Visualising Volume: Development of an app to help primary-aged children visualise volume of 3D shapes and learn by interaction

Emer Sweeney

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

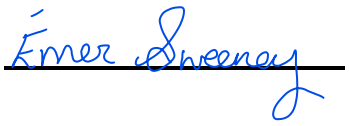
Thursday December 16th 2021

Abstract

Apps- and games-based learning is becoming an increasingly popular method of learning for young people as they adapt in a world heavily centred around technology. Visualising Volume is an application that seeks to teach young students (5-7 years old) concepts related to volume of 3D shapes by encouraging them to interact with shapes in games and experiment to learn. This project followed a Scrum framework to achieve design, testing and delivery of the Visualising Volume mobile application. Evaluation of the application demonstrated that, though there is room for improvement, the application behaves as intended and shows potential as a useful learning tool.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: Emer Sweeney Signature: 

Acknowledgements

Thank you, Chigozie Onyekaba, for being a patient and helpful guide throughout this project, and for always being willing to go beyond your obligations to provide support.

Thank you, also, Elise Forsyth, Marilena Charalambidou, Ewa Wanat, Karim Altom and Charlie Dexter. You have all made this a lovely year for me and eased the difficulties of learning in the unorthodox environment created by the pandemic.

Contents

Chapter 1

Introduction

1.1 Project Objectives

This section discusses the objectives of this project with consideration for the time scale, the target audience of the application and the intended features of the application. The main goal of this project was to develop, and document the development of, an iOS mobile application called Visualising Volume that helps young primary-aged children (5-7 years old) understand volume by visualisation; specifically, to help children understand (1) the connection between dimensions of 3D shapes and their capacity and (2) how volumes of different everyday objects can be compared.

To achieve specification (1), Type 1 games of the application enable users to make changes to shape dimensions and view the effect that has on volume. To visualise how the volume is affected, the 3D shapes have water stored inside them. The water level adjusts to fit the new shape dimensions, allowing users to see how the capacity of the shape is affected by the dimension change. To achieve specification (2), Type 2 games of the application enable users to pour water from everyday objects into a different object to work out how the volumes of the two objects compare.

The following objectives encapsulate the aims of the project and account for the needs specific to the target user group.

1. Develop an application that contributes to users' understanding of volume of various 3D shapes and objects, as described above.
2. Provide an attractive user interface that appeals to target user group (primary school children). In particular, provide a clean, colourful and simple user interface.
3. Provide an easy-to-use and accessible user interface that is easily learnable by target user group. In particular, provide a user interface with: buttons labelled intuitively and simply, with as few words as possible whilst maintaining clarity; clickable/draggable items clearly presented as clickable/draggable; as few actions as possible whilst providing the desired functionality.
4. Make the application 'classroom-friendly' by providing a means for a teacher to create student accounts and view student progress.

5. Design the application in such a way that code is very reusable and allows easy addition of similar games using existing code.
6. Ensure the application displays on all target devices correctly and is ready for deployment to iOS mobile devices.
7. Clearly lay out the project background, requirements and features in the report.
8. Effectively communicate in the report the software design principles employed in development of the application.
9. Conduct useful testing of the application to determine to what extent functional aims have been achieved and to what extent the application is valuable as a learning tool.
10. Evaluate, reflect on and discuss testing results and consider any changes that would be of value for future work on this application.
11. Organise and manage the project effectively, following a Scrum framework. This is a software development framework, part of the Agile family of frameworks and principles. It focuses on self-organisation of teams and consists of artefacts and events that help the team focus on one work iteration at a time - a 'sprint' - to help meet the product goal [14]. This objective is broken down as follows:
    - 11.1. Manage time by planning time allocations for sprints as a whole and user stories within sprints. User stories are functions of the system deemed required by considering the user's journey when using the system.
    - 11.2. Use a project management tool to organise sprints and store details on user stories such as acceptance tests and costs. Acceptance tests are critera that user stories must pass to be deemed done, and costs are a quantification of the effort and time required to complete a user story,
    - 11.3. Regularly update project supervisor in a professional manner and seek advice when difficulties arise.
    - 11.4. Document the process regularly so the report is a true reflection of the project journey.
    - 11.5. Begin working days with a brief independent meeting to consider plan for the day and what should be accomplished.
    - 11.6. Review sprints as they end to evaluate progress and make any changes for improved work in following sprints.

Please see **Appendix A** for User Stories.

1.2 Report Structure

This section discusses the structure of this report which aims to take the reader through the project process. It documents the design and development of the application, providing details and justifications of design decisions and evaluating the software and work process.

Chapter 2: Background describes the context in which this project sits, discussing in particular the effectiveness of app- and games-based learning and the importance of teaching volume to young

children. It also provides a brief overview and analysis of existing software similar to the product in this project.

Chapter 3: Product Requirements and Features describes the functional and non-functional requirements of the software product, which informed a list of prioritised features, also provided in this section. It informs the reader of the tools and technologies employed to meet these requirements and provides a justification.

Chapter 4: Design describes the system architecture and design of the software.

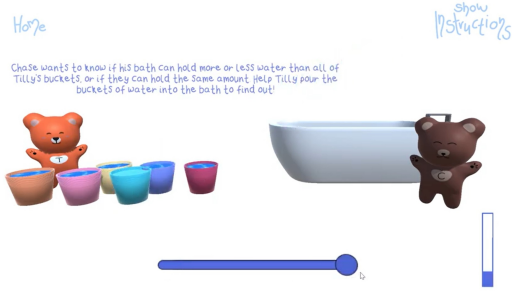
Chapter 5: Testing and Evaluation describes the testing and evaluation strategies for this project and their results.

Chapter 6: Conclusion concludes this report by reflecting on the strengths and weaknesses of the project, and provides considerations for further work.

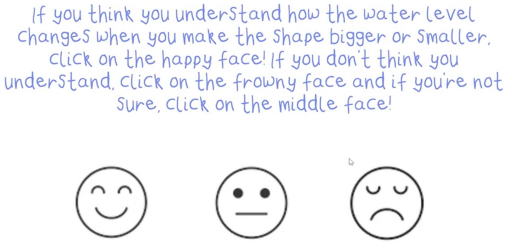
1.3 The Application

Figure 1.1 shows some screenshots from the application, particularly screenshots demonstrating each game type and screenshots demonstrating the student feedback feature, which the teacher can view. A link to a video demonstration of the application is available in **Appendix B**.

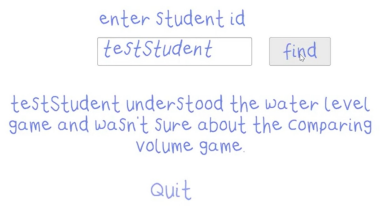
(a) Type 1 Game



(b) Type 2 Game



(c) Feedback Screen



(d) Teacher's Progress Check Screen

Figure 1.1: Application Screenshots

Chapter 2

Background

2.1 Importance of Spatial Skills in Mathematics and Other Fields

This section discusses the significance of spatial skills to ability in Mathematics and other fields, to investigate the relevance of what the Visualising Volume application aims to teach users. Studies have shown the importance of early spatial skills in the development of later mathematical skills. Interestingly, Verdine et al [17] discovered that early spatial skills are actually a better indicator of later mathematical skills than early mathematical skills. They also found that the link between early spatial skills and later mathematical skills is strengthened when early spatial skills are combined with executive function (the skills required of an individual to plan and reach their goals, including attention control, working memory and inhibition[1]). This suggests that learning spatial skills in a goal-oriented environment is especially important for contributing to a stronger foundation in mathematics. Goal-oriented environments for development of spatial skills can be created using tasks such as building a chosen shape out of smaller components or manipulating an object to match an image.

Research has also demonstrated the benefit of strong spacial skills in environments that are not purely mathematical. Wai et al [18] culminated 50 years worth of research on 400,000 participants and learned that spatial ability is crucial to developing skills and learning knowledge in STEM fields, such as chemistry. Kell et al [9] discovered an equally profound link between spatial skills of 13-year olds and their future creativity and technical innovation, suggesting the advantage of strong spatial skills in children reaches beyond the field of mathematics.

The importance of spatial skills to young learners, as demonstrated by this research, suggests the potential of the Visualising Volume application to be a valuable learning tool.

2.2 App- and Games-Based Learning

This section discusses the relevance of app- and games-based learning and their potential effectiveness as useful teaching tools. Mobile applications are an increasingly popular learning medium,

[1] As defined at https://developingchild.harvard.edu/science/key-concepts/executive-function/

with downloads of educational mobile applications being almost double in 2020 compared to 2017 [15]. Examples include *Google Expeditions* [1], an app allowing immersive learning by exploring a variety of environments, from historical sites to parts of the human body, and *Reading Eggs* [2], an award-winning app that provides an interactive reading experience for children. Research of app-based learning, and technology-based learning more generally, indicates the potential for educational software to have a positive effect in the classroom. A 2015 study [7] suggests that technology-based learning is actually more effective than traditional learning, with authors attributing this result to the more active, interesting learning environment created by the use of technology. A more recent review of multiple studies [8] yielded similar findings, with the authors concluding that there is evidence that use of interactive apps in learning is beneficial. Interestingly, they found this to be particularly true for early mathematics learning.

There is substantial evidence to suggest that games-based learning can be of great benefit to young students. In an experimental study conducted in Taiwan [4], children showed improved recall and problem-solving skills when they received educational instruction using a computer video-game. Another Taiwanese study [19] found that, in addition to problem-solving skills, students' motivation was improved by games-based learning. Similarly encouraging results were produced by a Greek study [12] which involved trying to teach students computing concepts using a video game. The researchers found that school students improved their knowledge of the concepts being taught and were more motivated to learn this way than by traditional teaching methods.

The positive value of using games and applications to help students learned, as demonstrated by the research discusses, suggests the relevance and potential value of the Visualising Volume application.

2.3 Analysis of Existing Software

This section discusses existing software with similar features to the Visualising Volume application and evaluates them. There exist applications allowing users to choose a 3D shape and enter the shape dimensions so the app can provide the shape's volume ([3], [13]), with static visuals that do not reflect any changes in user input. Research for this project did not yield any available software to fit the purpose of this project. There were no discoverable mobile or web apps that allowed users to manipulate 3D shapes and observe the effects on the shape's capacity, and no apps with dynamic visuals exhibiting the relative voluminal effects of changing one dimension of the shape and maintaining the others. This indicates that the Visualising Volume application may offer features not available on other applications.

A multi-platform software worth discussing is *Shapes 3D Geometry Learning* by Learn Teach Explore [20], an app designed for use in classrooms to teach children how 3D shapes are formed from 2D nets. It achieves an effective, interactive education on geometry of 3D shapes much like the style of education this project aspires to provide on volume of 3D shapes. *Shapes 3D Geometry Learning* achieves its quality as a multi-award winning app[2] by providing a space for users to experiment with building different nets and observe the effects of their choices. Users are given the aim of building their selected shape from various nets, providing a goal-oriented environment for learning. The success of this software is a positive indicator of the effectiveness of interactive learning tools to support mathematical education.

[2] Awarded Appaward 2014, EAS Recommended, Moms with Apps Proud Member and UK ED CHAT according to https://shapes.learnteachexplore.com/shapes-3d-geometry-learning/#buy

Chapter 3

Product Requirements and Features

3.1 Product Requirements

This section explains how requirements for this project were decided. Requirements can be divided into two sections: functional and non-functional. Functional requirements refer to services the system must provide, and can usually be inferred from the user stories, whilst non-functional requirements refer to the system's quality, such as its availability and performance. This section will begin by discussing functional requirements.

Gathering requirements for this project included thorough consideration of how best to demonstrate the relationship between shape size and shape capacity to the target consumer. The intended user of this application is a child aged 5 -7 years old, who most likely has not yet learned volume as a mathematical concept [5]. The purpose of this app is to introduce them to the concept, without introducing the mathematical reasoning behind it. It aims to help users make a connection between the size of a container and how much liquid it can hold, without the need for understanding properties of 3D shapes and mathematical formulae. At the early primary school age, children tend to understand from play that a visibly larger container can hold more liquid than a smaller one. This app aims to consolidate this understanding and build upon it in two ways. The first is encouraging users to manipulate shape dimensions, making a 3D cuboid or cylinder - with a constant volume of water in it - smaller to see water height rise (decreasing available space) and larger to see water height fall (increasing available space). The second is encouraging users to try and fill a larger shape with water from multiple containers (all of which are the same size, significantly and clearly less than that of the large container), in order to decide if x amount of small containers can hold less, the same or more water than 1 large container.

Once the most appropriate methods for demonstrating the discussed concepts were devised, consideration needed to be given to how users should interact with the app. Providing a means for users to learn was the first step; achieving an app that users want to interact with and can learn from effectively was the second. From background research, it was known that goal-oriented environments are effective for learning spatial-skills at an early age and increase motivation in learners. This informed the decision to make the app a game-like software, with users being encouraged to aim for a particular condition in each task.

Discerning the features provided by similar applications discussed in Chapter 2, and any features *not*

provided, helped inform what this app should offer. One type of existing application enabled users to calculate the volume of 3D shapes by providing dimensions, and in changing the dimensions users can see how the volume is impacted. However, these apps do not have graphics that reflect changes in dimensions and volume so it is not easy for young children to see how dimension changes affect the appearance and apparent capacity of a 3D shape. Another type of existing application (of which only one was found) allows users to experiment with 2D nets of 3D shapes and encourages users to try and build the 3D shape from its 2D net. This provides the "learn and play" environment this project aimed to reproduce, informed by the research discussed in the previous paragraph, but for learning how 2D nets relate to 3D shapes. Deliberation of the two discussed types of application led to the decision to try and merge the benefits of both by developing an app that (1) contributes to understanding of volume of 3D shapes and (2) provides a playful and experimental environment for learning.

Non-functional requirements of the system include that it must display and function correctly on iOS devices with screen resolutions 750 x 1334 to 2048 x 2732. This range covers most Apple iOS mobile devices (iPhones and iPads) and was informed by 3 factors: (1) 57% of UK children aged 5 - 7 years old have their own tablet (higher than any other device), and 77% use a tablet to go online (higher than any other device) ([10]); (2) the most popular type of tablet is an Apple iPad ([16]) and (3) including iPhone screen resolutions in the range significantly increases versatility and accessibility of the application.

A second non-functional requirement of the app is it must appear to respond immediately to user input so a young user can discern that their action resulted in a particular outcome. A delay between cause and effect might impede the aim of the product to help users establish a relationship between shape dimension changes and voluminal changes.

Finally, the visual design of the product was considered. A colour palette consisting of various colours of a medium-high brightness was chosen to create a fun, attractive user interface (a 2016 study actually showed that using saturated colours creates a more joyful experience for users playing video games, as does using a range of colours [6]). Keeping the colours below a very high brightness was decided as a way to keep a calm environment. The learning environment should be playful and experimental - using intense colours might imply stressful situations to the users. For consistency and to reduce the need for instructions on screen at all times, it was decided colours would be used in a meaningful way: for example, any indicator of a target (e.g., arrow pointing to target water height) will always be the same colour so users associate the colour with being a goal. White was chosen as the only colour that would be used as a background in order to keep scenes clean and colours easily discernible.

3.2 Product Features

This section provides a list of features of the Visualising Volume product, informed by the requirements discussed in the previous section. Features are aspects of the application that achieve a requirement, or set of related requirements. They are allocated a priority based on the *MoSCoW* grading system: *must-have, should-have, could-have* and *would-have*, each decreasing in priority. Please note *Type 1* games refer to *reach the target water height* games and *Type 2* games refer to *compare the volume* games.

1. *Must-have* features:

    1.1. Compatible with screen resolutions 1536 x 2048 - 2048 x 2372. This covers most Apple iPad screen resolutions, the most popular device amongst 5 -7 year olds.

    1.2. Responds to user input with no perceptible delay.

    1.3. Two Type 1 games: cylinder and cuboid. User is shown the shape partially-filled with water and is encouraged to drag one face of shape to manipulate water height to reach target.

    1.4. Two Type 2 games: bath & buckets scenario and teapot & teacups scenario. User is shown an empty large container and multiple filled small containers, and must work out whether the large container has higher or lower volume than the multiple small containers combined.

    1.5. Easy-to-use and accessible user interface that is easily learnable by target user group. In particular, a user interface with: buttons labelled intuitively and simply, with as few words as possible whilst maintaining clarity; clickable/draggable items clearly presented as clickable/draggable.

    1.6. Ability to register for an account and log into that account.

    1.7. Ability for users to provide feedback on their understanding of concepts.

2. *Should-have* features:

    2.1. Compatible with screen resolutions 750 x 1334 - 1536 x 2048. This covers most Apple iPhone screen resolutions.

    2.2. Clean and simple user interface.

    2.3. Constant colour scheme with meaningful usage of colour.

    2.4. Randomisation of target water height in relevant games.

    2.5. Ability to register for and log in to a teacher account that has privilege to view student account details.

3. *Could-have* features:

    3.1. Game characters to add a fun element that will appeal to young users.

4. *Would-have* features:

    4.1. Level selection screen for users to choose which game they play.

    4.2. Two further Type 1 games: sphere and cone. These are more difficult shapes for children to work with as cross-sectional radius is not constant, so would be considered a "challenging" extra game.

    4.3. Two further Type 2 games: variation with two single containers, instead of one large and multiple small. For example, two very differently-shaped bottles. The purpose of this would be to encourage users to consider how different-looking shapes and everyday objects might have the same volume.

Would-have features are of lowest priority and were not implemented in this project due to time constraints.

3.3 Tools & Technologies

This section justifies the tools and technologies used in this project and explains their usefulness in achieving the project objectives. The following were important considerations for this project when deciding on the tools to use:

- The app uses 3D shapes that must be manipulable.
- The app must be deployable to iOS mobile devices.
- It would be useful to experiment with user interface and view changes quickly instead of programming and compiling to know how screen looks.
- The project has a 12-week time restriction which must include time taken to learn new software/languages.
- Version control should be used to prepare for any cases of lost progress.
- A project management tool should be used to keep track of sprints, stories and issues.

With these in mind, it was decided that game engine Unity would be the most appropriate platform to use. It is a well-known tool with particular usefulness for projects simulating physics and real-life mechanisms. It enables shapes to easily be attributed physical properties, causing them to behave realistically, such as responding to gravity and collisions. It has many "drag-and-drop" UI features making it easy to try out lots of different options very quickly and without the need for code. Unity has a core feature called Play Mode that enables users to run projects inside the Unity Editor simulating how it would run in a build. It also allows custom scripts to be created and attached to objects in game scenes enabling custom behaviours to be created. Scripts can be easily reused with different GameObjects by making certain attributes serialised, making Unity an ideal option for this app which uses repeated behaviours.

Programming in Unity uses C#, an object-oriented language close to Java. With prior knowledge of Java, it was decided learning to program sufficiently in C# was an achievable task in the time frame of the project. Microsoft Visual Studio Code was used to program in C# and provided benefits of inline warnings and error notifications. To create any 3D models required for the project, free software Blender was used.

A database was set up using Google Firebase to enable user authentication and to store user data. This was chosen as it integrates well with Unity and provides very useful documentation on configuration and integration with an application. GitHub was used to facilitate version control for this project. It helps keep track of changes by enabling descriptions with commits and stores any pushed versions of the software in case a restoration was needed. Git LFS (Large File Storage) was used in addition to GitHub to track files too large to be stored in the GitHub repository.

Having been used effectively in previous projects by the author, project management tool Jira was employed to help with following a Scrum framework. For testing, surveys were created using Microsoft Forms as it enabled participants to be emailed interactive forms that are quick and easy to fill out, and has useful visual displays of response information.

Chapter 4

Design

4.1 System Architecture

This section discusses the architecture of the Visualising Volume application, an overview of which is provided in Figure 4.1. *Architecture* as used in this report is a high-level description of the system's components, as opposed to a code-level description which will be provided in Section 4.2. It encompasses the main *layers* of the system, which in this case are the client (application on user device), the game engine (Unity) and the cloud-hosted database (Firebase). The purpose of this section is to outline the significant components and justify their use in this project in alignment with the project objectives.

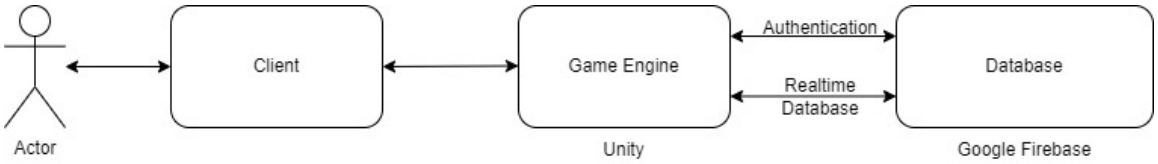


Figure 4.1: Diagram of System Architecture

4.1.1 Unity and .NET Framework

The code for this project was written in C#, in the Visual Studio Code IDE. An IDE (Integrated Development Environment) is a software that provides a text editor for writing code along with some developer tools such as highlighting code errors and scripting wizards (a feature that displays a list of API[1] calls dependent on what the user is typing, making them quick and easy to generate). The MonoBehaviour scripting wizard in Visual Studio Code, in particular, makes it very straightforward to interact with Unity's API and make use of features inherent in `MonoBehaviour`, Unity's base class.

Unity makes use of Microsoft's .NET framework which provides the CLR, the common language runtime environment. This runtime environment provides portability. For example, the Unity-independent components of this project in the `Model` classes could be removed from the Unity

[1] An API, Application Programming Interface, is a means to access existing software via code.

and, by the CLR, could communicate with objects programmed in a different language. This makes Unity, with its relationship with .NET, an ideal choice of game engine when prioritising scalability.

4.1.2 Google Firebase

Firebase provides many cloud-hosted services, two of which were employed in this project. Firstly, it provides a realtime database that the Visualising Volume application uses to store user data, specifically their evaluation of their understanding of the two different voluminal concepts demonstrated in the games. Secondly, it provides user authentication which the application uses to create and store user accounts. The user database allows user permissions to be altered and is easily scalable as the number of users grow.

Firebase supports integration with mobile applications which is a requirement for the next stages beyond this project scope, which would see the application deployed to iOS mobile devices (please see Section 4.1.3 for further details). It does not guarantee stability and expected behaviour when integrated with desktop applications, however it does provide a subset of the software development kit that allows the database and services to be used with desktop applications for testing purposes. This was employed effectively during this project as evidenced by repeated use of the desktop version of Visualising Volume yielding expected results in the Firebase database set up for the application.

Unlike many popular database services, Firebase is not a relational database but what is known as a *noSQL* database. Relational databases are named so as they arrange data as entities related to each other; in this sense the data is structured. NoSQL databases store unstructured data, the benefits of which make noSQL databases an ideal choice for mobile applications. Unstructured data, though it can result in redundancy (the very thing relational databases seek to eliminate), is much easier to partition across multiple servers meaning it is very easy to scale noSQL databases. For mobile applications that do not require the integrity enforcement of relational databases, noSQL databases offer fast-retrieval of data and scalability, which is important for applications that seek to grow, particularly across multiple platforms.

4.1.3 Target Devices

The target devices, as defined in Features 1.1 and 2.1, are Apple iPads and iPhones due to their popularity amongst the target user group (see Chapter 2 for details). Unity has a feature that facilitates building iOS applications and allows you to view the display of many different screen resolutions whilst in development mode, so the developer can check the compatibility of the game layout with target devices. Development of the application was conducted in iOS build mode to ensure compatibility with all iOS devices. It was not possible, however, to deploy the application to iOS devices as deployment requires an Apple Developer licence, the cost of which was not within the budget of this project. Fortunately, Unity provides the ability to test applications via wired connection to a device which is running the iOS Unity Remote application. This enabled user testing to be conducted with iPads and iPhones, giving participants an accurately simulated experience of using the application deployed on the device. For this reason, testing can be considered an accurate reflection of testing with a deployed application.

4.2 Design and Implementation Details

This section outlines the critical design decisions made in development of Visualising Volume, particularly regarding scalability of the application, class structures and class relations. It justifies design patterns used and details their implementation, and explains decisions made to facilitate certain features.

4.2.1 Reusability and Scalability

A main objective of this project was to produce reusable code and a scalable application. This implied making as much code as possible independent of game-specific aspects, such as the objects used in the game scene or the logic handling the game flow. Much of the planning stage of the project involved deciding on class hierarchy and how to structure the code effectively to reduce workload in the future, allowing for scalability and easier maintenance of code. Using the Model-View-Control pattern (discussed in Subsection 4.2.3) helped with separation of concerns, with all of the game engine specific behaviour being handled by the view class and all of the game engine-independent data modelling handled by the `Model` class. This improved reusability of code as the `Model` class could be used across multiple platforms, within the constraint that the platform compiles C# code.

4.2.2 MVC Pattern

Since this project had a significant focus on the user interface of the application, the Model-View-Controller pattern was employed, and slightly adapted, for designing scripts (Unity classes or class collections) and dividing class responsibilities. Patterns in software design are templates or rules for arranging classes and objects such that code flexibility, reusability and maintainability are optimised.

When implementing the MVC (Model-View-Controller) pattern, there is usually a `Main` class to instantiate `View` and `Model` classes, responsible for display and data modelling, respectively, as well as the `Controller` class that coordinates `Model-View` interactions. For this application, since Unity begins by running *scenes* not scripts (though scripts can be instantiated immediately if a part of the scene), there was no need for both a `Main` and a `Controller` class. Instead, a `Main` class was designed for each game type that acted as a controller class. This figure always implies the Subject-Observer relationship between the `Model` and `View` classes, which is discussed in Section 4.2.3.

In addition to its usefulness for graphical user interfaces, the Model-View-Controller template presented a structure that helped to create an application whose data logic is not Unity-dependent. To improve scalability, all data modelling was designed to be independent of Unity-specific features, with the `View` classes ensuring that effective Unity-specific features are used to optimise usability and data display.

4.2.3 Class Hierarchies and Observer Pattern

Class hierarchy refers to a programming class *tree* or levels system where classes inherit from other classes. It is important in Object-Oriented Programming (OOP), which is a style of programming based on well-defined objects and their interactions [11] . C#, the programming language of Unity, is an OOP language and has a base class `Object` from which all other C# classes derive. This translates in Unity to all objects the Unity editor can reference being derived from the C# `Object` class.

The benefits of OOP arise by encouraging modularity of behaviours - encapsulating related behaviours in a reusable unit of code - which eases code maintenance. If a particular functionality needs amended, OOP helps ensure that only the class of the object with that functionality will require the amendments, not everything else related to it. If a particular functionality is required for an object and already exists in another object, inheritance allows the former object to reuse code already implemented in the latter and reduces the redundancy that would arise by having this functionality coded in two separate classes.

To benefit from these advantages, classes were designed according to object-oriented principles. In particular, any behaviour to be used more than once was encapsulated as a class or an interface, an abstract class that provides method headers for behaviours - with no implementation - and obliges child classes to implement them. This is useful when classes have the same types of behaviour that require differing implementations. For example, Type 1 games in Visualising Volume require methods to perform calculations, the parameters and logic of which are dependent on the shape type. Only two shapes were used in this project (one for each Type 1 game), so for simplicity it might have been reasonable to amend the `Model` class for each version of the game so that it could handle calculations specific to the shape used in that version. This would have reduced the number of classes created, however it would impede scaling of the application as a new `Model` class would need to be created for every shape added to the game. Instead, an interface, `ShapeVolume`, was created with method headers indicating the calculations they should perform. For each shape, a class was created that implements the `ShapeVolume` interface and provides the method behaviour depending on the shape. Each `ShapeVolume`-derived class has member variables for its dimensions and formulae for calculations specific to the shape. This decision enabled Type 1 games to require only one `Model` class which can delegate shape-specific calculations to `ShapeVolume` objects - by polymorphism, all classes that implement the `ShapeVolume` interface can be referred to as a `ShapeVolume` object. This means the `Model` class has a reference to a `ShapeVolume` object - which could be a `CuboidVolume` object or `CylinderVolume` object, or other so long as it implements `ShapeVolume` - which can be set accordingly for each new Type 1 game, and does not needed edited for every new game created.

Figure 4.2 is a UML (Unified Modeling Language) class diagram showing the structure of a Type 1 game. For clarity and emphasis of significant features, it has been simplified by omitting helper methods, such as getters and setters, and temporary data holders. A game uses the `CuboidVolume` *or* `CylinderVolume` class but both have been added here to exemplify the relationship with the `ShapeVolume` interface. This relationship is be replicated for every new shape added. `DragNotifier` is an events handling class that notifies `Main` of user input by calling the `updateModel` method, which in turn notifies `Model` that new data needs to be processed by calling its `updateShapeModel` method. Type 2 games have a similar structure.

`View` and `Model` classes follow an Observer-Subject pattern. This design pattern involves Ob-

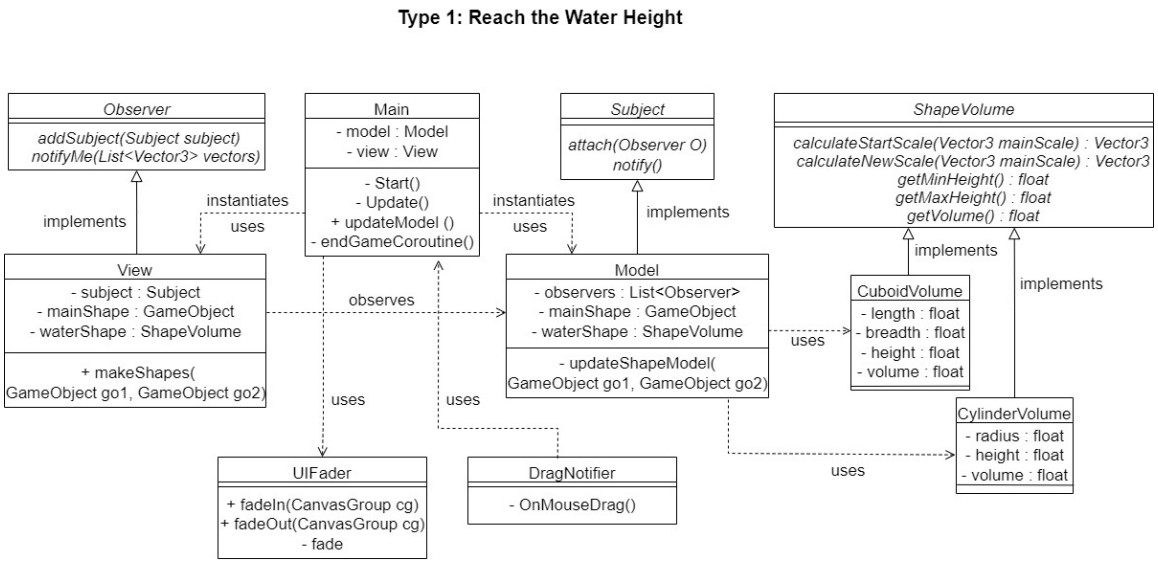


Figure 4.2: Simplified UML Class Diagram

server classes 'subscribing' to a Subject class, which notifies its 'subscribers' of a particular change through notify methods. This is a useful pattern as it removes the need for Observer classes to regularly check for the Subject changes and instead concentrates the notifying task in the Subject class. In Visualising Volume, each game type requires its own pair of `View` and `Model` classes, so to implement this a `Subject` interface was made with methods to add and notify Observers and an `Observer` interface was made with methods to register a Subject and be notified by a Subject.

Variation in Games

With the time restriction, it was not possible to create many games so a more reasonable way of producing variation in the application was to add an element of randomness to both game types. For Type 1 games, this was implemented by randomisation of the target water height, indicated by the height of the main game character Tilly the Bear. This character GameObject is instantiated at runtime of a size randomly chosen from a range of reasonable target heights. Similarly, variation was induced in Type 2 games by randomising the size of the main object. This varies the number of small objects needed to fill the bath, and therefore varies the outcome of each level between (1) small objects having higher combined volume than larger object; (2) large object having same volume as all small objects; and (3) large object having the higher volume. This randomisation was the strategy chosen to ensure users have varied experiences each time they use the application or replay games and allows them to consider different scenarios and further their understanding.

Database

Integration with Firebase required initialising the database, which was done through the `FirebaseManager` script, using the Firebase API. This script us attached to an empty GameObject in appropriate game scenes and handles writing to and reading from the database. This occurs when users submit feedback on their understanding and when teachers request to view this feedback information.

Chapter 5

Testing

5.1 Developer Testing

This section discusses manual developer testing, which consists of developers testing the software by using it themselves and evaluating it as a user. Throughout development the game was repeatedly run so functionality could be inspected manually. This was a helpful technique for ensuring general behaviour of the application and for testing the business logic, such as win conditions and correct order of scene transitions, and for making decisions about the applications appearance and user interface. It was also essential for determining if user story acceptance criteria were met, helping determine progress by signalling the formal completion of sprint tasks.This technique is limited, however, in its usefulness when it comes to debugging. To help improve the process of finding and eliminating errors, developer testing was paired with automated testing, discussed in 5.4.

5.2 User Testing: Evaluating Software Performance

This section discusses user testing, the first of two types conducted for this project, which focussed on how the software functions and looks. The purpose of this type of user testing was to help learn the extent to which Objectives 2 and 3 were achieved, providing less biased feedback than developers would provide. Objectives 2 and 3 concerned providing an attractive and easy to use application. It was also considered an important stage as user testing can highlight any edge cases that might not have considered by developers. Edge cases are 'extreme' usage cases that aren't expected behaviours of users but that may be dangerous or troublesome if not handled and prepared for.

For user testing, participants were briefed on the purpose of the testing and, having consented, were presented with the Visualising Volume application to play, either on an iPhone XR (pixel resolution: 1792 x 828) or 10.5" iPad Pro (2224 x 1668). Participants were emailed a Microsoft Forms survey to complete so their feedback could be easily gathered and analysed. A link to the survey is provided in Appendix B.

Determining the relative proportions of user responses was deemed an appropriate way to analyse the results as it clearly indicates how opinions are divided amongst participants and highlights any

significant results. As this type of testing is qualitative, it is difficult to determine the markers of success or failure. To provide a scale and formal evaluation system, it was decided that results will be categorised as follows :
Good: Positive statements with which over 70% of users agreed or strongly agreed, or negative statements with which less than 30% agreed or strongly agreed.
Satisfactory: Positive statements with which 50 - 70% of users agreed or strongly agreed, or negative statements with which 30-50% agreed or strongly agreed.
Unsatisfactory: Positive statements with which 50% of users (or less than) agreed, or negative statement with which more than 50% agreed.

87% of participants agreed or strongly agreed that all games are laid clearly, suggesting that the layout is well-considered and clear enough for the majority of users. Although no negative feedback was provided, the remaining 13% of participants indicated that their opinion was "neutral", suggesting there is room for improvement. This result categorises layout clarity as good.

75% of participants indicated that the colour scheme of the application was appealing, categorising this as good. 63% of participants indicated that the colour scheme was meaningful, suggesting efforts to associate colours with functions in the game was not achieved as well as was hoped but nonetheless categorises this area as satisfactory. 13% indicated that the colour scheme was confusing, which is not significant enough to warrant urgent reconsideration but significant enough to be considered in conjunction with other results. It might be useful to conduct further testing to investigate a potential link between the attempt to make the colour scheme meaningful and users finding it confusing. Nonetheless, achieving a clear colour scheme is categorised as good.

50% of users indicated that all text was easily readable, meaning half of users did *not* find all text easily readable. Furthermore 38% of users indicated that the text was too faint, and 38% indicated the font made text unreadable. This is disappointing but useful feedback suggesting the text styling requires urgent consideration and is categorised as unsatisfactory.

100% of users agreed or strongly agreed that instructions for all games are clear, categorising this as good.

50% of users were not aware that they could exit a game by selecting the 'Home' button, which is unsatisfactory and indicates further work needed. Only 13% were not aware they could access this function or the shoe/hide instructions functions, so this area is categorised as good. It is possible that this is connected to the issue of unclear appearance of text. Further testing should be done to investigate if a connection exists. It is also possible that naming the button 'Home' does not clearly indicate its purpose and requires rewording.

100% of users strongly agreed that the application displayed correctly on the device they used. This categorises this area as good. A limitation here, however, is that 80% of participants used the iPhone and only 20% used the iPad. The low representation of iPad usage was due to unforeseen lack of availability of the device arranged for testing. As no users indicated any issues with display, this is not deemed an area of deep concern however further testing should be conducted on a greater range of devices to ensure Objective 7 is achieved (app displays correctly on all target devices).

100% of participants agreed or strongly agreed the application responds as expected to user input. This categorises this area as good. However, since 50% of participants refrained from selecting the most positive option, consideration might be given in future to improving performance of the application.

The table below provides a summary of the results. It shows that most areas tested in this type of user testing are satisfactory, and some are indeed very comfortably above this level. There are two areas, however, that require significant consideration: making text readable and making home and instructions functions more clear.

Area	Unsatisfactory	Satisfactory	Good
Games are laid out clearly			✓
Colour scheme is appealing			✓
Colour scheme is clear		✓	
Colour scheme is meaningful		✓	
Text is easily readable	✓		
Instructions are clear			✓
Home and Show/Hide Instructions functions are clear	✓		
Application displays correctly on devices			✓
Application responds as expected to input			✓

5.3 Proxy User Testing: Evaluating Value as a Teaching Medium

This section outlines second type of user testing conducted, which was the strategy employed to determine if the application could provide benefit as a learning tool and discusses the results. There was insufficient time to achieve clearance from the ethics committee for user testing involving participants under 16 years of age. To gauge how useful the application is as a learning tool, a survey was conducted with participants with a background in mathematics teaching and/or primary school teaching. This was deemed an appropriate work-around as they have extensive understanding of how young people learn. This user testing was conducted exactly as the user testing discussed in Section 5.2 was. A link to the survey is provided in Appendix B.

To determine if the application had met the Objective 1: Develop an app that contributes to users' understanding of volume of various 3D shapes and objects, participants were asked two sets of questions. The first set assessed participants' opinion of the usefulness of Type 1 games, and the second set assessed their opinion of the usefulness of Type 2 games.

Participants were first presented with Scenario 1 (see Figure 5.1) and the statement "Before using the application, I think a child aged 5 -7 years old would know that increasing the length of this shape would decrease the height of the water level". Participants were asked how strongly they agree with the statement and were then asked how much they agree this would be the case after using the application. 100% of participants disagreed or strongly disagreed with the first statement, suggesting that participants do not believe 5-7 year olds would understanding how changing the length of a cuboid would affect the water level. 100% of participants agreed or strongly agreed that, after using Visualising Volume, 5-7 year olds are likely to understand how the water level is affected by changing the length of the cuboid.

Participants were then presented with Scenario 2 (in Figure 5.1) and the statement "Before using the application, I think a child aged 5 -7 years old would know how to compare the volume of 6 buckets combined to the volume of the cuboid by pouring the buckets into the cuboid and considering how many were emptied/how full the bath is after emptying them all". Participants were asked how strongly they agree with this and how strongly they would agree after a child has used the

(a) Scenario 1 (b) Scenario 2

Figure 5.1: Proxy User Testing Scenarios 1 and 2

application. 100% of participants disagreed with the first statement, suggesting 5-7 year olds would not understand how to compare the volume of different objects. 100% of participants agreed that, after using Visualising Volume, 5-7 year olds are likely to understand how to compare the volume of objects by using the technique shown in the application.

These results reflect very positively on the application and indicate that it has potential value as a teaching tool. A limitation of this type of testing, however, is that the survey did not specify exactly what ”using” the application means. It might have been interpreted as one use, or it might have been interpreted to mean after significant, repeated usage. This wouldn't reflect badly on the application but might suggest that improvements could be made to help children understand the concepts with less use of the application. Another limitation is that the questions only prompted participants to consider two specific scenarios. It is hoped that the participants opinion on a 5-7 year olds understanding of these scenarios applies generally to the concepts concerned and is not specific to the shapes used in this survey, however this cannot be claimed with certainty. To increase reliability of this inference, further testing should be done with various scenarios and shapes.

5.4 Automated Testing: Unit Testing

This section defines unit testing and discusses why it was chosen as a testing technique. Unit tests are programs that assess small areas of functionality. They are useful throughout to check behaviours are as expected and help spot areas of difficulty. Manual testing by developers cannot always pinpoint where issues are arising, as lots of interacting areas of coded functionality may be interacting to produce what the developer sees on screen. With unit tests, it is much easier to break debugging down into manageable chunks and spot where issues might be. This method of testing aligns with the Agile methodology as it allows developers to make minimal changes to code - they are able to pinpoint trouble spots more narrowly and perform less experimental code changes. It also helps enforce good object-oriented programming style: if objects have been defined properly, issues with a particular functionality should arise in a narrow area of the system as that functionality should be encapsulated in an object which is responsible for it. Unit testing can help developers consider the structure of their programs and make improved decisions.

Difficulty was experienced with ensuring expected behaviour of the shape acting as water in response to user input in Type 1 games. The initial solution was manual testing by the developer, meant making tweaks to parameters and trying to figure where errors were occurring. It was decided unit testing would be a more appropriate method for finding, and avoiding errors. Unit tests were written for functionality requiring mathematical accuracy which highlighted where unexpected behaviour was happening.

Chapter 6

Conclusion

This chapter evaluates the project as a whole and discusses notable aspects of the project journey. It provides a reflection on the process and suggests where improvements could have been made.

6.1 Outcome

At the end of this project, a few things have been achieved:

- An application, Visualising Volume, has been planned, designed, developed, tested and evaluated.
- A project has been successively managed, seen through from the planning stage of devising a product idea through the delivery of that product.
- A report has been completed documenting every stage of the project, evaluating the product and techniques used in the project.

Though these have not been executed perfectly, as will be discussed in Section 6.2, they have been attempted with enthusiasm and have provided a valuabe learning opportunity.

6.2 Limitations

This section discusses the most significant weaknesses of this project, and makes suggestions for future work. These are important to consider as, without critical evaluation, rooms for improvement cannot be identified and improvement cannot be made.

Generally, this project was well-managed. An impactful mistake was made early in the process, however, that requires consideration. When deciding to develop an application - a new feat to the author - it was assumed that the 12 week time frame would permit time to learn the required skills (C# and 3D modelling) and become familiar with new environments, such as Unity and Blender,

whilst not impeding the progress of the project. This didn't turn out to be the case, and the end result was an application that, though functional, fell short of the initial vision. Of the product-specific objectives laid out at the beginning of the project, one was left unmet: to create an easy-to-use application. The application was let down by unclear text and inaccessible or unclear functionality. On reflection, it is clear that more time should have been dedicated to checking accessibility of font style, size and colour, which would have greatly improved the usability of the application.

One objective was set regarding testing: to conduct useful testing that determines the extent to which functional aims have been achieved and the app is valuable as a learning tool. This was partially met but could have been achieved much more fully. This area of the project was a steep learning curve, and highlighted the value of having multiple types of testing to support each other and, most significantly, the importance of planning testing from early stages in the project. Automated testing should be implemented as soon as development begins to prevent time being wasted searching for errors that testing could find quickly. User testing should be more thoroughly planned to ensure all aspects of the product are tested.

With regards to the product itself, much room exists for improvement. A key suggestion for future work is to improve accessibility by the means suggested above, but perhaps also by implementing a text-to-speech function. This would be appropriate for the user group - children aged 5 - 7 years old - as not all will be capable of reading the instructions without the aid of a teacher or guardian.

Another key suggestion is to expand the use of the database to store user information. It would be helpful for teachers to view a record of user performance or feedback so they can track progress of students. This could be facilitated by storing 'Class' attributes with each child, and the records for these children being accessible by the teacher of that classroom. On this scale, security would need to be considered more seriously and stricter authentication checks would need to be implemented. In addition, although Firebase provides a degree of input checking, client-side safety measures should be implemented to ensure that date read to and written from the noSQL database is of the expected format to prevent unexpected behaviour.

6.3 Reflection

As a positive conclusion to this report, this reflection highlights the achievements of this project and the aspects to be proud of. The author was able to successively follow a Scrum framework, adapted to work for one person, and regularly review progress to make any changes necessary. The primary objective of this project was to produce an application that contributed to users' understanding of concepts related to volume of 3D shapes. User testing conducted with teachers as proxy users demonstrated the potential value of Visualising Volume as a learning tool, which was a very positive outcome of this project.

Appendix A

User Stories

1. As a user, I want to end a game when I achieve the win/end conditions.
    - Priority: very high
    - For Type 1: when water height matches bear height; for Type 2: when water bar is full or when all small objects have been emptied.
    - Indicate end of game: character(s) smile to indicate success. Type 1 zooms into character.
    - Test: reach win condition, game ends, correct 'solution' screen appears
2. As a user, I want to access basic game functions, specifically buttons to go home, see instructions, replay level, go to next level.
    - Priority: very high
    - On game screen: home and instructions (show/hide), instructions on screen for start of game and disappear after 5 seconds
    - Test: buttons produce desired affect and clearly visible on all games
3. As a user, I want games to be clearly laid out and easy to understand.
    - Priority: high
    - No unnecessary elements on screen, objects easily identifiable
    - Instructions easily fit on screen with large enough font but not distracting or interfering with game scene
    - Clear and readable font
    - Test: evaluate visually
4. As a user, I want draggable edge in Type 1 to be identified as so and be draggable.
    - Priority: very high
    - Edge of shape, add arrow
    - Test: drags according to input
5. As a user, I want small objects in Type 2 to be clickable, indicate when they've been clicked and indicate when their course of action has finished.

- Priority: very high
- Indicate when object has been clicked by colour change & slider reset to 0 when new object is clicked
- Objects not clickable until current object has finished course of action
- Object path mimics slider dragging
- Test: when object is clicked it displays behaviour to indicate so
- Test: click a second object when first is in course of action and check it doesn't affect anything or indicate it has been clicked
- Test: click object, drag slider and check object follows drag
- Test: click second object after first is complete object and check slider goes to 0
- Test: click object, complete course of action and see that object goes to state that indicates finished (i.e. emptied)

6. As a user, I want to be taken to an 'explanation screen' that explains the solution of the game I just played.
    - Priority: high
    - Type 1: explanation of why shape changes -¿ water level change (high priority: general explanation; low priority: explanation specific to target water height and user input)
    - Type 2: (1) offer a screen to make a choice with an explanation of the game scenario. (2) offer soution screen explaining if their answer was correct or not.
    - Test: explanation Type 1 appears after win conditions met
    - Tests : Type 2: choose answer appears after win conditions met, explanation of scenario is correct, correct user input identified, correct identification of whether user input is correct answer or an incorrect answer, explanation of solution is correct
7. As a user, I want Type 1 water height to vary correctly with water shape.
    - Priority: high
    - Water height should increase/decrease as shape length decreases/increase
    - Test: water height does down as shape length is increased & vice versa
8. As a user, I want my game to display correctly on my device.
    - Priority: high
    - iOS setting on Unity
    - Test: use on iPhone and iPad and check it renders properly
9. As a user, I want my games to have a degree of randomisation to provide variation.
    - Priority: medium
    - Type 1: target height varies
    - Type 2: large object size varies
    - Test: run each game multiple times and note target height/object size each time, and see that there is reasonably even distribution
10. As a user, I want to view all games and choose which one i want to play.

- Priority: low
- Level screen
- Test: check all games are there and links all work

11. As a user, I want to make an account and log in
    - Priority: medium
    - Store feedback in database as attributes for that user
    - Teacher has a privileged account that can see feedback
    - Test: play with multiple accounts and see that information is updated and correct

Appendix B

Second appendix

Link to video of Visualising Volume application in use: https://youtu.be/Mka6vA4yJ8M. Please note this video shows a desktop version of the application. This decision was made because mouse movements are indicated by the cursor on a desktop, a feature that is amiss on touchscreen iOS devices, which more effectively demonstrates the features of the application for the viewer.

Link to User Testing Survey: https://forms.office.com/Pages/ResponsePage.aspx?id=DQSIkWdsW0yxEjajBLZtrQAAAAAAAAAAAAa__a-HB1dUOFFGQzRDMzg2WFNJVTA0RDBERjVGUlZKTS4u.

Link to Proxy User Testing Survery: https://forms.office.com/Pages/ResponsePage.aspx?id=DQSIkWdsW0yxEjajBLZtrQAAAAAAAAAAAAa__a-HB1dUOFlSWEhYSksxS1g1RFVKODQ2MTRaOU1NVS4u.

School of Computing Science
University of Glasgow

Ethics checklist form for 3rd/4th/5th year, and taught MSc projects

This form is only applicable for projects that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, getting information about how a system could be used, or evaluating a working system.

If no other people have been involved in the collection of information, then you do not need to complete this form.

If your evaluation does not comply with any one or more of the points below, please contact the Chair of the School of Computing Science Ethics Committee (matthew.chalmers@glasgow.ac.uk) for advice.

If your evaluation does comply with all the points below, please sign this form and submit it with your project.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.

   Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback

2. The experimental materials were paper-based, or comprised software running on standard hardware.

   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, laptops, iPads, mobile phones and common hand-held devices is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

   Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives were offered to the participants.

   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

(a) Signed Ethics Checklist (Part 1/2)

5. No information about the evaluation or materials was intentionally withheld from the participants.
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

6. No participant was under the age of 16.
   Parental consent is required for participants under the age of 16.

7. No participant has an impairment that may limit their understanding or communication.
   Additional consent is required for participants with impairments.

8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. All participants were informed that they could withdraw at any time.
   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

10. All participants have been informed of my contact details.
    All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.

11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions.
    The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. In cases where remote participants may withdraw from the experiment early and it is not possible to debrief them, the fact that doing so will result in their not being debriefed should be mentioned in the introductory text.

12. All the data collected from the participants is stored in an anonymous form.
    All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Project title: Visualising Volume (for COMPSCI5018: MSc Development Project for IT+)

Student's Name: Emer Sweeney

Student Number: 2191611s

Student's Signature Émer Sweeney

Supervisor's Signature

Date 03/12/2021

Ethics checklist for projects

(a) Signed Ethics Checklist (Part 2/2)

Bibliography

[1] Google expeditions. https://edu.google.co.uk/expeditions/ar/#about.

[2] Reading eggs. https://readingeggs.co.uk/.

[3] A. Abubakar. *Volume calculator-3D Shapes, geometry calculator*. Version 1.2 [Mobile App], Published: 2021. Available at: Google Play, downloaded: 3 October 2021.

[4] T. Chuang and W. Chen. Effect of computer-based video games on children: An experimental study. *Education Technology & Society*, 12-2:1–10, 2009.

[5] Scottish Government Education Scotland. Curriculum for excellence: Experiences and outcomes. 2018.

[6] E. Geslin, L. Jégou, and D. Beaudoin. How color properties can be used to elicit emotions in video games. *International Journal of Computer Games Technology*, 2016, 2016.

[7] S. Ghavifekr and W.A.W. Rosdy. Teaching and learning with technology: Effectiveness of ict integration in schools. *International Journal of Research in Education and Science (IJRES)*, 1(2):175–191, 2015.

[8] S. F. Griffith, M. B. Hagan, P. Heymann, B. H. Heflin, and D. M. Bagner. Apps as learning tools: A systematic review. *Pediatrics*, 145(1), 2020.

[9] H. J. Kell, D. Lubinski, C. P. Benbow, and J. H. Steiger. Creativity and technical innovation: Spatial ability's unique role. *Psychological Science*, 24, No. 9, 2013.

[10] Ofcom. Children and parents: Media use and attitudes report. 2020/2021.

[11] Oracle. *Lesson 8: Object-Oriented Programming*. https://www.oracle.com/java/technologies/oop.html. Accessed 02/12/21.

[12] M. Papastergiou. Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation. *Computers & Education*, 52:1–12, 01 2009.

[13] V. Robert. *CalculateVolume*. Version 1.1 [Mobile App], Published: 2016. Available at: Apple App Store, downloaded: 3 October 2021.

[14] Scrum.org. What is scrum?

[15] Sensor Tower. Worldwide mobile education app downloads from 1st quarter 2017 to 1st quarter 2020, by platform (in millions). 2020.

[16] L. S. Vailshery. Global market share held by tablet vendors 2011-2021. 2021.

[17] B. N. Verdine, R. M. Golinkoff, K. Hirsh-Pasek, and N. S. Newcombe. Links between spatial and mathemtical skills across the preschool years. *Monographs of Society for Research in Child Development*, 82, No. 1, 2017.

[18] J. Wai, D. Lubinski, and C. P. Benbow. Spatial ability for stem domains: Aligning over 50 years of cumulative psychological knowledge solidifies its importance. *Journal of Educational Psychology*, 101, No. 4:817–835, 2009.

[19] Y. C. Yang. Building virtual cities, inspiring intelligent citizens: Digital games for developing students' problem solving and learning motivation. *Computers & Education*, 59(2):365–377, 2012.

[20] Learn Teach Explore Sp. z o. o. *Shapes 3D Geometry Learning*. Version 2.3.7 [Mobile App], Published: 2012. Available at: Apple App Store.