10.1145/1592761.1592779

Resnick, Mitchel, Malony, J., Rusk, N. , Silverman, B., & Eastmond, E Scratch. Retrieved 2012, June 23, from http://scratch.mit.edu/

Ritterfeld, Ute, Cody, Michael, & Vorderer, Peter. (2010). *Serious games: Mechanisms and effects*: Routledge.

Ryan, Richard M, Rigby, C Scott, & Przybylski, Andrew. (2006). The motivational pull of video games: A self-determination theory approach. *Motivation and emotion, 30*(4), 344-360.

Saltsman, Adam Flixel (Flash Game Engine). Retrieved 2012, October 28, from http://flixel.org/

Sandoval, William A., & Bell, Philip. (2004). Design-Based Research Methods for Studying Learning in Context: Introduction. *Educational Psychologist, 39*(4), 199-201. doi: 10.1207/s15326985ep3904_1

Shashaani, Lily. (1994). Gender-differences in computer experience and its influence on computer attitudes. *Journal of Educational Computing Research, 11*(4), 347-367.

Shein, Esther. (2014). Should <i>everybody</i> learn to code? *Commun. ACM, 57*(2), 16-18. doi: 10.1145/2557447

Shen, Yuzhong. (2009). *Teaching game development using microsoft XNA game studio*. Paper presented at the Proceedings of the 2009 Spring Simulation Multiconference, San Diego, California.

Skinner, B. F. (1953). *Science and human behavior*: The B.F. Skinner Foundation.

Snow, CR. (1992). *Concurrent programming* (Vol. 26): Cambridge University Press.

Software, Adobe Flash Builder. Retrieved 2012, September 12, from http://www.adobe.com/products/flash-builder.html

Sperl, Daniel, & Weissböck, Holger The Starling Framework - An open source game engine for Flash. Retrieved 2012, September 12, from http://gamua.com/starling/

Statistik Austria (2010). Zeitverwendungserhebung der Statistik Austria. Freizeit in Österreich [Sparetime in Austria]. Retrieved 2012, April 7, from http://www.statistik.at/web_de/presse/052105

Statistik Austria (2013). IKT-Einsatz in Haushalten 2013. Retrieved 2016, April 12, from http://www.statistik.at/web_de/statistiken/informationsgesellschaft/ikteinsatz_in_haushalten/index.html

Statistik Austria (2014). Haushalte mit Internetzugang 2014. from https://www.statistik.at/web_de/statistiken/energie_umwelt_innovation_mobilitaet/informationsgesellschaft/ikt-einsatz_in_haushalten/022213.html

Stowasser, Louis CraftyJS "Crafty, code like a fox". Retrieved 2012, October 7, from http://craftyjs.com/

Strickland, Bonnie. (2001a). Experimental Design. In B. Strickland (Ed.), *The Gale Encyclopedia of Psychology* (2nd ed. ed., pp. 234). Detroit: Gale.

Strickland, Bonnie. (2001b). Play. *The Gale Encyclopedia of Psychology*, 503-504. http://go.galegroup.com/ps/i.do?id=GALE%7CCX3406000503&v=2.1&u=43wien&it=r&p=GVRL&sw=w

Sung, Kelvin The Game-Themed Introductory Programming Project. Retrieved 2012, June 24, from http://depts.washington.edu/cmmr/Research/XNA_Games/index.php

Susi, Tarja, Johannesson, Mikael, & Backlund, Per. (2007). Serious games: An overview.

Swiss Educ Team Website - Kara Screenshots. Retrieved 2013, August 21, from http://www.swisseduc.ch/informatik/karatojava/kara/screenshots.html

Taylor, M. J., & Baskett, M. (2009). The science and art of computer games development for undergraduate students. *Comput. Entertain., 7*(2), 1-9. doi: 10.1145/1541895.1541904

Technical University of Vienna Curriculum for the bachelor studies computer science. Retrieved 2012, August 29, from http://www.informatik.tuwien.ac.at/lehre/studienplaene/informatik-archiv/informatik-studienplan-2011

Thomson, Elizabeth S, & Laing, Angus W. (2003). "The Net Generation": Children and Young People, the Internet and Online Shopping. *Journal of Marketing Management, 19*(3-4), 491-512.

Tiigi et. al., Tõnis LimeJS. Retrieved 2012, October 7, from http://www.limejs.com/

Unity Technologies Unity3D - Island Demo. Retrieved 2013, August 29, from http://download.unity3d.com/gallery/demos/demos/IslandDemo.zip

Unity Technologies Unity - Creating And Using Scripts. Retrieved 2013, August 24, from http://docs.unity3d.com/Documentation/Manual/CreatingAndUsingScripts.html

Unity Technologies Unity - Multiplatform. Retrieved 2013, August 24, from http://unity3d.com/unity/multiplatform/

Unity Technologies Unity - Store. Retrieved 2013, August 24, from https://store.unity3d.com/

Unity Technologies Unity game engine. Retrieved 2013, August 24, from http://unity3d.com/

Unity Technologies Unity Web Player. Retrieved 2013, August 29, from http://unity3d.com/webplayer

University of California Berkeley BSD License. Retrieved 2012, October 28, from http://opensource.org/licenses/BSD-2-Clause

University of Vienna - Senate, Commission for the curriculum Curriculum for the bachelor studies computer science. Retrieved 2012, August 29, from http://www.univie.ac.at/mtbl02/2005_2006/2005_2006_193.pdf

Vepsäläinen, Juho Game engine feature matrix. 2012, October 28, from https://github.com/bebraw/jswiki/wiki/Game-engine-feature-matrix

Volk, Daniel. (2008). *How to embed a game engineering course into a computer science curriculum*. Paper presented at the Proceedings of the 2008 Conference on Future Play: Research, Play, Share, Toronto, Ontario, Canada.

W3C DOM - Document Object Model. Retrieved 2012, October 28, from http://www.w3.org/DOM/

W3C Website - HTML 5 - Canvas element Retrieved 2012, October 28, from http://www.w3.org/TR/html5/the-canvas-element.html#the-canvas-element

Wang, Alf Inge, & Wu, Bian. (2011). Using Game Development to Teach Software Architecture. *International Journal of Computer Games Technology, 2011*. doi: 10.1155/2011/920873

Watson, J. B., & Rayner, R. (1920). Conditioned emotional reactions. *Journal of Experimental Psychology, 3*, 1-14.

Weber, Jeff (2013). Farseer Physics Engine. Retrieved 2013, August 13, from https://farseerphysics.codeplex.com/

Website - .NET framework. Retrieved 2013, August 14, from http://www.microsoft.com/net

Website - Achtung die Kurve! Retrieved 2013, August 14, from http://en.wikipedia.org/wiki/Achtung,_die_Kurve!

Website - AntMe Retrieved 2013, January 2, from http://antme.net/

Website - Content Pipeline Architecture. Retrieved 2013, August 22, from http://msdn.microsoft.com/en-US/library/bb447745(v=xnagamestudio.10).aspx

Website - Doom (series). Retrieved 2013, August 24, from http://en.wikipedia.org/wiki/Doom_(series)

Website - Greenfoot. Retrieved 2013, January 2, from http://greenfoot.org/

Website - iodoom3 - Open source, cross platform Doom 3 (GPLv3). Retrieved 2013, August 24, from https://github.com/iodoom/iod3

Website - iodoom3 FAQ - What is iodoom3? Retrieved 2013, August 24, from http://www.iodoom3.org/faq/

Website - KidsProgrammingLanguage KPL Retrieved 2013, January 2, from http://www.kidspl.de/lang/examples/grafik.php

Website - Lego Mindstorms. Retrieved 2013, January 2, from http://mindstorms.lego.com/en-us/default.aspx

Website - Microsoft Public License (Ms-PL). Retrieved 2013, August 24, from http://www.microsoft.com/en-us/openness/licenses.aspx

Website - Microsoft Visual Studio. Retrieved 2013, August 14, from http://www.microsoft.com/visualstudio/eng/downloads

Website - Mono - an open source implementation of Microsoft's .NET Framework. Retrieved 2013, August 24, from http://www.mono-project.com/Main_Page

Website - MonoDevelop Retrieved 2013, August 19, from http://monodevelop.com/

Website - Scratch 1.4 Infos on ScratchWiki. Retrieved 2013, August 20, from http://wiki.scratch.mit.edu/wiki/1.4

Website - Scratch 2.0 - Cloning. Retrieved 2013, August 20, from http://wiki.scratch.mit.edu/wiki/Cloning

Website - Scratch 2.0 - Cloud Data. Retrieved 2013, August 20, from http://wiki.scratch.mit.edu/wiki/Cloud_Data

Website - Scratch 2.0 - Procedures/Own Blocks. 2013, August 20, from http://wiki.scratch.mit.edu/wiki/Procedures

Website - Scratch 2.0 Wiki Retrieved 2013, August 20, from http://wiki.scratch.mit.edu/wiki/Scratch_2.02

Website - Squeak. Retrieved 2013, August 20, from http://www.squeak.org/

Website - Standard Content Importers and Content Processors. Retrieved 2013, August 22, from http://msdn.microsoft.com/en-us/library/bb447762(v=xnagamestudio.42).aspx

Website - Unreal series. Retrieved 2013, August 25, from http://en.wikipedia.org/wiki/Unreal_(series)

Website - What is Scratch? Retrieved 2013, August 14, from http://wiki.scratch.mit.edu/wiki/Scratch

Website - Xbox live - Indie Games. from http://xbox.create.msdn.com/en-us/home/about/how_it_works

Website - XNA Game Studio 4.0. Retrieved 2012, June 24, from http://www.microsoft.com/en-us/download/details.aspx?id=23714

Website - XNA Game Studio 4.0. (2011). Retrieved 2013, August 22, from http://msdn.microsoft.com/en-us/library/bb200104(v=xnagamestudio.40).aspx

Wendel, Daniel (project manager) (2010). Star Logo TNG. Retrieved 2013, January 2, from http://education.mit.edu/drupal/starlogo-tng

Wendel, Viktor, Göbel, Stefan, & Steinmetz, Ralf. (2010). *Game Design und Game Development in einer Serious Games Vorlesung.* Paper presented at the Mensch & Computer Workshopband.

Wilensky, Uri NetLogo. Retrieved 2012, June 23, from http://ccl.northwestern.edu/netlogo/

Williams, David, South, Joseph, Yanchar, Stephen, Wilson, Brent, & Allen, Stephanie. (2011). How do instructional designers evaluate? A qualitative study of evaluation in practice. *Educational Technology Research and Development, 59*(6), 885-907. doi: 10.1007/s11423-011-9211-8

Woei-Kae, Chen, & Yu Chin, Cheng. (2007). Teaching Object-Oriented Programming Laboratory With Computer Game Programming. *Education, IEEE Transactions on, 50*(3), 197-203. doi: 10.1109/te.2007.900026

Xamarin Inc. Website - Xamarin. Retrieved 2013, August 28, from http://xamarin.com/

Xamarin Inc. Website - Xamarin Studio. Retrieved 2013, August 28, from http://xamarin.com/studio

Yee, Nick. (2006). The labor of fun how video games blur the boundaries of work and play. *Games and Culture, 1*(1), 68-71.

Youngblood, G. Michael (n.d.). Using XNA-GSE Game Segments to Engage Students in Advanced Computer Science Education Retrieved 2012, September 2, from ftp://ftp.research.microsoft.com/pub/carlat/N14_Using%20XNA-GSE%20game%20segments%20to%20Engage%20Students.pdf

Zillich, Norbert. (2011). Pornography Consumption among Juveniles and Flexibilization of Gender Roles. *ZEITSCHRIFT FUR SEXUALFORSCHUNG, 24*(4), 312-325.

Appendix

Game development was embedded into a Moodle course. This course can be downloaded from http://comber.at/gamepins/mc_gamepins.mbz (September 30, 2015). The GamePinS sample projects, which include the template structures and necessary files, are available at http://comber.at/gamepins/Beispiele/GamePinS.zip (September 30, 2015).

The Moodle course can be visited online and accessed as a guest. The course can be found under http://www.comber.at/gpmoodle (September 30, 2015), and the password for guest access is *MC_GamePinS_T3mplate.*

Data and feedback from the 2012 game development

Results and original feedback from students can be found online. The username and password for all supplements are the same, i.e., username is *gameprog* and password is *2012GamePin$*.

General Questionnaire 2012 – 2015:
http://www.comber.at/research/results/fragebogen_allg.htm (October 30, 2015)

Game Development with XNA 2012:
http://www.comber.at/research/results2012/SurveyB_2012.htm (October 30, 2015)

Statements of students 2012:
http://www.comber.at/research/results2012/statements.html (October 30, 2015)

Game Development 2014:
http://www.comber.at/research/results2014/SurveyAandB_2014.htm (October 30, 2015)

Thank you for checking out the digital supplements above and helping to save paper!