

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**

# **Platform for Educational Games Generation**

**André Gomes Barbosa**



Mestrado Integrado em Engenharia Informática e Computação  
Supervisor: Ana Cristina Ramada Paiva

September 15, 2015



© André Gomes Barbosa, 2015

# **Platform for Educational Games Generation**

**André Gomes Barbosa**

Mestrado Integrado em Engenharia Informática e Computação

Approved in Public Examination by the Jury:

President: João Carlos Pascoal Faria

External examiner: João Miguel Fernandes

Supervisor: Ana Cristina Ramada Paiva

---

September 15, 2015



# Abstract

The development of games for education is a growing area of research. Over the past few years, studies point out that the use of games as a supplement to traditional learning can be much more efficient and motivational than just using the traditional teaching method. These are called serious games.

This work aims to improve a previous game developed to learn software testing, having the contents of the game are loaded from a XML file which would be separated from the game implementation. The game itself, *iLearnTest*, consists of several challenges which cover various themes regarding software testing learning.

Since the content is saved to and loaded from a separate file from the game implementation, it becomes easier to create and edit new levels or even new games for different topics, simply by editing a XML file. This new approach turns *iLearnTest* into a framework rather than a game in the sense that it is possible to use the same game(s) as templates for the learning process of a variety of themes.



# Resumo

O desenvolvimento de jogos para a educação é uma área de pesquisa em crescimento. Ao longo dos últimos anos, os estudos apontam que o uso de jogos como um complemento ao ensino tradicional é muito mais eficiente e motivador do que utilizando apenas o método tradicional de ensino. Estes são denominados jogos sérios.

Este trabalho tem como objectivo melhorar um jogo anteriormente desenvolvido para aprendizagem de testes de *software*, de maneira que os conteúdos do jogo sejam carregados de um arquivo XML, separado da implementação do jogo. O jogo em si, *iLearnTest*, é composto por vários desafios que abrangem diversos tópicos sobre aprendizagem de testes de *software*.

Uma vez que os dados são guardado e carregados a partir de um ficheiro isolado da implementação do jogo, espera-se que se torne mais fácil criar e editar novos níveis ou até mesmo novos jogos para temas diferentes, bastando simplesmente editar um ficheiro XML. Esta nova abordagem transforma o *iLearnTest* numa *framework* em vez de um jogo, no sentido em que é possível reutilizar os mesmos jogos como moldes para o processo de aprendizagem de uma variedade de temas.





# Acknowledgements

I would like to thank my family for the unconditional support they have given me ever since I can remember. Not only that, they have encouraged me and cheered me up through all the difficult moments which brought me to where and who I am presently. In short, I would like to thank: my father for opening my eyes to the things that really matter the most, my mother for her unwavering and constant positivity; my grandparents for aiding to raise me to the best of their capabilities; and my little brothers for all the laughs and joyous moments.

I have also made a lot of friends in these past five years to which I have to thank for more things than I realize myself. I would like to thank them for putting up with me for this long and enduring all the challenging moments alongside me; for the companionship that everyone displayed towards me and also for all the amusing little moments we shared. Finally, I would like to thank all of my friends for making my college life an unforgettable one.

I would like to thank my thesis supervisor, Ana Paiva, for her collaboration and readiness to help and motivate me throughout this research work. It is thanks to her guidance that I have somehow managed to strive for the objectives I had imposed to myself.

There are so many other people I have to extend my thanks to. To all those who cared for me and my success, I give you my most truthful thanks.

André Gomes Barbosa



*“You should enjoy the little detours to the fullest.  
Because that's where you'll find the things more important than what you want.”*

Ging Freecss



# Contents

<b>Introduction .....</b>	<b>1</b>
1.1 Context .....	2
1.2 Problem .....	2
1.3 Motivation .....	3
1.4 Structure of the document .....	3
<b>Literature Review .....</b>	<b>4</b>
2.1 Software Architecture .....	4
2.1.1 Multitier architectural style .....	5
2.1.2 Multilayered architectural style .....	6
2.1.3 Client/Server architectural style .....	8
2.1.4 Component-Based architectural style .....	9
2.1.5 Object-Oriented architectural style .....	9
2.1.6 Choosing a suitable architectural style .....	10
2.2 Frameworks for game development .....	10
2.2.1 Unity 3D .....	10
2.2.2 Construct 2 .....	11
2.2.3 GameMaker .....	11
2.2.4 Stencyl .....	12
2.2.5 GameSalad .....	12
2.2.6 Choosing a suitable framework .....	12
2.3 Existing games for software testing .....	13
2.4 iLearnTest .....	16
2.5 Conclusions .....	21
<b>Approach .....</b>	<b>22</b>
3.1 The “Construct 2” mechanisms .....	23
3.1.1 Construct 2: Event-based programming .....	23
3.1.2 Project structure .....	24
3.1.3 The XML and AJAX Plugins .....	25
3.1.4 Setting up the project .....	26

3.2	The solution.....	26
3.2.1	“Hangman” minigame.....	28
3.2.2	“Drag-and-drop” minigame.....	30
3.2.3	“Equivalence class” minigame.....	31
3.2.4	Quiz minigame.....	32
3.2.5	The solution’s particularities.....	33
3.3	XML Structure.....	34
3.4	Steps to generate a new XML-based game.....	38
	<b>Validation.....</b>	<b>40</b>
4.1	Validation of the tool as framework.....	40
4.2	Conclusion.....	42
	<b>Conclusions.....</b>	<b>44</b>
	<b>References.....</b>	<b>46</b>
	<b>MIME types for HTML5 games.....</b>	<b>49</b>
7.1	Fundamental MIME types.....	49

# List of Figures

Figure 1: Three-Tier Web Application Architecture	6
Figure 2: Multilayered Architecture	7
Figure 3: Two-Tier Client-Server Architecture	8
Figure 4: U-TEST game.	14
Figure 5: Bug Hunt's structure.	15
Figure 6: TestEG's interface.	16
Figure 7: iLearnTest initial menu.	18
Figure 8: Chapter selection example.	19
Figure 9: Chapter's Objectives example.	19
Figure 10: Original <i>iLearnTest</i> 's structure diagram.	20
Figure 11: Implementation's Structure.	22
Figure 12: New menu layout.	27
Figure 13: "Hangman" example.	28
Figure 14: "Drag-and-drop to boxes" example.	30
Figure 15: Equivalence Class exercise.	31
Figure 16: End-of-chapter quiz results.	32
Figure 17: XML Structure.	34
Figure 18: <i>iLearnTest</i> website.	38
Figure 19: "Game edition" page.	38
Figure 20: Uppercase-Lowercase example.	41
Figure 21: Uppercase-Lowercase XML structure.	41





# Acronyms

ACM	Association for Computing Machinery
XML	eXtensible Markup Language
P2P	Peer-to-Peer
IDE	Integrated Development Environment
HTML5	Hypertext Markup Language, version 5
GML	GameMaker Language
AJAX	Asynchronous JavaScript and XML
ADDIE	Analysis, Design, Development, Implementation, Evaluation
CSS	Cascading Style Sheet
JSON	JavaScript Object Notation
PHP	Hypertext Preprocessor
ISTQB	International Software Testing Qualifications Board
WWW	World Wide Web

## Chapter 1

# Introduction

The development of educational and training games is a research area that has grown quite a bit in the latest years, in a number of areas. We see it various times at the academic level, when trying to motivate students to learn a certain topic. There are also a number of games to help adults understand some concepts concerning areas such as healthcare, tourism, business management, defence, and so on. These are referred to as serious games, whose objective is to transmit educational content to the user, in a motivational and interactive way [1].

Serious games are designed for learning or for the practice of technical and non-technical knowledge, relevant to the individual's professional development. They include ludic and entertaining factors to increase motivation and enhance engagement in the learning process.

Educational games are designed to not only assist on learning certain subjects but also to expand on previously acquired knowledge, acting as a knowledge consolidation tool, while also allowing people to understand an historical event or culture, with the goal of enabling the acquisition of new skills in a more forgiving environment.

Some game-based learning benefits include a larger attention span from the students, the ability to reinforce knowledge in a friendly environment and to learn through experience and through mistakes [2]. For instance, simulator games not only reduce the risks imposed by real life situations but also reduce the costs associated with these risks, proper equipment, facilities and such. It is ideal for testing highly dangerous routines in a sandbox environment.

Another way to gauge the effectiveness of educational games is by analyzing our very nature. As children, human beings start their learning process with games in order to assimilate basic knowledge such as understanding numbers and words. The same principle can be applied to teens and adults; by using games, it is easier and a lot more enjoyable to acquire and to consolidate knowledge.

### 1.1 Context

Software testing requires a considerable amount of techniques and concepts; even if the theoretical contents are assimilated it might not be enough for students to apply said contents in a real life situation which makes it hard for companies to hire them [3]. The use of games as an alternative method to traditional learning usually leads not only to better assimilation of contents (since it requires more concentration) but also their application in real life problems [4]. This research project relates to the following areas: serious games, software testing and game-based learning.

ACM Classification:

- Applied computing → Computer games;
- Social and professional topics → Testing, certification and licensing;

Keywords:

- Serious games; Software Testing; Educational game; Game-Based Learning.

There are certain qualifications aimed at professionals who must acquire practical knowledge of the fundamental concepts of software testing. For instance, the Foundation level qualification provided by the International Software Testing Qualifications Board [5]. The basis of this Foundation is known as Foundation Level Syllabus and is provided to several examination bodies for them to approve the training providers and to derive examination questions in their local language. Training providers produce courseware and determine appropriate teaching methods for certification, and the syllabus will help candidates in their preparation for the examination. The game discussed in this research work, *iLearntest*, incorporates the syllabus contents and presents it in an engaging manner by including different challenges for each of the syllabus' topics.

### 1.2 Problem

With the development of this project, it was intended to comprise the theoretical contents of software testing in a game, incorporating challenges for each different topic to promote the user's engagement during the practice of software testing knowledge and techniques. The idea is also to offer the possibility of individual study so that each user can learn at its own pace. Other ways to motivate the user include a scoreboard system in which we would record the student's best score and the maximum score; this is an incentive for the student to reach the maximum score for each exercise while competing with other users.

## Introduction

The main purpose of the project is not only to develop and improve this game's non-functional requirements but to enlarge it into a framework by taking a more dynamic approach. For instance, reading the set of problems from a XML file which would be separated from the game implementation. The content of the files would be read into a set of game templates developed in Construct 2, a game development framework introduced in section 2.2.2. This way, it is possible to extend iLearnTest to more academic areas other than software testing learning.

### 1.3 Motivation

While software tests are seen as quality measures, testing techniques are still rarely applied by software development companies, factor which might be caused by lack of qualified and available professionals to implement these techniques [6]. This is a problem that affects not only software testing but also other areas that have a special need for highly competent personnel such as healthcare and business management. In order to surpass this recurring problem a new type of learning platform capable of adapting to each area's procedures is needed. This platform's purpose would be to generate educational games which should not only be easy to understand and utilize but also stimulate the learning process in a more pleasant and enjoyable manner.

This research work results in the proof of concept of a platform that allows users to learn the respective contents at their own pace and in a fun, motivational way. These sorts of measures should be reassuring for people entering the work market, where one's knowledge is of utmost importance.

### 1.4 Structure of the document

Besides the introduction, this document contains another four main sections. The following section describes the current state-of-art while mentioning other projects or problems that are relevant and interesting to some degree, regarding the problem at hand. This section includes a technological review of game development frameworks and software architectures that can or have been used for the development of similar projects. The document also goes over the work that has been done previously regarding the iLearnTest game: its structure, website, implementation details, the values that it transmits to the user and the approach applied during the most recent developments. There is also a section that includes the discussion regarding the experiment performed in order to validate the tool. Lastly, we draw some final conclusions based on the viability of the solution and possible future developments.

## Chapter 2

# Literature Review

This chapter contains the documentation of the results of the research on the current state-of-art which covers different possible software architectures for the problem at hand and how they are applied to educational software. We also address the existing development frameworks as well as games, including our very own iLearnTest.

### 2.1 Software Architecture

The main purpose of this research project is to develop a platform to generate educative games, which means it is necessary to separate the contents of these games from their implementation. The best way to do this is through the implementation of a suitable architecture.

Software architecture is the process of defining a structured solution that meets the technical and operational requirements while optimizing performance, security and manageability. The reason architecture is so important is that software must be built on a solid basis. The risks exposed by poor architecture include software that is unstable, is unable to support future business requirements, or is difficult to deploy or manage in a production environment. [7]

Systems should be designed with consideration for the user, the IT infrastructure, and the business goals and a balance must be found between competing requirements across these three areas. This is achieved by constructing an architectural solution that addresses relevant concerns such as:

- The quality attribute requirements such as security and performance;

## Literature Review

- How the application is going to be used;
- How is it going to be deployed and managed;
- How the application can be adaptable and flexible over time.

Application architecture's goal is to identify the requirements that affect the structure of the application and then link business and technical requirements by understanding use cases and finding ways to implement those use cases in the software. A good architecture means fewer risks associated with a technical solution and is sufficiently flexible to adapt to changes that may occur over time in hardware, user scenarios or requirements. In other terms, applications should be built to change instead to last by considering how it may need to change in face of new requirements. Even if the architecture may expose the structure of the system, it should hide the implementation details and grasp all of its use cases or scenarios while handling both functional and quality requirements.

For each architectural style, one must consider its key principles and major benefits in order to establish the best possible architecture for a given problem. The following subsections contain a list of architectural patterns and styles frequently used to provide a solution to frequently recurring problems. Each one of these patterns is basically a set of principles that improves partitioning and promotes design reprocess.

Note: The terms "tier" and "layer" are frequently used interchangeably. There is, however, a distinction between these expressions where 'tier' is used when representing the physical layout of various components in a system's structure, while 'layer' is used when representing the orientation of the different physical or conceptual elements that form an software solution. For example, a three-layer solution could easily be deployed on a single tier. [8]

### **2.1.1 Multitier architectural style**

This is fundamentally a client-server architecture in which presentation, processing and data management functions are physically separated. The most common multitier architecture is the three-tier architecture which is usually composed of a presentation, domain and data storage tier. This architecture segregates functionality into separate segments in a similar way as the layered style, but with each segment being a tier located on a physically separate component. [9]

- Presentation tier comprises the user interface which translates tasks and results to information the user can understand such as, for example, a browser;
- Logic tier coordinates the application, processes commands, makes logic decisions and performs calculations. It also processes data between the presentation and data tiers. An example of a logic tier component could be a framework such as the Construct2 or Unity tools;

- Data tier is where the information is stored and retrieved from. Usually takes the form of a database, file system or, in this case, XML file. The information is processed by the logic tier and then presented to the user by the presentation tier.

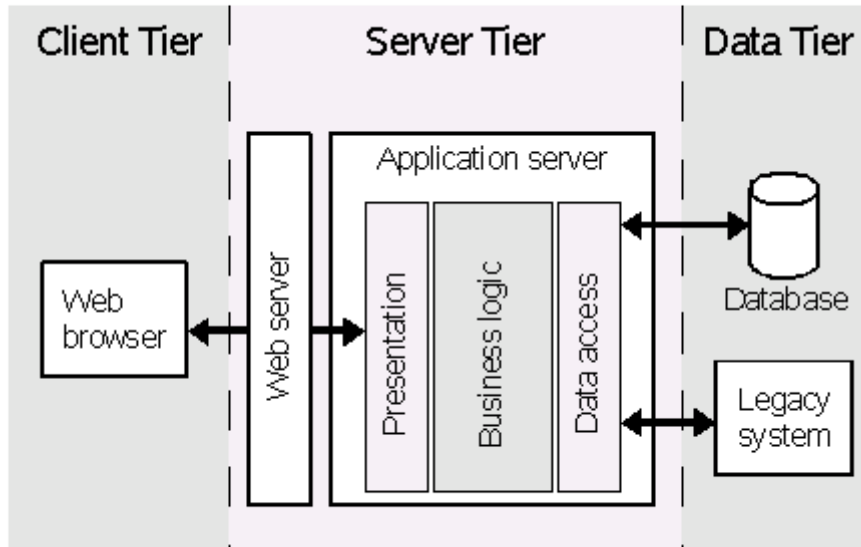


Figure 1: Three-Tier Web Application Architecture [10]

The main benefits that the N-tier/3-tier architectural style offers are:

- Maintainability, since each tier is independent of the other tiers and updates or changes can be carried out without affecting the application as a whole;
- Scalability of the application which is realistically feasible since tiers are based on the deployment of layers;
- Flexibility, because each tier can be managed or scaled independently;
- Availability since applications can exploit the modular architecture of enabling systems using easily scalable components.

### 2.1.2 Multilayered architectural style

This software architecture uses various layers for assigning the different responsibilities of a software product. In a logical multilayered architecture for a system with an object-oriented design, the following four layers are the most common [11]:

- Presentation layer consists in the user's interface that presents the information to the users in an understandable way. This layer is equivalent to the presentation tier in the multitier architecture;
- Application/service layer aims at providing middleware that serves third-party services and applications at a higher application layer;

## Literature Review

- Business/domain layer is the part of the program that encodes the real-world business rules that determine how data can be created, displayed, stored, and changed;
- Data access layer handles logging, networking, and other services which are required to support a particular business layer.

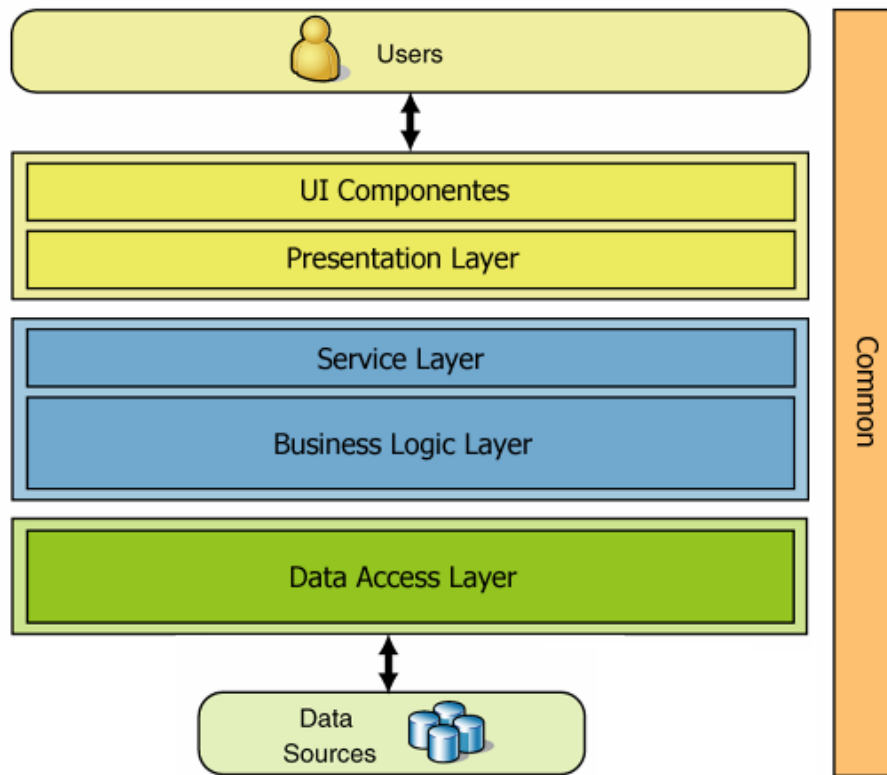


Figure 2: Multilayered Architecture [12]

The main benefits of the layered architectural style are abstraction, isolation, manageability, performance and reusability:

- Abstraction: Layers allow changes to be made at the abstract level. The level of abstraction used in each layer can be increased or decreased;
- Isolation: Technology upgrades can be isolated to individual layers in order to reduce risk and minimize impact on the overall system;
- Manageability: Separation of core concerns helps to identify dependencies, and organizes the code into more manageable sections;
- Performance: Allocating the layers over multiple physical tiers can improve scalability, fault tolerance, and performance;
- Reusability: Roles promote reusability by providing a user-customized view on to the same data and functionality;



- Testability: Increased testability from having well-defined layer interfaces.

### 2.1.3 Client/Server architectural style

The simplest form of client/server system involves a server application that is accessed directly by multiple clients, referred to as a 2-Tier architectural style. The client/server architectural style describes the relationship between a client and one or more servers, where the client initiates one or more requests, waits for replies, and processes the replies on receipt. The server typically authorizes the user and then carries out the processing required to generate the result. The server may send responses using a range of protocols and data formats to communicate information to the client [13].

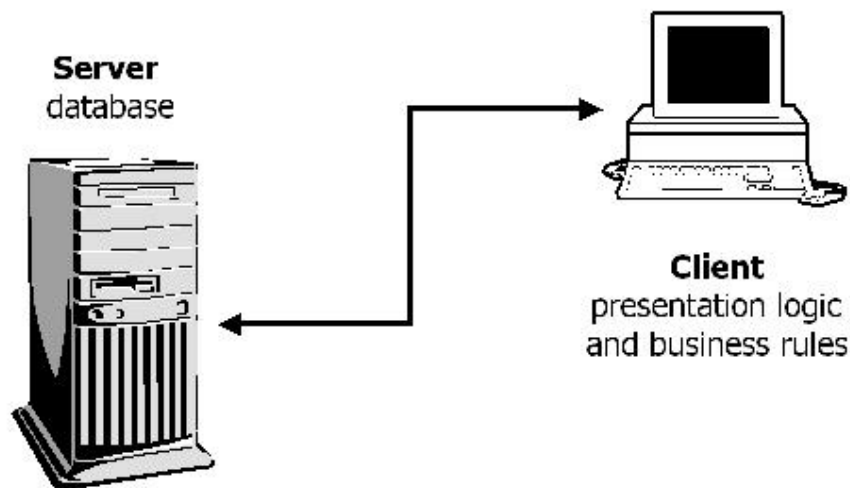


Figure 3: Two-Tier Client-Server Architecture [14]

The main benefits of the client/server architectural style are higher security, centralized data access and ease of maintenance. All data is stored on the server, which generally offers a greater control of security than client machines. Exactly because data is stored only on the server, access and updates to the data are far easier to administer than in other architectural style. Roles and responsibilities of a computing system are distributed among several servers that are known to each other through a network. This ensures that a client remains unaware and unaffected by a server repair, upgrade, or relocation.

Other variations on the client/server style also include Peer-to-Peer (P2P) applications. The P2P style allows the client and server to swap their roles in order to distribute and synchronize files and information across multiple clients. It extends the client/server style through multiple responses to requests, shared data, resource discovery, and resilience to removal of peers [13].

### **2.1.4 Component-Based architectural style**

Component-based architecture describes a software engineering approach to system design and development. It focuses on the decomposition of the design into individual functional or logical components that expose well-defined communication interfaces containing methods, events, and properties [13]. The key principle of the component-based style is the use of components that are:

- Reusable in different cases and scenarios;
- Replaceable;
- Not context specific, in order to be able to function in different environments and contexts;
- Extensible from existing components in order to provide new behavior;
- Encapsulated, as to not reveal details of the internal processes, variables or states;
- Independent or with minimal dependencies on other components to avoid affecting them or the system when deployed or replaced.

The main benefits from following a component-based architecture include ease of deployment, reduced costs of development and maintenance and improved reusability by using third-party components, ease of development and mitigation of technical complexity through the use of a component container and its services.

### **2.1.5 Object-Oriented architectural style**

An object-oriented architecture projects the system as a series of cooperating objects, instead of a set of routines or procedural instructions. Objects are discrete, independent, and loosely coupled. They communicate through interfaces, by calling methods or accessing properties in other objects, and by sending and receiving messages [13]. The key principles of the object-oriented architectural style are:

- Abstraction, which reduces a complex operation into a generalization that retains the base characteristics of the operation;
- Composition, since objects can be assembled from other objects and can choose to hide the internal objects from other classes or expose them as interfaces;
- Inheritance which makes maintenance and updates easier, as changes to the base object are propagated automatically to the inheriting objects;
- Encapsulation, as the objects' functionality is exposed only through methods, properties, and events, and hides the internal details such as state and variables from other objects;

- Polymorphism which allows to override the behavior of a base type that supports operations in the application by implementing new types that are interchangeable with the existing object;
- Decoupling of objects from the consumer by defining an abstract interface that the object implements and the consumer can understand.

The main benefits of the object-oriented architectural style are that it maps the application more closely to the real world objects, reusable through polymorphism and abstraction, testable through encapsulation and extensible due to changes in the representation of data not affecting the interfaces that the object exposes.

### 2.1.6 Choosing a suitable architectural style

While analyzing these architectures, it was decided that a three-tier based architecture would be more appropriate to improve *iLearntest*. Firstly, we assume that the data tier consist on our XML file. It is also possible to use a tool such as Construct2 to handle the logic behind the several minigames included in *iLearntest* and use them as templates which will load and process the data from the XML. The resulting information is then presented to the user in the form of a browser game (see Figure 11: Implementation's Structure in section 3).

The reason why the N-tier/3-tier architecture was selected over the remaining ones is because it is characterized by the functional decomposition of applications and their distributed deployment, providing improved scalability, availability, manageability, and resource utilization. Should it be needed, this style can also act as client/server architecture.

## 2.2 Frameworks for game development

This section is devoted to explaining the currently existing frameworks for game development. After conducting a thorough study on the topic, it was concluded that the main frameworks are:

### 2.2.1 Unity 3D

Unity 3D [15] is classified as a game development tool for intermediate and advanced users:

It is one of the most well-known frameworks in game development. The IDE is very polished and easy to use, but being a 3D tool means that a certain level of knowledge is needed before getting started. Unity supports three languages: UnityScript (which is similar to JS), C#, and Boo.

The pro version adds lots of features for more advanced game developers. It's also possible to export to consoles, such as the Xbox, PS3, Wii U, and more, although those licenses cost even more.

Unity has a Web player that relies on a plugin, so while it is possible to export a game to the web it will not be playable on mobile browsers. Finally, Unity is in the process of releasing a 2D workflow that will be a very useful tool for those not interested in developing 3D games.

### **2.2.2 Construct 2**

Construct 2 [16] is generally classified as a game development tool for beginners:

Construct2 is an easy, friendly tool for making games. It employs a drag-and-drop behavior system, where you build up game logic from pre-made scripts that are attached to game's elements.

It has its own Web-based tool and support for publishing to a number of different platforms. Construct2 games are built in HTML5 and, because of this, it is ideal for publishing a game on the Web. The only major down side to Construct2 is that there is no coding aspect of making the game, so the developer is fully dependent on what Scirra, Construct's fabricant, has provided. And, while it is possible to add additional functionality via plugins, it is not ideal if the developer wants to manually tweak the game.

### **2.2.3 GameMaker**

GameMaker [17] is classified as a game development tool for beginners and intermediate users:

It is a great tool for making 2D games since it is incredibly powerful. Some of well-known indie success stories got their start in GameMaker.

GameMaker is similar to Construct2 in ease of use, since it can perform drag-and-drop actions, event-based coding, and more advanced users can take advantage of its built-in scripting language called GML (GameMaker Language). GML is C based, so if one knows or ever programmed in C, JavaScript, Java, or C#, it will seem rather similar.

The language does have limitations, such as limited data structures and no classes. While the UI of GameMaker is a little rough around the edges, it is still an excellent tool for 2D games, and its support for publishing to desktop, mobile, and HTML5 should not be overlooked.

#### **2.2.4 Stencyl**

Stencyl [18] is classified as a game development tool for beginners:

Stencyl is a game creation platform that allows users to create 2D video games for computers, mobile devices, and the web. Games created in Stencyl can be exported to the web via Adobe Flash Player, and to personal computers as executable games, as well as onto various mobile devices as iOS and Android applications. The physics and collisions are handled by a two-dimensional physics simulator engine.

Besides being an authoring tool, Stencyl can also be classified as an integrated development environment (IDE) and it includes a number of modules in order to manage and edit the game's behaviors, entities/actors, tile sets and levels. Additional tools allow the user to import images as foregrounds and backgrounds in scenes, import and edit fonts, import sounds and music files, and alter game settings such as player controls and game resolution. Stencyl can alternatively be set up to utilize external image editors, such as Photoshop and GIMP, to modify images already loaded into a project.

Stencyl has a "design mode" which is an interface that allows users to create modular game logic for actors and scenes using a visual programming language. As such, users are not required to learn or type out a particular programming language, nor must they concern themselves with syntax. Rather, available actions are dragged and dropped from a palette of "code blocks".

#### **2.2.5 GameSalad**

GameSalad [19] is classified as a game development tool for beginners:

GameSalad is a tool that makes use of a drag-and-drop system in order to ease the development of games for users with no programming knowledge. It includes visual editors and a behavior-based logic which are often used for the development of quick prototypes and the auto-publication of multi-platformers.

This system provides a graphical interface used to describe the rules and behaviors of game objects, with no need for additional coding. The application contains a library of behaviors to choose from such as movements, attributes and collisions that can be grouped to form a new behavior. GameSalad includes features such as publication on multiple platforms, real-time editing, game preview, scene editor and expression editor for complex behaviors. The free version retains most functionality except publication in Windows 8 and Android.

#### **2.2.6 Choosing a suitable framework**

After analyzing all the alternatives, it was decided that the Construct 2 framework would be viable enough to provide a proof of concept of a XML-based game generation platform. This

decision was based on a number of factors in which Construct 2 surpasses the other mentioned frameworks. When compared to GameMaker, for instance, Construct 2 can define object actions as “behaviors”; since it is needed to define every single movement in GameMaker, this feature saves the developer a lot of time. The layout editor is also superior since it does not require the user to create a new sprite simply for the sake of resizing or repositioning it. Construct 2 also offers in-built object types that make it easy to add different elements to the game. These advantages presented by Construct 2 rule out other frameworks such as Stencyl, GameMaker and even GameSalad [20]. This last one has a couple of downsides such being limited to Mac only and not letting the users upload a HTML5 game wherever they want - it can only be uploaded to their arcade and not to a private server. The other strong contender would be Unity tools but it was concluded that, despite its advantages, Unity is far better suited for 3D games development than 2D.

Although Construct 2 has a considerable learning curve, it also offers some key benefits such as the plugins to inject content into the game from XML files through AJAX requests. The tool offers a powerful engine that destroys needless objects/elements and creates new ones during runtime while needing very little “garbage collection” (releasing unused memory blocks), which would otherwise take a lot of CPU time and it would slow down the game considerably. The engine also takes advantage of Javascript features like object sealing, preventing new properties from being added to game objects and marking all existing properties as non-configurable, which helps guarantee the browser will run the code as fast as possible [21]. The ability to export the project to HTML5 is a requirement in order to easily embed the game into the *iLearnTest* webpage. Since this framework has been used in previous implementations of this game, it is also possible to reuse some parts of the game such as the game layouts/interfaces and maintain its consistency with other game elements even if the logic behind said games is modified. Construct 2’s architecture has also high modularity in the sense that its functionalities are separated in plugins which can be easily extended; it also compatible to some degree with multi-level architectures since it allows for a separate data layer with the XML and AJAX plugins.

### **2.3 Existing games for software testing**

Throughout the research on the topic of educational games for software testing learning, we have found an interesting empirical study about the subject, by Alessandra Zoucas et al. [6]. In said study games to support Software Engineering teaching were sought, using terms such as software testing game learning, software testing game based learning and software testing teaching approach. It was observed a lack of specific games for software testing.

## Literature Review

As such, a new search was conducted directed at software engineering games revealing quite a number of simulation games for desktop such as SimSE [22], TIM [23], Planager [24], SESAM [25], X-MED [26] and Simules [27]. In the same article, it was found that, since there was no game solely devoted to software testing, “U-TEST” was developed to assist software testing teaching. It is a simulation game with special focus on unit tests; it is based on the ADDIE model [1] and it is destined for Software Engineer students. In the game, the player takes part in a project, building the test cases to the presented functions. The player will receive a ranking based on performance.

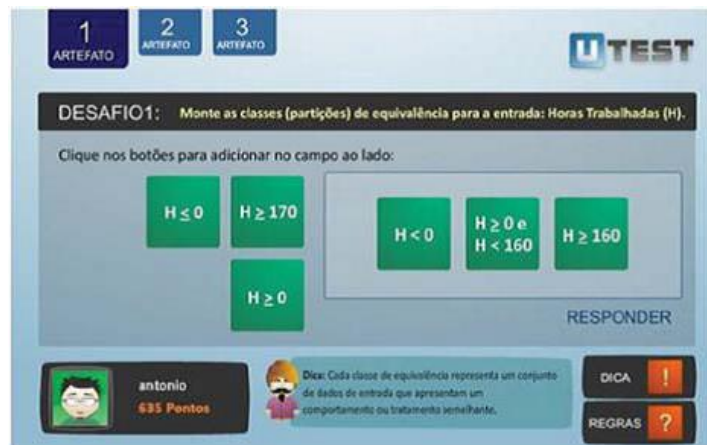


Figure 4: U-TEST game.

The player undergoes ten stages throughout the game with the following challenges:

- Presentation of artefact;
- Setup the equivalence classes;
- Define limit values for identified classes;
- Select the correspondent value to the identified value;
- Setup a cause-effect graph or decision tree;
- Project final feedback.

By the end of each stage, the user receives a score based on graph and commentary. When the game is cleared the player receives feedback on its overall performance and ranking position on the scoreboard. While going through other articles, there were found a couple of interesting software testing learning games which were quite relevant to the topic. Those games were Bug Hunt [28] and TestEG [29]:

- Bug Hunt [28] is an online tutorial, displayed on a web browser and developed to motivate students to learn software testing techniques. It contains various characteristics that entice both students and teachers:
  - Incorporating challenges for each separate class while providing immediate evaluation to motivate the student to practice fundamental knowledge;

## Literature Review

- The students can solve the problems at their own pace;
- The solution is configurable to a certain degree in order to fit the teacher's requirements;
- The teacher gets automatic feedback on the performance of the students.

During first-time usage, the users are presented with instructions on the objectives of the tutorial and a brief description on how to make use of it. Throughout the tutorial, the student goes through a series of classes using specific testing techniques in order to find the errors. The user's progress is evaluated through the number of inaccuracies detected. Each class contains a set of objectives, an exercise and, finally, the results. The exercise is composed by the following components: instructions, artifacts, tests and help tips. As soon as the tutorial is completed the user receives the overall evaluation based on performance. The following figure illustrates the whole process from the student's perspective.

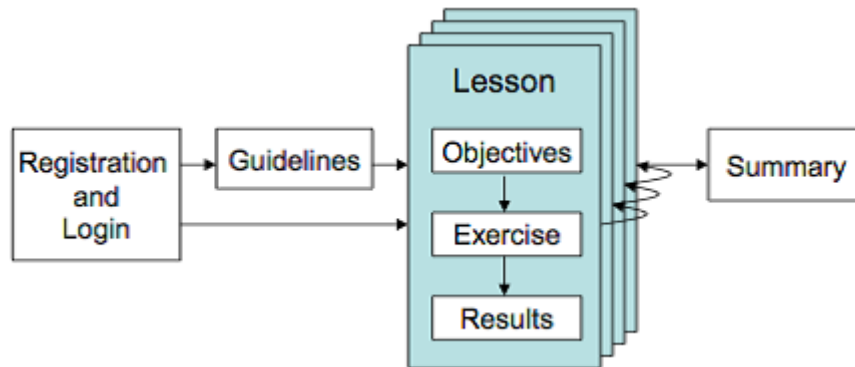


Figure 5: Bug Hunt's structure.

The basic structure contains four lessons. The first lesson gives a basic introduction about software testing with the objective of finding a flaw through the use of test cases. In the following lessons, the user learns about other relevant concepts such as black box and white box techniques. All the progress is saved by the end of the session.

Additionally, the teacher receives complete reports of the student's results. As mentioned, the teacher has the option to customize his own lessons by proposing the exercises the development team.

- TestEG [29] is an educational computer game in the shape of a software testing quiz:





Figure 6: TestEG's interface.

The game simulates a situation in a company, where the user is the manager of the testing department. The player receives a budget and has to hire three testers. Throughout the game the manager must help his team solve the problems they find. The player must answer ten questions in ten minutes while making sure to not run out of budget. The user can spend some of the budget training his team, verify their performance and read theoretical content on software testing. The game also contains a scoreboard.

This game is also teacher-oriented. As a teacher you can register your students and control the content transmitted to them so that you can evaluate them and have an idea of their difficulties.

## 2.4 iLearnTest

One of the purposes of the project is to improve, from a non-functional requirement point-of-view, a previous version of the game *iLearnTest* which is basically an online platform with authentication access that allows the users to play a multi-level game with software testing contents. It also shows a scoreboard with data on different users in order to promote competition between them, improving their motivation [30]. The “weakness” of this platform was that it only supported one pre-determined problem statement which is hard coded. In other words, the initial implementation did not support various sets of problems unless one would change it in the code.

## Literature Review

*iLearnTest* is also intended to be attractive to students. Not only it transmits the necessary knowledge, but it also intends to:

- Incorporate challenges of each content to promote the students engagement and the practice of software testing knowledge and techniques;
- Offer the possibility of individual study so that students can learn at their own pace;
- Have a score total for each problem/game in a manner that incentives the student to reach maximum scores, increasing their knowledge;
- Present the final results to students and the correct answers to the problems.

*iLearnTest* was developed according to the ADDIE methodology[1] which is learning-oriented and includes five stages:

- Analysis: identifying the targets of the system (in this case, computer science students who needed to polish their software testing skills) as well as the study of the state-of-art which includes existing tools, games and methodologies;
- Design: The objectives of the application are specified; In this case, it includes:
  - Review Process;
  - Static analysis;
  - Test conception techniques and categories;
  - Black-box based techniques;
  - White-box based techniques.
- Development: Interface and content creation. The contents revolve around the objectives discussed above while the interface was created via Construct 2 due to the possibility to export the project to HTML5 and its easy-to-use drag-and-drop interface; Note: Since the free version has a limited number of events, the upgraded version was used, which required a fee;
- Implementation: Game programming and web site development. The website contains:
  - An entry page, with required authentication;
  - An initial page, where the user can access the *iLearnTest* game;
  - An information page, with details regarding *iLearnTest*;
  - A contact page, which allows to user to contact the developer(s) regarding questions or suggestions.

Since the game was already exported to HTML5, the game runs on every browser. CSS was used for page style definition, while JavaScript and jQuery helped implement dynamic content on the client's side. JSON allowed for an easier data transmission between the server and the pages. Finally, PHP was used in the server side for a better connection between the client side and the database, which was created in SQLite;

- Evaluation: Validation of the development tool with real users and documentation of the results.

The following is a brief overview of *iLearnTest*'s original structure.

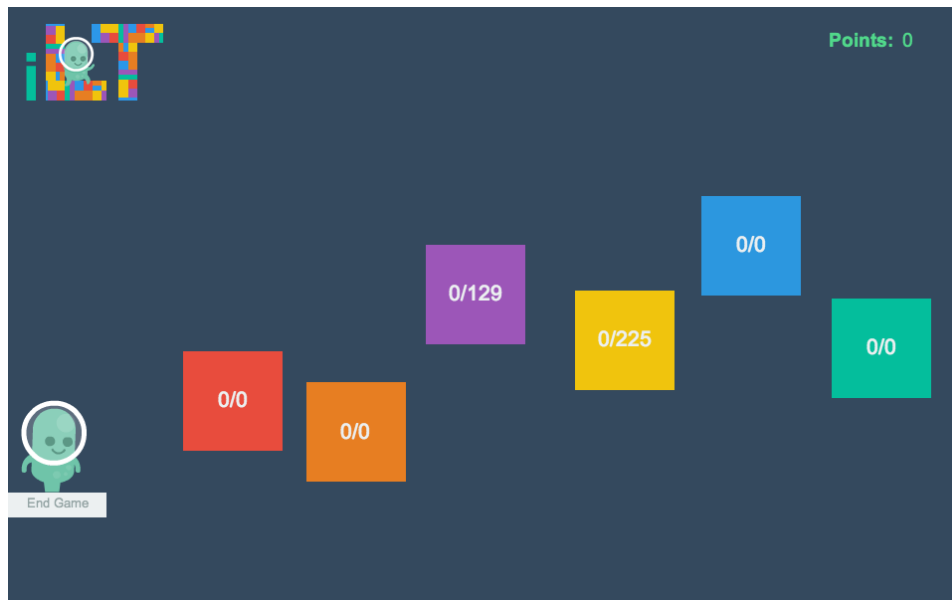


Figure 7: iLearnTest initial menu.

The game currently follows the structure previously provided by Foundation Level Syllabus [5]:

- Testing fundamentals;
- Testing through the software's life cycle;
- Static techniques;
- Test conception techniques;
- Test management;
- Test support tools.

These contents are represented by the six colored platforms in the figure above, which correspond to Syllabus six main chapters. By placing the character on a platform, a balloon pops up showing the respective title. It's also possible to check the player's score in comparison with the maximum score. In the example provided by figure 7, only chapters 3 and 4 are playable with maximum scores of 129 and 225, respectively.

## Literature Review



Figure 8: Chapter selection example.

Inside each chapter, there are the correspondent learning objectives, theoretical content and the game/challenge for that content.

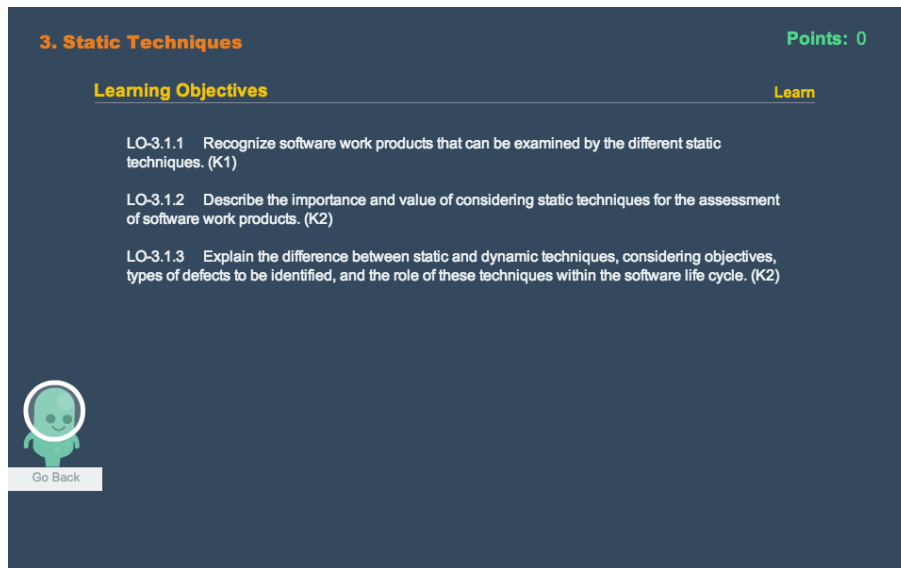


Figure 9: Chapter's Objectives example.

For each chapter there are different types of games. The available games include:

- Guess the concepts / “hangman” game: the player must guess the concepts that are described by the given hints with focus on formal review; For instance, a typical formal review has the following main activities: planning, kick-off, individual preparation, review meeting, redo work and follow-up;
- Drag-and-drop into boxes: the player must organize the contents in two separate boxes; this exercise is mostly used to distinguish between black-box and white-box techniques.
- Grab the right concepts / cascade-type of game: the player controls the character and has to grab the correct terms while ignoring and dodging the wrong ones in a certain context;
- Find the way / coverage games: the player must find the correct way through a maze in the form of graph. This game focuses on test/statement coverage contents.
- Equivalence classes / Equivalence partitioning: The player has to complete a equivalence matrix with the correct values or expressions according to the problem

## Literature Review

statement;

- Data flow games: This game is a bit more specific. The players have a segment of code and, for every line of this segment, they have to assign a data flow attribute to the variables of the code. Attributes can be “d” (define), “u” (undefine) or “r” (reference);
- Color game / Static analysis: the game has the following legend: red means “dead code”, yellow is “infinite loop” and green is “syntax error”. The player is given six segments of code to classify with the respective colors;
- Finally, in the end of each main chapter there is also a quiz in order to test the user’s newly obtained knowledge. The quiz is composed of about six questions; the results are there shown to the user along with the correct answer.

The following diagram represents the structure of the *iLearntest* game as it was originally, its contents and the chapters where the mentioned mini-games are used.

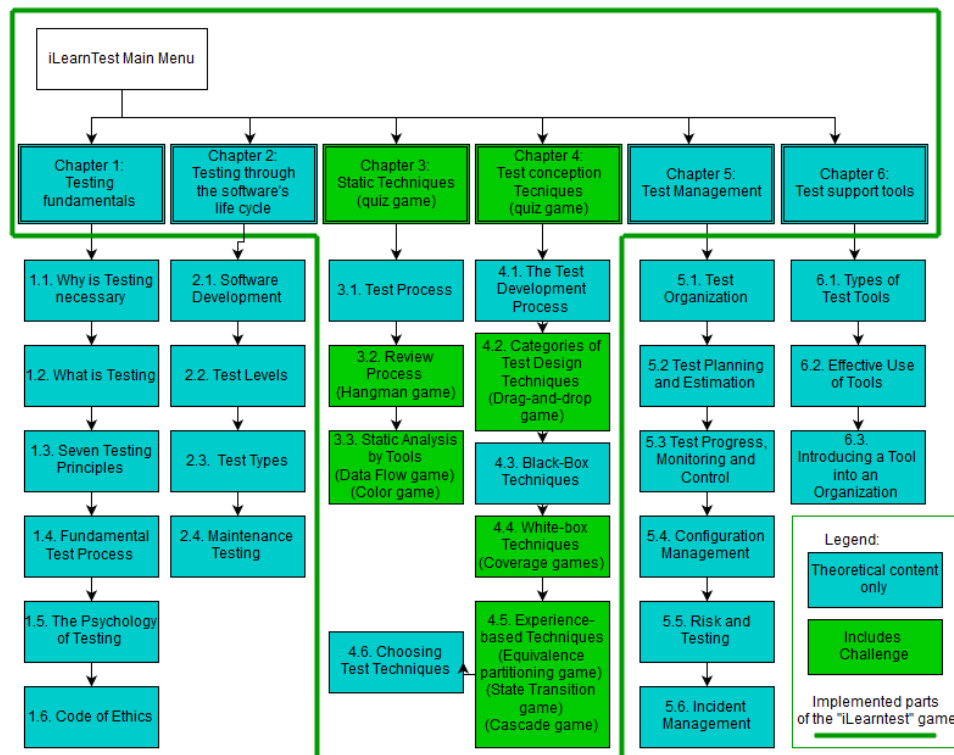


Figure 10: Original *iLearntest*'s structure diagram.

This diagram follows a similar structure to the provided by Foundation Level Syllabus and shows just how much of syllabus was implemented in the original *iLearntest* game. In this case, only chapters 3 and 4 were fully implemented as shown in the figure. There is a structure created for the remaining chapters; each chapter has its own screen with the respective subchapters displayed but empty. The double boxes in the diagram represent the six main chapters, while the blue boxes signify subchapters with theoretical content only. The green boxes represent the chapters where challenges have been implemented successfully. Only fully implemented chapters have the “final test” section.

It is important to stress that the *iLearntest* game has no data independence since the content is directly coded in the game layout. As perceived from the previous diagram, the game follows

a slightly rigid structure with six main chapters and its contents separated in subchapters. The problem with the original structure is that, due to not separating contents from the game engine, implementing a similar game would actually require to develop a whole new one from the beginning. Every chapter and screen of the game is “custom made” with the contents typed directly in the game’s layouts and it does not rely on any sort of databases other than the list of players. For instance, it is possible to expand the game despite the highly time-consuming efforts but it wouldn’t be possible to adapt it to other topics or educational areas without revisiting and altering every single screen in order to support a different context. This problem can be solved by remodeling the game, creating screens and mini-games prepared to receive the respective contents from a separate file rather than having the content hard coded into the game.

## 2.5 Conclusions

This study of the state-of-art provides a clearer insight of the most used frameworks on the development of serious games for education as well as the most commonly utilized software architectural patterns.

It was reasonable to conclude that the Construct 2 framework would be viable enough to provide a solution for the problem at hand, despite its learning curve. Some of its key benefits include the ability to inject content into the game from XML files through AJAX requests rather simply as well as the capability to export the project to HTML5 in order to easily embed the game into the *iLearnTest* webpage. This framework has been used in previous implementations, fact which allows reusing some of the game’s layouts/interfaces and maintaining its consistency with newer game elements.

Once it was decided to use Construct 2, the solution naturally took the form of a n-tier/three-tier based architecture. The original architecture of the *iLearntest* would benefit the most from a three-tier architecture which is typically characterized by the functional decomposition of applications and their distributed deployment, providing improved scalability, availability, manageability, and resource utilization. The three-tier architecture consists of data, logic and presentation tiers; For instance, the data tier consists on the XML content file while the logic behind the game is handled by the Construct 2’s engine; the game’s layouts presented through the browser serve as presentation tier.

## Chapter 3

# Approach

The approach for this project was mainly to change the previous implementation into a more dynamic one; for instance, reading the set of problems from a XML file which would be separated from the game implementation. The main idea is to have a set of game templates that can read the content of the exercises from the file, so that we can use the same game for the learning process of a variety of themes. In short, the objective is the proof of concept of a platform that is capable of generating educational games based on content loaded from a separate file. This solution allows a platform capable of loading content into *iLearnTest* from XML via AJAX request, in order to ease future updates and development of the tool. The administrator would be able to edit these contents while the player would interact with the game via browser. The data of the players would be saved to a database as exemplified on the following figure.

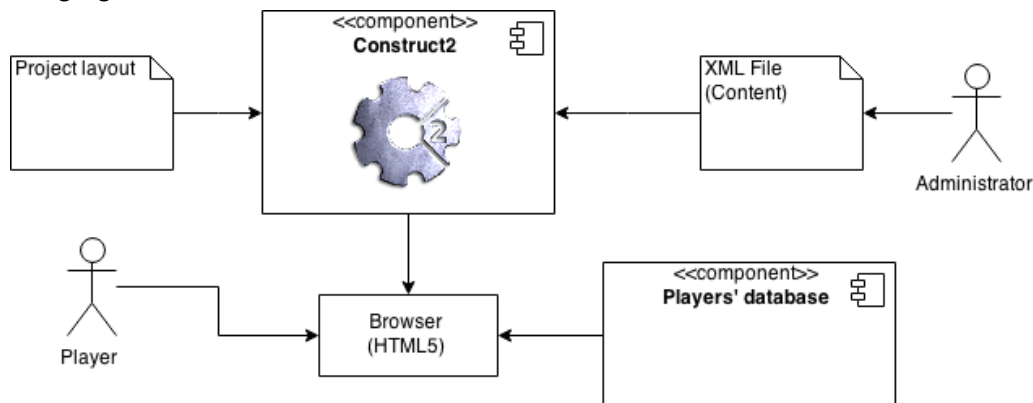


Figure 11: Implementation's Structure.

### 3.1 The “Construct 2” mechanisms

In order to better understand the implementation used to reach the current solution, it is fundamental to understand the mechanisms behind the Construct 2 framework which were used throughout the development phase to implement said solution.

#### 3.1.1 Construct 2: Event-based programming

Events are Construct 2's main feature. It is possible to define how the game works using a logical block system instead of using scripts or programming languages. Construct 2 uses the event system exclusively and does not support any scripting in the editor at all which can be rather restraining at times. However, the purpose of the event system is to be powerful enough to fully replace scripting and is extended by developers from time to time [31].

The basic concept of events is that conditions filter the instances meeting the imposed condition and the operations run for those instances only. This allows the user to control instances independently, especially when used with instance variables. Not every instance of the same object has to act the same; events provide the tools to filter out individual instances and run actions on those that met the conditions.

Events normally consist of conditions that must be met and actions to run under those circumstances. Optionally, sub-events can be added to an event to cover more conditions or run more actions. Actions can be formed by expressions such as the computation of values, retrieving or modifying the properties of objects and destroying unused ones. Events allow for multiple conditions which make it possible to run actions on instances meeting only several criteria in the same sense as a logical “AND”. On the other hand, when no conditions are used, every instance of a given object is affected by the event's actions; in this case, since there are no conditions to limit the instances picked, all of them are picked. However, it is important to keep in mind that after an event ends, the next event begins anew; its conditions will start picking from all instances again. The exception to this rule is the sub-events mentioned above.

There are few important peculiarities to know about events. For instance, events always run from top to bottom in a given event sheet, which is a list of events. This means the order the events are placed on the sheet is of great importance; every event is checked once per tick (usually sixty times per second). The screen is only draw after every event has been run which means if two events accidentally cancel each other out nothing will happen. This happens to be the source of many problems when debugging the projects for unexpected issues. Conditions and actions in an event follow the same logic: first, the conditions are checked from top to bottom and then the actions are executed in the same fashion.



### 3.1.2 Project structure

Construct 2 projects typically consist of the following elements: layouts, event sheets, object types, the System object and project files [31].

- Layouts may also be referred to as scenes, rooms, frames or stages. These are associated an event sheet which controls the behavior of the objects in the scene. Layouts also consist of multiple layers, which can be used to arrange objects in to background and foreground layers;
- Event sheets are a list of events that define the game logic. Events are the alternative to programming or scripting. As mentioned above, layouts have an associated event sheet which handles their logic. Sheets can either be associated with layouts or imported into other sheets;
- Object types define classes of objects. Multiple instances of an object type can be created. Every instance shares the events and possibly the artwork for its object type. Events can affect only certain instances of given object types by imposing a number of conditions before applying actions. Should no conditions be imposed, a single action will affect every instance of target object type. This is partially the reason why instance variables are supported; instance variables are added to an object type and store numbers or text *per instance*. Additionally, every instance of every object type has an UID (unique ID). This ID is, as the name suggests, unique to every element of the game and can be used to directly target said elements and modify their instance variables;
- System object has no instances and exists in every project to provide access to built-in functionalities of the Construct 2's engine that fall into three categories:
  - System actions which are used to manage the properties of layouts and canvas, move between layouts, manage global variables and save / load game states from JSON in games with savedatas.
  - System conditions which are used compare to compare angles, global and local variables; check for a certain unique ID's of given objects; check value types (string, number); check the condition of a layout or layer (empty, visible); pick instances of objects by comparison, evaluation or number of instance; perform loops ("for", "for each", "while", "repeat" blocks); perform special conditions ("else" and "or" blocks) and perform actions at given times such as start / end of layouts;
  - System expressions which contains libraries with math, text and time functions.

## Approach

- Project files which comprise any external file that can be imported to the project; in this case, the XML file with the game contents and some picture files in PNG format used in the game.

It is important to note a couple of factors: in these projects there are a number of values such as angles, speeds and sizes which tend to always use the same units for consistency. For instance, positions are in pixels with the origin at the top-left of the layout; sizes are also in pixels, times in seconds, speeds in pixels per second and accelerations in pixels per second per second. Finally, all objects using a number of an item in a list or array (indices) start from zero instead of one. Since zero-based indexing is consistent with a number of programming languages, it is actually more convenient in many cases than one-based indexing.

### 3.1.3 The XML and AJAX Plugins

The XML plugin can parse and read data from XML documents and uses XPath to access the XML document, which is a kind of query language for XML similar to how SQL is a query language for databases. It is important to note that the XML plugin allows read-only functions; this means the game can read content from the XML file but not modify it [31].

XML content must be loaded as a string with the “Load” action. This is where the AJAX plugin steps in since it is necessary to request the XML file from a server using the AJAX object type. The AJAX plugin allows web pages or files to be requested during the execution of the game.

In order to make an AJAX request the “Request” action is used with the respective URL/path. A moment after, the “On completed” event will trigger returning the “AJAX.LastData”, an expression which can be used to access the content of the response. When the AJAX request concludes, it is possible to pass “AJAX.LastData” into the “Load” action and start reading the data that was received. Some of the most helpful XML conditions and expressions when handling these responses include:

- “For each node”: Repeats an event for each node returned by an XPath query; usually used with a query that returns multiple nodes to select every node of a certain kind which are then iterated;
- “NodeCount”: Returns the number of nodes returned by an XPath expression such as the number of elements with a certain name. This expression is particularly helpful since it works even with browsers that do not support XPath queries as is the case with Internet Explorer;
- “StringValue”: Since most of the contents used are text-based this expression is a must; returns a string from an XPath expression. In case multiple values are selected returns the first one or the current one if it is a “For each node” event.

### 3.1.4 Setting up the project

This subsection describes the steps performed to embed the final version of the game in the iLearnTest website (see section 3.4) and the required setup for the server's configurations.

In order to embed the game in a website it is necessary to include an iframe in the webpage's HTML code. An iframe is essentially a portal to another webpage so it allows opening up and displaying multiple pages within one webpage. In order to do this the game is exported to HTML5 and uploaded to an available web host; the iframe box is built by using the address of where the "index.html" file from the exported folder had been uploaded. For instance:

```
<iframe src="<URL to the project's index.html>" name="<Name of your Iframe>"
width="???" height="???" frameborder="0" scrolling="no" ><p>Your browser does not
support iframes.</p> </iframe>
```

However, it is important the server hosting the project is correctly configured. Specifically, it is important that the server sends certain types of file with the correct MIME type. Having the wrong MIME type set can result in the failure of AJAX requests which would be of utmost importance in this case. The required MIME types can be consulted in "Appendix A – MIME types for HTML5 games"

Finally, it is important to define beforehand which files should be cached or not. Since the XML is susceptible to modification in order to modify or update the game, it is best not to cache it. However, other files such as image files should be selected for offline caching in order to speed up the game's loading times and avoid slowdowns.

## 3.2 The solution

The XML contains the chapters, its sections, learning objectives and challenges for each section and the respective theoretical contents. Using Construct2 and its XML plugin, the problem statements can be loaded fairly easily into the game. Despite the advantage of exporting the code to HTML5, the problem resided on the game's side: there is no simple way to generate a game layout/screen based only on the XML contents, let alone a full level of the game. Since the game's structure is divided in chapters, it was decided to set a maximum limit to the number of chapters and subchapters. The structure of the game establishes an upper limit of six main chapters and each one of them can have up to six subchapters and another layer of subchapters below those. Hence, the game can hold a maximum of  $6^3$  (two hundred and sixteen) chapters altogether. The reason why this limitation happens is that the contents must be read into existing game elements in order to access their properties through their unique IDs. Each chapter and subchapter has a layout with a menu similar to the initial menu unless it is only comprised of contents and does not have any further subchapters. Should a given chapter

## Approach

have less than six subchapters, the unnecessary box/link objects in the layout are destroyed. These layouts show, for each chapter, the respective subchapters and the points the player earned in each one of them, the overall points obtained throughout the whole game and the access buttons to the chapter's final test and learning objectives. As a side note, it is important to clarify that all the scores, titles and texts are obtained from the XML file during execution and, because they are loaded into pre-created textboxes, there is a limit to the numbers of characters for each type of box. Also, only the main chapters have a “final test” section.

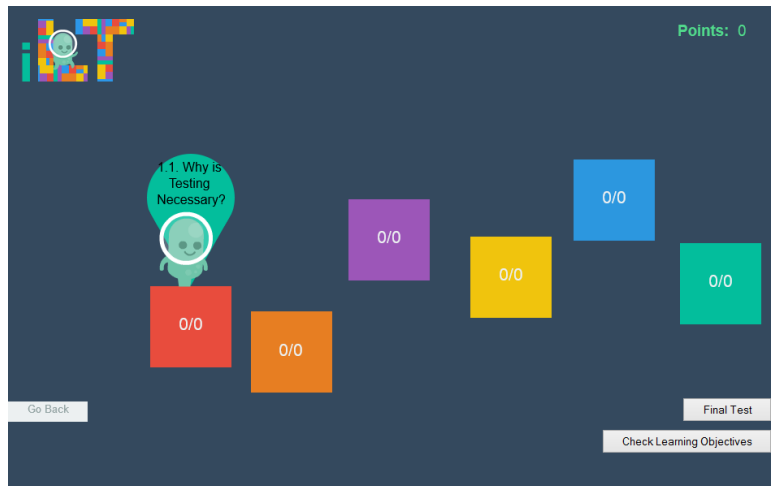


Figure 12: New menu layout.

The player can take the final test of the chapter at any point but it is advised to go through the remaining sub-chapters, if it is the case. Should the player choose to check the chapter's objectives, he shall be redirected to the text layout, which is a single layout that shows the theoretical contents of the respective chapter. By using a couple of global variables, it is easy to find out which chapter the text layout should load and to which layout the game should go back to. If a certain chapter does not have any further subchapters, the game will redirect the player to the chapter's contents. After going through these contents, the player can attempt to solve the chapter's challenge which is usually an exercise in the form of a mini-game. From the original set of mini-games that iLearnTest provided, these were implemented dynamically: “guess the concepts” (or rather, the hangman game), “drag-and-drop to boxes”, and an equivalence classes exercise where the player fills the empty blocks of an equivalence matrix.

### 3.2.1 “Hangman” minigame

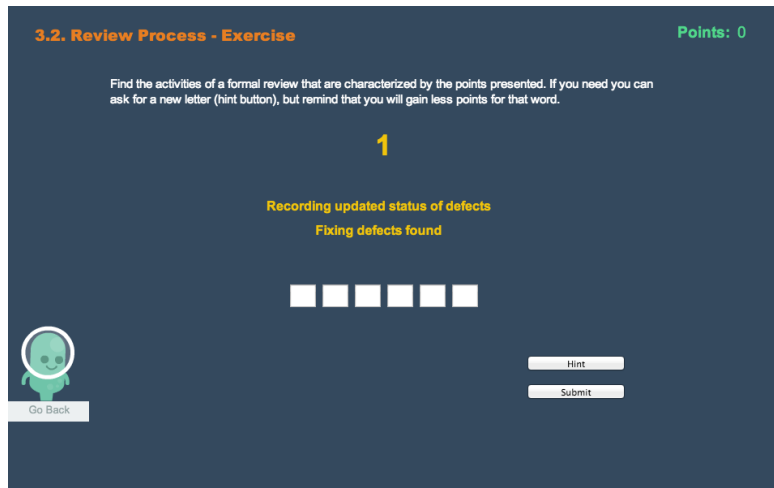


Figure 13: "Hangman" example.

Hangman’s limitations/rules: The player has to guess the correct word(s) through the tips given. There is a maximum of four tips per word but no limit to the number of expressions for the mini-game. The player can ask for further hints in the form of letters that compose the final word at the cost of some points. The problem statement, expression, tips and hints are all loaded from the XML file. Each expression can be composed of a maximum of two words with sixteen characters each. This exercise corresponds to the challenge originally displayed in *iLearntest’s* chapter 3.2.

Hangman’s implementation details:

Each box shown in figure 13 is actually an instance of a textbox object; they can be accessed and modified directly through their unique IDs but they can also be simultaneously modified which is useful to choose how many boxes to show at a given time, for instance. On start of layout some variables are initialized such as counters for the number of inputs, the number of correct inputs and the number of user hints and a variable to save the user’s score. The XML content is requested using the AJAX object. On request completed, The “AJAX.LastData” is loaded into the XML object. The title and problem statement of the exercise are loaded into existing text objects through their unique IDs.

An event runs on background to make sure each box allows only one character. The tips for the current word/expression are loaded into the respective text objects through their UID’s; the XPath expression to do so incorporates a counter that represents the number of the expression the player is trying to find. Since events are checked multiple times per second this event will automatically update the presented tips just by incrementing the counter that represents the current expression.

For every instance of the textbox object, it is checked if a given box should remain visible and enabled by using the length of the word loaded from the XML; the number of boxes

## Approach

surpassing a given length are set disabled and invisible and the “letter” attribute is set to null; otherwise the “letter” attribute will be set to the corresponding letter from the XML. This is used later to clarify which boxes should be accounted for during validation and to ease the comparison between the user’s input and the correct answer. Every correct input on an enabled box will award the user a certain number of points; to determine if the user guessed the expression correctly, the number of correct inputs is compared to the length of the expression. Should the user fail to guess it, the answer will be set visible on screen before moving on to the next one. The number of hints used is decreased from the user’s score even if given the correct answer.

The number of expressions the user has to guess is known simply by using *XML.NodeCount*; once the counter equals this value, the “hint”, “submit” and “next” buttons are disabled and a brief text congratulating the user is set visible.

The “next” button is only enabled after pressing the “submit” button in each round. The “submit” button triggers the validation events; the “next” button clears the multiple instances of textbox object and sets all of them enabled again; the counter for the current expression is increased and values such as the number of used hints are reset; the “hint” and “submit” button are enabled again while the “next” is disabled.

The “hint” function checks the “letter” attribute of a random instance of the textbox object; should this attribute be different from null and the “text” attribute is empty, then it fills the text attribute, disables the respective box and increments the number of used hints. Should the “letter” attribute be null then it is assumed that instance is not being used and another random instance is picked. The number of hints per expression is limited, disabling the respective button; this happens so that the user doesn’t spam the hint button for negative scores even after filling all the boxes.

The event sheet also contains the basic events for the player object movement to function correctly and for the “Go Back” button which disables colisions, makes the player object “disappear” into the background and moves to the previous layout. The game knows which layout to move to since the structure is hierarchical and the saves the previous layout’s name to a global variable set for that effect.

### 3.2.2 “Drag-and-drop” minigame

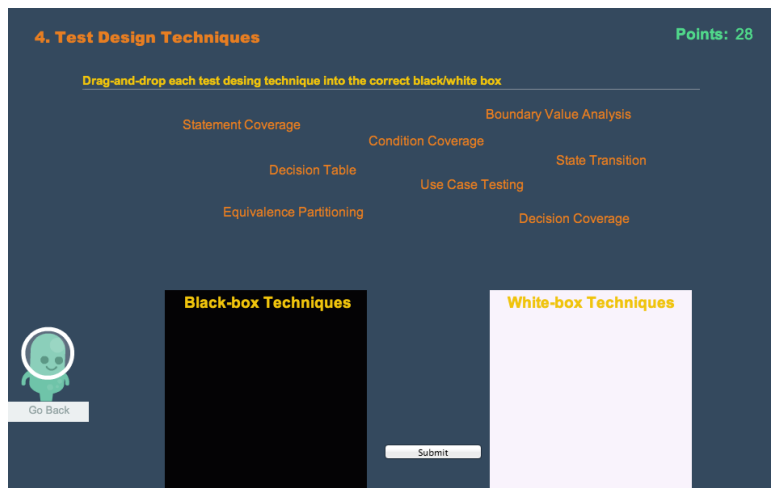


Figure 14: "Drag-and-drop to boxes" example.

Drag-and-drop’s limitations/rules: In this example, the player has eight words/expressions to classify as “black-box” or “white-box” techniques. The exercise itself can have up to ten words and can be applied to any sort of problem that requires the user to divide a set of concepts into two different categories. It is possible to set the expressions in the XML file as well as the box they belong to. This exercise corresponds to the challenge originally displayed in *iLearntest*’s chapter 4.2.

Drag-and-drop’s implementation details:

On start of layout, a system variable is set to save the scores obtained in the exercise and initialized at zero; the XML content is requested using the AJAX object. On request completed, The “AJAX.LastData” is loaded into the XML object; the title and problem statement of the exercise are loaded into existing text objects through their unique IDs, as well as the words used in the respective problem and the boxes’ names. The text objects with null text are destroyed on runtime. Since the text object has to be pre-created to be accessed and modified through its UID, it is limited to ten instances; if there is no corresponding XML node that means the textbox is unused and can be destroyed. Otherwise, the box would still exist, though empty and therefore invisible, and produce faulty scores.

The validate button will trigger the events that verify each text object position in order to find out if they are in the correct box. Should it be correct, the text will be colored green; otherwise it will be red. The verification is performed for each text box using its X and Y coordinates and comparing them to the big boxes’ positions; in the XML file it is specified the box that each word belongs to as “white” or “black”. According to the results, a text object is set visible to congratulate the user which is awarded a certain number of points for each correct answer. The validation button is destroyed to make sure the user can’t validate the answers a

## Approach

second time in a row, manipulating the result to their favor. This way they can retry the exercise until they achieve maximum score.

Once again, the event sheet also contains the basic events for the player object movement to function correctly and for the “Go Back” button which disables collisions, makes the player object “disappear” into the background and moves to the previous layout. The game knows which layout to move to since the structure is hierarchical and the saves the previous layout’s name to a global variable set for that effect.

### 3.2.3 “Equivalence class” minigame

4.3.1. Equivalence Partitioning - Exercise Points: 0

The New Bus Company as different prices according to the client age.

All babies, under 2, don't pay.  
Young people pay \$10. Adults should pay \$15 per trip.  
And Senior citizens have to pay \$5.

Young people age varies between 2 and 17 years old. A citizen is considered an adult when he

Fill the table below with the equivalence classes for each row according with the problem.

	Baby	Young	Adult	Senior
Age	<input type="text"/>	<input type="text" value="2,18"/>	<input type="text"/>	<input type="text"/>
Price	<input type="text"/>	<input type="text"/>	<input type="text" value="\$5"/>	<input type="text"/>


 Go Back Submit

Figure 15: Equivalence Class exercise.

Equivalence classes’ limitations/rules: The player has to complete the equivalence matrix according to the problem statement. The matrix itself can go up to a maximum size of 3 rows and 5 columns. The problem statement, size and attributes of the matrix are loaded from the XML. This exercise corresponds to the challenge originally displayed in *iLearntest*’s chapter 4.5.

Equivalence classes’ implementation details:

On start of layout, a system variable is set to save the scores obtained in the exercise and initialized at zero; the XML content is requested using the AJAX object. On request completed, The “AJAX.LastData” is loaded into the XML object; the exercise’s title and problem statement are promptly loaded into the respective text objects via unique ID. The number of rows and columns of the matrix are also saved to two variables. The names of each column and row are also loaded in the same manner. A couple of verifications take place: should the number of rows or columns be zero the matrix is disabled, which is possible because each square of the matrix is actually an instance of the same textbox object and therefore can be selected at the same time; if



## Approach

the exercise does not necessarily need the full size of the matrix, it can be partially culled by destroying the boxes whose UID's fall into a unused column or row which is the reason why the row/column numbers are collected early on.

For each box of the matrix, it is also verified if the box should be enabled; this information is also obtained by using the mentioned *XML.StringValue*. Should the box be disabled its color will be changed to grey with the respective value filled in; the box is literally disabled in the game so the player can't interact with it. A disabled box is not accounted for when calculating the final score nor should it. A simple trick to do this was to put the respective text into the box's placeholder rather than its text area. Although the difference in appearance is subtle it works since the validation events check each box's text areas to confirm the answers and not their placeholders. The validation event checks each and every box to compare its text to the correct answers in the XML file. Each correct box is changed to green and awards a given number of points; otherwise it will be colored red if incorrect and remain grey if disabled. When the validation is done, the validation button and the matrix are both disabled and the user gets a text with their score marking the end of the exercise. The user can always retry to try to attain the maximum score.

Once again, the event sheet also contains the basic events for the player object movement to function correctly and for the "Go Back" button which disables collisions, makes the player object "disappear" into the background and moves to the previous layout. The game knows which layout to move to since the structure is hierarchical and the saves the previous layout's name to a global variable set for that effect.

### 3.2.4 Quiz minigame

The screenshot shows a quiz interface with a dark blue background. At the top, the title '4. Test Design Techniques - Final Test' is displayed in orange, and the score 'Points: 43' is in green. There are six questions listed, each with a number and a description. To the right of each question is a text input field and a green checkmark or red X indicating the result. A 'Go Back' button with a green character icon is located at the bottom left.

Question Number	Question Text	Answer	Result
1	Equivalence partitioning, boundary value analysis, decision tables, state transition testing and use case testing are white box techniques.	False	Correct (Green Checkmark)
2	A state table shows the relationship between the states and inputs, and can highlight possible transitions that are invalid.	True	Correct (Green Checkmark)
3	Which black box technique divides inputs into groups that are expected to exhibit similar behavior?	Equivalence partitioning	Correct (Green Checkmark)
4	Which technique is often considered as an extension of other black box test design techniques?	Boundary value analysis	Correct (Green Checkmark)
5	The strength of decision table testing is that it creates combinations of conditions that otherwise might not have been exercised during testing.	True	Correct (Green Checkmark)
6	What is needed to be met for a use case to work successfully?	Review	Incorrect (Red X)

Figure 16: End-of-chapter quiz results.

Final test's limitations/rules: Each main chapter has, or rather, can have a "final test" section. The exercise takes the form of a quiz that will consist of six questions. Each one of

## Approach

these questions may have any variable number of possible answers to choose from. The player gains a certain amount of points for each correct answer. The questions and answers are loaded from the XML. Just like the text/content layout, the final tests consist of a single layout that loads the respective chapter's test. This is determined by a global variable in the game that is constantly updated accordingly to the user's actions. This exercise is accessible through the "final test" button in each of the main chapter screens. In the original *iLearntest* it is accessible in the main screen of chapter 3 and chapter 4.

Final test's implementation details:

This is actually the simplest minigame as it takes the form of a regular quiz. On start of layout, the XML content is requested using the AJAX object and several counters are set at zero. Basically for each question, while the respective counter is below the number of possible answers for said question then keep adding them to that question's dropdown list of answers. The number of possible answers is easily obtained using *XML.NodeCount*. However, the dropdown list has to be filled one answer at the time or the possible answers would be treated as a single one. The reason why multiple counters are used instead of using just one that is reset to zero is because of the order on which the events run. This way it is guaranteed that the counters don't get mixed up with each other. As for the validation each answer is checked by comparing them with the "answer" value in the XML using *XML.StringValue* once again. Depending on answer the corresponding sprite is set visible in front of it, as shown in figure 16. Once the validation is done, the validation button is destroyed. The user can revisit the quiz minigame and retry from the start.

Once again, the event sheet also contains the basic events for the player object movement to function correctly and for the "Go Back" button which disables collisions, makes the player object "disappear" into the background and moves to the previous layout. The game knows which layout to move to since the structure is hierarchical and the saves the previous layout's name to a global variable set for that effect.

### 3.2.5 The solution's particularities

On a final note, none of these exercises may seem overly complicated to implement at first. However, considering some of the framework's limitations and learning curve, the result is overall positive (see chapter 4) since the research work's aim was to deliver a solid proof-of-concept of the desired platform. Bearing in mind the number of abstractions that were included since the implementation of the original game, it is fairly feasible to extend it to other areas at this point even if there are still some obvious restrictions such as how much content or how many options it is possible to fit into a single minigame.

The reasoning through the development of the tool was to have a layout per chapter where a certain order of actions that lead to the next chapter; this order of actions had to be somehow limited to finite number of events in order to support a completely different order dynamically.

### 3.3 XML Structure

This section briefly presents the current structure of the XML file used as backend / database for the game.

```

<?xml version="1.0"?>
<ilearntest>
  <chapter title points>
    <subchapter title points>
      <learningobjectives>
        <learningobjective number> </learningobjective >
      </learningobjectives>
      <texts>
        <paragraph></paragraph>
      </texts>
      <games>
        <dragdropboxes white black >
          <word box > </word>
        </dragdropboxes>
      </games>
      <ssubchapter title points>
        <texts>
          <paragraph> </paragraph>
        </texts>
        <games>
          <equivalenceclass rows columns pointsperanswer>
            <problem> </problem>
            <column> </column>
            <row> </row>
            <expression enabled row column> </expression>
          </equivalenceclass>
        </games>
      </ssubchapter>
    </subchapter>
  </chapter>
</ilearntest>

```

Figure 17: XML Structure.

The main tag is “ilearntest” with the chapter’s title and maximum possible score. The number of “chapter” and “subchapter” ranges from zero to six.

## Approach

“learningobjectives” contains the learning objectives for said chapter and “texts” comprises the theoretical contents; the tag “games” can lead to the previously mentioned mini-games and lastly the “finaltest” which comprises the chapter’s quiz.

“ssubchapter” does not contain learning objectives as we assume they are pretty much the same as the subchapter above although it still retains all the other tags from “chapter” and “subchapter”.

The tag “games” contains the respective tags for the currently implemented mini-games, for instance: “equivalenceclass”, “hangman” and “dragdropboxes”.

The tag “dragdropboxes” can have up to ten tags “word”; this tag has a class “box” which can be “black” or “white”, for the sake of verifying which box is the correct one, and the node’s text is the word or expression to load into the game. It is possible to name the black and white boxes through the class “black” and “white”, respectively. The game is displayed to the user in a similar fashion to the example presented in figure 14. The following is a XML example of the drag-and-drop minigame:

```
<games>
  <dragdropboxes black="Black-box Techniques" white="White-box Techniques">
    <word box="white" >Statement Coverage</word>
    <word box="white" >Condition Coverage</word>
    <word box="black" >Boundary Value Analysis</word>
    <word box="black" >Decision Table</word>
    <word box="black" >Use Case Testing</word>
    <word box="black" >State Transition</word>
    <word box="black" >Equivalence Partitioning</word>
    <word box="white" >Decision Coverage</word>
  </dragdropboxes>
</games>
```

“Hangman” has a tag “problem”, which node’s text is the problem statement for that game, and a tag “word” that can be used any number of times; the latter has several classes: “answer” which is the correct word/expression and “size”, “size1” and “size2” which is respectively the total size of the expression without spaces, the size of the first and second words of the expression should that be the case. The tag “word” can have up to four sub tags “tip”, which contain the tips for the player, and a tag “l” as in “letter” that is to be used for each character of the expression; it’s not an elegant implementation but Construct lacks the function to target a character of a certain text variable when performing these verifications. The game is displayed to the user in a similar manner to the example presented in figure 13. The following is a XML example of the hangman minigame:

## Approach

```

<games>
  <hangman>
    <problem>Find the activities of a formal review that are characterized by the
points presented. If you need you can ask for a new letter (hint button), but remind that you will
gain less points for that word.</problem>
    <word answer="rework" size="6" size1="6" size2="0">
      <tip>Recording updated status of defects</tip>
      <tip>Fixing defects found</tip>
      <l>r</l><l>e</l><l>w</l><l>o</l><l>r</l><l>k</l>
    </word>
    <word answer="review meeting" size="13" size1="6" size2="7">
      <tip>Discussing or logging, with documented results</tip>
      <tip>Noting defects, and making decision about them</tip>
      <tip>Evaluating and recording issues</tip>
      <l>r</l><l>e</l><l>v</l><l>i</l><l>e</l><l>w</l>
      <l>m</l><l>e</l><l>e</l><l>t</l><l>i</l><l>n</l><l>g</l>
    </word>
  </hangman>
</games>

```

The tag “equivalenceclass” has number of rows and columns to build the matrix as well as the points per correct answer. Similar to “hangman”, it has a tag with the problem statement. It also can have up to five “column” tags and three “row” tags with the names of the respective classes. There is also an “expression” which contains the correct value for a certain “row” and “column” which on their turn are classes for this tag. The class “enabled” determines if that position of the matrix awards points or is already filled. The game is displayed to the user in a similar manner to the example presented in figure 15. The following is a XML example of the “equivalence class” minigame:

```

<games>
  <equivalenceclass rows="2" columns="4" pointsperanswer="3">
    <problem>The New Bus Company as diferent prices according to the client
age. All babies, under 2, don't pay. Young people pay $10. Adults should pay $15 per trip. And
Senior citizens have to pay $5. Young people age varies between 2 and 17 years old. A citizen is
considered an adult when he makes 18 years old. After completing 65 years old, is considered a
senior. Consider that the maximum age a person can have is 120 years old. </problem>
    <column>Baby</column>
    <column>Young</column>
    <column>Adult</column>

```

## Approach

```
<column>Senior</column>
<row>Age</row>
<row>Price</row>
<expression enabled="Yes" row="1" column="1">[0,2[</expression>
<expression enabled="No" row="1" column="2">[2,18[</expression>
<expression enabled="Yes" row="1" column="3">[18,65[</expression>
<expression enabled="Yes" row="1" column="4">[65,120]</expression>
<expression enabled="Yes" row="2" column="1">0</expression>
<expression enabled="Yes" row="2" column="2">10</expression>
<expression enabled="No" row="2" column="3">15</expression>
<expression enabled="Yes" row="2" column="4">5</expression>
</equivalenceclass>
</games>
```

As mentioned above, the tag “chapter” has a tag “finaltest” which can also be considered a mini-game in the form of quiz. The game is displayed to the user in a similar manner to the example presented in figure 16. The following is an example of the XML used for a quiz minigame with a couple of questions:

```
<finaltest>
  <question answer="False">
    <paragraph>Equivalence partitioning, boundary value analysis, decision
    tables, state transition testing and use case testing are white box techniques.</paragraph>
    <option>True</option>
    <option>False</option>
  </question>
  <question answer="Equivalence partitioning">
    <paragraph>Which black box technique divides inputs into groups that are
    expected to exhibit similar behavior?</paragraph>
    <option>Decision Table</option>
    <option>Equivalence partitioning</option>
    <option>Use case testing</option>
    <option>State transition testing</option>
  </question>
</finaltest>
```

This tag contains up to six tags “question” each one with a class “answer” which is the correct answer for the respective question. It contains a sub tag “paragraph” which is the problem statement as well as a tag “option” which can be used any number of times per question and each contains an option for the list of possible answers.

### 3.4 Steps to generate a new XML-based game

The common way for the final user to generate a new game structure based on a XML file is through the iLearnTest website. This site, as mentioned before, is based on an authentication method in order to keep track of each user's progression. It contains a page with the embedded game, exported from the Construct 2 framework, a scoreboard, a contact and an information page and the recently added “game edition” page.

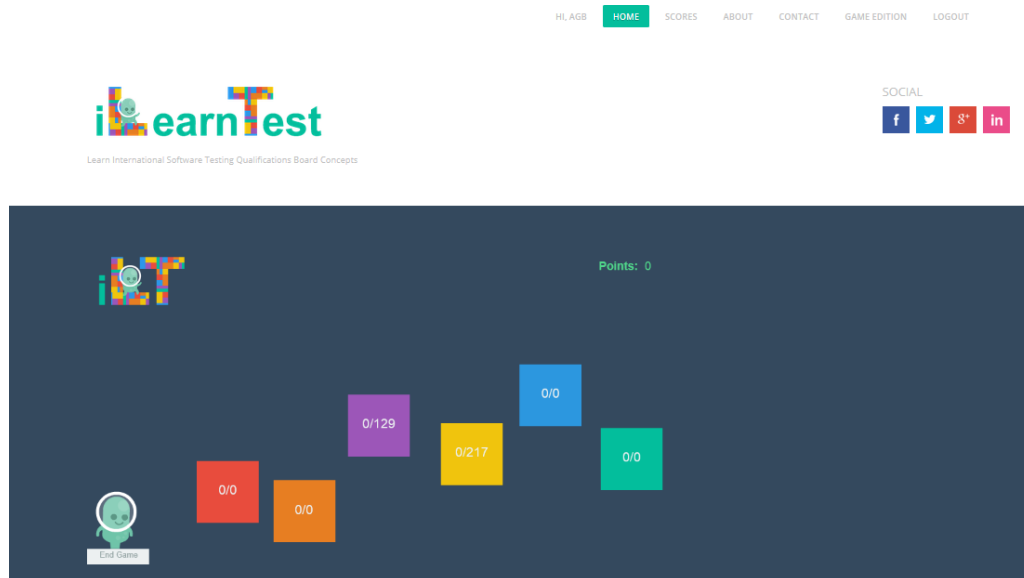


Figure 18: iLearnTest website.

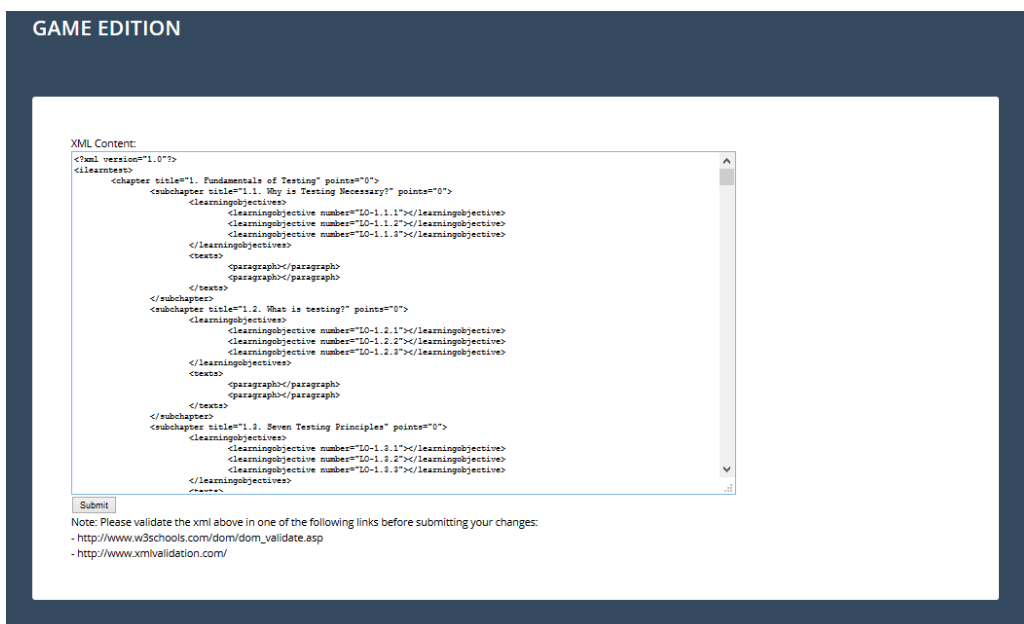


Figure 19: “Game edition” page.

In a normal scenario, this last page is only accessible by the page's administrator(s) in order to prevent poorly made modifications by users with little XML knowledge. After logging

## Approach

into the website, the administrator can access the “game edition” page which presents a text editor with the contents of the current XML file. In this page it is possible to modify this same file which will lead to an automatic and instantaneous update of the embedded game’s structure. Using the structure provided in the section above, the user can even create a completely new file with its own contents and using the available minigames. The edition of the XML is performed using rather simple PHP methods.

On a side note, the edition page consists of a HTML text area that opens with the current XML using the method `file_get_contents($file_path)`. When the changes are submitted the file is modified through a file handle such as:

```
$file_handle = fopen($file_path, 'w');  
fwrite($file_handle, $changes);  
fclose($file_handle);
```

As the user saves the changes to the content file, the game will automatically update. However, the page does not contain an inbuilt XML validation tool. It is strongly advised, as pointed out in the webpage, to apply any commonly used validation tool or site before saving the modifications. Thankfully, an invalid XML file does not cause the game to break due to, for instance, a possible bug. However, the game would load devoid of content, as it can’t extract correct information from the XML. Should the XML be valid, then the modification is complete and successful.



## Chapter 4

# Validation

When addressing the validation of results, one must consider the validation of the tool as a dynamic framework for creating games of the sort. The idea is to take each one of the implemented games and apply them to other possible contents whether it is software testing or other specific areas. For instance:

The “hangman” can be used in any context in which the player has to guess a key concept with a few tips; for example: “Guess the programming language that has the following syntax.”

The “drag-and-drop” mini-game can be used in a context where the user has to separate a number of concepts, objects or others into two major categories as if to classify them, for instance, classify the following equations as polynomial or linear equations.

“Equivalence class” is a logic exercise that requires the user to interpret the problem statement by parts in order to correctly fill the matrix. Its difficulty fluctuates accordingly with the complexity of the problem statement.

“Final test” is a simple quiz mini-game and can be applied to almost any content in which there’s a need to cover a considerable amount of contents with only a few questions.

### 4.1 Validation of the tool as framework

In order to validate the most recent implementation of iLearnTest as a framework a small experience was performed using both: only Construct2 and the XML framework developed. This experience consists on a small exercise where the user has to create a game with three layouts for learning objectives, theoretical contents and a small drag-and-drop game (similar to Figure 14). This exercise does not need to have any extravagant elements such as background,

## Validation

player sprites or others. The following is an example of the expected results in which the drag-and-drop minigame is used to teach elementary students how to distinguish uppercase from lowercase letters and the respective XML file.

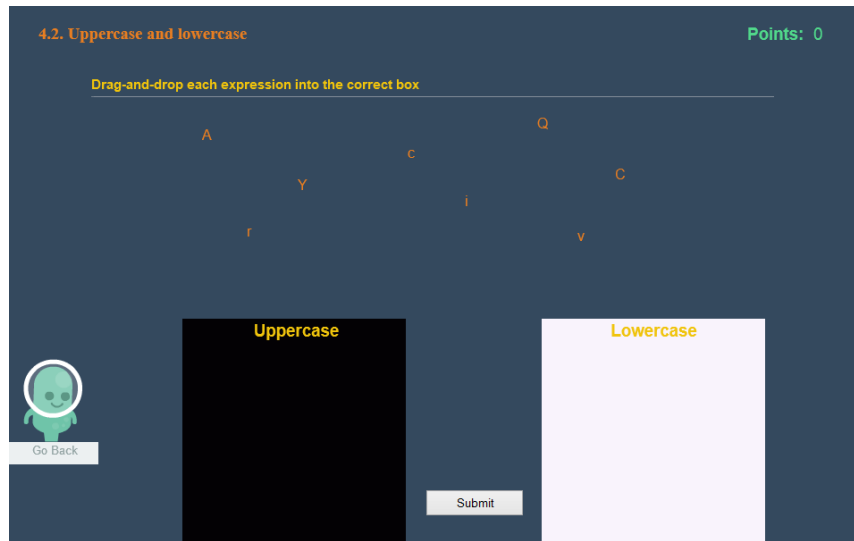


Figure 20: Uppercase-Lowercase example.

```
<subchapter title="4.2. Uppercase and lowercase" points="16">
  <learningobjectives>
    <learningobjective number="LO-4.2.1">Objective1</learningobjective>
    <learningobjective number="LO-4.2.2">Objective2</learningobjective>
    <learningobjective number="LO-4.2.3">Objective3</learningobjective>
  </learningobjectives>
  <texts>
    <paragraph>paragraph1</paragraph>
    <paragraph>paragraph2</paragraph>
  </texts>
  <games>
    <dragdropboxes black="Uppercase" white="Lowercase">
      <word box="white" >r</word>
      <word box="white" >c</word>
      <word box="black" >A</word>
      <word box="black" >Y</word>
      <word box="black" >Q</word>
      <word box="black" >C</word>
      <word box="white" >v</word>
      <word box="white" >i</word>
    </dragdropboxes>
  </games>
</subchapter>
```

Figure 21: Uppercase-Lowercase XML structure.

We had six students participating in the experience. They were about twenty-two to twenty-three years old and finalists in Informatics and Computing Engineering at Faculty of Engineering of the University of Porto. The experience considers the time need to install the project, which is almost negligible in both cases, the time needed to learn to use the tool and implementation of the drag-and-drop exercise. The average results are as follows:

## Validation

Time	XML Framework	Construct2
Installation	1 min	2 min
Learning	4 min	20 min
Implementation	4 min	22 min
Total	9 min	44 min

Table 1: Experience results of the tool as framework.

There were a few facts that were pointed out during this exercise. Firstly, it was noted that Construct2 has a steep learning curve. Since the time to perform this experience was limited, the necessary functionalities were exposed to the students during a 20-minutes session. In this case, the user only had to perform basic tasks such as creating a layout with textboxes and small sprites which took about twenty minutes to explain; if it was needed to use more specific functionalities, these twenty minutes would not suffice, most likely. On the other hand, anyone with basic knowledge on annotation languages can quickly understand how to use the developed framework with just a brief explanation of the file's structure and create their own game with little help. It is also easier to modify and update the contents through the xml file because the game as suitable templates for its contents; the downside to the Construct2 implementation is that it is hard coded and can be only modified through the Construct before exporting/releasing the game to HTML5. As a side note, it is important to note that the number of participants is, once again, limited. Therefore the time ratios between both parts of the experience might suffer slight changes with a larger population.

## 4.2 Conclusion

- Despite the small amount of students gathered, it is demonstrated through the experiment that the developed framework can make the construction and generation of games more efficient, both time and effort wise;
- The framework has a rather friendly learning curve even when compared with a game development framework such as Construct 2. Looking at the efficiency ratios obtained, we can conclude that learning to perform a task in the Construct 2 can take up to five times the time need to perform the same task in the platform. The implementation time ratio is about 5.5 times for a small task such as this one and is expected to scale with the task's difficulty. In other words, the more complex the exercise, the more difficult it should be to implement in Construct 2 than using the platform;
- Even though a relatively accessible exercise was selected, most students struggled to complete the exercise using the Construct 2 as it was their first contact with the tool;

## Validation

- Ultimately, this experiment puts emphasis in the contrast between an implementation which easily allows for modifications, updates or expansions to a tool's content by separating it to the game engine, and the previous implementation where the content was hard coded into the game. The experiment shows to a certain degree how important it is to code an application to be able to adapt, rather than code it to last;
- As a proof-of-concept, the developed platform shows promise.

## Chapter 5

# Conclusions

This document presents an extension and improvement of a previous game which was originally hard coded for the most part, into a more dynamic tool; for instance, reading the set of problems from a XML file which would be separated from the game implementation. In order to do this there is a set of game templates that can read the content of the exercises from the file, so that we can use the same game(s) for the learning process of a variety of themes. This solution consists in a platform capable of loading content into *iLearnTest* from XML via AJAX request, in order to ease future updates and development of the tool. The structure of this XML file was presented and thoroughly explained in section 3.3; using this structure as a guide, it is possible to edit or create educational games by editing the XML in the *iLearnTest* website.

The mini-games were implemented using the Construct 2 framework using event-based programming and can be adapted to educational contents other than just software testing. These mini-games include: the “hangman” game where the player has to guess the correct word(s) through given tips; the “drag-and-drop” game where the player places a set of expressions into their respective boxes, according to context; the “equivalence class”, or rather equivalence partitioning exercise, where the player fills a matrix with values and information from the problem statement; and the quiz mini-game which is used as a sort of final test for each chapter of the game to determine if the player assimilated the theoretical contents of said chapter. The implementation of these mini-games is thoroughly described in section 3.2.

The state-of-art provided a clearer insight of the most used frameworks on the development of serious games for education as well as the most commonly utilized software architectural patterns. It was reasonable to conclude that, from the list of frameworks for game development discussed, Construct 2 offered the advantage of being able to inject content into the game from XML files through AJAX requests rather easily and to export the project to

## Conclusions

HTML5, which can be integrated in the current website. The use of this framework also allowed reutilizing some of the previous layouts/interfaces of the game. The solution naturally took the form of a n-tier/three-tier based architecture which consists of data, logic and presentation tiers; for instance, the data tier consists on the XML content file while the logic behind the game is handled by the Construct 2's engine; the game's layouts presented through the browser serve as presentation tier. This architecture is typically characterized by the functional decomposition of applications and their distributed deployment, providing improved scalability, availability, manageability, and resource utilization.

It is worth to mention that the tool had already been validated once before as software testing learning game [32] and, despite the limited number of participants, the results were quite satisfactory. This time around, trying to validate the game construction through content available in a XML file we concluded that the division of the content from the game engine is beneficial considering the time efficiency of the new approach.

The experiment conducted as validation of the tool consisted in a small exercise where the user had to create a mini-game using both the developed platform and the Construct 2 framework. The exercise included the creation of three simple layouts for learning objectives, theoretical contents and a small drag-and-drop game, similar to Figure 14. This exercise does not need to have any extravagant elements such as background, player sprites or others. The time taken to learn the tool and to implement the mini-game would then be compared in order to obtain an effectiveness ratio of the new platform. While conscious that this experiment was performed at a much reduced scale, more exhaustive experiments are intended to be performed in the future for further validation of the tool.

On a final note, it is important to take into account that the current solution is actually a proof-of-concept that it is fairly possible and feasible to implement a platform for the creation of educational computer games. The solution that resulted from this research work can be further enhanced in the future by implementing more and new mini-games or challenges with the capability of adapting to multiple contexts.

# References

1. Malamed, C., The Gamification of Learning and Instruction: Game-Based Methods and Strategies For Training And Education. eLearn Magazine, 2012. 2012: p. 3.
2. Ledda, R. "Benefits of using Game-Based Learning in Education." from <http://www.educatornetwork.com/HotTopics/gamesbasedlearning/benefits>.
3. Fernandes, J.M. and Sousa, S.M., PlayScrum - A card game to learn the scrum agile method, in 2nd International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2010. 2010. p. 52-59.
4. Shneiderman, B., Designing for Fun: How Can We Design User Interfaces to Be More Fun? Interactions, 2004. 11: p. 48-50.
5. ISTQB. Certified Tester, Foundation Level Syllabus. 2011; Available from: <http://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html>.
6. Thiry, M., Zoucas, A., and Silva, A.C., Empirical study upon software testing learning with support from educational game, in SEKE 2011 - Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering. 2011. p. 481-484.
7. Microsoft (2015). "What is Software Architecture?" Microsoft Enterprise Architecture, Patterns, and Practices. from <https://msdn.microsoft.com/en-us/library/ee658098.aspx>.
8. Microsoft (2015). "Deployment Patterns." Microsoft Enterprise Architecture, Patterns, and Practices. from <https://msdn.microsoft.com/en-us/library/ms998478.aspx>.
9. Eckerson, W.W. (1995). "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." Open Information Systems.

## References

10. Semančík, R. (2002). Internet applications security. Computer Systems, Slovak University of Technology.
11. Domain-Driven Design, the Book pp. 68-74. Retrieved from <http://www.domaindrivendesign.org/books#DDD>
12. Novikov, D. (2010). "Open Source Extensible Enterprise n-tier Application Framework with Multilayered Architecture." from <http://www.codeproject.com/Articles/132838/Open-Source-Extensible-Enterprise-n-tier-Applicati>.
13. Microsoft (2015). "Architectural Patterns and Styles." Microsoft Enterprise Architecture, Patterns, and Practices. from <https://msdn.microsoft.com/en-us/library/ee658117.aspx>.
14. Ramirez, A.O. (2000). "Three-Tier Architecture." from <http://www.linuxjournal.com/article/3508>.
15. Unity Technologies, Unity 3D. 2014; Available from: [www.unity3d.com](http://www.unity3d.com).
16. Scirra. Construct 2. 2007; Available from: [www.scirra.com](http://www.scirra.com).
17. YoYo Games Ltd., GameMaker: Studio. 2013-2015; Available from: <http://www.yoyogames.com/studio>.
18. LLC. Stencyl. 2014; Available from: [www.stencyl.com](http://www.stencyl.com).
19. GameSalad. GameSalad. 2011; Available from: [www.gamesalad.com](http://www.gamesalad.com).
20. Scirra (2011). "Construct 2 vs GameMaker vs GameSalad vs Stencyl." from <https://www.scirra.com/blog/59/construct-2-vs-gamemaker-vs-gamesalad-vs-stencyl>.
21. Scirra (2011). "Construct2 vs Javascript." from <https://www.scirra.com/blog/52/construct-2-vs-javascript>.
22. Navarro, E., SimSE: a software engineering simulation environment for software process education, in Vasa. 2006. p. 321.
23. Dantas, A., Barros, M., Werner, C., Treinamento experimental com jogos de simulação para gerentes de projecto de software. Simpósio Brasileiro de Engenharia de Software - SBES, 2004: p. 23-38.
24. Rosa, R., Kieling, E., Planager - Um Jogo para Apoio ao Ensino de Gerencia de Projetos de Software. TCC Bacharelado em Sistemas de Informação PUC RS, 2006.
25. Ludewig, J., Models in software engineering - an introduction. Software and Systems Modeling, 2003. 2: p. 5-14.



## References

26. Gresse von Wangenheim, C., Thiry, M., Kochanski, D., Empirical evaluation of an educational game on software measurement. *Empirical Software Engineering*, 2008. 14: p. 418-452.
27. Figueiredo, E., Lobato, C., Dias, K., Leite, J., Lucena, C., Um jogo para o ensino de engenharia de software centrado na perspectiva de evolução. *XV Workshop sobre Educação em Computação (WEI)*, 2007: p. 37-46.
28. Elbaum, S., et al., Bug hunt: Making early software testing lessons engaging and affordable, in *Proceedings - International Conference on Software Engineering*. 2007. p. 687-697.
29. Oliveira, B., *TestEG - Um Software Educational para o Ensino de Teste de Software*. 2013.
30. Ribeiro, T.P.B., *iLearnTest: Jogo Educativo para Aprendizagem de Testes de Software*. 2014, University of Porto: FEUP.
31. Scirra (2015). "Official Construct 2 Manual." from <https://www.scirra.com/manual/1/construct-2>.
32. Ribeiro, T.P.B., Paiva, A.C.R. (2015). *iLearnTest - Jogo Educativo para Aprendizagem de Teste de Software*. 10ª Conferencia Ibérica de Sistemas y Tecnologías de la Información (CISTI'2015).
33. Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.
34. Esquivel, J. (2013). "SAP 3 Architecture." from <http://pt.scribd.com/doc/126195474/SAP-3-ARCHITECTURE#scribd>.

## Appendix A

# MIME types for HTML5 games

### Fundamental MIME types

Basic types:

.js files: application/javascript

.png files: image/png

.jpg files: image/jpeg

.jpeg files: image/jpeg

.css files: text/css

.html files: text/html

For offline support to work correctly:

.appcache files: text/cache-manifest

For web app manifests to load correctly:

.webapp files: application/x-web-app-manifest+json

.manifest files: application/manifest+json

Ensure AJAX requests or other external resources load correctly:

.txt files: text/plain

.xml files: text/xml

## MIME types for HTML5 games

.csv files: text/csv

.json files: application/json