Patterns in Mainstream Programming Games

Ander Areizaga Blanco[1], Henrik Engström[2]

Division of Game Development
University of Skövde, Sweden
[1]*anderareizaga1996@gmail.com*
[2]*henrik.engstrom@his.se*

Abstract

Studies have found serious games to be good tools for programming education. As an outcome from such research, several game solutions for learning computer programming have appeared. Most of these games are only used in the research field where only a few are published and made available for the public. There are however numerous examples of programming games in commercial stores that have reached a large audience. This article presents a systematic review of publicly available and popular programming games. It analyses which fundamental software development concepts, as defined by the ACM/IEEE Computer Science Curricula, are represented in these games and identifies game design patterns used to represent these concepts.

This study shows that fundamental programming concepts and programming methods have a good representation in mainstream games. There is however a lack of games addressing data structures, algorithms and design. There is a strong domination of puzzle games. Only two of the 20 studied games belong to a different genre. The eleven game design patterns identified in this study have potential to contribute to future efforts in creating engaging serious games for programming education.

Keywords: Serious Games, Programming, Learning, Game Design Patterns

1 Introduction

The use of serious games in formal education has been studied with different perspectives [1, 2] and for different subject areas. Programming is no exception and there are several examples of how games have been used to teach programming concepts [3–6].

Most studies of programming games are presenting games developed within a research context [7]. The focus of such studies is mainly to assess the learning outcome. The experiential aspects of these games are given less focus, as is the case for serious games in general [8, 9]. Miljanovic and Bradbury [7] review 36 studies of programming games. Player engagement was evaluated only in 11 of those studies. They also highlight the lack of access to games developed in research projects and argue for the need of third-party evaluations to avoid self-confirmatory bias.

This article presents a third-party evaluation of programming games with an explicit focus on games that are publicly available, and that have reached a mainstream audience. The assumption is that popular programming games have managed to provide an engaging player experience. In the study presented in this article, 20 programming games have been systematically selected from the commercial stores *Steam* [10] and *Google Play* [11]. These games have been analysed with respect to the ACM/IEEE Computer Science Curricula [12] and its

section *Software Development Fundamentals* to identify programming concepts covered by these games. They have also been analysed with respect to their gameplay and 11 game design patterns [13] have been identified. The result of this work shows that there exist approaches to represent programming concepts in games that have received a large player interest. The study also shows that there are programming areas where none, or very few such examples exist.

2 Background

Programming for software development is a task of understanding a certain existing problem followed by the process of designing and coding a suitable solution. The last step involves three main competences: comprehension; writing and debugging [14]; and, maintenance [15].

In recent years, there has been an increased interest in teaching programming outside traditional software development. Wing [16] proposed that *computational thinking* is a skill that is applicable in many other areas. This idea has been accepted and incorporated into technology education. There have also been objections that question the notion of computational thinking, how it is defined and how it is measured [17]. The present article is focused programming from a software development perspective with a *traditional computational thinking*, as discussed by Denning [17]. For the purpose of this article, we will use the ACM/IEEE Computer Science Curricula [12] and the section *Software Development Fundamentals* to define programming concepts that potentially can be covered by a game.

2.1 Software development fundamentals

The *software development fundamentals* [12] in the ACM/IEEE Computer Science Curricula is divided into 4 main units (Table 1): algorithms and design; fundamental programming concepts; fundamental data structures; and, development methods. Under these main units, several central topics are listed.

The unit *algorithms and design* relates to fundamental principles of programming. This includes general aspects connected to computational thinking [17].

The unit *fundamental programming concepts* is related to basic concepts of programming languages. This includes syntax and semantics of control structures, functions, variables and data types. The topics represent concepts recurring in many different computational models. This includes variables used to store data. The fundamental unit covers primitive data types such as integers, floating point numbers, Boolean, and characters. The unit also covers the programming structure composed of statements, such as expressions and assignments, but also control structure mechanisms, such as iterations and conditional statements. The concept of functions is also central for the control flow. Functions allows program execution to be declared and invoked using parameters which will control the result of the function. Recursion is an important special case of functions, where the definition of the function is self-referential [18]. The last topic in the fundamental programming concepts is related to input and output (I/O) from, and to secondary storage.

The unit *fundamental data structures* focuses on structures for representing and storing data. The topics relate to common data structures and strategies for selecting them. Common data structures such as arrays (a list elements of the same type) and structs (a collection of elements that can be of different types) are included as well as the representation and processing of strings which are used to represent text. Abstract data types (ADT) can be seen as algebraic structures with a set of values and a set of operations. Common examples of ADTs include set, stack, and queue.

Table 1: *ACM/IEEE Computer Science Curricula, software development fundamentals [12]. The rightmost column shows patterns identified in this article that cover the corresponding topic.*

Unit	Topics	Patterns
Algorithms and design	Concept and properties of algorithms	
	Role of algorithms in the problem-solving process	
	Problem-solving strategies	
	Fundamental design concepts and principles	
Fundamental programming concepts	Basic syntax and semantics of a higher-level language	10
	Variables and primitive data types	
	Expressions and assignments	1, 5
	Simple I/O including file I/O	9
	Conditional and iterative control structures	1
	Functions and parameter passing	2
	The concept of recursion	2
Fundamental data structures	Arrays	
	Records/structs	
	Strings and string processing	
	Abstract data types	
	References and aliasing	
	Linked lists	
	Strategies for choosing the appropriate data structure	
Development methods	Program comprehension	1, 6, 7
	Program correctness	8
	Simple refactoring	7, 11
	Modern programming environments	
	Debugging strategies	4, 8
	Documentation and program style	3, 6

The unit *development methods* relates to the working practices of programmers, covering topics related to the tools, procedures and conventions that have evolved in the relatively short history of software development.

2.2 Learning programming

Programming has traditionally been considered a difficult topic [19]. This is one of the main reasons that several research projects have tried to find a working learning process for programming. Vahldick et al. [14] identify three essential competencies for the learning process to be complete: comprehension, writing and debugging. These competencies are transversal, which means that the different skills may overlap and it will not be possible to acquire a whole competence at a time. Lau and Yuen [20] identify seven pedagogical approaches to teaching computer programming, based on literature: Structured programming approach; Problem solving approach; Software development approach; Small programming approach; Language teaching approach; Learning theory approach; and, Other approaches. Each of these approaches is based on rules defined for novices to start their learning process. It is not specified which one of the approaches that will be the most effective. Lau and Yuen highlight that classroom evaluation is lacking for many approaches. However, it is addressed that the effectiveness depends on the individual. Some approaches are successful for weaker students but give very little challenge to stronger students. One observation made by Lau and Yuen is that the trend in programming teaching is to use visual and media-rich environments to a larger degree. Games are however not mentioned [20].

2.3 Serious games for programming teaching

Games can be used in two ways to learn programming: by creating them or by playing them. Of these two approaches, the second one is closest to the definition of serious games and it is the focus of this article.

Vahldick et al. [14] and Miljanovic and Bradbury [7] review several game solutions for teaching programming. The latter extends upon the former and includes more games and a broader analysis. Most of the games reviewed in both studies come from research projects. A minority of the studied games come from commercial stores like *App Store* [21] and *Google Play* [11]. In order to find what topics related to programming are being taught with these games, they analyse them against the ACM/IEEE Computer Science Curricula from 2013. The results from these studies reveal some gaps with respect to the curricula. A problem highlighted [7, 14] is that most games presented in research projects are unavailable for download and evaluation. Most evaluations of such games are conducted by the researchers that have developed the game, which introduces the risk for self-confirmatory bias [7]. The evaluations in previous game reviews are only partly based on playtests. For example, less than half of the games evaluated by Miljanovic et al. [7] were accessible for playtest.

2.3.1 Programming game examples

Galgouranas and Xinogalos [4] present *jAVANT-GARDE*, a serious game solution for teaching programming with Java. The game element in jAVANT-GARDE is a combination of quizes and puzzle that requires code to be written to progress. An evaluation with 42 high school students show that most students prefer the game to other teaching alternatives. jAVANT-GARDE is available at Google Play but it has been downloaded less than 1000 times.

Lindberg and Laine [5] present *Minerva* which is a 2D game including both action and puzzle elements. The game has been evaluated on 6th grade students with mixed results. Minerva is not available for public download.

Tsarava et al. [22] present three board games designed to teach primary school children computational thinking. They present a feasibility study using adult players that gave some positive indications for the approach.

Coelho et al. [3] present a platform for deploying different programming games. They present a system that is partly realised and where some initial tests indicate that the prototype game needs additional features. The platform is not available for the public.

Finally, RoboBug is a serious game intended for students that are learning to debug programs. It is a puzzle-type game where the puzzles that appear inside the game will require the player to search for different bugs in the programs and solve them [6]. The source code for RoboBug is publicly available through GitHub.

2.4 Game design patterns

The concept of game design patterns was developed by Björk, Lundgren and Holopainen [13]. They define game design patterns as "descriptions of reoccurring interaction relevant to game play" [13, p.180]. Game design patterns are proposed as tools that can be used not only for problem-solving but also for supporting creative design works. For identifying a game pattern, they developed a structural framework. This framework consists of the following sections: name, description, consequences, using the pattern and relations [13].

An example of a common game design patterns is *Rock-Paper-Scissors*. This pattern is based on the non-transitive relation of three different options inside a game: A (e.g. rock), B (e.g. scissors), and C (e.g. paper). The relation is such that A wins over B, B wins over C, and C wins over A. This pattern is used in many games, for example, Pokémon and Quake. This pattern appears in the type-system of Pokémons where each of them will belong to a certain type or combination of types. There is a non-transitive relation between types that defines if a type will be weak or strong against another. The same happens in Quake where the weapons will belong to a certain type that will deal more or less damage depending on the monster it is used against.

3 Problem and Method

Games specifically targeting programming teaching have become common. Many of these games (e.g. [5, 6]) have been developed in a research context. A problem that appears when evaluating such games, is that, sometimes the authors do not offer readers an opportunity to play them. As an example, Miljanovic and Bradbury [7] states that *May's Journey* was not available to download during their evaluation process. Even if these games are available, they usually fail to reach a larger audience. The reason for this may be limitations in the distribution channel, lack of promotion, or simply that the game-play is not sufficiently appealing. As an example, White, Thian, and Smith [23] reported that 75% of their participants found the programming game they studied to be "not at all entertaining". None of the participants found it to be "very entertaining". There are however many examples of programming games that have been developed outside a research context and that target leisure users. These games are available at commercial stores such as Steam [10], App Store [21], and Google Play[11]. Some of these games have reached a very large audience.

The aim of this article is to analyse the most popular programming games at commercial game stores. A serious game, as any game, should provide an interesting experience to players. This is very hard to evaluate and is many times overlooked in academic studies of serious games [8]. As an example, Perttula et al. [9] conducted a systematic literature review where they studied the use of flow evaluations in serious games research. The conclusion of this review was that "surprisingly limited number of empirical studies on flow in serious games

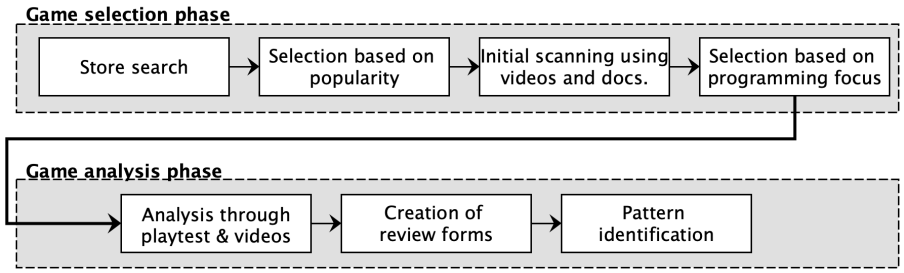

Figure 1: *Search and review process.*

context was found" (p.68). One of their identified examples on how flow can be evaluated is a flow framework for educational games, proposed by Kiili et al. [24, 25]. Although flow may be the most commonly used conceptualisation, it does not alone capture the full spectrum of gameplay experience [26]. By focusing on the most popular programming games, our approach has a potential to capture successful game designs, irrespective of player motivation. The other aspect of a serious game, that it should contribute to the achievement of some purpose, is addressed in this study by analysing how games represent software development fundamentals. This resembles the approach taken in [7]. The actual short- and long-term learning from these games are outside the scope of this study.

The main goal is to analyse successful programming games and explore which topics are covered in them, and to identify game design patterns [13], used to cover such topics. The research question of this study can be formulated as: "which fundamental programming concepts are represented in successful programming games and what game design patterns are used to represent them?".

3.1 Search and review process

The method used in this study is based on the approaches presented by Miljanovic and Bradbury [7] and Vahldick et al. [14]. However, in these articles searches were mostly focused on research databases and the review focused on the result from studies. In this study, the games come from commercial stores and the empirics primarily come from game analysis by playing each game.

The process used is presented in Figure 1. It was divided in two major phases. In the first phase games were selected and in the second phase these games were played and analysed.

3.1.1 Game selection

The first list of games was acquired through a search in the commercial stores Steam [10] and Google Play [11]. The choice of stores is motivated by their number of users and that they represent two different hardware platforms – PC and mobile. Steam is the most popular store used for acquiring commercial games for computers, with peaks of 16 million concurrent users [27], while Google Play is the most popular store for downloading games to Android devices – the biggest mobile platform. The Apple App Store was excluded due to the large overlap of games with Google Play store.

In Steam, games were selected using the tag *programming* and the tag *game*. They were sorted based on the approximate number of downloads (primary) and the popularity rating (secondary). This was made using steamspy.com [28] and Playtracker [29]. It generated a list of 30 games that were manually reviewed in the subsequent step.

In Google Play store, games were searched using the keywords *programming* and *games*. This search did not only result in actual games but it also returned other types of apps focused

on teaching programming. These non-game apps were discarded. The games were sorted by the total number of downloads and the 15 most downloaded games were manually reviewed in the subsequent step. The reason for selecting fewer games from Google Play compared to Steam was that there are substantially more games for the PC platform.

The initial scanning of the two lists, 30 games from Steam and the 15 games from Google Play, used descriptions given by developers and game videos containing any kind of gameplay or explanation of the game. The description was taken from store pages and the videos were taken from YouTube and store pages. The evaluation of a game was summarised in a protocol, including a rating of the programming focus of the game along a 5-graded scale: Low, Low-Medium, Medium, Medium-High and High. This classification was used to select 10 PC games (Table 2) and 10 mobile games (Table 3) with the highest ranked programming focus. These 20 games were played and carefully analysed according to the procedure presented below.

3.1.2 Game analysis

The analysis of the 20 games was a combination of playtesting and reviews of playthroughs on YouTube. All games were installed and playtested. The amount of time spent playing each game depended on the nature of the game. Determining a sufficient playtime is problematic in game analysis in general [30] as the player is part of the dynamics and there may be mutual exclusive parts of the game. In the present study, a game was played until all main concepts needed for playing were introduced and more complex levels started appearing. This mostly happened when more than half of the game was completed. This took between one and three hours. For the larger games, information about the later stages was collected from YouTube videos and articles describing the games.

Within this analysis, the following three main questions are answered for each of the games:

- What is the content of this game? What is its purpose?
- How is the game designed?
  - User interface
  - Game mechanics
  - Characters
  - Representation of the player
- How are programming concepts represented in this game?

A review form has been created for each game. These forms are used in the analysis of software development fundamentals [12] and to identify patterns. The identification of game design patterns was using the framework presented by Björk, Lundgren and Holopainen [13] specifying information regarding the pattern, containing examples and explanations of their usage. All this means that for every pattern identified, the information regarding their name, description, consequences, using the pattern and relations to be placed in a list for each selected topic. However, for this study the framework is slightly modified as it only has a description of the pattern, an explanation on how it is used and examples of games where it was found.

Table 2: *PC games analysed.*

Game	Price	Rating	# Reviews	Purchases[1]
7 billion humans	€12.49	95%	674	20k-50k
Else Heart.Break()	€22.99	83%	219	200k-500k
Exapunks	€16.79	96%	611	20k-50k
Human resource machine	€12.48	94%	1627	200k-500k
Marvellous Inc.	€4.99	75%	12	0-20k
Robots:Create AI	€0.99	90%	9	0-20k
Shenzhen I/O	€14.99	95%	1840	100k-200k
Spacechem	€9.99	96%	1923	500k-1000k
Tis-100	€6.99	97%	2360	200k-500k
While True:learn()	€9.99	92%	2808	100k-200k

1. These are estimates taken from steamspy.com [28].

4 Result

The selection phases resulted in the games presented in Table 2 and Table 3. The popularity of the PC games (Table 2) ranged from less than 20,000 installs (*Marvelous* and *Robots*) to 1,000,000 (*Spacechem*). The mobile games (Table 3) ranged from 10,000 installs (*Programming for Kids*) to 1,000,000 (*Gladiabots* and *Lightbot*).

The ratings for the PC games are in the range 75%-97% with a large majority over 90%. This indicates that these games have been successful not only in sales but also in user perception. The average rating for Steam games is 86% [31]. The mobile games have a greater variation in rating ranging from 3.1 to 4.5. As a comparison, Hu et al. [32] presents an analysis of review consistency between Google Play and App Store. This study shows an average between 3.7 and 3.9 on Google Play for a set of popular apps. Another, less scientific source, states that the average score for all Google Play apps is 4.1 [33]. This means that the majority of the analysed games have a rating above average. It is important to consider the total number of reviews when interpreting a rating. *Marvellous Inc.*, *Robots: Create AI*, and *Programming for kids* have a notable lower number of reviews compared to the other games. For this reason the ratings of these games should be considered less reliable.

4.1 Analysed games

The analysis of each game took on average 2-3 hours and resulted in a 1-3 page protocol. A brief description of each game is presented in an Appendix.

The most common genre among the analysed games was puzzle-based games. This genre is based around different levels or mechanics where the player has to solve different challenges to proceed [34]. As these games were based around programming, the most common puzzle is to create a program that completes certain tasks when executed. There are some games that innovate by introducing new ways of solving puzzles. One such game is *While True: learn()* which presents an immersive environment as well as provides links to sources containing information of the corresponding concept in real programming environments.

The only difference found between PC games and mobile games is that the latter tended to be more simplistic and childish and these games were generally advertised as programming games for kids. Several PC games had mature themes, such as romance (Figure 2), and targeted mature players.

Table 3: *Mobile games analysed.*

Game	Price	Rating	# Reviews	Downloads
Algorithm City: Coding game for kids	Free	4.2	1507	100k
Coddy: World on algorithm Free[1]	Free	-	-	100k
Coding game for kids – Learn to Code with Play	Free	3.4	619	100k
Coding planets	Free	4.3	2135	100k
Coding planets 2	Free	3.7	361	50k
Gladiabots	Free	4.2	15032	1000k
Human resource machine	€4.49	4.5	1812	50k
Lightbot: code hour	Free	4.4	15806	1000k
Programming for kids – Learn coding	Free	3.1	33	10k
Spritebox: code Hour	Free	4.3	875	100k

1. This game has been discontinued since the review.

Figure 2: *An example of the conversation interface in the game Else Heart.Break().*

4.2 Game design patterns

The analysis of the 20 games led to the identification of 11 Design patterns (Table 4). The identified patterns are briefly described below. These game design patterns have been inspired by the patterns available at the Gameplay design patterns collection [35]. The identification of patterns was made using the review forms that summarised the playtests and analysis of all games. A part of this work was also to associate each pattern with the topics of the software development fundamentals.

Table 4 shows which topics each pattern covers. The same information is presented in Table 1 but in the reverse direction – from topics to patterns.

Pattern #1: Block-based programming

This pattern is the use of blocks that represent certain actions inside a game. The advantage of this is that it simplifies programming by removing the possibility to make syntax mistakes. At the same time they provide a system that uses programming for solving problems. These blocks represent, for example, assignments of variables or different actions that can be done through the game. They are also used for making conditionals and loops as well as structuring

Table 4: *Identified game design patterns.*

#	Name	Found in	Topics covered
1	Block-based programming	7 billion humans, Human resource machine, While True: learn(), and 11 more.	Expressions and assignments; conditional and iterative control structures; program comprehension
2	Resource limitation	Lightbot: Code Hour, 7 billion humans, Human resource machine, and 5 more.	Functions and parameter passing; recursion
3	Commentary writing	Else Heart.Break(), 7 billion humans, Human resource machine, and 6 more.	Documentation and program style
4	Program output log	Robots: Create AI, 7 billion humans, Human resource machine, and Marvellous Inc.	Debugging strategies
5	Registers' representation	Marvellous Inc., 7 billion humans, Human resource machine, and 3 more.	Expressions and assignments
6	Program examples levels	TIS – 100, 7 billion humans, Else Heart.Break(), and 5 more.	Program comprehension; documentation and program style
7	Device hacking	Else Heart.Break(), and Exapunks.	Program comprehension; simple refactoring
8	Step by step program execution	7 billion humans, Human resource machine, Shenzhen I/O, and 6 more.	Program correctness; debugging strategies
9	Visual input and output representation	TIS – 100, Spacechem, While True: learn(), and 4 more.	Simple I/O including file I/O
10	Instruction-based programming	Shenzhen I/O, Marvellous Inc., Exapunks, and TIS – 100.	Basic Syntax and Semantics
11	Hidden game programming challenges	7 billion humans, Else Heart.Break(), Human resource machine, and 4 more.	Simple refactoring

the program. It lets the player configure set-ups and link different blocks of code to execute. This leads to a dynamics where the players are given a certain freedom to develop the program as they feel and to experiment with what the game provides.

This pattern covers the curricula topics *expressions and assignments*, *conditional and iterative control structures*, and *program comprehension*. For example, in *Human resource machine* blocks are used to represent actions that make characters in the game move, grab things or operate with variables. This corresponds to basic operations such as assignments and arithmetic expressions. Blocks are also used in *Human resource machine* to create conditionals and loops so certain actions are repeated or played when the conditional is fulfilled. This connects closely to *conditional and iterative control structures*. To work with these blocks, players have to develop program comprehension. Another example is *Gladiabots* that represents different conditionals using blocks. Block based programming gives a good visual representation on how to assign values or store data inside other blocks to use them in a program. Visuals make assigning values more intuitive. However, one of his weaknesses is that the player is only able to work with the options available. Block-based systams typically lacks many constructs available in more complex programming languages.

The pattern hence covers several topics but it is at an introductory level. Most of the advance aspects are not represented.

Pattern #2: Resource limitation

This game design pattern is based on limiting of resources available to the players. This will force players to reuse parts of a program with tools like functions. The functions available in games allow players to repeat several actions that are needed inside the levels. This pattern adds a challenging part to the game, limiting the freedom to solve problems. It makes the game more challenging and requires players to have a more thorough understanding of what they are doing.

This pattern covers the topics *Functions and parameter passing* and *recursion*. It is, for example, used in *While True learn()* where players have a limited amount of blocks they can use to solve different puzzles. As their resources are limited, players have to find the best way to combine them in order to achieve what they want. Another example is *Lightbot* where certain levels will not have enough space unless the player uses different functions available. In some cases the player has to create a function that calls itself, and in such way make use of recursion. Resource limitation is a somewhat artificial way to make players aware of the strength of functions. They are required to synthesising their program into repeatable parts that will be used more than once in a program. It has a potential to introduce players to the power of functions and their many advantages.

Pattern #3: Commentary writing

This game design pattern provides the players with a way of documenting their programs by writing comments in the code. The objective is to make programs easier to comprehend. This commentary writing can be in the form of specific blocks where players can write text or make drawings. These blocks do not affect the functional aspects of the code. The writing lets the player develop documentation of their code to be used later by the player herself or to share to other players. The purpose for the player could be to make the progress through the game easier but it can also be for amusement. For example, to write jokes or to create ASCII-art.

This pattern covers the topic *Documentation and program style* and is, for example, used in *Else Heart.Break()* where the player has the option to write comments in programs located in computers they can hack. In this game, comments are written by using the special character '#' at the start of the line. Another example is *Marvellous Inc.* where players can write

comments in the programs but they are also provided with comments in the existing programs. In most cases, the pattern reflects program documentation as it is made in real programming environments. One exception is *Human resource machine* where the players are only able to document with small drawings. This limits in explanation but also gives liberty to players to draw and enables more design-oriented documentation such as diagrams and graphs.

Pattern #4: Program output log

This game design pattern presents the players with the outcome of a program in form of text or visual representation. This provides players with feedback they can use to detect and solve errors in a program. Program output log simulates the outcomes from an executing program. It enables players to debug and analyse the behaviour of different constructions.

This pattern covers the topics *Debugging strategies* and it is, for example, used in *Robots*, which provides a button that will open a *code log tab*. This will show the movements and decisions the robot has made during the running of the program. Another example is covered in *Human resource machine* by giving the players a panel with all the actions the different employees have taken while the program was running. The program output log reflects the common use of program logs and traces used for debugging programs. A weakness in the studied examples is that games typically only give information on executed actions. They do not give error messages or contain error handling which is commonly used in real programming environments.

Pattern #5: Registers' representation

This game design pattern is to give players a way for storing data inside a game. This data storage simulates registers in a processor. Normally these registers can store a singular piece of data at a time. In games, these registers are commonly represented as a space, that can be full of data. It lets the players see what is happening in that space throughout the execution of their solution.

This pattern covers the topic *Expressions and assignments* and is, for example, used in *Marvellous Inc.* where a player is provided with registers in consoles and in the main character. They are used to store numbers that might be of interest for doing some mathematical operations. Another example is in *TIS-100* where a player has a fixed amount of registers for storing data. The main objective in this game is to fix a processor so the game has a close resemblance with a real processor. The strength of the pattern is that players can see in real time what they are storing. It lets players track the state of their data to know what the values are they storing at the moment they are using them. The main weakness is that registers only address the most basic forms of expressions and assignments.

Pattern #6: Program examples levels

This game design pattern is to start certain levels with existing code. This code can explain how new elements work or make a first introduction on how to proceed. Program examples levels give players different ways to learn how to proceed in later levels or to learn a new concept. It allows them to learn how to do basic operations in order to solve more complex later levels. In many example levels, commentaries are given as an explanation. Sometimes these levels will be half-completed so the players have to "fill the gaps" and fix the program.

This pattern covers the topics *Program comprehension*, *documentation*, and *program style*. It is, for example, used in *TIS-100* where code is present from the start in some levels. This gives information to a player on how specific entities work. Another example is *Spacechem* that gives players levels where some parts are working and others are not. In addition to

develop the program comprehension of players, examples reflect a programming style and can give examples on the importance of documentation. This has a potential to convey insights into aspects beyond core functionality. However, an easy example might not be enough for explaining complex concepts and the players might get lost even in the example. Reading code can be challenging for beginners.

Pattern #7: Device hacking

This game design pattern is to include hacking devices in the game. The players can alter a program that is running inside such a device. Sometimes, the players are provided with comments to the code and what the intended purpose of the program is. This hacking pattern can be seen as a combination of the patterns *Program examples levels* and *Commentary writing*. This gives the feeling that what they are doing has an impact in the game, and also that the code is an integral part of the game world. Hacking a device is associated with danger which can give the player a thrilling experience.

This pattern covers the topics *Program comprehension* and *simple refactoring*. *Device hacking* is, for example, used in *Else Heart.Break()* to hack different systems in the game world. These devices will sometimes contain a complex program that the players will have to understand in order to modify it. Some of these modifications can contain refactoring. Another example is from *Exapunks* where a player needs to create "viruses" that will steal data from the computer they are in. To do this, players are required to first understand the existing code, and secondly modify it until they achieve the wanted solution. A potential problem with this pattern is that it includes an illegal activity and it may be problematic to include it in certain educational contexts.

Pattern #8: Step by step program execution

This pattern is based on giving the players the option to run their solution step by step. This enables the player to search for a specific error in the program or to see what each line of the program does.

This pattern covers the topics *Program correctness* and *debugging strategies*. The latter is, for example, represented in *7 billion humans* where a separate button is present, next to the play button, that will run the main program step by step. A player is then able to debug an erroneous program be inspecting it action by action. A similar solution is provided in *Shenzhen I/O* by providing the players with several buttons that let them execute the program step by step. The pattern represents the topic in a good manner as it simulates the fundamental debugging tools available in programming environments. However, modern environments provide the developer with much more advanced options, e.g. the use of breakpoints and profilers, that are not present in the studied games.

Pattern #9: Visual input and output representation

With this pattern, the objective in the game is to represent the different inputs and outputs that a program has as visual objects inside the game. This also represents the process these objects or variables take inside the game, like the place they come from, and the place where the results must go. The visual representations contribute to the aesthetics of the game and amplify the feedback reward.

This pattern covers the topics: *Simple I/O including file I/O* and is, for example, used in *TIS-100* where input and output zones are visible to players right after they enter the puzzle. In this case they are represented as an arrow with a name on it. In most of the levels, a player will receive numbers that they will have to operate with and deliver to the output. Another

example is *Spacechem* which gives a player different places, marked as outputs or inputs, where various chemicals will appear in the circuit. The final combination of chemicals should be placed structured in a specific orientation. This pattern is useful for explaining a simplified version of what data inputs and outputs are. It does not get fully into file input and output so that is one of its bigger weaknesses. However, it is a good introduction for novices trying to learn how inputs and outputs work in programming.

Pattern #10: Instruction-based programming

This game design pattern is to use procedural based programming language inside the game to simulate assembler programming. This means that the whole programming part is based around some instructions with parameters that will do certain things. As an example of an instruction, `MOV` is commonly used for moving data from a register to another position. With this pattern, as with the pattern of block-based programming, the objective is to set an environment where the players can structure their programs.

This pattern covers the topic *Basic Syntax and Semantics* and is, for example, used in *Shenzhen I/O* where instructions are used to make programs for circuits. Shenzhen I/O contains a manual that explains the use of the different instructions as well as examples on how to use them. The pattern is also used in *Exapunks* by giving a player different instructions that belong to the pseudo-code used in the game. As the pattern is mainly represented by a pseudo-code that simulates a real-life programming languages, it gives a good representation of syntax and semantics used in real life instruction-based programming. However, this type of programming languages is only popular in low-level environments. It differs a lot from object-oriented programming languages, but it is still useful for learning how to program.

Pattern #11: Hidden game programming challenges

This game design pattern is to provide players with extra objectives that are not explicitly presented as goals in the game. This can be seen as sidequest that are common in many games. These challenges can give players a reward or badge but they usually do not give anything apart from self-satisfaction. A challenge can, for example, be to optimise a program solution by reducing the amount of code lines needed to solve a problem. Hidden game programming challenges give the players an extra thing to do and the possibility to create self-defined challenges. This will make the game harder for skilled players and therefore it will make them use all the tools they have available at that particular moment to solve the problem.

The only topic covered by this pattern in the studied games is *Simple refactoring*. This is, for example, utilised in *While True:learn()* by giving extra levels where the players are required to optimise previous solutions. It is also used in *7 billion humans* to make a competition between players. Simple refactoring matches this pattern well in that it provides alternative solutions to the same problem, where different solutions can optimise different characteristics. It should be noted that other topics are possible to cover with this pattern even if it was not the case in the studied games.

Giving the players new challenges are usually a good way to further their learning process. The big problem with this pattern is that these challenges are mostly introduced indirectly and that many players may decide not to complete them.

4.3 Relation to the ACM/IEEE Computer Science Curricula

Table 5 shows how well the main units of the ACM/IEEE Computer Science Curricula are covered in the studied games. As can be seen in the table, all studied games address funda-

mental programming concepts. A majority also covers development methods. Interestingly, algorithms and design is only covered in a single game (5% of the 20 studied games).

Each presented unit contains topics (as shown in Table 1) and in the following subsections we highlight topics that have a good representation in the studied games and topics that are absent or only covered sporadically.

Table 5: *Percentage of analysed games that cover the main units of the ACM/IEEE Computer Science Curricula.*

Main unit	Percentage (n=20)
Algorithms and design	5%
Fundamental programming concepts	100%
Fundamental data structures	30%
Development methods	65%

4.3.1 Topics covered

Among *fundamental programming concepts*, there are three topics that are present in more than 50% of the games: Conditional and iterative control structures (100%); Functions and parameter passing (65%); and, Simple I/O including file I/O (55%). The remaining topics in this unit are relatively well represented (30-45%) except *Variables and primitive data types*, which is highlighted only in one game.

In the unit *development methods*, two topics are present in a majority of the games: Debugging strategies (60%) and Program correctness (55%). The remaining topics in this unit are relatively well represented (35-40%) except *Modern programming environments*, which is not present in any game.

This indicates that half of the four units have good representation in successful games and that these games can be candidates in teaching of the involved topics. It remains to study the actual learning effects from using them.

4.3.2 Topics not covered

For the units *Algorithms and design* and *Fundamental data structures* there is almost no topic that is represented in more than one game. For the former unit, there is only one game, *While True: learn()*, that addresses it. On the other hand it addresses all four topics. For the latter unit, three topics are covered: Arrays (30%); Records/Structs (10%); and, Strings and string processing (5%). The remaining four topics are not present in any of the 20 studied games.

The collective analysis shows that 5 out of 24 topics listed in Table 1 are absent in all studied games. There is hence room for successful approaches that address: Modern programming environments; Abstract data types; References and aliasing; Linked lists; and, Strategies for choosing the appropriate data structure.

4.4 Discussion

The patterns identified in this article have been observed in a number of successful programming games. These patterns can serve as an inspiration for future programming games aiming to teach the associated topic. The strength of the presented study, compared to previous studies of programming games [7, 14], is that it is focused on successful games. This increases

the chances that patterns, observed in several such games, are actually capturing good game design principles.

The topics covered by the identified patterns are all under the units *Fundamental programming concepts* and *Development methods*. This means that no pattern was identified for the units *Algorithms and design* or *Fundamental data structures*. This reflects the result from the analysis (Table 5) which shows an imbalance between computer science topics they contain. All games address fundamental programming concepts in some way but very few cover data structures, algorithms and design. The lack of games focused on algorithms may be surprising, as algorithms are inevitable in programming. There was however only one game where the concept, design and properties of algorithms were highlighted.

A similarity in most studied games was that they were based on puzzle mechanics. This means they have levels inside the game containing puzzles. There are two clear exceptions from this. One is *Gladiabots* which is a simulation game where players construct AI for robots that fights in an arena. The second exception is *Else Heart.Break()* (Figure 2) which is a story driven game with point and click mechanics where the main objective is to hang around in a city. Even if this is an adventure game, elements involving programming tasks are mainly based on puzzle mechanics.

The domination of the puzzle genre is not as clear in other studies of programming games [14] or in general serious games [8]. Lindberg and Laine [5] highlight the potential importance of considering different player types and learning styles when developing and using programming games. They propose that different genres should be used to address this. Our study indicates that the current mainstream programming games do not provide a rich resource with respect to this.

From the studied games, it can be deduced that there is some relation between the topics covered by a game and the way these topics are represented. For example, games that focused on a more complex programming concepts are using more complex interfaces where players need to use a text-based language. This language could be specifically made for the game or based on an existing one. However, games focused on more fundamental concepts make more use of block-based programming systems where the textual syntax is not a challenge to players.

4.5 Limitations

The focus in this article is on popular programming games to analyse which programming concepts they represent, and to identify game design patterns used to convey them. The effectiveness of these games as means to learn programming has not been evaluated. We have also not analysed the demographics of players. It is possible that some programming games are popular among programmers or persons that have a special interest in the type of problem solving involved. Further studies are needed to analyse if players can learn the identified topics from these games and if they are appreciated by a broad demography.

The method used to search for games was limited to keyword search with only a few terms. In practice the search was limited to games with programming in its description or name. It would have been possible to extend the search query to include additional terms, for example, from the topics of the ACM/IEEE curricula. Some samples from this list however indicated that no additional games were found using those terms.

5 Conclusions and Future Work

This article presents a review of mainstream programming games with the aim to analyse them using the ACM/IEEE curricula and to identify game design patterns. A difference with previ-

ous studies in this field (e.g. [14]) is that our review is mainly based on first-hand play-sessions where each game is installed and played for approximately $2\frac{1}{2}$ hours. Another difference is that game selection is based on number of downloads. This is done under the assumption that popular games have managed to create a rewarding game experience. Games presented in research articles are rarely evaluated with respect to player engagement [7].

One result of this study is the identification of 11 game design patterns used in studied games. All these patterns relate to some topics of the ACM/IEEE computer science curricula. There are many patterns that covers topics related to fundamental programming concepts and development methods. These units are well covered in the studied games. There is however very little representation of algorithms, design and data structures. This result resembles that reported in similar reviews reported by Miljanovic et al. [7] and Vahldick et al. [14]. These reviews had an emphasis on research projects. Both studies highlight the problem that many of the games reported in research articles are unavailable. Many previous studies of programming games are conducted by the same persons that have developed the game being studied. This gives a risk for self-confirmatory bias. Miljanovic et al. [7] identifies this problem and calls for third-party evaluations. The study presented in this article is such a third-party evaluation and it is exclusively based on mainstream games available at big commercial app stores. An important contribution of this study is the systematic identification, analysis and presentation of successful games. The majority of the games studies in this article have been played by hundreds of thousand of players, some even by millions. This is never the case for research prototypes.

The design patterns in this article are presented as methods to improve gameplay quality in programming games. Most previous studies have a focus on the learning quality of games. It remains to evaluate how games using these design patterns can be integrated in programming teaching. There are also future research possibilities in designing games focused on software development concepts, such as linked lists and abstract data types, that are not present in any of the studied games. Finally, there are game design challenges to include programming elements in game genres other than the puzzle genre, which is heavily dominating in mainstream programming games.

References

[1] B. Berg Marklund, "Unpacking digital game-based learning: The complexities of developing and using educational games," Ph.D. dissertation, University of Skövde. ISBN 978-91-981474-8-3 2015.

[2] S. Arnab, R. Berta, J. Earp, S. De Freitas, M. Popescu, M. Romero, I. Stanescu, and M. Usart, "Framing the adoption of serious games in formal education." *Electronic Journal of e-Learning*, vol. 10, no. 2, pp. 159–171, 2012. [Online]. Available: https://files.eric.ed.gov/fulltext/EJ985419.pdf

[3] A. Coelho, E. Kato, J. Xavier, and R. Gonçalves, "Serious game for introductory programming," in *International Conference on Serious Games Development and Applications*. Springer, 2011. doi: https://doi.org/10.1007/978-3-642-23834-5_6 pp. 61–71.

[4] S. Galgouranas and S. Xinogalos, "jAVANT-GARDE: A cross-platform serious game for an introduction to programming with java," *Simulation & Gaming*, vol. 49, no. 6, pp. 751–767, 2018. doi: https://doi.org/10.1177/1046878118789976

[5] R. S. Lindberg and T. H. Laine, "Formative evaluation of an adaptive game for engaging learners of programming concepts in k-12," *International Journal of Serious Games*, vol. 5, no. 2, pp. 3–24, 2018. doi: https://doi.org/10.17083/ijsg.v5i2.220

[6] M. A. Miljanovic and J. S. Bradbury, "Robobug: a serious game for learning debugging

techniques," in *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ACM, 2017. doi: https://doi.org/10.1145/3105726.3106173 pp. 93–100.

[7] ——, "A review of serious games for programming," in *Joint International Conference on Serious Games*. Springer, 2018. doi: https://doi.org/10.1007/978-3-030-02762-9_21 pp. 204–216.

[8] E. A. Boyle, T. Hainey, T. M. Connolly, G. Gray, J. Earp, M. Ott, T. Lim, M. Ninaus, C. Ribeiro, and J. Pereira, "An update to the systematic literature review of empirical evidence of the impacts and outcomes of computer games and serious games," *Computers & Education*, vol. 94, pp. 178–192, 2016. doi: https://doi.org/10.1016/j.compedu.2015.11.003

[9] A. Perttula, K. Kiili, A. Lindstedt, and P. Tuomi, "Flow experience in game based learning–a systematic literature review," *International Journal of Serious Games*, vol. 4, no. 1, pp. 57–72, 2017. doi: https://doi.org/10.17083/ijsg.v4i1.151

[10] Valve corporation, "Steam," https://store.steampowered.com.

[11] Google, "Google play," https://play.google.com/store.

[12] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. New York, NY, USA: ACM, 2013. ISBN 978-1-4503-2309-3 999133.

[13] S. Björk, S. Lundgren, and J. Holopainen, "Game design patterns," in *DiGRA - Proceedings of the 2003 DiGRA International Conference: Level Up*, 2003. ISSN 2342-9666. [Online]. Available: http://www.digra.org/wp-content/uploads/digital-library/05163.15303.pdf

[14] A. Vahldick, A. J. Mendes, and M. J. Marcelino, "A review of games designed to improve introductory computer programming competencies," in *2014 IEEE frontiers in education conference (FIE) proceedings*. IEEE, 2014. doi: https://doi.org/10.1109/FIE.2014.7044114 pp. 1–7.

[15] J.-M. Hoc, T. Green, R. Samurcay, and D. Gilmore, *Psychology of programming*. Academic Press, 2014. ISBN 0-12-350772-3

[16] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33–35, 2006. doi: https://doi.org/10.1145/1118178.1118215

[17] P. J. Denning, "Remaining trouble spots with computational thinking," *Communications of the ACM*, vol. 60, no. 6, pp. 33–39, 2017. doi: https://doi.org/10.1145/2998438

[18] C. Rinderknecht, "A survey on teaching and learning recursive programming," *Informatics in Education-An International Journal*, vol. 13, no. 1, pp. 87–120, 2014. [Online]. Available: https://files.eric.ed.gov/fulltext/EJ1064322.pdf

[19] L. Mannila, M. Peltomäki, and T. Salakoski, "What about a simple language? analyzing the difficulties in learning to program," *Computer Science Education*, vol. 16, no. 3, pp. 211–227, 2006. doi: https://doi.org/10.1080/08993400600912384

[20] W. W. Lau and A. H. Yuen, "Toward a framework of programming pedagogy," in *Encyclopedia of Information Science and Technology, Second Edition*, M. Khosrow-Pour, Ed. IGI Global, 2009, pp. 3772–3777. ISBN 9781605660264

[21] Apple, "App store," https://www.appstore.com.

[22] K. Tsarava, K. Moeller, and M. Ninaus, "Training computational thinking through board games: The case of crabs & turtles," *International Journal of Serious Games*, vol. 5, no. 2, pp. 25–44, 2018. doi: https://doi.org/10.17083/ijsg.v5i2.248

[23] R. White, F. Tian, and P. Smith, "Code lab: A game that teaches high level programming languages," in *Proceedings of the 30th International BCS Human Computer*

Interaction Conference: Fusion! BCS Learning & Development Ltd., 2016. doi: https://doi.org/10.14236/ewic/HCI2016.76 p. 57.

[24] K. Kiili, S. De Freitas, S. Arnab, and T. Lainema, "The design principles for flow experience in educational games," *Procedia Computer Science*, vol. 15, pp. 78–91, 2012. doi: https://doi.org/10.1016/j.procs.2012.10.060

[25] K. Kiili, T. Lainema, S. de Freitas, and S. Arnab, "Flow framework for analyzing the quality of educational games," *Entertainment computing*, vol. 5, no. 4, pp. 367–377, 2014. doi: https://doi.org/10.1016/j.entcom.2014.08.002

[26] E. H. Calvillo-Gámez, P. Cairns, and A. L. Cox, "Assessing the core elements of the gaming experience," in *Game user experience evaluation*, R. Bernhaupt, Ed. Springer, 2015, pp. 37–62. ISBN 978-3-319-15984-3

[27] Valve corporation. (2019) Steam game and player statistics. [Online]. Available: https://store.steampowered.com/stats/Steam-Game-and-Player-Statistics

[28] S. Galyonkin. (2019) Steamspy. [Online]. Available: https://steamspy.com

[29] Marijan. (2019) playtracker insight. [Online]. Available: https://playtracker.net/insight

[30] C. Fernández-Vara, *Introduction to game analysis*. Routledge, 2019. ISBN 978-0-8153-5183-2

[31] D. Lin, C.-P. Bezemer, Y. Zou, and A. E. Hassan, "An empirical study of game reviews on the steam platform," *Empirical Software Engineering*, vol. 24, no. 1, pp. 170–207, 2019. doi: https://doi.org/10.1007/s10664-018-9627-4

[32] H. Hu, C.-P. Bezemer, and A. E. Hassan, "Studying the consistency of star ratings and the complaints in 1 & 2-star user reviews for top free cross-platform android and ios apps," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3442–3475, 2018. doi: https://doi.org/10.1007/s10664-018-9604-y

[33] AppBrain, "Ratings of apps on google play," https://www.appbrain.com/stats/android-app-ratings.

[34] E. Adams, *Fundamentals of game design*. New Riders, 2014. ISBN 0-321-92967-5

[35] S. Björk. (2019) Gameplay design patterns collection. [Online]. Available: http://virt10.itu.chalmers.se

Appendix

PC Games

7 Billion humans

7 billion humans is a puzzle based game where the player has to program the behaviour of the workers. When the player press the button to run their solution, each of the workers will execute the program at the same time. The objective of each level depends on the task that the manager tells the player to do.

Figure 3: *7 billion humans.*

The game is designed around levels that contain different puzzles or tasks. In each of these levels the player will be provided with different blocks that represent different actions the workers will perform (Figure 3). The actions will be run from top to bottom in the order that the player places them. The right part of the screen represents the program where the player place the action blocks. In the left down part of the screen there is a play button. There are also buttons to run one step at a time and to stop the program from running, as well as one to reset the workers.

Patterns in this game: Block-based programming; Resource limitation; Commentary writing; Program output log; Registers' representation; Program examples levels; Step by step program execution; Visual input and output representation; Hidden game programming challenges.

Else Heart.Break()

Else Heart.Break() starts with the main character Sebastian receiving a job offer in another city. When he arrives, strange things are happening with computers and electronics in the city. After some time talking to the people in the city (Figure 2), he can buy a device called modifier that lets the player hack different systems and modify their code. The player has freedom to talk to people, go to the café, sell soda etc. In some cases, the player will be given certain tasks or information from other characters that progress the plot of the game. The main task of the game is to find friends within the hacker- and activist community to try to stop the people that are running the city.

Figure 4: *Else Heart.Break().*

The game is mainly a point and click game. Some interactables can be hacked to achieve something relevant in the city. Hacking is done in a different UI (Figure 4) that allows for modification of the code for the object. This new UI will enable the player to write, compile or debug the code, undo their changes and to get some help. The programming language used for coding is a pseudo-code called *Sprak*, invented by the developers.

Patterns in this game: Commentary writing; Program examples levels; Device hacking; Hidden game programming challenges.

Exapunks

Exapunks is a puzzle game where the player will have to complete several hacking tasks by programming the behaviour of some small robots. These tasks are accessing systems in order to get the files inside, sometimes to modify them, drop them in the output box or delete them and then disappear without leaving a trace (Figure 5).

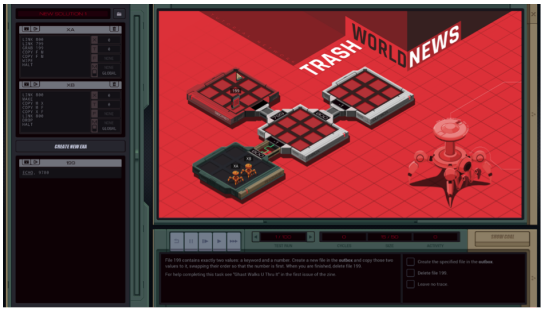

Figure 5: *Exapunks.*

The left side of the screen is where the different files available in the level appear, with information about the data they contain. The top left side of the screen is where the robots with their name and program appear. The players can write the program of the robots to do as they want. Each of the robots has access to two register to store one piece of data each and can grab only one file. The actions of the robots are shown in the centre of the screen.

Patterns in this game: Resource limitation; Registers' representation; Device hacking; Instruction-based programming.

Human resource machine

Human resource machine is a puzzle-based game based around levels. In each level the manager gives the player a certain task that the workers available in that level must do. Orders are given to workers by writing a program that will define their behaviour. This is done by placing different blocks that represent certain actions inside the main program table placed in

the right side of the screen (Figure 6). Normally these worker tasks involve the grabbing and placing of blocks that contain data.

Figure 6: *Human resource machine.*

The UI in the game provides several buttons for running the program made, running it step by step, increasing the speed at which the program plays, going back one step and stopping the current run. In each of the levels some secret challenges are give to the player to make their program as efficient as possible.

Patterns in this game: Block-based programming; Resource limitation; Commentary writing; Program output log; Registers' representation; Program examples levels; Step by step program execution; Visual input and output representation; Hidden game programming challenges.

Marvellous Inc.

Marvellous Inc. is a game where the player must solve puzzles presented as tasks that are given by emails from a company, co-workers or users (Figure 7). These tasks will normally require the player to program the behaviour of a robot. In some of them, the robot will have to move and use objects or go to consoles to do mathematical operations.

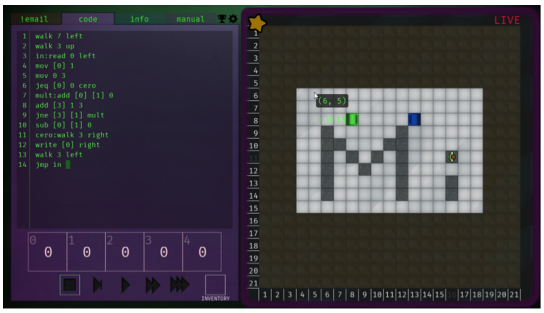

Figure 7: *Marvellous Inc..*

The registers available to the player will let them write and make operations with different numbers, using instructions like `ADD` and `MOV`. The game also includes a manual and an info tab where the player will have access to information regarding the different instructions available. There is also an option to give up on a task, if it is too hard to complete.

Patterns in this game: Commentary writing; Program output log; Registers' representation; Program examples levels; Step by step program execution; Visual input and output representation; Instruction-based programming; Hidden game programming challenges.

Robots: Create AI

Robots: Create AI is a puzzle-based game where the player has to program the artificial intelligence of a robot to solve puzzles. The game does not have any story and the only objective is to solve these puzzles.

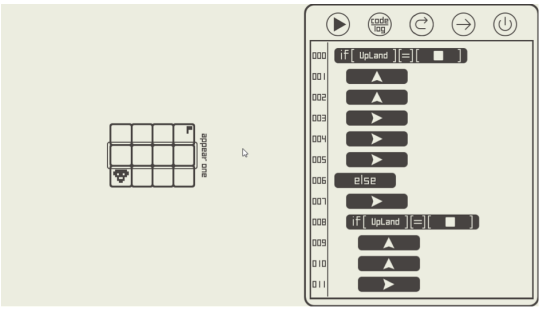

Figure 8: *Robots:Create AI.*

The programming is done by adding `ADD` blocks in the right part of the window (Figure 8). The player has access to an option called *code log*. This code log will show what movements and decisions the robot has taken while running the program. This is useful for finding errors that made the program to fail.

Patterns in this game: Block-based programming; Program output log; Step by step program execution.

Shenzhen I/O

In Shenzhen I/O the player works at a company that develops different circuits for various machines. It is a puzzle solving game where the player is given tasks via a message service, where bosses and co-workers will tell them what they must do. These puzzles are generally based around developing circuits for the company. To develop these circuits, the player will have to use parts available in each of the levels and program them so the circuit can work. Each component they use has a cost, and the game has some challenges where the player is asked to develop the circuit as cheap as possible, as fast as possible and with as few lines as possible.

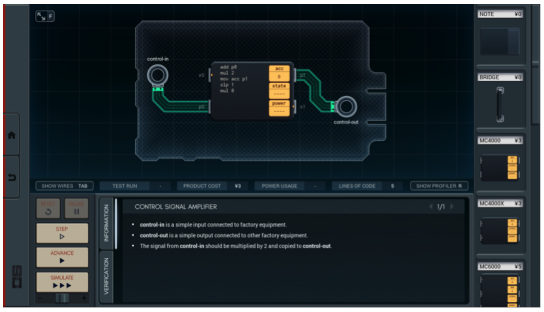

Figure 9: *Shenzhen I/O.*

Each level gives access to an UI (Figure 9) containing fields where the player can place different components for developing the circuits. The panel to the right contains all components available for that level. The lower panel provides information about the task, the cost, the power usage, the lines of code, the verification of the circuit and several buttons for running the tests.

In the manual of the game, there is a complete explanation of the different functions of the system. There is also an explanation of the components available in the game to give an understanding of how the program behaves in each of the components and what registers they have access to in them.

Patterns in this game: Resource limitation; Commentary writing; Program examples levels; Step by step program execution; Instruction-based programming; Hidden game programming challenges.

Spacechem

Spacechem is a puzzle-based game where the player must make systems for producing the requested chemicals using a set of available chemical ingredients. This is done using two circuits, one blue and one red (Figure 10). These two lines will run concurrently, so the player must be careful to not collide the chemicals while they are running.

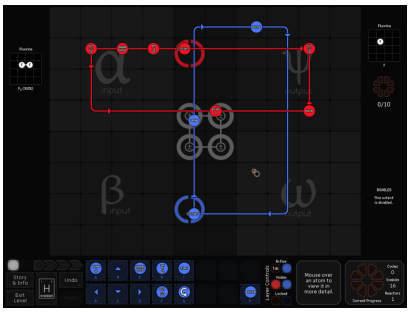

Figure 10: *Spacechem.*

A puzzle is solved by dragging and dropping chemicals in the correct place and in the correct state. In Spacechem the programming concepts are not shown directly in the game, but programming thinking is involved in the process. Player must consider that the lines run concurrently. This implies that sometimes the player will have to synchronise the lines for making certain things. In later levels, several factories will be run at the same time so the synchronisation between the inputs and the outputs of the different factories will need to be handled too.

Patterns in this game: Block-based programming; Program examples levels; Visual input and output representation.

TIS-100

TIS-100 is a puzzle-based game where the player must try to fix a broken unknown machine called TIS-100. For doing so they have to solve the different parts of the machine that are presented as puzzles. In these puzzles they can see how some segments are corrupted and they can investigate the error by debugging it. For solving the puzzles, the player must write code in the segments in order to get the inputs to the outputs as asked in the task of that level.

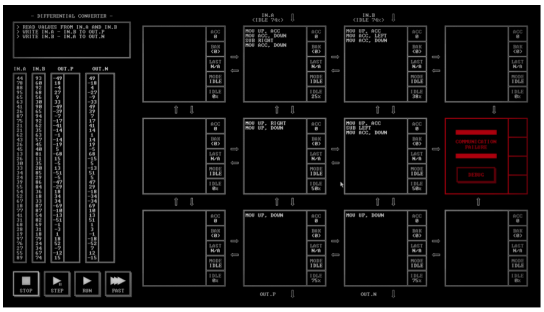

Figure 11: *Tis-100.*

In each of the levels, a screen like the one in Figure 11 will appear. The top, left side shows the task that has to be completed. The bottom, left part represents the different inputs and the expected outputs the player is supposed to generate. For doing that they have access to instructions like those used in an assembler-like programming language.

Patterns in this game: Commentary writing; Registers' representation; Program examples levels; Step by step program execution; Visual input and output representation; Instruction-based programming.

While True: learn()

This game presents a scenario with a cat that is very intelligent and that the player should try to understand. The game is based on different levels that will present certain tasks to fulfil. These tasks are related to machine-learning and data processing. The player gets access to different blocks (Figure 12) that filter data in different ways.

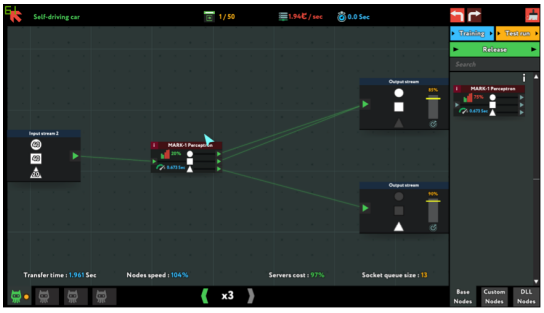

Figure 12: *While True:learn().*

Some of these blocks will, for example, filter the data by its form, some by their colour, some will reduce the errors. It is also possible to save a solution to use in future levels. While running the programs, the player can increase the speed at which they are playing so they do not have to wait a lot for the program to finish.

Patterns in this game: Block-based programming; Resource limitation; Registers' representation; Step by step program execution; Visual input and output representation; Hidden game programming challenges.

Mobile Games

Algorithm City: Coding game for kids

Algorithm City is a puzzle-based game where the player should program a penguin to grab the different coins that appear in the levels. For doing so they have access to several blocks that represent certain actions in the game.

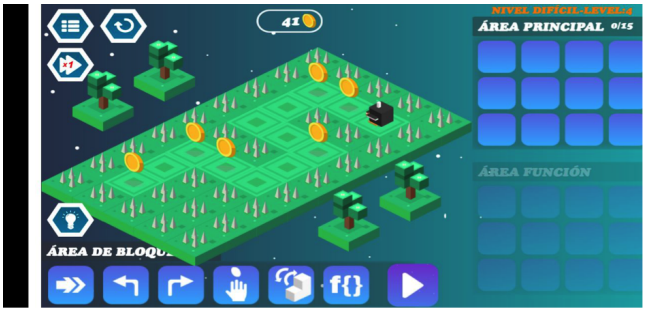

Figure 13: *Algorithm City: Coding game for kids.*

The player has access to a zone placed in the right part of the screen (Figure 13) that represents the main function that will be played when the play button is pressed. There are several other functions that can be used inside the main one. The player has access to different blocks that represent certain actions. These blocks, when placed inside a function, will run step by step, making the penguin grab the different coins that appear in the level.

Patterns in this game: Block-based programming; Resource limitation.

Coddy: World on algorithm Free

Coddy is a puzzle-based games where the player must write the program of a robot. They have access to several blocks that represent certain robot actions.

Figure 14: *Coddy: World on algorithm Free.*

A level (Figure 14) is based on a UI containing different blocks of code, panels with the functions and buttons that let the player run the program. In the right part of the screen the player has access to the different functions where they can place the blocks of actions. In the lower part of the screen, they have access to the blocks that represent the actions the robot will make, explained by graphics. In the left part of the screen they have access to the buttons that play and pause the execution of the program as well as access to block of configuration and level skipping. The main objective in all levels is to make the robot catch all the stars and get to the final, blue field.

Patterns in this game: Block-based programming; Resource limitation; Step by step program execution.

Coding game for kids – Learn to Code with Play

Coding game for kids is a puzzle-based game with levels in different categories where some of them teach certain programming concepts (Figure 15).

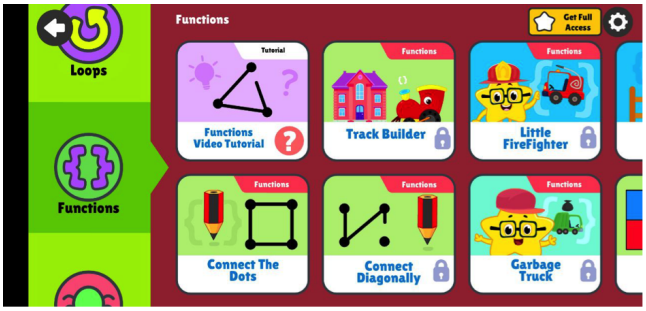

Figure 15: *Coding game for kids – Learn to Code with Play.*

Inside a level the player has access to blocks that represent certain actions as well as a main function where they can place these blocks. For solving the task, a player has to make the program with the resources in the proper order.

Patterns in this game: Block-based programming.

Coding planets and Coding planets 2

Coding planets and Coding planets 2 are games where the player is provided with several blocks that are combined in a main program. This will make the main character, a little robot, move through the field (Figure 16). The objective in each level is to acquire every gem in the field.

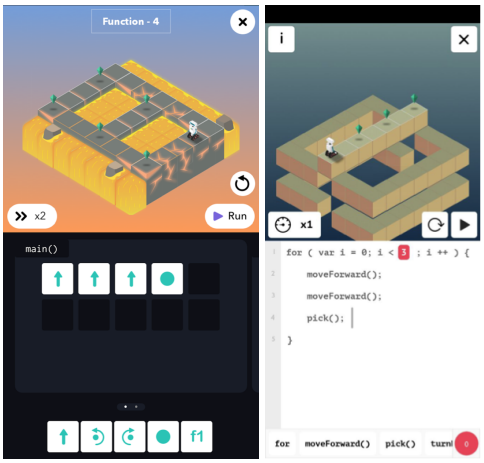

Figure 16: *Coding planets (left) and Coding planets 2 (right).*

The game is divided in 3 worlds where the first one will be an introduction to the concepts of the game and to the simplest tools available. Each of the other worlds will introduce a totally new concept for solving new puzzles with new challenges.

The player is provided with an interface (Figure 16) where the top part of the screen is the representation of the puzzle they have to solve. The middle part is the main function with buttons to run the solution at different speeds. The bottom part is the area for creating the solution. In Coding planets (Figure 16, left) the player uses blocks that represent the parts of the code. To solve a puzzle, the player has to place the blocks in the correct way for the robot

to obtain all the gems and complete the level. In Coding planet 2 (Figure 16, right) the blocks have been replaced with a text-based coding interface.

Patterns in these games: Block-based programming.

Gladiabots

Gladiabots is a puzzle-based game where the player must create the tree behaviour of the artificial intelligence of several robots. The objective in levels is to kill the enemy robots and acquire the different orbs that appear. There is also a multiplayer mode where a player fights other players. The robots have different weapons to give diversity to the game.

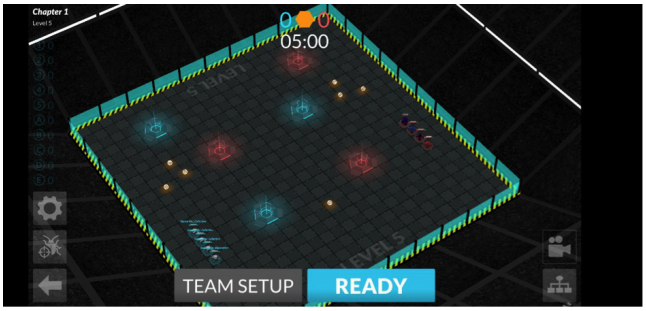

Figure 17: *Gladiabots.*

In the levels of the game (Figure 17), the robots are placed on the board. The blue robots represent the player and the red robots represent the enemy. The other items on the board should be brought to the home areas to score points. Some levels has a time limit. To complete a level, the player must create a behaviour tree for the different robots. This is done by placing blocks in a certain order. This programming screen is not shown in Figure 17.

Patterns in this game: Block-based programming.

Human resource machine

Human resource machine is a puzzle-based game. In each of the levels, a factory manager will give the player a certain task that the workers must execute. For giving orders to their workers, the player write a program that will define their behaviour. This is done by placing different blocks that represent certain actions inside the main program table placed in the right side of the screen (Figure 18). Normally these tasks involve grabbing and placing blocks of data.

Figure 18: *Human resource machine.*

The UI in the game provides several buttons for: running the program; running it step by step; increasing the speed at which the program plays; going back one step; and, stopping the

current run.

Patterns in this game: Block-based programming; Resource limitation; Commentary writing; Program output log; Registers' representation; Program examples levels; Step by step program execution; Visual input and output representation; Hidden game programming challenges.

Lightbot: Code Hour

Lightbot is a puzzle-based game where the player must program a robot to complete the level.

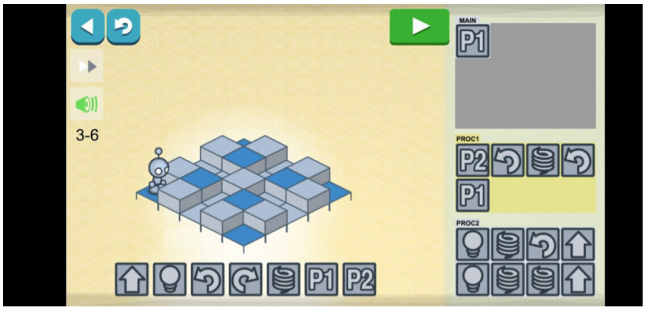

Figure 19: *Lightbot: code hour.*

In each of the levels, the robot will have to light the bulb in the blue colour fields (Figure 19). When all the blue field are lighted the puzzle will be completed.

To make a program, the player has access to different blocks that represent different actions in the game (Figure 19). These actions can be placed in one of the several functions the player has access to. When the program is run, it will launch the main function and what is inside of it.

Patterns in this game: Block-based programming.

Programming for kids – Learn coding

Programming for kids is a puzzle-based game where the player creates a program that solves the puzzle that appears in each level. For doing so, they have access to different blocks that represent certain action inside the game. These blocks can be placed inside the main function or in other functions. There exist 3 categories: functions, sequences and loops.

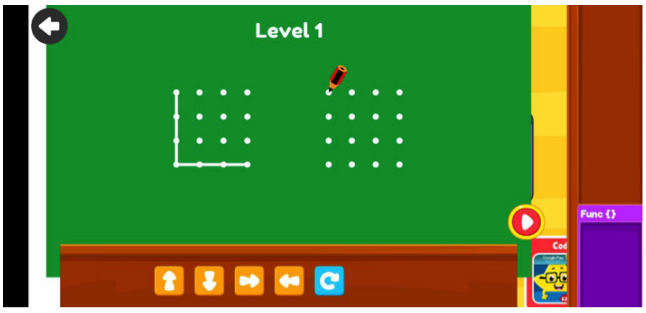

Figure 20: *Programming for kids – Learn coding.*

The example level in Figure 20 is a puzzle in the form of a pencil that needs to draw the same lines as in the left image. For doing so, the player has access to blocks that represent certain actions (lower part of the screen), and a main function zone (right side) where they can

place them in the order they want them to execute. There is also a function zone that can be used if needed.

Patterns in this game: Block-based programming.

Spritebox: code Hour

Spritebox is a platformer game with some programming puzzles. The player can move the main character from left to right to obtain stars and to save all the little whale-shaped things (Figure 21). Throughout the levels, several puzzles containing programable parts appear. In order to continue with the level the player has to solve the puzzle.

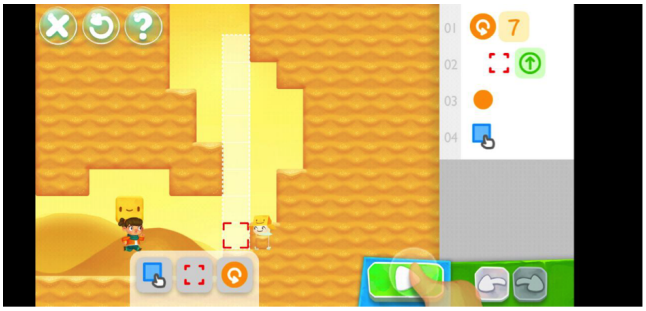

Figure 21: *Spritebox: code Hour.*

In the programming puzzles that appear throughout Spritebox, the player can make different loops that will run certain actions the specified number of times. Conditional actions are not present in the game.

Patterns in this game: Block-based programming.