Adapting Educational Content to Maximise Reuse Across Knowledge Groups via Interactive Experiences

Teodora Militaru
2055695

Supervisor: Alexander Dixon

Year of Study: Third Year, 2022 - 2023

Abstract

Computer games have become an increasingly popular means of delivering educational content usually targeting beginner knowledge groups. Since this type of interactive experience can be valuable for experienced users as well, mainly throughout their revision process, it is important that it caters to such groups as well. However, augmenting content such that it is suitable for these categories is an expensive, inefficient and time consuming process.

This project investigates different methods of adapting educational content such that it is suitable for all knowledge groups, in the context of a proof of concept escape room-style game centred around four topics in Computer Science: Object Oriented Programming, Logic and Verification, Database Systems and Web Development.

The proposed approaches to content adaptation are: modifying the probability of appearance of different questions, providing hints for beginners, using a time constraint for the experienced players and generating puzzle content that is affected by the chosen difficulty.

Keywords: educational games, game-based learning, randomisation algorithms, Unity, C#

Contents

1 Introduction **6**

2 Background **8**

2.1 The Escape Room Format 8

2.2 Educational Escape Room Experiences 11

2.3 Methods of Maximising Content Reusability 14

2.3.1 Learning Objects 14

2.3.2 Difficulty Adaptation 16

2.3.3 Randomisation Algorithms 21

3 Project Management **22**

3.1 Methodology 22

3.2 Project Planning 22

3.3 Project Objectives 23

3.4 Risk Management 25

3.5 Use of Tools 25

3.5.1 Unity Asset Store 26

3.5.2 Blender 26

3.5.3 Unity 27

3.5.4 ProBuilder 27

4 Project Design **27**

4.1 Structure of the Game 27

4.2 Storyline 28

4.3 Puzzle Design 29

5 Project Implementation **31**

5.1 3D Assets Creation 31

5.2 Environment Creation 32

5.3 Scene Management 33

5.4 Room Randomisation 34

5.5 Puzzle Randomisation 36

5.6 Room Population 40

5.7 Character Controls 41
5.8 Timer Implementation 44
5.9 Title Screen and Options Menu 46
5.10 Drag and Drop Puzzle Mechanics 46
5.11 Random Logic Expression Algorithm 48
5.11.1 Generation 49
5.11.2 Validation 50
5.12 Candidates for Logical Equivalence Generation 52

6 Testing 53

7 Evaluation 54
7.1 Project Execution 54
7.2 Author's Assessment of the Project 55
7.3 Areas for Future Work 56

8 Conclusions 57

9 Legal, Social, Ethical and Professional Issues 58

Appendices 66

A Functional Requirements 66

B Non-functional Requirements 67

C Extensions 67

Acknowledgements

First and foremost, I would like to thank my project supervisor, Alexander Dixon, for inspiring me to pursue this particular project and for his continuous support and guidance throughout development. I would also like to thank my second assessor, Jane Sinclair for the valuable presentation feedback, which helped me improve the project and prioritize different aspects when writing this report.

In addition, I would like to thank the members of the Warwick Game Design Society their help throughout the development and testing phases of the project. Last but not least, I am grateful to my friends for their continuous support during this challenging year.

1 Introduction

Throughout the years, with the rise in popularity of computer games, their impact has generated heated discussion among behavioural experts, with both positive and negative arguments being made for the use of such interactive experiences in the educational field [1]. Some of the benefits of game-based learning that were identified during research include the lessons being more interactive, students being able to practically apply different concepts and the learning process being more enjoyable [2]. Moreover, such interactive experiences have been proven to have positive effects on cognition skills as they demonstrate good use of learning principles. In his article, James Paul Glee [3] explores a few of the most important such principles and the way they are applied in the context of computer games. Most notably, educational games provide context along with information which allows the student to apply the knowledge in a practical way, improving the memorisation process [4].

As a result of the positive image that educational games have garnered, many institutions decided to invest into developing their own interactive experiences to act as supportive material in their lessons. When designing any form of media to support the learning process, it is important that the learning objectives are clearly stated, ensuring the final product achieves all the initially set goals. In addition, because these types of learning tools are tailored to fit each use case, there are three important considerations before the development process can begin: **prior knowledge**, **learning and retention** and **potential transfer** [5].

The question of whether the players are required to have any prior knowledge before engaging in the interactive experience is essential in determining the trajectory of the educational game. In most cases, the target audience for educational games are inexperienced students, making these materials unsuitable for people who are already knowledgeable in the subject areas present in the game. Because computer games can be a space to not only learn, but practice certain concepts, it is important that such resources account for other knowledge groups as well. For example, experienced students might benefit from an educational game tailored to their skills during their preparation for exams.

Another essential question in designing a game-based learning experience is what are the skills that can be expected to be learnt by the end of the educational game (learning and retention). Since the expectations are different for each learning group, content would have to be augmented to fit all their diverse requirements, an expensive, inefficient and time consuming process.

A solution to this issue could be finding different methods of adapting the same content such that it is suitable for all knowledge groups. Currently there have not been many attempts at this task, which is why this project is proposing and aiming to investigate four techniques to maximise the reuse of educational content across all knowledge groups:

1. modifying the probability of appearance of different questions depending on the chosen difficulty,
2. providing hints for inexperienced players,
3. using a time constraint for the experienced players,
4. generating puzzle content that is affected by the chosen difficulty.

In order to test the effectiveness of the chosen methods, the project also involves producing a proof of concept escape room-style game centred around four topics in Computer Science: Object Oriented Programming, Logic and Verification, Database Systems and Web Development. These particular topics were chosen as they represent the foundation of Computer Science, having many practical applications that can easily be translated into an interactive experience. Moreover, these topics constitute an area of interest for a diverse group of users, with varying prior knowledge, goals and expectations, motivating the requirement for content adaptation to fit these various knowledge groups.

2 Background

2.1 The Escape Room Format

Escape Room experiences have been growing in popularity over the years, especially between 2015 and 2018, with both physical and digital variations being sought after by enthusiasts [6]. These types of games usually consist of a couple of puzzles which the players need to solve in order to "escape" a number of rooms, most often in a limited time. In an attempt to define the structure of the escape room format, Markus Wiemker, Errol Elumir, and Adam Clare [7] identified the following three core elements of such an experience: **a challenge**, **a solution** to overcome the challenge and **a reward** for completion (figure 1). Even though this structure may seem rigid at first, the challenges (or puzzles) are highly customizable making every game unique and this feature is the primary reason why the escape room format is so widely favoured in game-based learning.

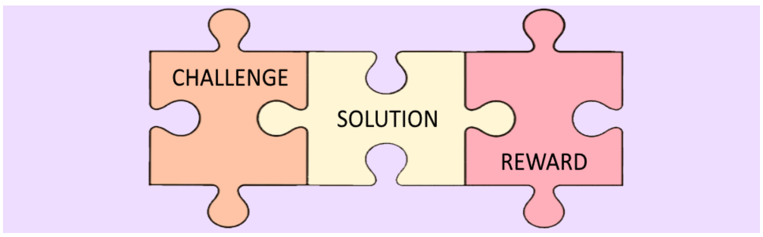


Figure 1: Structural components of an escape room experience

In addition, the article written by Markus Wiemker, Errol Elumir, and Adam Clare [7] states that escape room puzzles can be classified in two groups: **mental** and **physical**. Mental puzzles refer to those that require critical thinking and knowledge in order to solve the task, while physical puzzles consist of combining and using objects to overcome the challenge. An example of the latter could be finding a key and using it to unlock a door, an exercise that requires minimal thinking and is usually used to make the game more challenging when a time restriction is also applied. Mental puzzles are cognitive puzzles, which require the player to investigate clues and use previous knowledge to overcome the challenge. These puzzles are the most commonly used type for educational games since they engage the player's thinking skills and require knowledge of the educational content involved, thus, in this project, the focus will be on content adaptation in the context of mental puzzles only.

Another key aspect of designing an escape room experience is choosing the suitable approach for definining the order in which puzzles must be solved in order to successfully complete the room. This order is defined by Markus Wiemker, Errol Elumir, and Adam Clare [7] as a **"puzzle path"**, with three common implementation approaches being identified: **linear path**, **open path** and **multi linear path**.

The most popular method, especially in physical escape room experiences, is the **linear puzzle path**, which requires the puzzles to be solved in a predefined order. As it can be seen in figure 2, solving the first puzzle on the path provides the player with an object or hint (the reward) needed to solve the next puzzle.

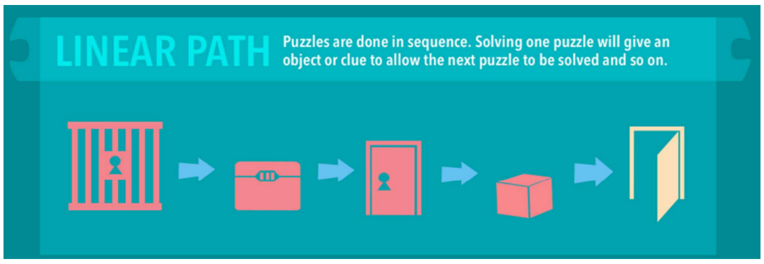

LINEAR PATH
Puzzles are done in sequence. Solving one puzzle will give an object or clue to allow the next puzzle to be solved and so on.

Figure 2: Example of a linear puzzle path [7]

Even though this approach has the advantage of a simple design that is easy to implement, it generates a few issues in the context of educational games. Most notably, learning experiences are usually designed for larger groups of people in order to allow for collaboration. With linear puzzle paths, all the players are forced to work on the same puzzle until they can reach the next one. This becomes an issue especially when the questions are meant for only one person to answer, case in which the rest of the team has to wait until they can move on to the next challenge, process that is not time efficient and can lead to boredom and disengagement with the experience.

A solution to this issue is the **multi linear puzzle path**, which allows for more flexibility in the ordering of the puzzles to be solved. This approach consists of a set of linear puzzle paths which can be solved in parallel or can intersect with each other (figure 3), depending on the desired structure of the interactive experience.

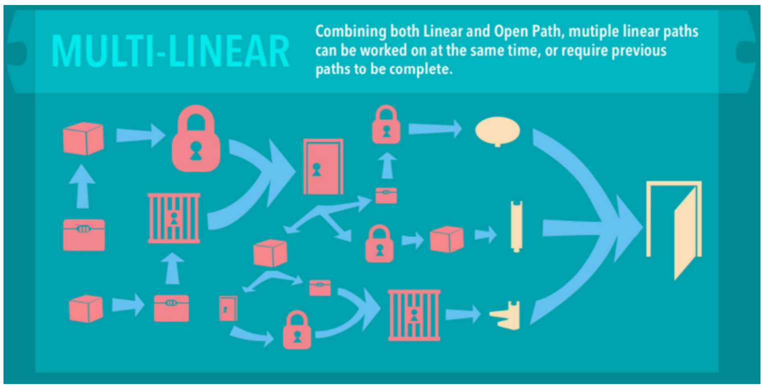

MULTI-LINEAR
Combining both Linear and Open Path, mutiple linear paths can be worked on at the same time, or require previous paths to be complete.

Figure 3: Example of a multi linear puzzle path [7]

While this method allows for members of the team to split up and work separately, its implementation is considerably more complex, especially in the context of this project, which targets maximising reuse of content. Because the room and puzzle designs are highly customizable, many components are randomised at each encounter, which makes creating multi linear random paths extremely elaborate and not worth the benefits.

A way to allow the players to have freedom in choosing the order in which to attempt the puzzles, while also accounting for the randomisation of content is to use the **open puzzle path** design pattern (figure 4).

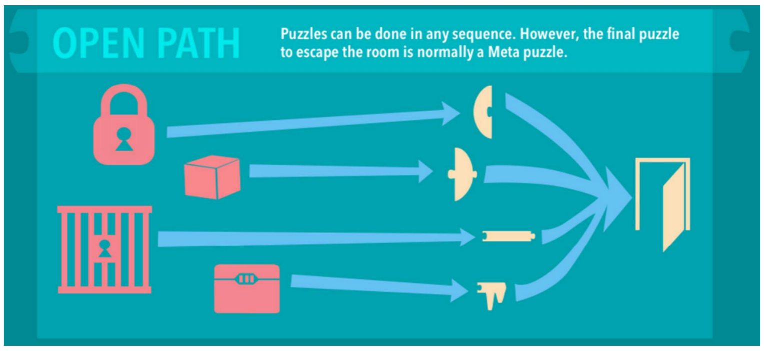

OPEN PATH
Puzzles can be done in any sequence. However, the final puzzle to escape the room is normally a Meta puzzle.

Figure 4: Example of an open puzzle path [7]

This approach eliminates the dependency between puzzles and removes the risk of bottleneck that can occur with the other two methods. Usually, in order to make the game more cohesive, solving the final puzzle in the room requires objects and hints acquired from the other challenges in the room. This type of puzzle is called a **"meta puzzle"** and it combines elements from both the **mental** and **physical** puzzle variants.

Because of the flexible structure that maximises the independence of the game objects, this approach is the most suitable for this project. Since the proof of concept game has educational content at its core and only uses mental puzzles, the meta puzzle at the end can be replaced by a simple space constriction, which prevents the player from exiting a room without having solved all of the available puzzles.

2.2 Educational Escape Room Experiences

With the growing demand for educational games, especially in the STEM field, many have adopted the escape room format with slight modifications to fit the resources available and their target groups. While most of the early escape room experiences used multiple rooms that the players were locked in, this would not be feasible in a school setting due to the limited space available and the large number of students [8]. As a result, slight modifications had to be made, such as constraining the interactive experience to one room and highlighting the objects involved in the game.

To prove how efficient escape room-style games are in the educational context, a couple of case studies were conducted, targeting different subject areas, age and knowledge groups. One of the most relevant such reports in the context of this project is **"Escape Classroom - The Leblanc Process"** [9]. This was an experimental physical escape room experience focused on teaching chemistry. As it can be observed in figure 5, the game used the answers to the questions to generate codes that opened the locks to the room. Since it was a physical interactive experience, a teacher was present to relay information and guide the students. At the end of the experiment, the participants were surveyed and 60% reported that the game increased student participation in the class and 93% recognised that they felt more motivated to learn by the experience.

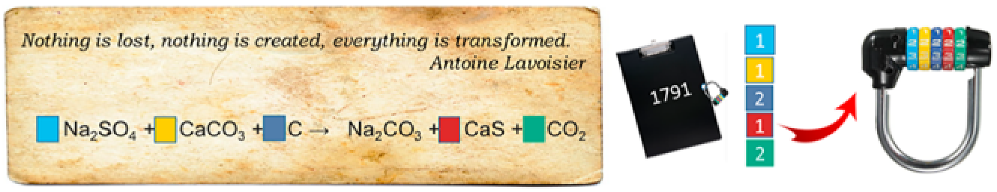
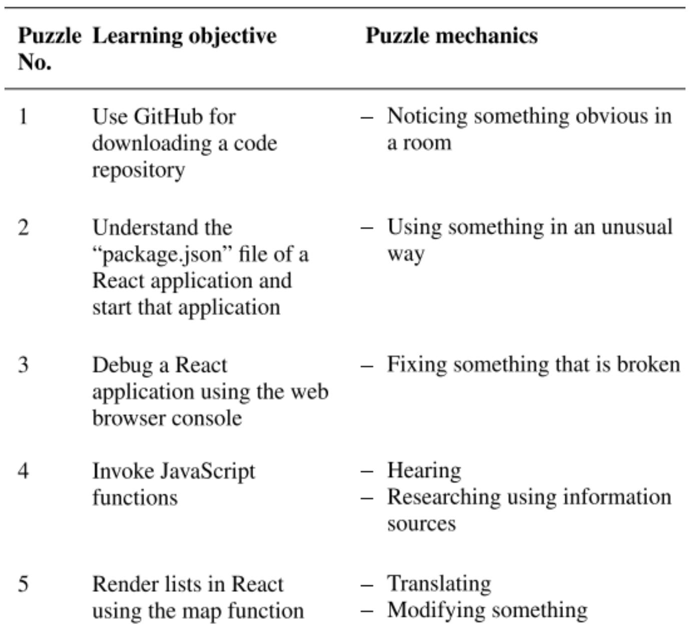

Nothing is lost, nothing is created, everything is transformed.
Antoine Lavoisier
Na_2SO_4 + $CaCO_3$ + C → Na_2CO_3 + CaS + CO_2
1791
1
1
2
1
2

Figure 5: Example of a puzzle from "Escape Classroom - The Leblanc Process" [9]

Another example of a case study targeting an educational escape room experience was performed by the UPM (Universidad Politécnica de Madrid) [10] and involved a digital escape room game centered around programming topics in a higher education setting. In order to ensure the learning objectives would be met by the interactive experience, different puzzle mechanics were designed for each of them (figure 6).



Puzzle No.	Learning objective	Puzzle mechanics
1	Use GitHub for downloading a code repository	– Noticing something obvious in a room
2	Understand the "package.json" file of a React application and start that application	– Using something in an unusual way
3	Debug a React application using the web browser console	– Fixing something that is broken
4	Invoke JavaScript functions	– Hearing – Researching using information sources
5	Render lists in React using the map function	– Translating – Modifying something

Figure 6: Some of the learning objectives targeted by the escape room experience developped by UPM [10]

Because the interactive experience took place during the usual lab times, lecturers were not able to constantly supervise and assist all students, thus a linear puzzle path was used. Even though the game was catering to experienced users, hints were still made available yet the players were required to solve smaller questions to gain access to them. In order to fully engage the participants in the interactive experience, the game used the intriguing narrative of deactivating a bomb, a theme commonly used throughout escape room experiences. Since the benefits of using storytelling to motivate the learning process have been proven in various studies [11], including this one, the proof of concept game developed throughout this project also makes use of such a tool in order to be more appealing to players.

Another resource worth noting is the **Breakout EDU** platform [12], which allows for educators to create their own escape room experiences, tailored to the skills and needs of their students. In addition, the website provides a range of pre-designed interactive learning experiences (figure 7), in both digital and physical format, which can be used and customized to enhance student engagement with the educational materials.

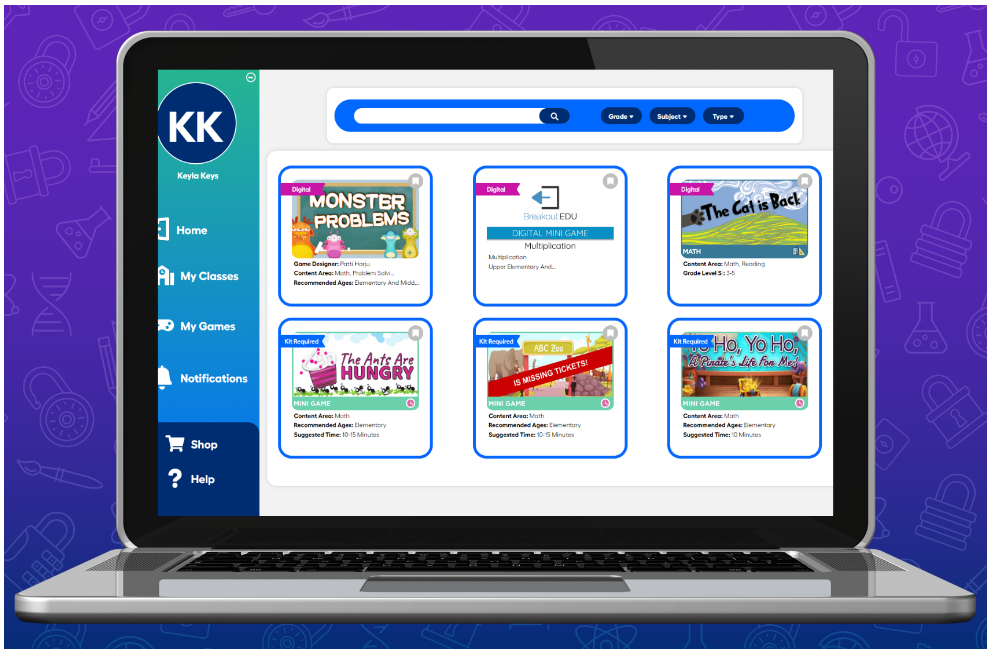


Figure 7: Example of the BreakoutEDU user interface [12]

As it can be seen from the previous examples, most educational escape room games are usually designed to integrate within the lecture plan and to either introduce content to the inexperienced students or test already taught material. This process, even though having the benefit of a teacher being present to provide guidance, can be time consuming and expensive (due to the materials used). This issue, along with the digitalization of education in the past years due to the COVID-19 pandemic, resulted in an increased interest in developing digital stand-alone escape room experiences [13].

Moreover, if such a learning tool can be made suitable for all knowledge groups, experienced students could also benefit throughout their revision for exams, where they require questions of higher difficulty. However, most existing educational interactive experiences either do not attempt to cater to all knowledge groups at all or use iterative methods that are time consuming and inefficient. This is why this project aims to fill in the gap of educational games tailored to fit all knowledge groups, while also proposing methods to adapt content in an efficient way.

2.3 Methods of Maximising Content Reusability

2.3.1 Learning Objects

Learning Objects are the result of one of the earliest attempts at content adaptation in the educational field, with the ultimate goal of maximising reuse. They are small and self-contained digital resources that can be combined and reused to generate bigger learning experiences. To help educators find suitable such materials quicker, the learning objects are usually organised in repositories based on a common set of metadata that describes their purpose, instructional context and learning outcomes [14]. Even though learning objects come in various forms, from videos to educational games, their content usually abides by some standards, such as the **Sharable Content Object Reference Model (SCORM)** or **IEEE Learning Object Metadata** [15].

The **Sharable Object Reference Model** defines a set of rules for content development and delivery, with the goal of ensuring e-learning content created by different entities could be shared across various Learning Management Systems that are compliant with the SCORM model. The main functions of SCORM include defining the communication process between a learning object and the system's back-end, consistently describing the contents of the course through the use of metadata and methods of structuring courses based on the user's needs. Before the introduction of this standard, the development of e-learning materials was slow and inefficient as they could not be used across different platforms. However, SCORM has its limitations, including only focusing on browser-based learning management systems, which require constant Internet connection and not supporting platform transitions, such as computer to mobile [16].

The **IEEE Learning Objects Metadata** is an internationally recognized standard, providing a framework for the metadata used to describe learning objects, in order to facilitate the discovery and acquisition of such educational resources. This model is composed of nine main categories of metadata elements, each with their own sub-elements, forming a hierarchy, as shown in figure 8.

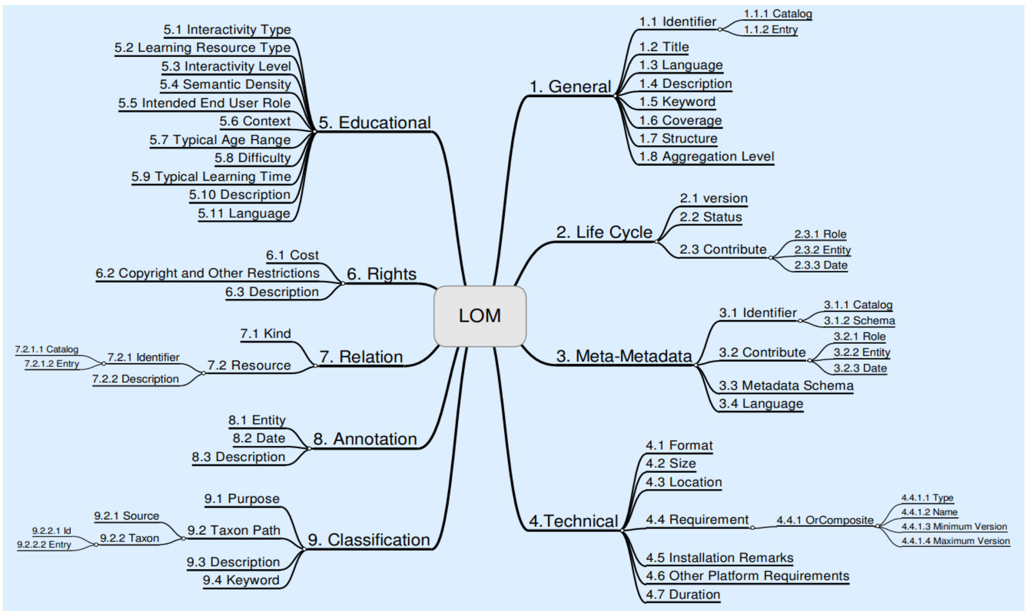


Figure 8: Representation of the hierarchy of elements in the IEEE Learning Objects Metadata Model [17]

Most notably, the **Technical** and **Relation** categories, provide information about the compatibility of any learning objects with different platforms, thus making the process of finding educational material that can integrate well within a particular system much quicker [17].

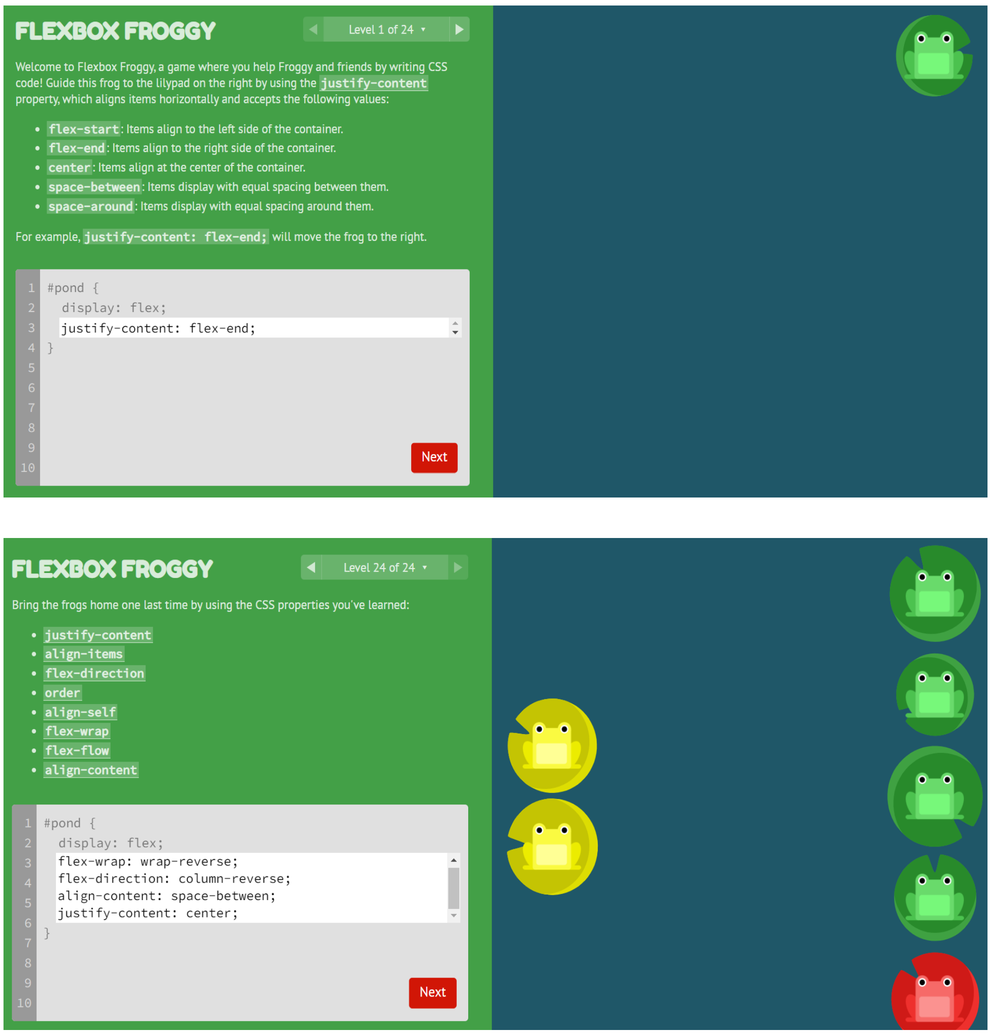
However, even though they provide high reusability across different systems and, through standardised metadata, ensure they are well organised and easy to find, learning objects have the high disadvantage of being of varying qualities and only catering to specific knowledge groups [18]. A solution to this would be combining different learning objects in the same interactive experience such that all levels of experience are accounted for, however, finding such resources and ensuring they are cohesive is a difficult and time-consuming process.

2.3.2 Difficulty Adaptation

One of the most important features of interactive experiences and the main reason for their rising popularity is their ability to challenge the player's thinking skills, while also not inducing frustration or anger [19]. This aspect becomes crucial in the case of educational games, especially those that cater to more than one knowledge group, since the game objectives must be achievable for beginners, but still challenging for the experienced players. When attempting to classify the methods of adapting the content based on the chosen difficulty, two categories can be identified: **static** and **dynamic** [20], which can be used either separately or simultaneously.

Static difficulty adaptation is **level-based** and involves the game having a couple of different levels, which are sorted in ascending order of difficulty. This method of adapting content is easy to implement, however, it requires separate materials to be developed for each difficulty, which is an expensive and time consuming process. An example of an educational game that uses this mechanism is Flexbox Froggy [21], a web development interactive experience that teaches a few CSS commands. In order to tailor the contents of this game to all knowledge groups, multiple levels are available and the player can choose to skip to any level they find suitable (figure 9).

Dynamic difficulty adaptation is **experience-based** and requires the game components to adapt to the player's skills. This type of content adaptation is less encountered in educational games due to the difficulty of the implementation and the fact that game-based learning is usually targeted to only one knowledge group, not accounting for the user gaining knowledge throughout the experience. Even though this approach is complex, it can have many benefits in an educational context as every user experience would be uniquely designed to adapt to their skills and needs.


FLEXBOX FROGGY

Level 1 of 24

Welcome to Flexbox Froggy, a game where you help Froggy and friends by writing CSS code! Guide this frog to the lilypad on the right by using the `justify-content` property, which aligns items horizontally and accepts the following values:

- `flex-start`: Items align to the left side of the container.
- `flex-end`: Items align to the right side of the container.
- `center`: Items align at the center of the container.
- `space-between`: Items display with equal spacing between them.
- `space-around`: Items display with equal spacing around them.

For example, `justify-content: flex-end;` will move the frog to the right.

```
#pond {
  display: flex;
  justify-content: flex-end;
}
```

FLEXBOX FROGGY

Level 24 of 24

Bring the frogs home one last time by using the CSS properties you've learned:

- `justify-content`
- `align-items`
- `flex-direction`
- `order`
- `align-self`
- `flex-wrap`
- `flex-flow`
- `align-content`

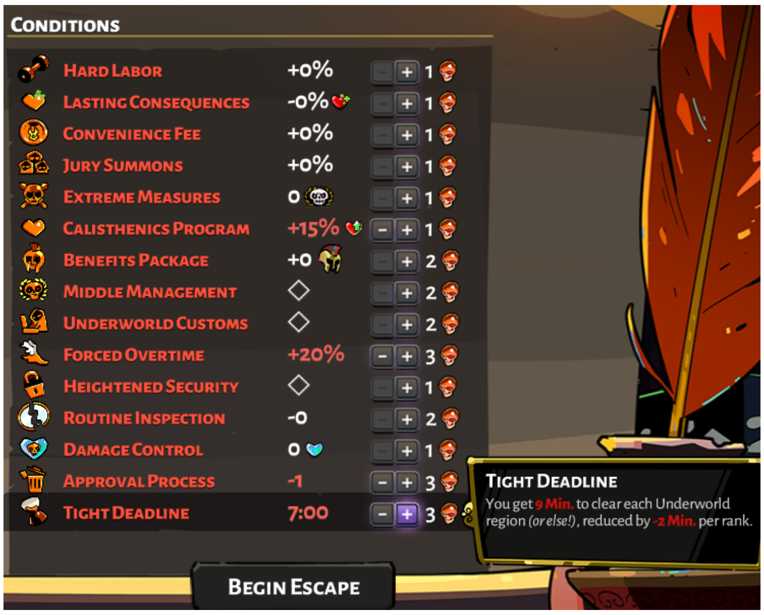
```
#pond {
  display: flex;
  flex-wrap: wrap-reverse;
  flex-direction: column-reverse;
  align-content: space-between;
  justify-content: center;
}
```

Figure 9: Screenshots from the educational game Flexbox Froggy [21], showing two different levels

An example of a game that uses a combination of static and dynamic difficulty adaptation methods is the game **"Hades"** [22], released in 2020 by Supergiant Games [23]. The game follows the story of Zagreus, the son of Hades, as he attempts to escape from the Underworld with the help of the Olympian Gods. Each attempt involves moving through four regions, each containing a random number of rooms, by defeating enemies, which become increasingly stronger with every new region (static difficulty adaptation). This game immensely inspired the structure of the proof of concept interactive experience developed throughout this project, as it demonstrates efficient ways of maximising content reuse in general and across different experience levels.

Firstly, the game uses probabilistic randomisation techniques to make the structure of each escape attempt unique. By shuffling the power-ups, the rewards and the number and design of rooms, the outcome of each escape attempt becomes highly unpredictable, technique proven to increase the difficulty of the game [24], while reusing the already existing content (dynamic difficulty adaptation).

Secondly, to account for the user gaining experience, after a few successful escapes, a list of constraints, called "The Pact of Punishment" is provided (figure 10).

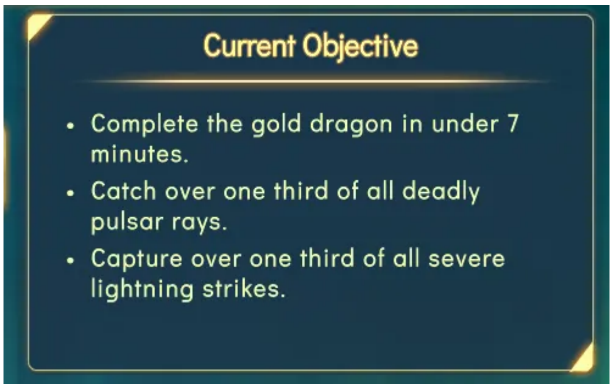

CONDITIONS

HARD LABOR	+0%	1
LASTING CONSEQUENCES	-0%	1
CONVENIENCE FEE	+0%	1
JURY SUMMONS	+0%	1
EXTREME MEASURES	0	1
CALISTHENICS PROGRAM	+15%	1
BENEFITS PACKAGE	+0	2
MIDDLE MANAGEMENT	◇	2
UNDERWORLD CUSTOMS	◇	2
FORCED OVERTIME	+20%	3
HEIGHTENED SECURITY	◇	1
ROUTINE INSPECTION	-0	2
DAMAGE CONTROL	0	1
APPROVAL PROCESS	-1	3
TIGHT DEADLINE	7:00	3

TIGHT DEADLINE
You get 9 Min. to clear each Underworld region (or else!), reduced by -2 Min. per rank.

Figure 10: Screenshot from "Hades" [22] showing the constraints that can be applied after the user gains experience

One of the most notable such constraints is the time limitation, a commonly used technique to increase the difficulty of a game, without requiring the creation of new content. Another example of a game which employs a time limitation when the player reaches an appropriate level of experience, is **"Spiritfarer"** [25], released in 2020 by Thunder Lotus Games [26]. While the timer used in "Hades" increases by a certain number of minutes for every new region, the time limitation in "Spiritfarer" is a fixed value, which applies only to a particular game challenge (figure 11).

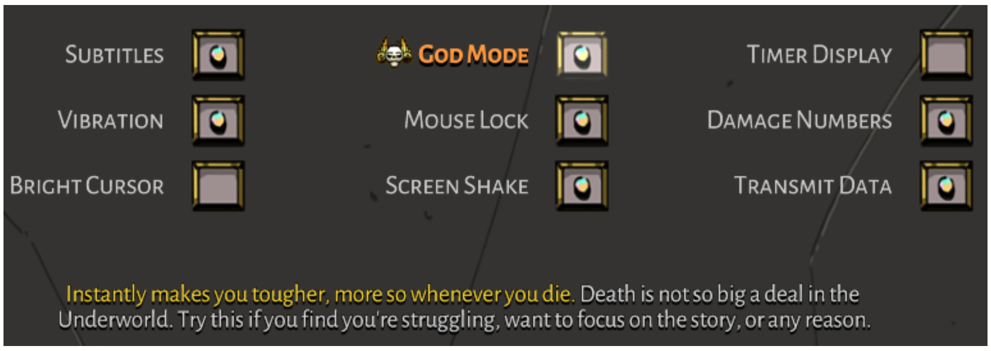


Current Objective

- Complete the gold dragon in under 7 minutes.
- Catch over one third of all deadly pulsar rays.
- Capture over one third of all severe lightning strikes.

Figure 11: Screenshot from "Spiritfarer" [25] displaying the time constrained challenge

A feature that both of the previously mentioned games have in common is that, aside from the constraints provided for the higher difficulty, they also account for the player being too inexperienced to complete the game. As a result, "Hades" offers the additional option of "God Mode" [27], which increases the character's resistance to the attacks by 10% with every loss, until the threshold of 80% is reached (figure 12).



SUBTITLES	✓	GOD MODE	✓	TIMER DISPLAY	
VIBRATION	✓	MOUSE LOCK	✓	DAMAGE NUMBERS	✓
BRIGHT CURSOR		SCREEN SHAKE	✓	TRANSMIT DATA	✓

Instantly makes you tougher, more so whenever you die. Death is not so big a deal in the Underworld. Try this if you find you're struggling, want to focus on the story, or any reason.

Figure 12: "God Mode" feature from "Hades" [22]

In the case of “Spiritfarer”, when the challenge is lost, the player receives the option of increasing the time limit to a more suitable value, as it can be seen in figure 13.

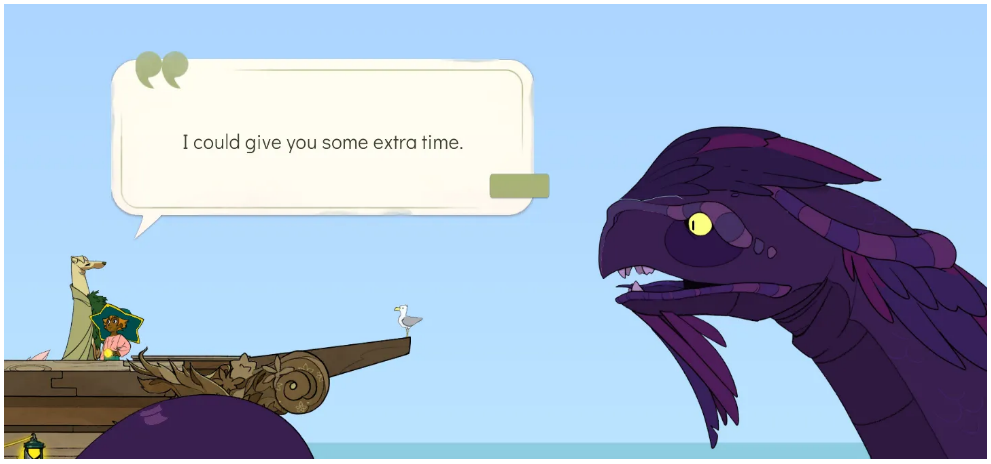


Figure 13: Screenshot from “Spiritfarer” [25] showing the option to receive more time in case of failure

Other implementations of dynamic difficulty adaptation techniques mainly focus on games that use artificial intelligence agents and machine learning methods to design player profiles and adapt the behaviour of the agents to their skills and experience [28]. Implementing such a complex mechanism in the context of educational games, while bringing multiple advantages such as tailoring the interactive experience to the needs of the user, would require extensive training and testing of the model on a large number of users, resources that are not easily accessible in the context of this project’s development. However, certain principles of dynamic difficulty adaptation, similar to the ones present in “Hades”, have been used during implementation as they promote reuse of content, with no iterative development of separate levels for each knowledge group being required.

On the other hand, aside from the complexity of the implementation, another concern with dynamic difficulty adaptation, especially in the context of educational games, is needing some type of prior assessment of the user knowledge before choosing how to adapt the content. To account for this, the proof of concept game uses principles from static difficulty adaptation as well, by having the player select a difficulty level (beginner, intermediate or advanced) before starting the game, then modifying the chance of appearance of certain puzzles depending on how suitable they are for the chosen level. Even though some puzzles are designed with a particular knowledge group in mind and their chance of appearance for other groups is lowered, they might still be encountered and made suitable through hints and time constraints, depending on the situation.

2.3.3 Randomisation Algorithms

As previously discussed, one way in which games adapt their content to fit different difficulty levels is through randomisation of the environment and other game features. The most common way in which this is achieved in escape room-style games is by randomising some of the codes required to solve the puzzles. In the context of this project, to maximise reuse of content, the rooms and puzzles will be randomised, based on the conditions mentioned in the Introduction section.

In their article, Alex Grasas, Angel A. Juan, Javier Faulin, Jesica de Armas and Helena Ramalhinho [29] explore algorithms for biased randomisation, showcasing their strengths and weaknesses. The main issue with most existing methods is the fact that they use a uniform probability distribution when choosing a random value. This means that, even though the candidate elements may initially be ranked based on some criteria, the decision process is entirely random. As a result, they all have the same chance to be picked and the original ranking is neglected.

A solution to this is introducing bias to the randomisation process, by using a couple of "priority rules". In the context of this project, these rules represent how suitable the puzzle is for the selected difficulty level, how often it has been encountered by the user and how much experience the player has gained during gameplay. These are reflected in a *probability of appearance* value assigned to each puzzle in the game. The **Parameterized Biased Random Sampling** algorithm presented in the previously mentioned article uses this probabilistic value to extract the highest ranking such elements and add them to a *candidates* subset. In order to introduce randomness to this process, a few other elements having the lower probabilistic values are then added to the *candidates* subset, from which a random value will be extracted. Thus, the algorithm is able to successfully incorporate the initial ranking into the shuffling process.

3 Project Management

3.1 Methodology

Ensuring the success of the project requires choosing the appropiate development methodology in order to guarantee the timely completion of all the required deliverables. Because the scope of the project was large and subject to change, due to research being done during development, an agile approach was the most suitable. Particularly, Scrum [30] allowed for the use of incremental development, through sprints, to slowly implement new game features every one or two weeks. In addition, due to the requirement for robust documentation, some plan-based elements were also adopted, such as principles from the PRINCE2 methodology. Most notably, planning and development were product-oriented, therefore when scheduling issues appeared, the objectives were re-prioritized ensuring the deliverables (the proof of concept game and the documentation) would be completed on time.

3.2 Project Planning

The **Work Breakdown Structure (WBS)** is a hierarchical decomposition of a project into smaller tasks called work packages, making it easier to manage and estimate the time and resources needed for completion. Figure 14 shows the WBS that was constructed for the development of the proof of concept game, dividing it into six deliverable oriented small packages: assets, rooms, puzzles, player, UI and the memory fragment system.

By using this deliverable-oriented WBS, the tasks could be better organised to ensure all core objectives of the project would be achieved by the end of development. Figure 15 shows the sprints that were devised and their predicted duration, in order to implement all features included in the WBS.

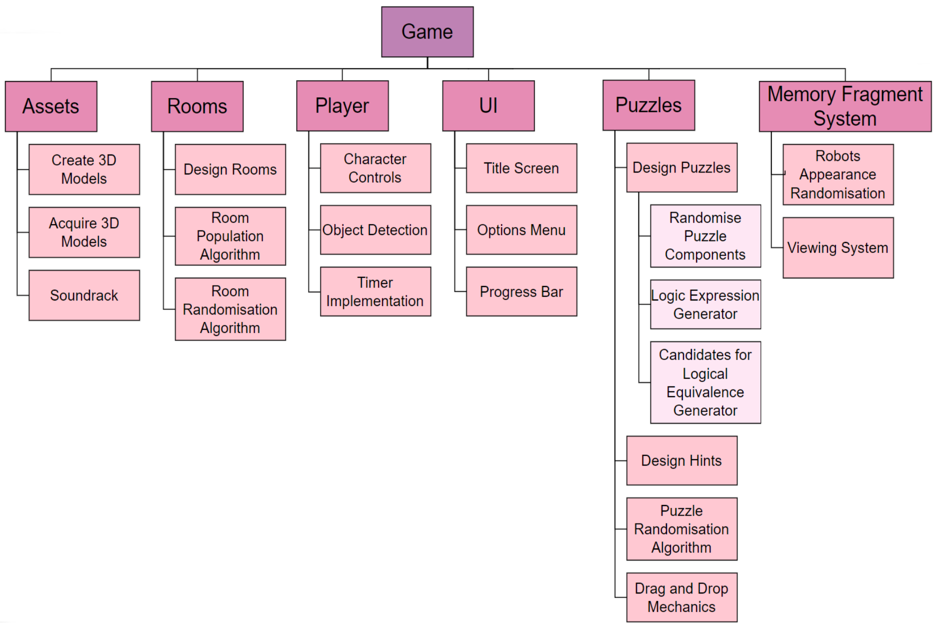


Figure 14: Work Breakdown Structure (WBS) of the project

Sprint	Work Package	Time Allowance
Sprint 1	Assets	2 weeks
Sprint 2	Rooms	1 week
Sprint3	Player	2 weeks
Sprint 4	UI	1 week
Sprint 5	Puzzles	2 weeks
Sprint 6	Memory Fragment System	1 week

Figure 15: Organisation of sprints

In order to account for possible delays or major changes in the project's scope, some of the plan was left flexible. If all game features were to be completed on schedule, this time would have been used to implement some of the extensions or to improve the assets used in the game.

3.3 Project Objectives

To ensure the project would achieve all its goals, measurable objectives had to be defined. In order to better manage the large resulting scope and ensure that development was focused on the key features of this project, MoSCoW prioritization [31] was used. This technique consists of assigning each objective one of the following values, depending on its contribution to the project:

- **Must** have: crucial requirements, needed for a project to be considered a success
- **Should** have: important features that, even though are not vital, add value to the project
- **Could** have: desirable features, but not necessary for the project to be considered a success
- **Won't** have: features that will not be implemented in the current version of the project, but may be in the future

The tables containing the functional requirements, non-functional requirements and extensions, along with their assigned priority after MoSCoW analysis can be found in Appendices A, B and C respectively.

For the functional requirements, it can be observed that most of them are assigned the *Must* or *Should* priorities. This is because they represent valuable features that are imperative for the project to achieve its goals. Most notably, all requirements related to the four methods of content adaptation mentioned in the Introduction section have been assigned *Must* as, without their implementation, the project would not be considered a success. In addition, all requirements referring to user experience and game functionality have been assigned *Should* as, even though they provide significant value to the final product, they do not add anything to the problem this project is trying to solve.

Because the development of the proof of concept game is time restricted, it is important to be able to identify features which, while providing value to the product, are unrelated to the main aims of this project. Therefore, most of the requirements assigned *Could* refer to the quality of the assets used in the game and aspects of the user experience which are not essential in solving the issues previously presented and have less of an impact on the functionality of the interactive experience. The requirements that have been assigned *Won't* can still add value to the product, but their implementation would not be feasible in the limited time allowed for this project's development. However, due to their potential, they should still be considered as an improvement if the project is to be continued in the future.

3.4 Risk Management

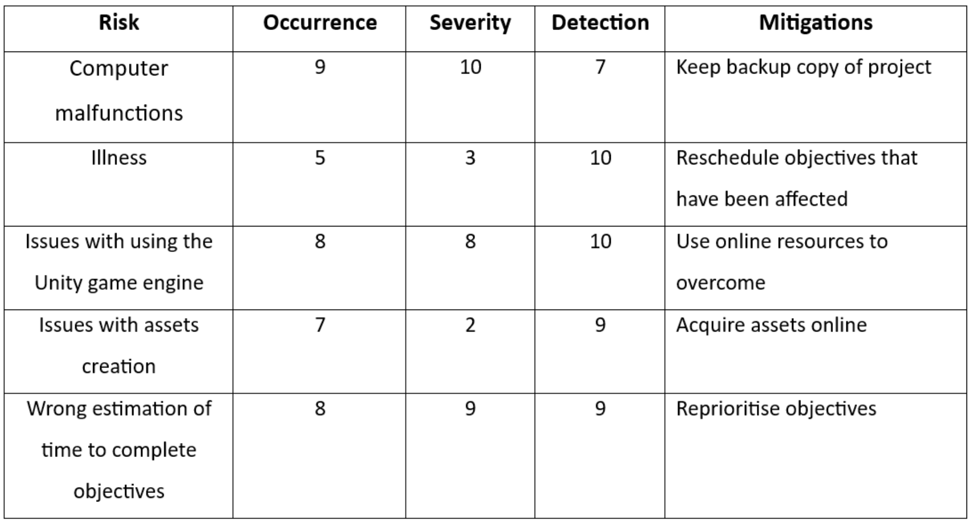
An essential part of the Planning Stage is the identifying and managing the potential risks. This ensures that, if problems do occur, solutions are put in place to minimise any delays in the development process. To this end, *FMEA analysis* [32] was performed to compute, for every identified risk:

- the likelihood of it occurring,
- its potential impact on the development process,
- how easy it is to detect,
- mitigations.

The table in figure 16 shows the results of the *FMEA analysis.* As it can be observed, one of the most impactful risks to the project's development is the wrong estimation of time for different objectives. This situation is very likely to happen due to inexperience with using the Unity game engine and the research stage that overlaps with development, as new objectives can be added to the scope or the project can incur major changes. As previously mentioned, the timetable for term 2 was left flexible in an attempt to minimise the effects of this risk occurring.

3.5 Use of Tools

To guarantee the success of the project, it is important that suitable tools are identified and utilized to their full potential.



Risk	Occurrence	Severity	Detection	Mitigations
Computer malfunctions	9	10	7	Keep backup copy of project
Illness	5	3	10	Reschedule objectives that have been affected
Issues with using the Unity game engine	8	8	10	Use online resources to overcome
Issues with assets creation	7	2	9	Acquire assets online
Wrong estimation of time to complete objectives	8	9	9	Reprioritise objectives

Figure 16: FMEA Analysis of potential risks

3.5.1 Unity Asset Store

An essential part in creating an immersive interactive experience are the game objects used. Since the scope of this project was large, time had to be spent wisely on implementing game features so that all the objectives could be achieved. This is why finding good resources for importing 3D assets was essential. The Unity Asset Store [33] is an online marketplace where developers can buy and sell assets for use in the Unity game engine. This particular resource was the preferred choice for asset acquisition in the context of this project due to the presence of free to use 3D models made specifically to be compatible with Unity. By using mostly pre-made game objects, the development process was significantly accelerated.

3.5.2 Blender

Blender [34] is a free to use 3D modelling software, widely utilized for the creation of 3D game assets. The main benefits of using Blender include the large variety of features available and the active community that provides support, tutorials and resources. However, Blender has the drawback of being challenging for beginners to learn due to the complex interface and many features available. Moreover, compatibility issues were encountered when trying to export some of the textures in the Unity game engine. As a result, in this project, Blender was only used for the specific 3D objects required in the game that could not be acquired from other sources.

3.5.3 Unity

Unity [35] is one of the most popular game engines, due to its large range of supported platforms and its flexibility, allowing for a wide variety of interactive experiences to be developed. This game engine was chosen for this project mainly because of the large collection of resources and tutorials available and the active community of developers. Most notably, the Create With Code [36] tutorials made by Unity Technologies helped expand understanding of basic game development concepts and ways to implement commonly used features, such as character controls and title screens.

3.5.4 ProBuilder

ProBuilder [37] is a toolset recently acquired by Unity, which introduces features for 3D modelling as well as level design tools, allowing for quick structure prototyping. One major advantage of using ProBuilder instead of any other tool for environment creation is the seamless integration with the Unity game engine. Therefore, importing and exporting assets and textures can be done easily without worrying about compatibility issues. Moreover, due to the smooth integration with Unity, environments created with ProBuilder can be tested and tweaked without having to reimport the models. Since the proof of concept game developed throughout this project is designed after the escape room format, quick environment creation was necessary, therefore this resource proved essential to ensure the project was progressing as expected.

4 Project Design

4.1 Structure of the Game

The proof of concept game respects the escape room format previously mentioned in the Background section and consists of a couple of *"runs"*, defined as a random number of rooms that must be escaped. Each room consists of a random number of puzzles which the player needs to solve, in order to move to the next environment. In order to cater to all knowledge groups, there are three possible difficulty levels the player can choose from: **beginner**, **intermediate** and **advanced**. To ensure the content of the game is appropriate for the chosen difficulty while maximising reusability, various methods of content adaptation were used, some static and some dynamic.

To ensure the inexperienced players are able to solve the puzzles, hints were provided and placed in the rooms for the *beginner* difficulty level only. To help the users develop their cognitive skills, the hints were designed as smaller versions of the questions in the puzzles, with a provided solution. Because it is important that the game is challenging for the experienced players, no hints are provided for the *intermediate* and *advanced* difficulties. Moreover, for the *advanced* players a time constraint is applied to make solving the puzzles more difficult. To avoid the possibility of a player being stuck on a puzzle and unable to leave the room, a "Give up" option is available which returns the player to the initial spawning place and decreases the chance of appearance of the unsolved puzzles in the previous room, thus introducing dynamic difficulty adaptation principles. Moreover, because some of the players might wrongly choose the difficulty in the beginning, they are allowed to change it anytime during gameplay, by accessing the options menu.

The process of creating content for the game, such as puzzles, hints or rooms, has been simplified by using different scenes for all such elements. Due to this, individuals that were not involved in the development process could still implement their own educational content that would seamlessly blend in the interactive experience.

4.2 Storyline

As previously discussed, an important aspect of interactive learning experiences is creating an engaging environment that motivates the player to apply or learn certain skills. To ensure the proof of concept game is appealing and a reward (one of the main structural components of the escape room format) is provided, after a random number of successful *runs* the player encounters a robot (figure 17) that carries a *memory fragment* containing part of the storyline.

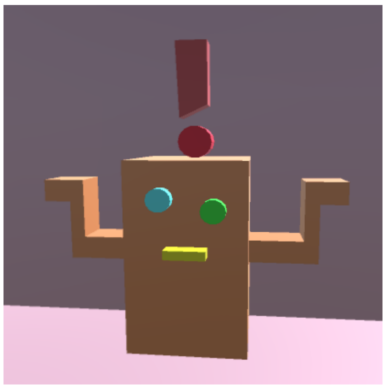


Figure 17: Screenshot from the proof of concept game, showing the robot carrying memory fragments

Because one of the objectives of this project is providing a space for the player to track their progress, the *memory fragment* can only be obtained after solving a higher difficulty puzzle than the others in the game. Since no hints are provided, this puzzle is optional and can be completed later in the game.

The story revolves around a programmer that lost his daughter and created a virtual reality experience to be able to relive their memories together. However, the program malfunctioned, which results in the character losing his memory and unknowingly getting stuck in the virtual experience. By using the memory fragment system previously described, these story parts are able to act as educational milestones for the player, while also providing the game with a well-defined goal and ending. The title of the game is *"Project Donut"* in reference to the 3D donut that was created during tutorials to become more familiar with Blender.

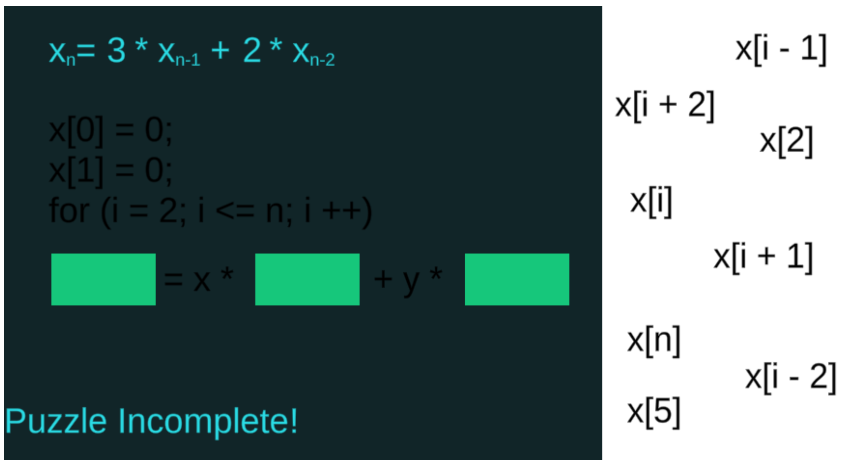
4.3 Puzzle Design

There are four Computer Science topics present in this game in the form of puzzles: object oriented programming, logic and verification, database systems and web development. However, each of these topics have a different number of questions dedicated to them depending of how well the content could be randomised and adapted to fit the objectives of this project. For example, throughout development, potential to randomly generate logic expressions was discovered and added to the project scope. Due to this, highly randomised logic and verification questions could be generated from the same content and adapted to fit the different difficulties. On the other hand, not many interesting content randomisation techniques were discovered for the Web Development and Database Systems categories, thus not many such questions are present in the final game.

There are 3 puzzle formats used in this interactive experience:

- Drag and Drop Puzzle
- Input $\Rightarrow$ Output Puzzle
- Correct Choice Puzzle

The Drag and Drop format is used across several puzzles in the game and involves the user having to click and drag different objects in their correct places. As it can be seen in figure 18, the player can use the code parts from the right side to complete the program on the left.



$x_n = 3 * x_{n-1} + 2 * x_{n-2}$

```
x[0] = 0;
x[1] = 0;
for (i = 2; i <= n; i ++)
        = x *        + y *
```

Puzzle Incomplete!

Figure 18: Example of a Drag and Drop puzzle from “Project Donut”

The Input $\Rightarrow$ Output puzzle format involves providing the user with code and input values and requesting the resulting output. In order to make these puzzles reusable and ensure they can appear in the game multiple times with different solutions, most variables are randomised each time the puzzle is loaded, a common technique in escape room style games.

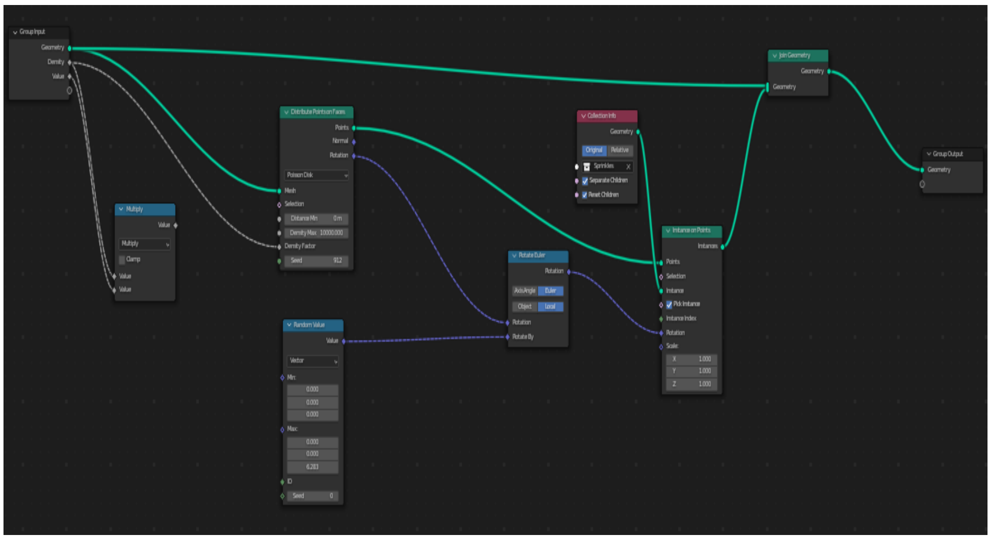
The Correct Choice puzzle involves giving the user a simple question and a few possible answers. To ensure the player does not try each answer until a correct one is reached, the puzzle is reloaded with every wrong attempt, having its content randomised and a different solution.

5 Project Implementation

5.1 3D Assets Creation

When developing the proof of concept game, the first step was creating the 3D models that would be used in the interactive experience and could not be acquired from other sources. As mentioned in the previous section, Blender was used to create the donut used in the title screen and the robots which contain the memory fragments. An essential feature used in the creation of models that present symmetry, such as the robots, is the *mirror modifier* [38]. This allows for the model to be duplicated on any of the three axes (X, Y and Z) and any changes made to be reflected on both sides. By mirroring the robot on the Y axis, the need to create both hands individually while ensuring they were symmetrical was eliminated.

When creating the donut model, a series of YouTube tutorials by Blender-Guru [39] were followed in order to become more familiar with the Blender interface and the various features available. Most notably, the *geometry nodes* [40] tool can be used to procedurally alter an object's properties, such as colour, material or geometry. In the case of the donut, they were utilized to automatically randomly place the sprinkles, without having to alter the entire model later if changes were to be made. Currently, three variations of the sprinkles are placed on the donut, by using the *Weight Paint* mode [41] to choose the areas with more or less sprinkles. Because of the geometry nodes, the process of editing one of the variations or adding a new one requires only one model to be altered or created.



Distance Min 0 m
Density Max 10000.000
Seed 912

Min: 0.000 0.000 0.000
Max: 0.000 0.000 6.283
Seed 0

X 1.000
Y 1.000
Z 1.000

Figure 19: Geometry nodes used to create the sprinkles on the donut

5.2 Environment Creation

Another big step in the implementation stage was designing the rooms which contain the puzzles. Because the game follows the escape room format, these are essential for the interactive experience to be considered complete. As mentioned in the Use of Tools section, ProBuilder was used to simplify the proccess of environment creation and ensure the rooms had diverse structures, without too much time being spent towards this aspect.

In edit mode, ProBuilder allows for four selection modes: **object**, **vertex**, **edge** and **face**, each presenting a different set of functions that can be used to modify the objects. A notable feature that was used in designing all the rooms is the *Flip Normals* function [42], which allows for the rooms to be modelled from the exterior, then turned into interior spaces. In addition, the *Extrude* methods [43] available for all selection modes were used to create unique shapes for all the rooms.

5.3 Scene Management

Because the game revolves around escaping from a couple of rooms which are designed as separate scenes, a script to manage the way these are created and loaded was crucial to its functionality. To this end, an instance of the *ManageScenes* script was used in every room, by attaching it to a *Manager* object prefab. Figure 20 shows the code used to ensure that there is exactly one instance of this managing tool in each room, without having to manually add the Manager prefab to every new design that is created. This approach has the advantage of making the creation of new environments in the game accessible to individuals that were not involved in the development of this project, thus applying principles from the learning objects approach and making the product valuable for use in other educational contexts.

```
public class ManageScenes : MonoBehaviour{
        public static ManageScenes Instance;
        public void Awake(){
                if (Instance != null){
                        Destroy(gameObject);
                        return;
                }
                Instance = this;
                DontDestroyOnLoad(gameObject);
        }
}
```

Figure 20: Code showing how the instance of the ManageScenes class is created in every scene

Aside from managing the ways in which the scenes are loaded, the script also contains functions for the randomisation of rooms and puzzles in the game in accordance to the algorithm explored in the Background section, as well as variables that should be globally available to access and modify, such as the selected difficulty.

5.4 Room Randomisation

Since the project aims to maximise reuse of content, an important step of the implementation stage is represented by the Room Randomisation Algorithm, which has the following goals:

- randomly choose the next room to be loaded
- ensure that the same room is not encountered twice in the same run
- the chance of the player returning to the main room increases with every newly encountered space
- there is at least one room in a run
- there are no more than five rooms in a run

To achieve this, the algorithm begins by creating three lists of rooms: **visitedRooms**, **backupRooms** and **mainRooms**. These lists are initialized in a separate script that is only run when the main room is loaded. This is because the end of the run is defined as the player reaching the main room, thus the lists should be reinitialized in preparation of the next run. At the beginning of the game, the *visited* list is empty and five random rooms are added to the *backup list*, the rest being added to the *main list*.

In the *SceneManagement* script, a randomisation function is used to select the next room that would be loaded after the player successfully escapes the current environment. This function creates a *candidateRooms* list (figure 21) containing:

- all rooms in the main list,
- a randomly chosen room from the backup list,
- no rooms from the visited list.

The *candidateRooms* list does not contain elements from the *visitedRooms* list to ensure that the player does not encounter a room more than once in each run, to avoid the game becoming repetitive.
The *backup list* is used as a way to add the chance of the run ending early

```
//empty candidates list
candidateRooms.Clear();

//add all the rooms from the main list to the candidates list
candidateRooms.AddRange(mainRooms);

//get one room from the backup list and add it to the candidates list
backupRoom = backupRooms[Random.Range(0, backupRooms.Count)];
candidateRooms.Add(backupRoom);

//get a random room from the candidates list
nextRoom = candidateRooms[Random.Range(0, choiceRooms.Count)];
```

Figure 21: C# code showing the creation of the candidates list

(by encountering the main room), without this event being as probable as the choice of any other room in the game. To this end, the probability of the main room being chosen is increased with every selection by adding an instance of its corresponding scene to the *backup list*. To guarantee a run does not exceed five rooms, each room selected by the algorithm is removed from its original location, then added to *visited list*. Before the randomisation algorithm is called again, the number of elements in the *visited list* is checked and, if it is equal to five, the main room is loaded (figure 22).
When a run ends and the player is back in the main room, the room lists are recreated (figure 23):

1. all instances of the main room are deleted from the backup list,
2. the rooms in the backup list are moved to the main list,
3. the rooms in the visited list are moved to the backup list,
4. the backup list is supplemented with rooms from the main list as needed (until the number of elements is five).

The reason why the already visited rooms are added to the *backup list* instead of the *main* one is because the player should have a higher chance of encountering new rooms at each run. Therefore, by only choosing one

```
//add the current room to the visited list
visitedRooms.Add(currentScene);

//remove the current room from its original location
backupRooms.Remove(currentScene);
mainRooms.Remove(currentScene);

//add an instance of the main room to the backup list
backupRooms.Add(2);

//check if the run needs to end
if (visitedRooms.Count == 5)
{
    SceneManager.LoadScene(mainRoom);
}
```

Figure 22: C# code showing the actions taken before a room is chosen

room out of the *backup list* to become a candidate for selection, the chance of rooms being repeated in consecutive runs is as low as possible.

5.5 Puzzle Randomisation

The Puzzle Randomisation Algorithm is similar to the algorithm used for shuffling the rooms, but it adds more probability to the process. Because some of the contents of each puzzle are randomised, a player can encounter the same question twice in a run as the answer will be different every time and they will still be required to apply their critical thinking skills. Moreover, it is a well known fact that repetition represents the basis of the learning process [44]. Thus, by allowing the player to encounter a puzzle multiple times during gameplay, a revision process is simulated, allowing the educational content to be absorbed more efficiently.

In addition, because the project aims to maximise reuse across all knowledge groups, the content that is considered of unsuitable difficulty for a category of users should still have a chance of being encountered in order to combat iterative creation of materials for each separate knowledge group. To solve the issue of the interactive experience becoming too easy or difficult, the

```
//remove all instances of the main room from the backup list
backupRooms.RemoveAll(x => x == mainRoom);

//move all remaining rooms from the backup list into the temporary list
toMove.AddRange(ManageScenes.Instance.backupRooms);

//clear the backup list
backupRooms.Clear();

//add the rooms from the visited list to the backup list
backupRooms.AddRange(visitedRooms);

//clear the visited list
visitedRooms.Clear();

//suplement the backup list with rooms from the main list if needed
while (backupRooms.Count < 5)
{
    room = mainRooms[Random.Range(0, mainRooms.Count)];
    backupRooms.Add(room);
    mainRooms.Remove(room);
}

//move the rooms from the temporary list to the main list
mainRooms.AddRange(toMove);
```

Figure 23: C# code showing the modifications made to the room lists at the end of a round

conditions of the game are altered depending on the selected difficulty. As such, beginners are provided with hints to aid in solving the more difficult puzzles that may appear and experienced players are constrained by a time limit.

Another important feature is the use of dynamic difficulty adaptation techniques, which involve the game increasing in difficulty as the player gains experience. To implement this, as the player progresses through the game, the probability of appearance of puzzles with higher difficulty is slowly increased. Because the chance of encountering previously solved puzzles is lowered, after some time, the player will come across more difficult puzzles, without having to alter the game options manually.

Considering all of the above, the Puzzle Randomisation Algorithm has the following goals:

- puzzles of suitable difficulty have a higher chance to be encountered,
- puzzles of unsuitable difficulty still have a chance to be encountered, but it is lower
- as the player gains experience, the probability of higher difficulty puzzles increases
- the player has less chance of encountering previously solved puzzles

To achieve these, the algorithm bases itself on the algorithm proposed by Alex Grasas, Angel A. Juan, Javier Faulin, Jesica de Armas and Helena Ramalhinho in the article discussed in the Background section. The Puzzle Randomisation algorithm divides the puzzles into sets, also called probability pools, each with a different assigned chance of being selected, depending on the chosen difficulty level. Therefore, suitable puzzles will be assigned a probability of 100, while unsuitable ones will be given the value 50. This is achieved through a list of *Puzzle* structs (figure 24), that specify two values for each element: **puzzle** (indicating the index of the scene containing the puzzle) and **probability** (the assigned probability).

```
public struct Puzzle
{
    public int index;
    public int probability;
}
```

Figure 24: C# code showing the definition of the *Puzzle* struct

The similarity to the Room Randomisation Algorithm is given by the process of building the *candidates list*.Thus, the Puzzle Randomisation Algorithm selects all the elements from the *main list*, which in this case, is a list of the puzzles with maximum probability values and a lower number of elements from the *backup list*. In this case, the *backup list* is represented by all the puzzles with lower probabilities. In order to emulate these given ranking, without losing the randomisation factor, the number of elements chosen from each probability pool will vary in accordance with the assigned value of that particular subset. To implement this in an efficient way, supposing **P** is the assigned probability of the current set and **N** is the number of elements in the set, the algorithm will choose $\mathbf{N/(100/P)}$ elements. In this way:

- when P = 100: N elements will be selected
- when P = 50: N/2 elements will be selected (half of them)
- when P = 10: N/10 elements will be selected (one if N = 10)

After the *candidates list* is complete, one puzzle is chosen at random and returned by the function.

Because the game also aims to increase the chance of more difficult puzzles as the player gains experience (dynamic difficulty adaptation), the chance of appearance for puzzles needs to be updated after every run, depending on the circumstances. This is handled by a separate script attached only to the main room. This is because when a player reaches that room the run is considered to have ended, either:

- **successfully**, with all the puzzles in the rooms being completed,
- **unsuccessfully**, when the timer runs out for the advanced difficulty or
- **unsuccessfully**, when the player gives up.

To differentiate between these scenarios, a *boolean variable* is used, taking the **true** value for successful and **false** value for unsuccessful. A record is kept of all the puzzles the player has completed and, if the run was successful, the chance of appearance of those questions is decreased. Whenever the player gives up or the timer runs out, the unsolved puzzles from the last room are retrieved and their probability of appearance is lowered. Moreover, for each puzzle that was not encountered in the run, the chance of appearance is increased, allowing for new content to have a higher chance of being chosen.

5.6 Room Population

Since the game reuses the same room designs and just randomises the order in which they appear, measures have to be taken in order for the game not to become repetitive. Therefore, the **RoomPopulation** script was created to ensure that each time the player encounters a room, the puzzles and hints are found in different randomised places. Moreover, the script also adapts the number of puzzles in a room, depending on the selected difficulty, in the following way:

- between 1 and 3 puzzles for the beginner level,
- between 2 and 4 puzzles for the intermediate level,
- between 3 and 5 puzzles for the advanced level.

In order to randomise the number of puzzles present into a room, without altering the design of the room, all the objects were given appropriate tags [45], with the most important being *"Puzzle"* and *"Hint"*. At the start of the RoomPopulation script, a random number of puzzles to be activated is chosen, by using the **Random.Range** function [46], according to the selected difficulty, as previously discussed. Afterwards, two separate lists are created: objects in the room tagged *"Puzzle"* and objects in the room tagged *"Hint"*. From this lists, random elements are chosen depending on the desired number of active puzzles and their tags are changed to *"ActivePuzzle"* and *"ActiveHint"* respectively.

After this, in order to link the game objects to their respective scenes, a struct named *GamePuzzle* was used (figure 25), containing four values: **pLocation** (the game object containing the puzzle), **pContent** (the puzzle scene), **hLocation** (the game object containing the hint) and **hContent** (the hint scene).
After a puzzle has been solved, its linked game object's tag is changed back to *"Puzzle"*, the GamePuzzle struct is removed from the active puzzles list and the number of active puzzles in the room is decreased.

```
public struct GamePuzzle
{
public GameObject pLocation;
public int pContent;
public GameObject hLocation;
public int hContent;
}
```

Figure 25: C# code showing the *GamePuzzle* struct

5.7 Character Controls

After the rooms were created, the way the player would move and be able to interact with the environment had to be defined. The proof of concept game was designed in a 3D format in order to simulate the real life escape room ambience and seamlessly blend with the story of a virtual reality experience the game revolves around. Therefore, the most appropriate type of character controls was a first person view that would allow the player to move in all directions and look around the room.

To allow the player to move forwards, backwards, left and right, input from the keyboard keys is taken and multiplied with a *speed* variable and the *Time.deltaTime* variable [47], which is used to scale the movement based on the amount of time that passed from the last frame (figure 26).

```
void FixedUpdate(){
horizontalInput = Input.GetAxis("Horizontal");
verticalInput = Input.GetAxis("Vertical");

transform.Translate(Vector3.right * horizontalInput
                    * Time.deltaTime * speed);
transform.Translate(Vector3.forward * verticalInput
                    * Time.deltaTime * speed);
}
```

Figure 26: C# code showing the implementation of the character movement

Moreover, the player should be able to look around the environment by using the mouse, which is implemented in a similar way as the previous feature. As it can be seen in figure 27, the appropriate mouse input is retrieved: input on the *x-axis* (for looking left and right) and input on the *y-axis* (for looking up and down). The input is then multiplied by a *sensitivity* variable, which acts similarly to the *speed* variable used in the previous example. The *Time.deltaTime* variable is used once again to ensure the movement is scaled based on the time between frames. Because the player should not be able to rotate more than 90°on the y-axis, as it is humanly impossible, the rotation is clamped using the *Mathf.Clamp* function [48].

```
void FixedUpdate(){
mouseX = Input.GetAxis("MouseX") * sensitivity * Time.deltaTime;
mouseY = Input.GetAxis("MouseY") * sensitivity * Time.deltaTime;

clampedY = - mouseY;
clampedY = Mathf.Clamp(clampedY, -90f, 90f);

transform.localRotation = Quaternion.Euler(clampedY, 0f, 0f);
character.Rotate(Vector3.up * mouseX);
}
```

Figure 27: C# code showcasing the implementation of the LookAround functionality

Since the mouse is used to look around the environment, the cursor should be locked in the scene at the start of the script to avoid accidentally clicking outside of the game screen, as seen in figure 28.

```
void Start(){
Cursor.lockState = CursorLockMode.Locked;
}
```

Figure 28: C# script showing how the cursor is locked in the scene

Because the game constantly switches from walking around the environment to solving the puzzles, which require clicking on different UI elements, cursor unlocking in these situations is also implemented. This is handled by a function in the ManageScenes script (since it should be available across all scenes), which sets the lockState property of the cursor to CursorLockMode.None [49].

In order to access the puzzles and hints hidden in the room, the player needs to be able to detect and interact with the surrounding game objects, depending on whether they contain puzzles or hints, which requires the use of raycasting. **Raycast** [50] is a Unity Physics function that consists of a ray, transmitted from a set origin, which returns a boolean value depending on whether it hits an object or not. If a game object was hit, information about it, such as the distance, position or a reference to its Transform is stored in a Raycast Hit variable [51]. To detect the type of object the player is looking at, the origin of the ray was set to the location of the camera, which is a child of the character object (figure 29). Because the game objects were given appropriate tags, as previously discussed, their information could be accessed and an appropriate action could be triggered.

```
void Update(){
//create the ray
Ray mouseRay = Camera.main.ScreenPointToRay(Input.mousePosition);
//fire the ray
hitData = FireRay(mouseRay);
//get information about the object that was hit
hitPoint = hitData.point;
hitDistance = hitData.distance;
}
```

Figure 29: C# script detailing how a raycast is fired from the position of the camera

Since one of the most important principles of usability of digital applications is **consistency** [52], all interactions with the objects in the rooms are done by looking at them, then using the left mouse click to trigger an action. To make the interactable objects visible to the player, whenever the character is facing such an object, a prompt is displayed giving instructions of how to proceed (figure 30).
After the left click mouse input is detected, depending on the type of object that was detected (given by the tag), a different action is taken. For the objects containing hints or puzzles, the correct scenes linked to them through the RoomPopulation script are loaded. To do this, the **FindIndex** func-

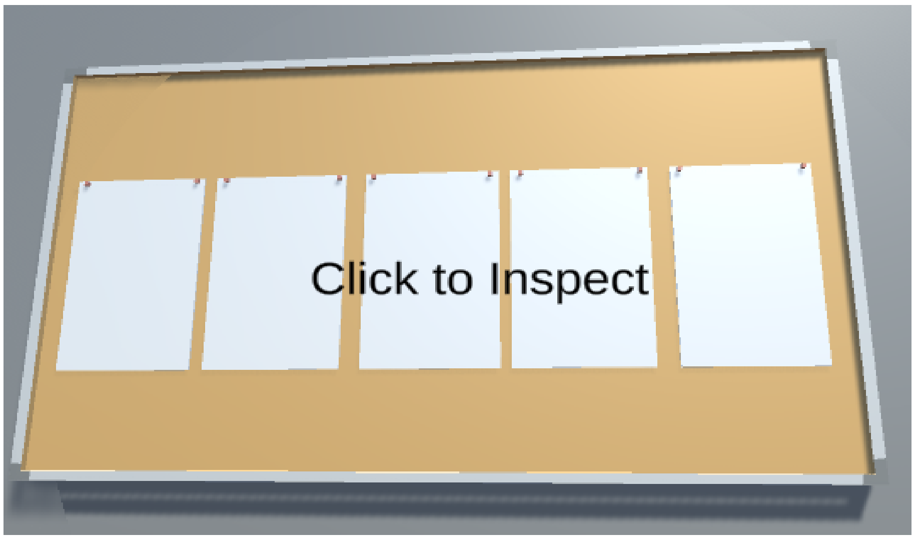


Figure 30: Screenshot from "Project Donut" showing how inspectable objects are highlighted

tion [53] has been used to look into the list of active puzzles (created in the RoomPopulation script previously discussed) and retrieve the GamePuzzle struct that contains the object detected by the ray (figure 31).

```
index = activePuzzles.FindIndex(x => x.pLocation == hitObject);
SceneManager.LoadScene(activePuzzles[index].pContent);
```

Figure 31: C# script showing how the scene corresponding to a puzzle is found, then loaded

5.8 Timer Implementation

As previously mentioned in the Background section, one of the methods of adapting content to make it suitable for all knowledge groups is using a time constraint for the experienced players, technique used by many games to increase the difficulty without having to change the content. Consequently, in the proof of concept game, the player will be allowed the limited time of five minutes to escape a room. This timer gets reset with every room and, unlike in the game "Hades", the remaining time is not added to the next room's limit. This is because the rooms do not increase in difficulty, therefore the conditions of solving the puzzles should stay the same throughout the entire run of the game.

The implementation of the timer for the advanced difficulty level uses the *Time.deltaTime* variable provided by the Unity game engine. In order to make the timer reset on every occasion that the player enters a new room, the script is attached to the *Player prefab.* The total allowed time in a room is assigned to the *remainingTime* variable in the Start() function so that every time the script is run, when the *Player prefab* is loaded into a scene, the timer is reinitialized. In the *Update()* function, the remaining time is checked at every frame and, if there is still time (the value is bigger than 0), the *remainingTime* variable is decreased by **Time.deltaTime**.

Because when the timer runs out a **"Game Over!"** should be triggered, the script uses a *timerRunning* boolean variable that is initialised with the value "true" in the Start() function, its value being changed to "false" when the timer runs out (remainingTime is less than 0). Moreover, because the timer only applies to the advanced difficulty, this timerRunning variable is set to "false" for any other difficulty. By handling this in the Start() function, whenever the player opens the options menu to change the difficulty, the value of the timerRunning variable is changes, making the effects of the user's action immediately visible.

To display the remaining time of the screen, the value in remainingTime needs to be converted into **"minutes:seconds"** format. This is achieved through a **DisplayTime()** function, called after the time decrements in the Update() function. As it can be seen in figure NO, the minutes are calculated by dividing the time with 60 and the seconds are calculated from the remainder of this division. The values are then displayed using a TextMeshPro game object, specifying the formatting with the **String.Format** method from the System library [54].

5.9 Title Screen and Options Menu

The title screen is an important feature of every game. As it can be seen in figure 32, the title screen for "Project Donut" gives the player the following options: play the game, access the options menu, challenge mode (which is a feature to be added in the future) and leave the game.

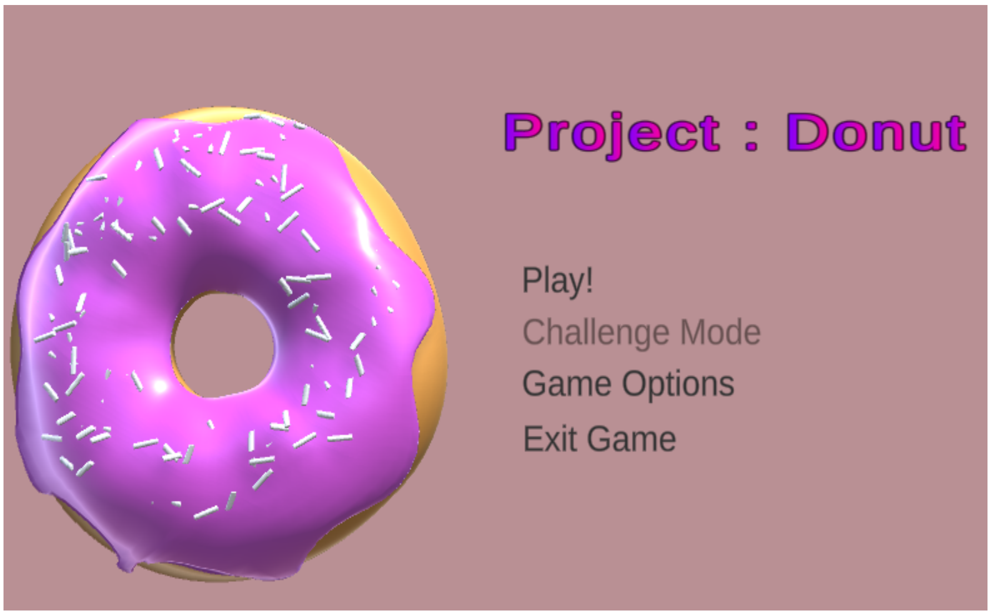


Figure 32: Title screen example

The options menu contains functionality for increasing or decreasing the volume of the sound effects in the game, changing the difficulty and resetting the progress. The options menu can also be accessed while playing the game to allow for the difficulty to be changed at any time, with modifications immediately being reflected. The menu can be accessed by pressing the escape key, also offering a "Back to title" option when accessed in this way, giving the player a means to exit the application.

5.10 Drag and Drop Puzzle Mechanics

As previously mentioned in the Project Design section, the drag and drop puzzle mechanics involve clicking on an object and moving it to the desired place. To implement this, two separate scripts were used:

- the *Draggable* script, for moving an object around

- the *DropLocation* script for the locations the object can be placed in

To detect when the user clicks on an UI elements and begins the drag action, the built-in Unity function *"OnBeginDrag()"* [55] was used in the *Draggable* script. This method is part of the *"IBeginDragHandler"* interface and receives a *"PointerEventData"* parameter that contains information about the position of the cursor when the user clicked the left mouse button. Inside this function, as it can be seen in figure 33, the current parent of the clicked object is saved in a variable called *parentAfterDrag*. The current parent needs to be retrieved because, if the drag action is cancelled or the drop location is invalid, the object needs to return to its original location.

```
 public void OnBeginDrag(PointerEventData eventData)
{
        parentAfterDrag = transform.parent;
}
```

Figure 33: C# code showing how the current parent is retrieved

In order to make the object follow the cursor while it is being dragged, its position is updated to that of the cursor in the built-in *"OnDrag()"* function [56], which is called every time the cursor moves during the dragging action (figure 34).

```
void OnDrag(PointerEventData eventData)
{
        transform.position = Input.mousePosition;
}
```

Figure 34: C# code showing how the position of the object is changed to that of the cursor

When the drag action ends, the UI element needs to be positioned in the new location if it is valid, otherwise returned to the original position. To this end, inside the built-in *"OnEndDrag()"* function [57], which is called when the drag action ends, the parent of the element is updated to the *parentAfterDrag* variable (figure 35).

```
 public void OnEndDrag(PointerEventData eventData)
 {
        transform.SetParent(parentAfterDrag);
}
```

In order for the *parentAfterDrag* variable to be updated to the drop location, if it is valid, its value is modified inside the *DropLocation* script, by using the *"OnDrop()"* function [58] (figure 36). This method is called when an element is dropped on an object which was set up to support the Drag and Drop action.

```
public void OnDrop(PointerEventData eventData)
{
        draggedItem.parentAfterDrag = transform;
}
```

Figure 35: C# code showing how the parent of the dragged object is changed after the dragging action

To make the UI elements be automatically arranged when they are dropped on a valid location, the *grid layout group component* [59] has been attached to such elements. This component transforms the game object into a grid of a specified size and ensures that all child element are arranged in order, depending on the selected features. This feature is highly customisable, including options such as padding, alignment, start corner and orientation.

5.11 Random Logic Expression Algorithm

The Random Logic Expression Algorithm is an implementation of the idea of generating content based on the selected difficulty, one of the main methods of content adaptation explored in this project. Traditionally, in order to create a challenge based on solving a logic expression, the contents had to be manually created for each instance of the puzzle for every target knowledge group. This process is repetitive, time-consuming and inefficient since the puzzles created for one category of users may prove unsuitable for another. The Random Logic Expression Algorithm provides a way to automate this operation, as well as a method for computing the correct answer to a given question. Therefore, its implementation can be divided in two stages: **generation** and **validation** of the logic expression.

5.11.1 Generation

In order to take into account the selected difficulty when randomly generating the logic expression, its length will depend on the target knowledge group. This is because, with a longer, more complex logic expression, more than one operators may be present and the player will be required to account for the order of operations while solving the question. Moreover, with the time constraints applied for the advanced difficulty, the user has to be more familiar with the truth tables of the different operators involved to be able to solve the question quicker. By considering the most simple example of a logic expression (two variables connected by a logic operator), the length of such a formula can be defined as the number of logic operators it contains (figure 37). Therefore, depending on the selected difficulty, the length of the expressions generated will be:

- between 1 and 3 for beginner,
- between 3 and 5 for intermediate,
- between 4 and 6 for advanced.

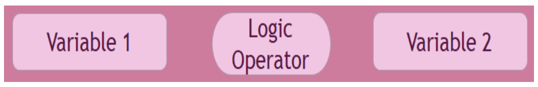


Figure 36: Structure of a simple logic expression

The algorithm uses a recursive function, *Generate()*, which takes four parameters: **l** (the current length of the expression), **L** (the desired length), **operators**[] (an array of the logic operators to be used) and **variables**[] (an array of variables to be used in the expression). The last two parameters are used to allow for easy customisation of the function if new symbols or variables are to be introduced. At the beginning of each call to the *Generate()* function, the elements of the expression are randomly chosen from the provided arrays, by following the structure previously described, and the length of the expression (l) is increased (figure 37).

```
var1 = variables[Random.Range(0, variables.Length)];
var2 = variables[Random.Range(0, variables.Length)];
op = operators[Random.Range(0, operators.Length)];
l++;
```

Figure 37: C# code showing how the variables and operator are randomly chosen

Afterwards, if the current length l is smaller than the desired length L, one of the variables in the expression is randomly chosen for expansion. The expansion process consists of recursively calling the *Generate()* function and replacing the value of the chosen variable with the result (figure 38).

Figure 38: C# script showing a recursive call to the *Generate()* function

Outside of the previously described if statement, the algorithm contains code for randomly choosing whether to negate a variable or not, in order to introduce more complexity to the expression. This is achieved through a function called *Decision()* which returns a random boolean value, by using the built-in *Random.Range* function. In order to actually construct the logic expression, the *var1*, *var2* and *op* strings are concatenated and, to aid in the evaluation algorithm, are separated by a *","* character. When displaying the expression in the game, this character can be omitted by using the built-in *Replace()* function [60] for strings (figure 39).

```
displayExpression.text = expression.Replace(",", "");
```

Figure 39: Code showing how the *Replace* function is used

5.11.2 Validation

The validation algorithm is used to compute the result of a logic expression with given values for the variables used. The implementation of this algorithm consists of **two stacks** [61]: one for *values* and one for *operators*. The stack is a data structure which uses the **"last-in-first-out"** principle to store its elements, offering the following operations:

- **Push**: adds an element to the top of the stack
- **Pop**: returns and removes the element on top of the stack
- **Peek**: returns the element on top of the stack, without removing it
- **Size**: returns the number of elements in the stack
- **isEmpty**: checks if the stack is empty

In order to simplify the process of constructing the stacks from a given expression, its components are separated by the *","* character. By using the built-in *Split()* function [62] for strings, all elements of the formula can be retrieved in an array (figure 40). The values in the array are then checked and pushed to their respective stacks.

```
string[] components = expression.Split(",");
```

Figure 40: Code showing how the *Split()* function is used

Similarly to the generation function, the evaluation function is also called recursively, in order to account for the presence of parentheses, case in which the expression inside needs to be evaluated first. The function takes only two parameters, the *values* and *operators* stack, and does not return anything. This is because the result of the evaluation will be the only element remaining in the values stack after the function finishes running.

The algorithm consists of a do while loop with the exit condition being the operators stack being empty (as there would be no more operations to perform) or a closing parentheses being encountered. At the start of each loop, the element on top of the operators stack is retrieved and an action is taken depending on its value.

If the operator is a negation, the algorithm peeks in the operators stack, checking whether an open parenthesis is at the top. This helps identify if the negation applies to a variable, case in which the element on top of the values stack is retrieved, negated and pushed back to the *values* stack. On the other hand, if, by using the **Peek()** function on the operators stack, an open parenthesis is discovered, it is removed from the stack, the *Evaluate()* function is recursively called, then the element on top of the values stack is retrieved, its value is negated and added back to the *values* stack (figure 41).

```
operations.Pop();
//call function recurssively
EvaluateExpression(ref operations, ref values);
//negate result
 var1 = (bool)values.Pop();
values.Push(!var1);
```

Figure 41: Section of the code to compute the result of a logic expression

When the element on top of the operators stack is a normal logic operator, such as **AND** or **OR**, the algorithm peeks in the operators stack searching for an open parenthesis, case in which the element from the top of the values stack is retrieved and the Evaluate() function is recursively called. If there are nor parentheses detected in the *operators* stack, the algorithm retrieves two elements from the *values* stack and applies the logic operator, pushing the result back to the *values* stack.
In this way, when the algorithm finishes running, there will be no elements in the *operators* stack and only one element in the *values* stack representing the result of the logic expression. This can then be compared to input from the user and suitable actions can be taken.

5.12 Candidates for Logical Equivalence Generation

Another short algorithm that was implemented in the game to showcase the concept of content generation based on the chosen difficulty is the Candidates for Logical Equivalence algorithm. This algorithm generates a random conditional expression, which could be found inside an if statement or a while loop and, based on the selected difficulty, generates a varying number of candidate expressions. The player's task is to identify which of these expressions is equivalent to the given one. Even though this algorithm is not as efficient and smart as the previous one, it is still an useful tool to reduce the number of separate puzzles that would have to be manually implemented without its help.

6 Testing

To ensure a project is functional and meets all requirements, user testing is an important stage in the development process. Initially, the project followed an iterative development strategy, with two testing phases being planned. However, the development encountered significant delays, therefore, the plan had to be adjusted accordingly. As a result, after the implementation of each type of puzzle, user testing was performed with a small group of participants from the Warwick Game Design society. During this process, they were asked to rate the difficulty of the puzzles and, considering their experience in the presented educational topics, the puzzles were divided into the beginner and advanced categories used in the randomisation algorithm. Moreover, the feedback from the participants was useful in making some changes in the user interface. For example, before a puzzle can be accessed, instructions are provided to help define the type of input it requires (figure 42). **ADD SS OF INSTRUCTIONS**

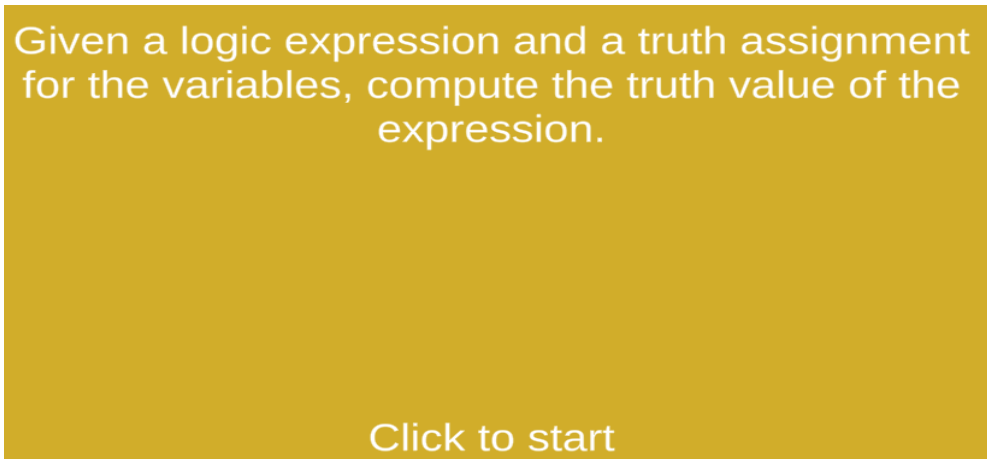

Given a logic expression and a truth assignment for the variables, compute the truth value of the expression.

Figure 42: Example of instructions

7 Evaluation

7.1 Project Execution

The Gantt charts in figure NO shows the development timeline of the project. As it can be observed, against the initial plan, there was a sig-

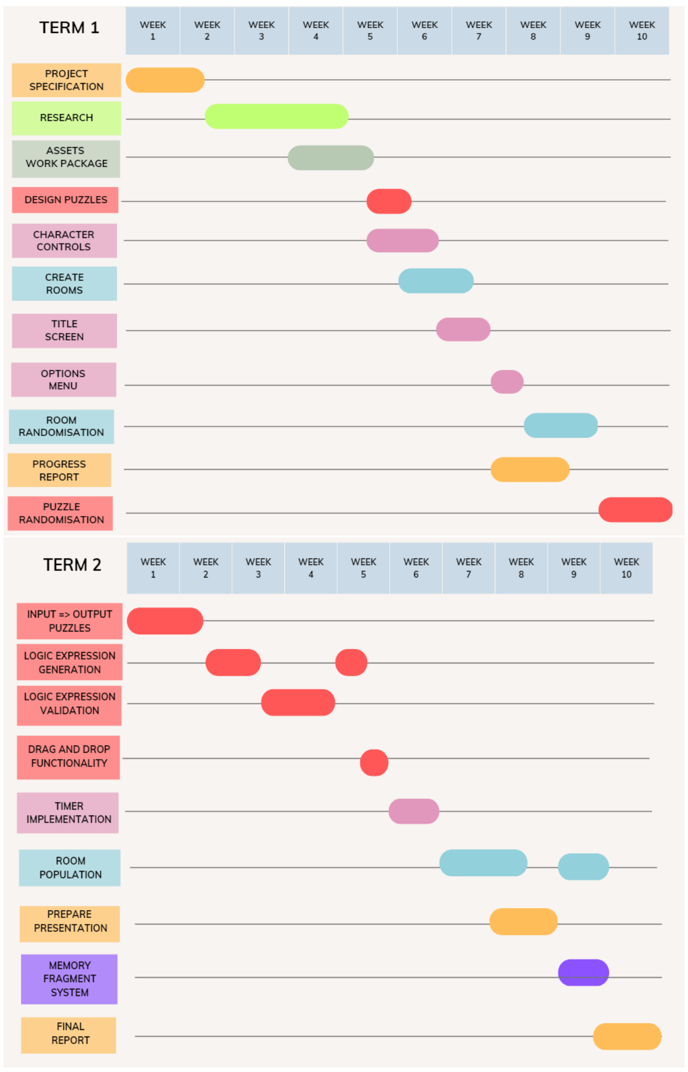


Figure 43: Gantt charts showing the project timeline

nificant delay in implementing features from the *Puzzle* and *Room* work packages. This was partly due to wrong initial estimation of the time required to develop the features, but also some technical issues that arose due to inexperience with using the Unity game engine.

The delay in implementing the puzzles resulted from an overambitious initial scope focusing on too many educational topics to include. While the original plan assumed only some puzzle components would be randomised, this would not fit the concept of reuse and would require development of many separate puzzles. Therefore, the objectives have been adapted to fit better with content reuse across knowledge groups and more effort was put into developing efficient ways of generating and randomising content than creating puzzles for all the proposed topics.

The agile development methodology allowed for the plan to be altered in accordance with these changes without affecting most of the core features of the game. Therefore, the main unaccomplished objectives are the ones which have been assigned *"Could"* or *"Won't"* during the MoSCoW analysis. The only objectives assigned a higher priority value that have been affected are those in relation to the "Memory Fragment System" work package. Even though the robot design was created and functionality has been added to spawn the object randomly as previously mentioned, no higher difficulty puzzles have been implemented. To this end, some of the already existing content that can adapt to the difficulties (such as the random logic expression generator or the candidates for logical equivalence algorithm, could be used as puzzles for the memory fragment robots.

7.2 Author's Assessment of the Project

As mentioned in the Background section, the concept of adapting the same content to be suitable for all knowledge groups is new in the educational games field, with little existing literature attempting to achieve this. Therefore, the main contribution of this project is to fill in this gap, through proposing and implementing four methods usually encountered in informal interactive experiences.

In addition, the proof of concept game that was developed is highly customizable, mainly due to how the rooms and puzzles were implemented. As a result, there is potential for other interactive experiences to be built on top of the existing source code, without many compatibility issues.

With regard to the technical difficulty of this project, it can be seen from the Implementation section that many different features had to be implemented in order for the game to be fully functional and achieve its objectives. To accomplish this, appropriate data structures had to be used and much time had to be spent researching built-in functions that could help make the methods more efficient. Moreover, the code went through many modifications, in order to make it more efficient and simple to understand.

A big shortcoming of this project is the low number of puzzles that were implemented. Because of the time constraint, large scope and unforeseen personal problems, priority had to be given to the actual technically challenging components of the game, instead of simple puzzle content creation. This limitation however, motivated finding efficient ways to reuse the little content that was created, such as the Random Logic Expression Generator, which resulted in interesting features that were not initially in the scope to be developed. Therefore, even though the game does not have as much educational value as intended, the generation of content is done more intelligently and displays more adaptation based on difficulty as initially planned.

7.3 Areas for Future Work

As previously mentioned, because of the significant delays caused by the puzzle implementation, potential to automatically generate content based on the selected difficulty was discovered. Aside from the generation of logic expressions (which was completed), random generation of programming questions was also attempted. This however, proved too complex to implement in the limited time remaining, but could be an interesting feature to work on in the future.

In addition, the project could also benefit from the implementation of some of the extensions that were assigned "Could" or "Won't" after the MoSCoW analysis. Particularly, the addition of puzzles that would affect the environment could be an interesting feature that would enhance the user experience and provide a more practical method of learning.

Another interesting technical challenge, briefly mentioned in the Background section, is the use of machine learning techniques for dynamically adapting the difficulty to the user's skills and needs. This could result in a more tailored learning experience, increasing the educational value of the game.

8 Conclusions

As previously mentioned, there is currently a gap in the research of content adaptation techniques in an educational context. This project has successfully implemented four such methods through the use of a digital escape room-style interactive experience, a type of system which has seen a huge rise in popularity since the COVID-19 pandemic. Therefore, it is strongly believed that the project holds significant value in its potential to be used for building further more efficient systems, as mentioned in the Future Work section.

Even though development encountered several significant delays, all high priority objectives have been achieved and agile methodology features were efficiently used to adapt the plan to any changes. As a result, while there is a lack of puzzles and not all intended Computer Science topics have been explored completely, more technically challenging features were implemented, which relate more closely to the main goals of this project: content adaptation to maximise reuse across knowledge groups.

Moreover, the project provided a good opportunity for the author to become familiar with the Unity game engine and other resources used in the game design industry. As the project required the implementation of complex features, extensive research of existing libraries and tools had to be performed to ensure the completion of all core objectives.

9 Legal, Social, Ethical and Professional Issues

During the testing phase, participants were asked to give feedback on their experience with the game. Since all questions related strictly to the contents of the interactive experience and no personal information was required, there are no legal, social, ethical or professional concerns.

References

[1] Miguel de Aguilera and Alfonso Mendiz. “Video games and education: (Education in the Face of a “Parallel School”)”. In: *Computers in Entertainment* (2003). DOI: 10.1145/950566.950583. URL: https://doi.org/10.1145/950566.950583.

[2] Monika Simkova. “Using of Computer Games in Supporting Education”. In: *Procedia - Social and Behavioral Sciences* 141 (2014), pp. 1224–1227. DOI: 10.1016/j.sbspro.2014.05.210. URL: https://www.sciencedirect.com/science/article/pii/S1877042814036349.

[3] James Paul Gee. “What video games have to teach us about learning and literacy”. In: *Computers in Entertainment* (2003). DOI: 10.1145/950566.950595. URL: https://doi.org/10.1145/950566.950595.

[4] Risyatun Naziah et al. “The Effects of Contextual Learning and Teacher’s Work Spirit on Learning Motivation and Its Impact on Affective Learning Outcomes”. In: *Journal of Educational Sciences* (2020). DOI: 10.31258/jes.4.1.p.30-43. URL: https://jes.ejournal.unri.ac.id/index.php/JES/article/view/7930.

[5] Vincent Aleven et al. “Toward a Framework for the Analysis and Design of Educational Games”. In: *2010 Third IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning.* 2010, pp. 69–76. DOI: 10.1109/DIGI.2010.55.

[6] Panagiotis Fotaris and Theodoros Mastoras. “Escape rooms for learning: A systematic review”. In: *Proceedings of the European Conference on Games Based Learning.* 2019, p. 236.

[7] Markus Wiemker, Errol Elumir, and Adam Clare. “Escape room games: ”Can you transform an unpleasant situation into a pleasant one?””. In: *Game based learning* 55 (2015), pp. 55–75.

[8] Chantal Lathwesen and Nadja Belova. “Escape Rooms in STEM Teaching and Learning—Prospective Field or Declining Trend? A Literature Review”. In: *Education Sciences* 11 (2021). DOI: 10.3390/educsci11060308. URL: https://www.mdpi.com/2227-7102/11/6/308.

[9] Nicolas Dietrich. “Escape Classroom: The Leblanc Process—An Educational “Escape Game””. In: *Journal of Chemical Education* (2018). DOI: 10.1021/acs.jchemed.7b00690. URL: https://pubs.acs.org/doi/10.1021/acs.jchemed.7b00690.

[10] Sonsoles López-Pernas et al. “Examining the Use of an Educational Escape Room for Teaching Programming in a Higher Education Setting”. In: *IEEE Access* (2019), pp. 31723–31737. DOI: 10.1109/ACCESS.2019.2902976.

[11] Olaf H. Graven and Lachlan MacKinnon. *Using Games Technology to Develop Reusable Virtual Learning Environments-an Embedded Approach.* June 2012. URL: http://0-search-proquest-com.pugwash.lib.warwick.ac.uk/conference-papers-proceedings/using-games-technology-develop-reusable-virtual/docview/1326324789/se-2.

[12] *Breakout EDU - Educational Games.* URL: https://www.breakoutedu.com/ (visited on 05/01/2023).

[13] Agoritsa Makri, Dimitrios Vlachopoulos, and Richard A. Martina. “Digital Escape Rooms as Innovative Pedagogical Tools in Education: A Systematic Literature Review”. In: *Sustainability* (2021), p. 4587. DOI: 10.3390/su13084587. URL: https://www.mdpi.com/2071-1050/13/8/4587.

[14] Rory McGreal. “Learning objects: A practical definition”. In: (2004). URL: https://auspace.athabascau.ca/handle/2149/227.

[15] Eli Cohen and Malgorzata Nycz. “Learning Objects and E-Learning: an Informing Science Perspective”. In: *Interdisciplinary Journal of E-Learning and Learning Objects* 2.1 (2006), pp. 23–34. URL: https://www.learntechlib.org/p/44811/.

[16] Zameer Gulzar and A Anny Leema. “Sharablecontent Object Reference Model: An Overview”. In: *National Conference on Computing Technologies Todays and Beyond (NCCTTB’15).* 2015, pp. 71–75.

[17] Phil Barker. “What is IEEE learning object metadata/IMS learning resource metadata”. In: *CETIS Standards Briefing Series, JISC (Joint Information Systems Committee of the Universities’ Funding Councils)* (2005).

[18] Patrick E. Parrish. “The trouble with learning objects”. In: *Educational Technology Research and Development* 52.1 (2004), pp. 49–67. DOI: 10.1007/BF02504772. URL: https://doi.org/10.1007/BF02504772.

[19] Carlo Fabricatore. “Gameplay and Game Mechanics: A Key to Quality in Videogames”. In: 2007. URL: http://www.oecd.org/dataoecd/44/17/39414829.pdf.

[20] Sertaç Ögüt and Barbaros Bostan. “Game challenges and difficulty levels: lessons learned From RPGs”. In: *International Simulation and Gaming ...* (2009). URL: https://www.academia.edu/795475/Game_challenges_and_difficulty_levels_lessons_learned_From_RPGs.

[21] *Flexbox Froggy.* URL: https://flexboxfroggy.com (visited on 04/24/2023).

[22] *Supergiant Games.* URL: https://www.supergiantgames.com/games/hades/ (visited on 04/24/2023).

[23] *Supergiant Games.* URL: https://www.supergiantgames.com/ (visited on 05/01/2023).

[24] Glen Berseth et al. “Characterizing and optimizing game level difficulty”. In: *Proceedings of the 7th International Conference on Motion in Games.* Association for Computing Machinery, 2014, pp. 153–160. DOI: 10.1145/2668064.2668100. URL: https://doi.org/10.1145/2668064.2668100.

[25] *Spiritfarer.* URL: https://thunderlotusgames.com/spiritfarer/ (visited on 05/01/2023).

[26] *Thunder Lotus Games — Montreal Game Developers with an Indie Heart.* URL: https://thunderlotusgames.com/ (visited on 05/01/2023).

[27] Jeffrey Parkin. *Hades guide: God Mode, explained.* 2020. URL: https://www.polygon.com/hades-guide/21456205/god-mode-difficulty-easy-damage-reduction (visited on 05/01/2023).

[28] Miguel Gonzalez-Duque, Rasmus Berg Palm, and Sebastian Risi. "Fast Game Content Adaptation Through Bayesian-based Player Modelling". In: *2021 IEEE Conference on Games (CoG).* 2021, pp. 01–08. DOI: 10.1109/CoG52621.2021.9619018.

[29] Alex Grasas et al. "Biased randomization of heuristics using skewed probability distributions: A survey and some applications". In: *Computers & Industrial Engineering* 110 (2017), pp. 216–228. DOI: 10.1016/j.cie.2017.06.019. URL: https://www.sciencedirect.com/science/article/pii/S036083521730270X.

[30] *What is Scrum?* URL: https://www.scrum.org/resources/what-scrum-module (visited on 04/25/2023).

[31] *MoSCoW Prioritization.* URL: https://www.productplan.com/glossary/moscow-prioritization/ (visited on 04/25/2023).

[32] D. H. Stamatis. *Failure Mode and Effect Analysis.* Quality Press, 2003.

[33] *Unity Asset Store - The Best Assets for Game Making.* URL: https://assetstore.unity.com/ (visited on 04/26/2023).

[34] *blender.org - Home of the Blender project - Free and Open 3D Creation Software.* URL: https://www.blender.org/ (visited on 04/26/2023).

[35] *Unity Real-Time Development Platform — 3D, 2D, VR & AR Engine.* URL: https://unity.com (visited on 04/26/2023).

[36] Unity Technologies. *Create with Code.* URL: https://learn.unity.com/course/create-with-code (visited on 04/26/2023).

[37] *Build Virtual Worlds From Scratch — Unity ProBuilder.* URL: https://unity.com/features/probuilder (visited on 04/26/2023).

[38] *Mirror Modifier — Blender Manual.* URL: https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/mirror.html (visited on 04/26/2023).

[39] *Blender Beginner Donut Tutorial - YouTube.* URL: https://www.youtube.com/playlist?list=PLjEaoINr3zgFX8ZsChQVQsuDSjEqdWMAD (visited on 04/26/2023).

[40] *Geometry Nodes — Blender Manual.* URL: https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/index.html (visited on 04/26/2023).

[41] *Weight Paint — Blender Manual.* URL: https://docs.blender.org/manual/en/latest/sculpt_paint/weight_paint/index.html (visited on 04/26/2023).

[42] Unity Technologies. *Flip Normals — ProBuilder — 4.1.2.* URL: https://docs.unity3d.com/Packages/com.unity.probuilder@4.1/manual/Object_FlipNormals.html#:~:text=The%20Flip%20Normals%20tool%20flips,shape%20into%20an%20interior%20space. (visited on 04/26/2023).

[43] *Extrude Edges — ProBuilder — 4.3.1.* URL: https://docs.unity3d.com/Packages/com.unity.probuilder@4.3/manual/Edge_Extrude.html (visited on 04/29/2023).

[44] Robert F. Bruner. *Repetition is the First Principle of All Learning.* 2001. URL: https://papers.ssrn.com/abstract=224340 (visited on 04/27/2023).

[45] Unity Technologies. *Unity - Manual: Tags.* URL: https://docs.unity3d.com/Manual/Tags.html (visited on 04/29/2023).

[46] Unity Technologies. *Unity - Scripting API: Random.Range.* URL: https://docs.unity3d.com/ScriptReference/Random.Range.html (visited on 04/29/2023).

[47] Unity Technologies. *Unity - Scripting API: Time.deltaTime.* URL: https://docs.unity3d.com/ScriptReference/Time-deltaTime.html (visited on 04/29/2023).

[48] Unity Technologies. *Unity - Scripting API: Mathf.Clamp.* URL: https://docs.unity3d.com/ScriptReference/Mathf.Clamp.html (visited on 04/29/2023).

[49] Unity Technologies. *Unity - Scripting API: CursorLockMode.* URL: https://docs.unity3d.com/ScriptReference/CursorLockMode.html (visited on 04/29/2023).

[50] Unity Technologies. *Unity - Scripting API: Physics.Raycast.* URL: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html (visited on 04/26/2023).

[51] John French. *Raycasts in Unity, made easy.* 2021. URL: https://gamedevbeginner.com/raycasts-in-unity-made-easy/ (visited on 04/26/2023).

[52] Tania Schlatter and Deborah Levinson. *Visual Usability: Principles and Practices for Designing Digital Applications.* 2013.

[53] *List¡T¿.FindIndex Method (System.Collections.Generic).* URL: https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.findindex?view=net-8.0 (visited on 04/27/2023).

[54] *String.Format Method (System).* URL: https://learn.microsoft.com/en-us/dotnet/api/system.string.format?view=net-8.0 (visited on 04/29/2023).

[55] Unity Technologies. *Unity - Scripting API: EventSystems.IBeginDragHandler.OnBeginDra* URL: https://docs.unity3d.com/2018.1/Documentation/ScriptReference/EventSystems.IBeginDragHandler.OnBeginDrag.html (visited on 04/30/2023).

[56] Unity Technologies. *Unity - Scripting API: EventSystems.EventTrigger.OnDrag.* URL: https://docs.unity3d.com/2018.2/Documentation/ScriptReference/EventSystems.EventTrigger.OnDrag.html (visited on 04/30/2023).

[57] Unity Technologies. *Unity - Scripting API: EventSystems.IEndDragHandler.OnEndDrag.* URL: https://docs.unity3d.com/2018.2/Documentation/ScriptReference/EventSystems.IEndDragHandler.OnEndDrag.html (visited on 04/30/2023).

[58] Unity Technologies. *Unity - Scripting API: EventSystems.EventTrigger.OnDrop.* URL: https://docs.unity3d.com/2018.2/Documentation/ScriptReference/EventSystems.EventTrigger.OnDrop.html (visited on 04/30/2023).

[59] *Grid Layout Group — Unity UI — 1.0.0.* URL: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-GridLayoutGroup.html (visited on 04/26/2023).

[60] *String.Replace Method (System)*. URL: `https://learn.microsoft.com/en-us/dotnet/api/system.string.replace?view=net-8.0` (visited on 04/30/2023).

[61] *C# Stack¡T¿*. URL: `https://www.tutorialsteacher.com/csharp/csharp-stack` (visited on 04/30/2023).

[62] *String.Split Method (System)*. URL: `https://learn.microsoft.com/en-us/dotnet/api/system.string.split?view=net-8.0` (visited on 04/30/2023).

Appendices

A Functional Requirements

Requirement	Priority
The player will be able to select a difficulty level from the provided list: beginner, intermediate and advanced	MUST
The player will be able to move around the world from the perspective of a controllable character	MUST
It will be possible to change the difficulty at any time	SHOULD
The difficulty will adapt as the player gains experience	MUST
Beginner level players will be able to find hints around the rooms to aid them in solving the puzzles	MUST
Advanced level players will have a time limit to solve the puzzles	MUST
The players will be able to see a countdown of how much time they have left	SHOULD
Minimalistic sound effects will be used to make the experience more immersive	SHOULD
The player will be able to complete a run of the game by escaping from 3 to 5 rooms	MUST
The player will be able to obtain memory fragments from completing tasks given by some robots	SHOULD
The robots will pe spawned after a random number of successful escapes	SHOULD
The player will be able to view their memory fragments by using the TV in the main room	SHOULD
Each room will contain 2 to 5 puzzles that require computer science knowledge	MUST
The number of puzzles in a room will depend on the selected difficulty	MUST
Each of the puzzles will have elements that can be randomly generated to ensure the chances of encountering the same puzzle twice are as low as possible	MUST
The content of some of the puzzles will be automatically generated based on the selected difficulty	MUST
The rooms will contain a few basic escape-room style puzzles	COULD
The objects containing puzzles which the player can interact with will be highlighted	SHOULD
The player will be able to pick up different objects in the room	WON'T
The objects which the player is able to pick up will be highlighted	WON'T
The player will be able to access an inventory consisting of objects found in the room	WON'T
The objects in the inventory can be used to solve some of the puzzles	WON'T
After a player encounters a certain room, the chance of it reappearing will be lowered	MUST
The player will be able to modify the volume of the music and sound effects playing in the background	SHOULD
The game will have a title screen offering a few options: load game, new game, set difficulty, game options	MUST
The player will be able to reset their progress	COULD

B Non-functional Requirements

Requirement	Priority
The game will be entertaining	MUST
Users with no previous computer science knowledge will be able to play the game	MUST
The game will be challenging for users with previous computer science knowledge	MUST
The storyline will be engaging	SHOULD

C Extensions

Requirement	Priority
More intricate sound effects could be used to improve the atmosphere of the game	COULD
New computer science topics could be added	WON'T
The 3D assets could be improved to make the experience more immersive	COULD
A glitch effect could be applied to all rooms and, as the story progresses, it could slowly be removed	WON'T
The TV display could be changed to the last memory fragment earned when the player is in the main room	COULD
The player could have a notebook in which to add screenshots of the different hints in the game	COULD
A challenge mode where the player would have to escape as many rooms as possible in a set time	WON'T