Game Development for Smart Twisty Puzzles

by

Jacob Hreshchyshyn

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved June 2023 by the
Graduate Supervisory Committee:

Ajay Bansal, Chair
Alexandra Mehlhase
Tyler Baron

ARIZONA STATE UNIVERSITY

August 2023

ABSTRACT

There exists extensive research on the use of twisty puzzles, such as the Rubik's Cube, in educational contexts to assist in developing critical thinking skills and in teaching abstract concepts, such as group theory. However, the existing research does not consider the use of twisty puzzles in developing language proficiency. Furthermore, there remain methodological issues in integrating standard twisty puzzles into a class curriculum due to the ease with which erroneous cube twists occur, leading to a puzzle scramble that deviates from the intended teaching goal. To address these issues, an extensive examination of the "smart cube" market took place in order to determine whether a device that virtualizes twisty puzzles while maintaining the intuitive tactility of manipulating such puzzles can be employed both to fill the language education void and to mitigate the potential frustration experienced by students who unintentionally scramble a puzzle due to executing the wrong moves. This examination revealed the presence of Bluetooth smart cubes, which are capable of interfacing with a companion web or mobile application that visualizes and reacts to puzzle manipulations. This examination also revealed the presence of a device called the WOWCube, which is a 2x2x2 smart cube entertainment system that has 24 Liquid Crystal Display (LCD) screens, one for each face's square, enabling better integration of the application with the puzzle hardware. Developing applications both for the Bluetooth smart cube using React Native and for the WOWCube demonstrated the higher feasibility of developing with the WOWCube due to its streamlined development kit as well as its ability to tie the application to the device hardware, enhancing the tactile immersion of the players with the application itself. Using the WOWCube, a word puzzle game featuring three game modes was implemented to assist in teaching players English vocabulary. Due to its incorporation of features that enable dynamic puzzle generation and resetting, players who participated in a user survey found that the game was compelling and that it exercised their critical thinking skills. This demonstrates the feasibility of smart cube applications in both critical thinking and language skills.

DEDICATION

To my family, for anchoring me during turbulent times and for their continuing demonstrations of their love of life.

ACKNOWLEDGMENTS

I would like to thank my advising committee, consisting of Dr. Ajay Bansal, Dr. Alexandra Mehlhase, and Dr. Tyler Baron, for their guidance, expertise, and patience shared throughout this endeavor. I would also like to thank my colleague and friend, Joseph Hale, whose contagious enthusiasm for twisty puzzles sparked my own interest. Finally, I would like to thank Dr. Kiser for guiding me towards the expression of the noble pursuit of selflessly pursuing the ideal.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This thesis presents a twisty puzzle (a.k.a. twizzle) application developed for the WOWCube [24], a novel alternative to other smart cube devices and applications that rely on a Bluetooth connection to a single-screen mobile device. The application titled *WordBox* is a word puzzle game that challenges players to rearrange the letters rendered on each screen such that the letters form a four-letter word on each face of the cube. With this application, this thesis shall demonstrate how *WordBox* improves upon the educational/entertainment value of existing twisty puzzle toys and devices with respect to heightening critical thinking and language skills.

Chapter 2 outlines the initial goals of the project and how those goals evolved as development platforms changed. Chapter 3 frames these project goals in context with the domain of twizzles and their intersection with computer science and application development. Chapters 4 and 5 outline the process, advantages, and challenges of leveraging React Native with and without Expo for game application development on mobile devices while Chapter 6 outlines the same when leveraging the WOWCube SDK for application development on its platform. Chapter 7 offers an evaluation of *WordBox* based on the results of a qualitative user survey. Chapter 8 responds to the evaluation of the application by presenting the future works that will improve upon the existing application. Chapter 9 concludes the thesis by concentrating the information provided in the previous sections to justify the preference of the WOWCube over existing smart cube/smartphone applications by focusing on the development experience, app design flexibility, player interaction ergonomics, and educational value.

CHAPTER 2

PROJECT GOALS

2.1 Initial Goals

The initial goal of this project was to work in collaboration with another 4+1 graduate student on a React Native application for Android and iOS devices. There would have been a data analytics focus and a game development focus. The data analytics focus on the project was to develop a system that would analyze how players solve 3x3x3 twizzles modeled after the Rubik's Cube and use that information to offer suggestions on how players can improve their solve time. The game development focus on the project was to develop a tactical game for the application that would teach players basic techniques on how to solve a cube using the topology of the level design. The React Native application would use a Bluetooth connection to link to one of several commercially available "smart cubes", which are physical twizzle toys that players can manipulate as with a regular twizzle to interact with a companion application.

The purpose of developing these application components was to facilitate both the process of learning how to solve twizzles and how to improve one's ability to solve twizzles given their current proficiency. These aims would contribute to the open-source twizzle community and would help in stoking the steady interest in traditional twizzles as well as the emergence of smart cubes like the GoCube [13].

These goals would ultimately change in response to the introduction of the WOWCube, a 2x2x2 twizzle featuring 24 LCD screens, one for each face square, used for displaying applications that respond to touch inputs and cube twists.

2.2 Updated Goals

The updated goal of the project is to leverage the WOWCube SDK to create a word puzzle game that challenges players to twist the cube to rearrange the letters into valid four-letter words. The purpose of developing this application is to document the process of working with the new device and to improve upon the educational value of the Rubik's cube while simultaneously making the cube easier to use as a teaching implement. In particular, the goal is to explore the cube's use as an educational tool for novel contexts such as language education. The attainment of these goals would allow developers new to WOWCube development to explore a set of suggestions on how they can navigate their own WOWCube project, demonstrate the versatility of the twizzle as a canvas for educational game development, and demonstrate the efficacy of smart cubes with integrated applications as an educational tool for basic language education.

To better illustrate the importance of the WOWCube as a 24-screen smart twizzle, it is important to consider the history of twizzles and the current state of the art in smart cube technology.

CHAPTER 3

OVERVIEW OF TWISTY PUZZLES

3.1 History of Twisty Puzzles

The history of twizzle cubes can be traced back to Dr. Larry D. Nichols, who invented the first 2x2x2 twisty puzzle cube and patented the toy in 1972 [1]. Aside from describing the familiar setup of a 2x2x2 twizzle comprising eight cubes with the aim of rearranging the cubes such that the arrangement creates a single-colored face on each face of the cube, the patent makes further claims about the possibility of a sphere-shaped 2x2x2 twizzle as well as a 2x2x3 twizzle in the shape of a rectangular prism [12] (see Figure 1).

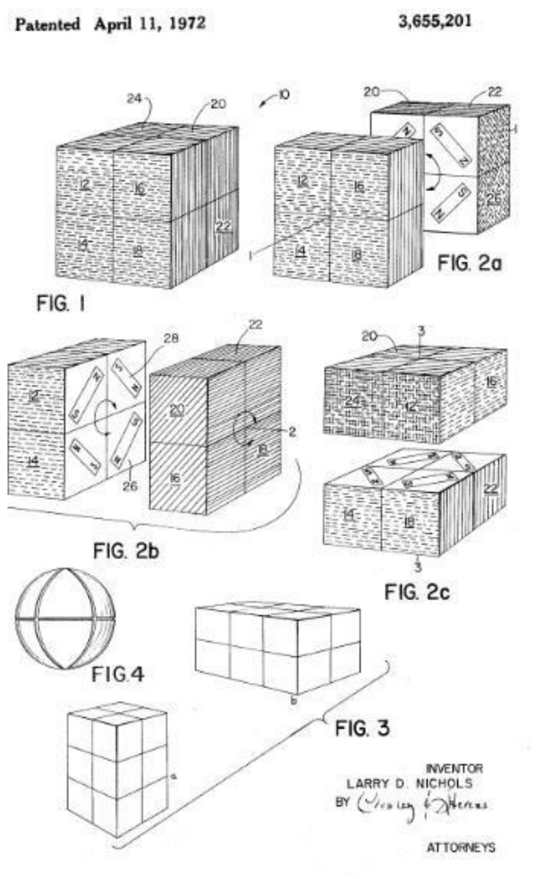


Figure 1. An image of Nichols's cube patent [12].

Despite the patent's attempt to claim the possibility of twizzles larger than the 2x2x2, Ernő Rubik would succeed in claiming his own 3x3x3 twizzle in 1975 [1], which would spark massive interest in the toy and lead to the emergence of a cubing community.

Notable figures in the cubing community include David Singmaster, an American-British mathematician who contributed to the collective pursuit of mastering the cube with a book that introduced cube move notation [18], which is commonly used today in the cubing community to assist enthusiasts in learning algorithms that can help in solving a cube, among other applications. Other prominent figures include Herbert Kociemba, who utilized this same notation to prove via computation that a 3x3x3 Rubik's cube requires 20 moves or fewer to be solved [17]. This number is also known as God's Number.

2017-2018 saw the introduction of the first smart cube devices, including one from GoCube on Kickstarter. The device would connect via Bluetooth to a companion application for mobile devices that would teach players how to solve a Rubik's cube and offer a host of cube-related games and activities (see Figure 2).

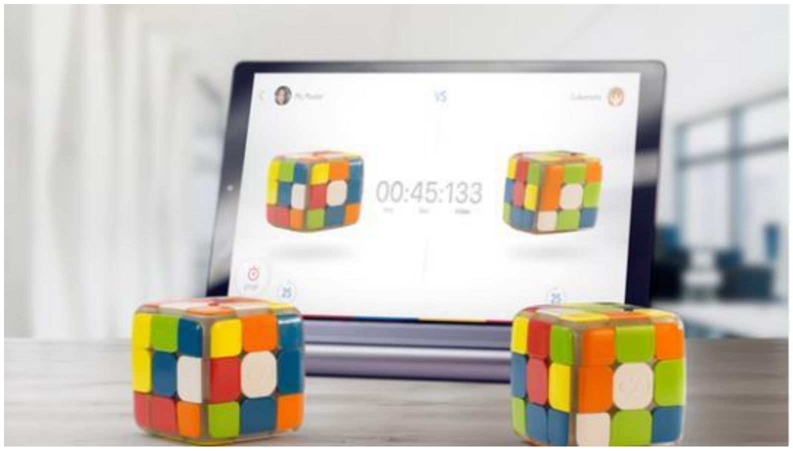


Figure 2. Promotional image of GoCube [14].

The product generated massive interest, resulting in GoCube's initial backing goal to be met just 30 minutes after the project was launched on Kickstarter [14]. During and since GoCube's introduction, other competitors would enter the market, including Giiker [8] and GAN [7], offering their own versions of the smart cube with varying tactile performance features and mobile applications.

In response to the emergence of smart cubes and their companion applications, a twizzle open-source community formed to allow cubers to hone their solving skills without being tied down to one individual product, since the main competitors in the smart cube market developed applications that lacked cross-cube compatibility. One of the individuals spearheading a branch of this movement is Lucas Garron, a Stanford graduate who collaborated with Tom Rokicki, one of Kociemba's collaborators who determined God's Number to be 20, to develop Twizzle.net [10] (see Figure 3).

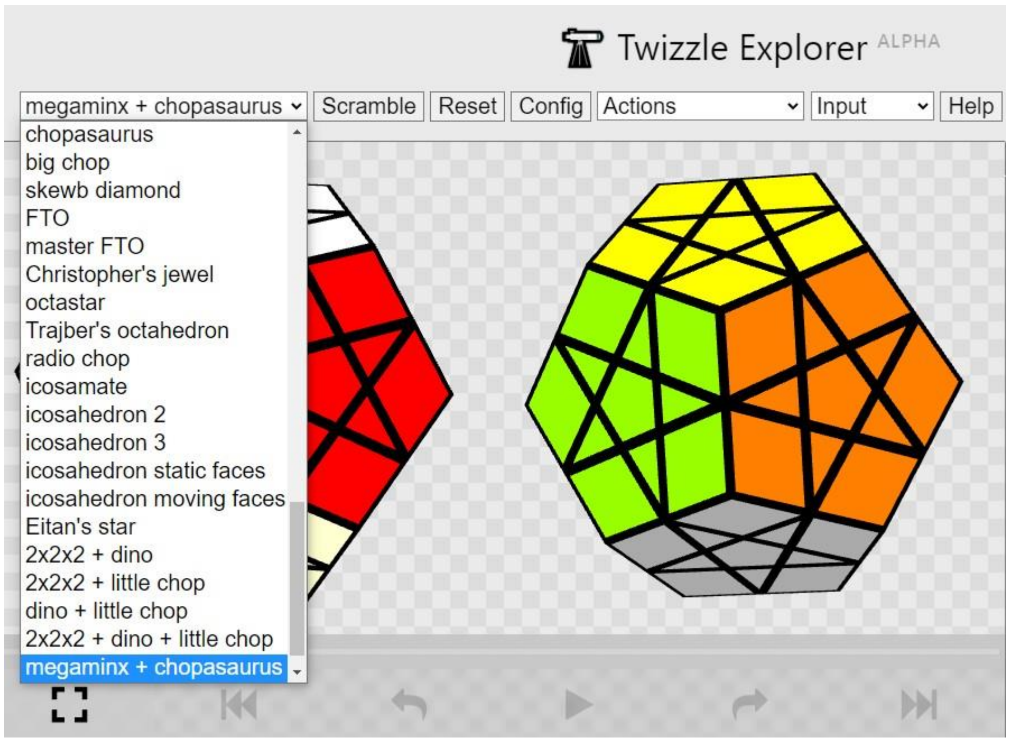


Figure 3. Twizzle.net rendering a non-WCA puzzle [22].

Twizzle.net would be designed as an open-source, universal virtual cube framework that focuses on consolidating the needs of most cubers, including timers, solve data visualization, and virtual versions of all sorts of twisty puzzles, including those supported by the WCA [23] as well as some that are not [22]. Importantly, Twizzle.net would be designed to make intuitive how moves and cube algorithms are performed in a virtual context, whether that be using a mouse to click and drag, using a keyboard, using a mobile device touch screen, or using a Bluetooth smart cube [10].

Meanwhile, inventor Ilya Osipov and his son Savva began development on a new type of smart cube called the WOWCube in 2016 [24] (see Figure 4).

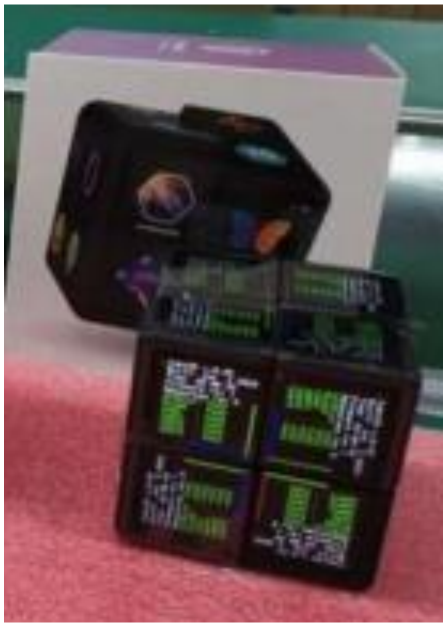


Figure 4. Promotional image of WOWCube [24].

The device looks and handles like a 2x2x2 Rubik's cube that features 24 3D articulated IPS screens, a rechargeable Lithium-ion battery set, 8 speakers, and a 6-axis IMU. It utilizes a custom operating system called CubiOS that runs on an SoC and even features wireless communication between cubes using Bluetooth 4.2 and Bluetooth LE

[25]. Designed as an entertainment system, the device features many novel variations of popular mobile games, such as *Cut the Rope* and *2048*, both of which leverage the unique topology of the twisty puzzle to innovate upon their respective gameplay loops [26].

Worth noting are some key functional differences between conventional 2x2x2 Rubik's cubes, Bluetooth smart cubes, Twizzle.net, and the WOWCube. These differences can be categorized as Application Integration, Ease of Resetting, and Scalability. Application Integration refers to how well an application associated with the puzzle (if a puzzle supports an application) integrates or interfaces directly with the physical topology of the cube. Ease of Resetting refers to how easy it is to revert the cube back to a solved state. The fewer manual inputs or mental effort needed to reset the cube, the easier it is to reset. Finally, scalability refers to how robust the puzzle is at scaling to higher puzzle dimensions. For example, a puzzle that is capable of representing a 2x2x2 puzzle in one instance and a 3x3x3 puzzle in another instance has some scalability. Table 1 summarizes these key differences.

	APPLICATION INTEGRATION	EASE OF RESETTING	SCALABILITY
CONVENTIONAL 2X2X2 CUBE	None	Difficult	Unscalable
BLUETOOTH SMART CUBE	Moderate	Moderately easy	Unscalable
TWIZZLE.NET	Moderate	Easy	Extensive
WOWCUBE	Extensive	Easy	Unscalable

Table 1. Comparison of twizzle functionality.

The conventional 2x2x2 puzzle is the least robust of these twisty puzzles since it features no companion application, requires manually solving the cube in order to reset the puzzle, and is unable to represent more complex puzzles due to its physical composition.

The Bluetooth smart cube puzzles, which include cubes from Giiker, GoCube, Gan, etc., have moderate application integration. Not only do they feature a companion application for their cubes, but they also respond to inputs from a physical cube device and project those responses onto a mobile application. If the application were built into the device itself, it would have a higher application integration. Furthermore, they are moderately easy to reset since most companion applications offer players instructions on how to solve their cube from the current state of the cube in order to assist in calibrating the cube for other application game modes. Finally, because of the physical composition of the smart cube device itself, these smart cubes are also not scalable.

Twizzle.net has moderate application integration because of its ability to receive inputs from existing smart cubes and projecting those inputs onto a separate screen. It is easy to reset since, at the click of a button, players can generate scrambles and reset scrambles to a solved state. Finally, Twizzle.net is extensively scalable since it features 68 types of puzzles to simulate and play with [22].

Finally, WOWCube has extensive application integration since the application is rendered directly on the device itself. The WOWCube is also easy to reset, as long as the application supports resetting and scramble generation as *WordBox* does. However, the tradeoff of extensive application integration is that the device does not scale well with other types of twisty puzzles.

3.2 Related Work

There exists limited research on the use of twisty puzzles in formal educational contexts. Most literature touches on the anecdotal benefits of leveraging Rubik’s cubes to supplement spatial reasoning and problem-solving abilities, as seen in Rohrig’s observations on high school science students [16], as well as the potential of Rubik’s cubes to assist in memorization, such as Mazzatenta’s experience in music learning and recital preparation [11]. Some research explores the benefits of using Rubik’s cubes to assist in teaching abstract mathematical concepts, such as group theory [4]. However, there does not yet exist any extensive research on the potential benefits of leveraging smart cube devices to assist in language learning.

CHAPTER 4

PROJECT DEVELOPMENT IN REACT NATIVE WITH EXPO

From the initial project goals, project development began as a React Native mobile application that would leverage Expo to facilitate development. React Native was initially considered since it is a JavaScript framework, which would theoretically allow for easy integration of other open-source JavaScript technologies that would boost development specifically for a twisty puzzle application. Such technologies include libraries associated with Twizzle.net, which is a framework written in vanilla JavaScript for web-based applications [5]. In addition to providing a host of functionality for rendering twisty puzzles and generating scrambles for those puzzles, these libraries would also enable Bluetooth support for various smart cubes, thereby signaling the potential of greatly reducing the overhead of developing these base features and enabling extended development on the earlier described cube solving analytics and tactical gameplay features.

Yet another step towards reducing overhead came in considering Expo as the project's development platform. Being an open-source framework for applications that run natively on Android, iOS, and the web, Expo would have simplified the use of libraries native to Android and iOS via its SDK, which contains a set of libraries compatible with both platforms [6]. Expo would not only have reduced the need to develop custom libraries to enable functionality for each mobile platform, but would also have enabled a more robust development environment since building iOS applications would not need a macOS development environment, but would instead leverage Expo's

EAS Build cloud service, all while using the Expo Go app on a physical or emulated device to test the application [6].

Unfortunately, while Expo does simplify the building and native compatibility issues of the two main mobile platforms, at the time of project development, it did not contain any libraries allowing for simple integration of Bluetooth Low Energy, which is necessary in order for the application to communicate with the smart cubes. Additionally, compatibility issues arose when attempting to incorporate other potentially useful frameworks, such as React Native Game Engine [2] and three.js [21]. For these reasons, project development with Expo was short-lived and led to the exploration of app development using Bare React Native.

CHAPTER 5

PROJECT DEVELOPMENT IN BARE REACT NATIVE

Development in Bare React Native offered its own strengths and drawbacks. The main push towards leveraging Bare React Native came in its flexibility of incorporating native Bluetooth Low Energy libraries that would enable turns on smart cubes to be detected and interpreted by the application. Further, with the right configuration, libraries such as three.js and those associated with Twizzle.net could also be incorporated into the project to decrease development overhead.

Unfortunately, for the purposes of this twisty puzzle game development project, the challenges of developing in Bare React Native proved too great to overcome. These challenges include the need to develop using both Android Studio and XCode in order to build and deploy apps for both Android and iOS devices. This limited the game development portion of the project to an Android-based application since there were no macOS development tool options available with which to build and test the app without compromising the development process for the data analytics portion of the app. Additionally, there remained persistent integration issues with the Bluetooth and Twizzle.net libraries. Twizzle.net was particularly challenging to integrate due to its use of web components for rendering puzzles. While such components could be wrapped in a WebView, this wrapping added yet another layer of complexity in terms of editing those wrapped components to fit the needs and layout of the rest of the app.

Figure 5 demonstrates this issue. It illustrates the incorporation of one of the Twizzle.net libraries in an early version of the React Native cubing app using a WebView to contain the rendering. Despite the app successfully rendering the cube player, its scaling and positioning is off, which is partially the result of the scaling logic being wrapped deeply in the WebView, limiting the ability to dynamically update the scale and contents of the cube.

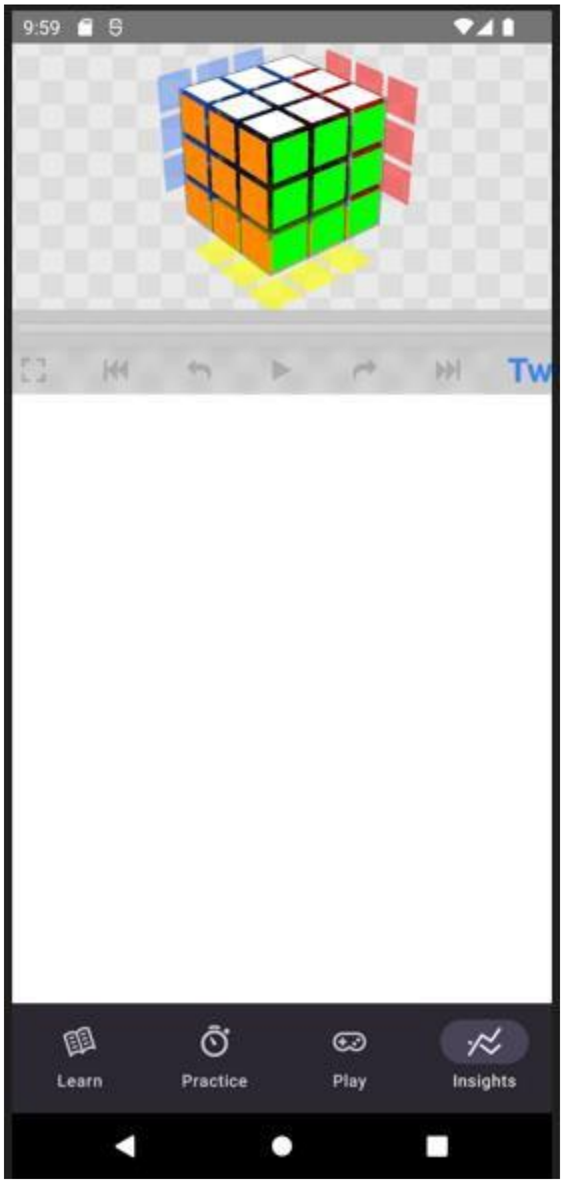


Figure 5. Rendering of Twizzle.net cube player in React Native.

Due to the compounding limitations imposed by the React Native development environment, research on existing smart cube technologies was revisited, leading to the exploration of Cubios Inc.’s WOWCube as a platform for novel edutainment games for twisty puzzles.

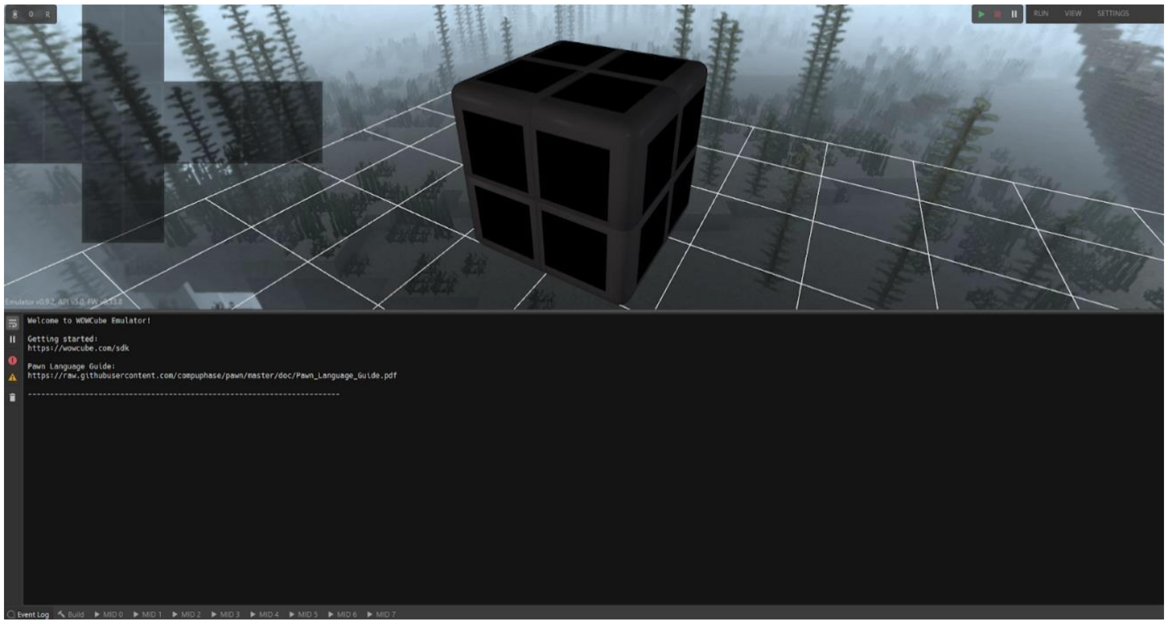
CHAPTER 6

PROJECT DEVELOPMENT IN WOWCUBE SDK

Ultimately, in order to better align with and realize this project's underlying goal, which is to develop an educational twisty puzzle game application, the WOWCube and its corresponding SDK was chosen. The WOWCube SDK aligns with this goal because of the features that facilitate the development of such an application. The primary features to consider separately are the WOWCube Emulator and the WOWCube API, which is accessed using the PAWN programming language [15].

6.1 WOWCube Emulator

One of the features included in the WOWCube SDK is the WOWCube Emulator, pictured in Figure 6.



```
Welcome to WOWCube Emulator!

Getting started:
https://wowcube.com/sdk

Pawn Language Guide:
https://raw.githubusercontent.com/compuphase/pawn/master/doc/Pawn_Language_Guide.pdf

----------------------------------------------------------------------
```

Figure 6. A depiction of the WOWCube Emulator.

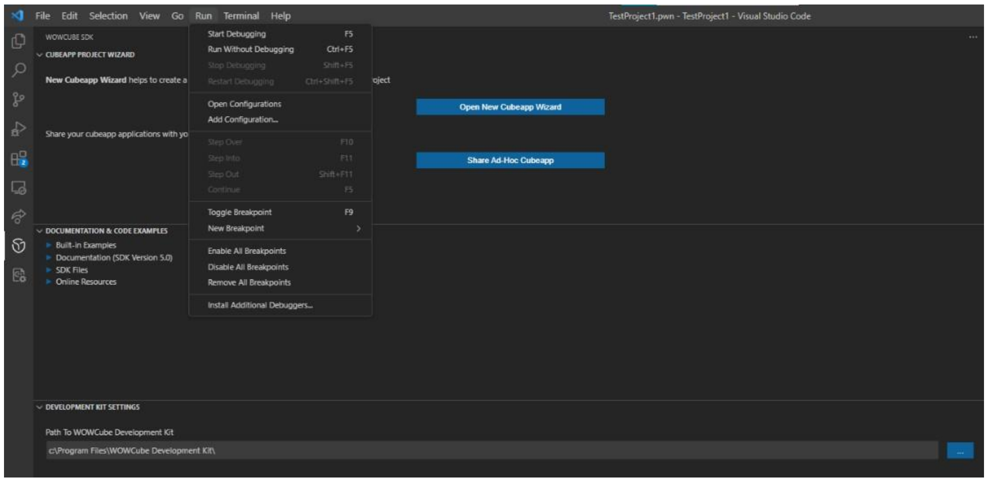
The WOWCube Emulator contains the following features:

- An interactive 3D model of the WOWCube, whose faces can be twisted by clicking and dragging with the mouse and whose 24 LCD squares can be tapped by clicking on the desired square.
- A collection of logging tools to facilitate debugging, including an Event Log, a Build Log, and eight module logs, each of which corresponds to one of the WOWCube's modules. More information on modules can be found in Section 6.2.
- An unwrapped view of the cube faces to assist in visualizing all of the cube at once.
- A collection of configuration settings to facilitate debugging and to modify how the rendering of the 3D cube model and environment is visualized.
- Start, Stop, and Pause commands to test the cube application at runtime. It functions very much like a runtime environment in a game engine, such as Unity [19].

Taken together, these features make up a robust game engine for testing WOWCube applications, greatly reducing the configuration overhead present in mobile application development, which would have required the development of a bespoke visualization and testing environment before development on the game itself could begin in earnest. Further assistance in reducing app development overhead came in the form of WOWCube's API and its corresponding documentation.

6.2 WOWCube API

Developers can work with the WOWCube API via a VSCode plugin, which allows app boilerplate code to be generated easily. The plugin also enables VSCodes's run tools to communicate with the WOWCube Emulator so that development and testing with the emulator can all be done in VSCode (see Figure 2). To assist in leveraging the created scaffold code for a new cube app, the plugin contains helpful links to example applications, documented source code, and other online resources on the SDK. Figure 7 also indicates the path to the WOWCube development kit, which enables VSCode to run the app within the WOWCube emulator.



New Cubeapp Wizard helps to create a … oject

Share your cubeapp applications with yo…

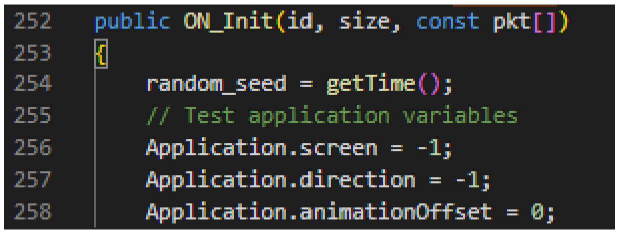
Path To WOWCube Development Kit

c:\Program Files\WOWCube Development Kit\

Figure 7. A picture of the WOWCube SDK plugin.

Worth noting is the SDK’s use of the PAWN programming language, which is a C-styled embedded scripting language intended to execute quickly and with a minimal footprint in memory. As stated in CompuPhase’s PAWN Language Guide, “PAWN is *quick* . . . PAWN is small . . . Unlike many languages, PAWN is not intended to write complete full-scale applications in. PAWN’s purpose is to *script* the functionality provided by an application or by a device. It is in purpose similar to Microsoft’s ‘Visual Basic for Applications’, only quicker and smaller (and without the installation hassle)” [15]. Given the WOWCube’s nature as a novel embedded device fitted with its own custom operating system on an “Energy Efficient SoC” [25], it is reasonable for a language like PAWN to be leveraged to script the applications designed for the device.

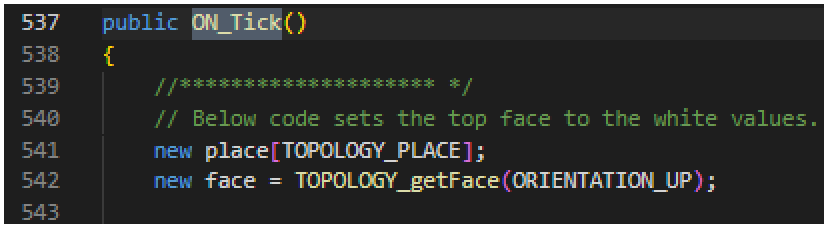
The API introduces several important callbacks necessary for most WOWCube applications, including *WordBox*. The first of these is ON_Init, which is the first routine to be called once the application starts (see Figure 8). Its purpose is to execute initialization instructions before the app begins running the main looping routines, such as ON_Tick, ON_PhysicsTick, and ON_Render. Note that some of Figure 3’s code is an artifact of implementing random cube scrambles, though this is implemented differently later in the callback and in other routines.



```
public ON_Init(id, size, const pkt[])
{
    random_seed = getTime();
    // Test application variables
    Application.screen = -1;
    Application.direction = -1;
    Application.animationOffset = 0;
```

Figure 8. A snippet of *WordBox’s* implementation of ON_Init.

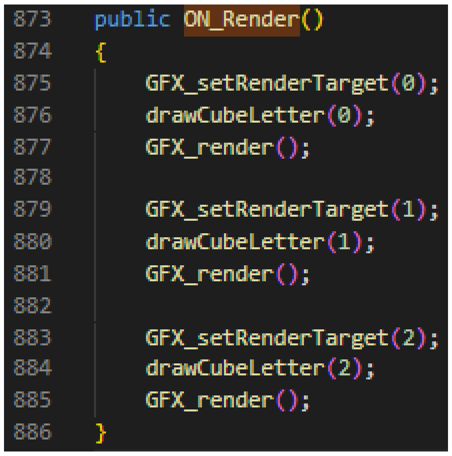
ON_Tick and ON_PhysicsTick are callbacks comparable to Unity’s Update and FixedUpdate event functions [20]. ON_Tick acts as the main application execution loop that executes its instructions as often as permitted by the application. ON_PhysicsTick behaves similarly, but updates as frequently as physics updates are permitted by the application. Since no physics calculations are required in *WordBox*, only ON_Tick was used as the application’s main execution loop (see Figure 9). The snippet here shows the beginnings of dynamically determining the orientation of the WOWCube’s squares in order to generate a solvable puzzle.



```
public ON_Tick()
{
    //******************** */
    // Below code sets the top face to the white values.
    new place[TOPOLOGY_PLACE];
    new face = TOPOLOGY_getFace(ORIENTATION_UP);

```

Figure 9. A snippet of *WordBox's* main execution loop.

The final callback pertinent to *WordBox* is ON_Render, whose primary responsibility is to draw images onto the screens of the WOWCube based on the set render target and the specified screen/module pairings. It is called immediately after execution of ON_Tick (see Figure 10). The implementation utilizes a custom function, drawCubeLetter, to render the appropriate letter to the correct square of the WOWCube. Notice the three separate calls for GFX_setRenderTarget, which is necessary to account for the three screens of each of the WOWCube's modules.

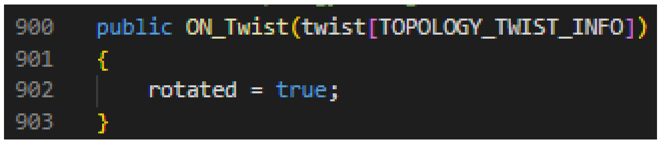


```
public ON_Render()
{
    GFX_setRenderTarget(0);
    drawCubeLetter(0);
    GFX_render();

    GFX_setRenderTarget(1);
    drawCubeLetter(1);
    GFX_render();

    GFX_setRenderTarget(2);
    drawCubeLetter(2);
    GFX_render();
}
```

Figure 10. *WordBox's* complete implementation of ON_Render.

Screen/module pairings are vital to specify when rendering images since these are used to uniquely identify each of the 24 squares, or "facelets", on the WOWCube. A module represents a single cubie, which is one of eight cubes that make up the entire 2x2x2 WOWCube. Each module contains three visible squares, or screens, and contains a unique hardware ID numbered 0 - 7. Using these topological properties of the WOWCube, combinations of module ID and screen number can be used to pick out individual facelets, thereby allowing developers to specify exactly where on the cube images are to be drawn.

The final set of API features important to *WordBox* is its collection of event handler callbacks. These include ON_Twist, ON_Shake, and ON_Tap. As the names suggest, ON_Twist executes whenever the topology of the WOWCube changes from a twist (see Figure 11). During development, experimentation was performed on this callback to implement gameplay features, such as changing the puzzle. Such logic would be refactored to the ON_Tap callback.



```
public ON_Twist(twist[TOPOLOGY_TWIST_INFO])
{
    rotated = true;
}
```

Figure 11. An implementation of the ON_Twist event handler callback.

ON_Shake executes its instructions whenever the WOWCube's IMU detects shaking (see Figure 12). Notably, this callback is capable of detecting how many times the WOWCube is shaken in a given period of time, allowing for different types of behavior. During development, experimentation was performed on this callback to implement a means of quitting the application and checking for whether a puzzle has been solved. The solution checker was moved to the ON_Tick callback. The quit functionality cannot be tested with the WOWCube emulator at this time, though Cubios Inc. plans to incorporate this in the emulator in the near future.

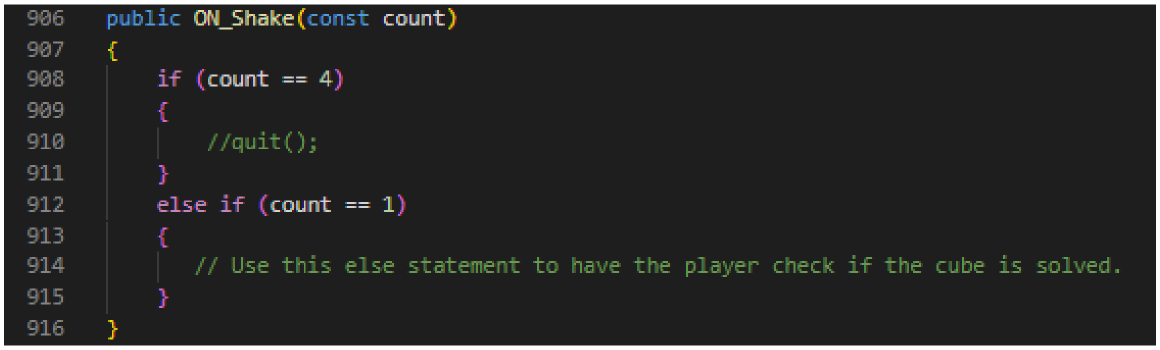

```
public ON_Shake(const count)
{
    if (count == 4)
    {
        //quit();
    }
    else if (count == 1)
    {
      // Use this else statement to have the player check if the cube is solved.
    }
}
```

Figure 12. An implementation of the ON_Shake event handler callback.

ON_Tap executes its instructions whenever a tap is detected on one of the WOWCube's 24 LCD facelets (see Figure 13). *WordBox* relies heavily on this callback to switch between three different game modes. This functionality is assisted by the callback's ability to determine how many taps are detected in a given period of time.

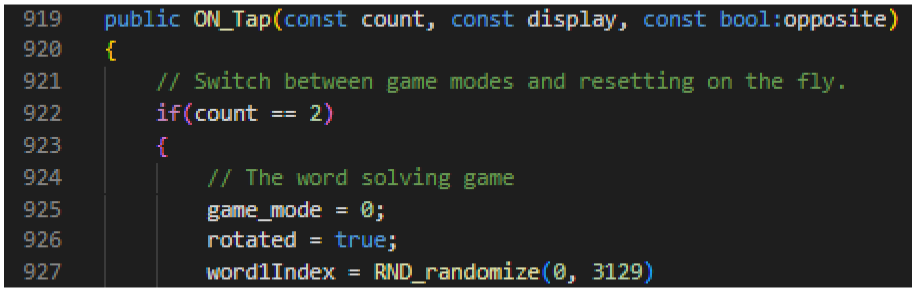
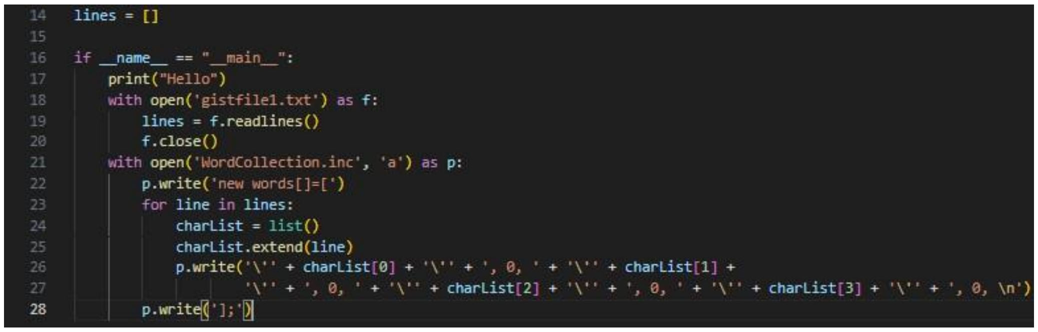

```
public ON_Tap(const count, const display, const bool:opposite)
{
    // Switch between game modes and resetting on the fly.
    if(count == 2)
    {
        // The word solving game
        game_mode = 0;
        rotated = true;
        word1Index = RND_randomize(0, 3129)
```

Figure 13. A snippet of an implementation of the ON_Tap event handler callback.

Using this combination of API tools, *WordBox* was developed to feature a word bank consisting of valid four-letter words and three unique game modes.

6.3 Implementation of the Word Bank

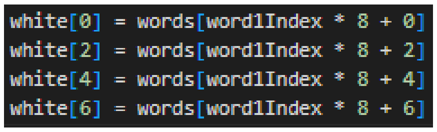
While the WOWCube is capable of network communication over Bluetooth with other WOWCubes, it does not currently support network communication over the internet, thereby hindering the ability of the WOWCube to access external APIs to receive word bank information. To circumvent this issue, it was necessary to enumerate a list of valid four-letter words and make it accessible as a dictionary stored in the WOWCube's memory. To avoid enumerating this list by hand, a simple Python script was used to read a file containing a complete list of those valid words [9] and write that data as a 1-dimensional list to a Pawn include file called WordCollection.inc (see Figure 14). Within the word collection itself, each word was formatted to be eight characters long in order to account for the possibility of accessing individual characters as strings. Thus, every other character has a terminating character, which is represented numerically with a zero.



```python
lines = []

if __name__ == "__main__":
    print("Hello")
    with open('gistfile1.txt') as f:
        lines = f.readlines()
        f.close()
    with open('WordCollection.inc', 'a') as p:
        p.write('new words[]=[')
        for line in lines:
            charList = list()
            charList.extend(line)
            p.write('\'' + charList[0] + '\'' + ', 0, ' + '\'' + charList[1] +
                    '\'' + ', 0, ' + '\'' + charList[2] + '\'' + ', 0, ' + '\'' + charList[3] + '\'' + ', 0, \n')
        p.write('];')
```

Figure 14. A simple Python script that reads word data from a text file and writes it to a Pawn include file.

This results in a dictionary containing 3,130 words, each character of which can be accessed by selecting a number between 0 and 3,129 to choose the desired word, multiplying that selection by an offset of 8, representing the number of characters in each word, and adding an offset of 0, 2, 4, or 6 to select the first, second, third, or fourth character, respectively (see Figure 15).



```
white[0] = words[word1Index * 8 + 0]
white[2] = words[word1Index * 8 + 2]
white[4] = words[word1Index * 8 + 4]
white[6] = words[word1Index * 8 + 6]
```

Figure 15. An example of accessing and storing word dictionary characters.

Using this means of storing and accessing word characters, puzzle scrambling algorithms can be applied to the selected words.

6.4 Implementation of the Scrambling Algorithms

In order to produce an intuitive set of scrambling algorithms that align with commonly used expressions for different types of puzzle twists, cube notation was employed as the underlying source of inspiration (see Figure 16).

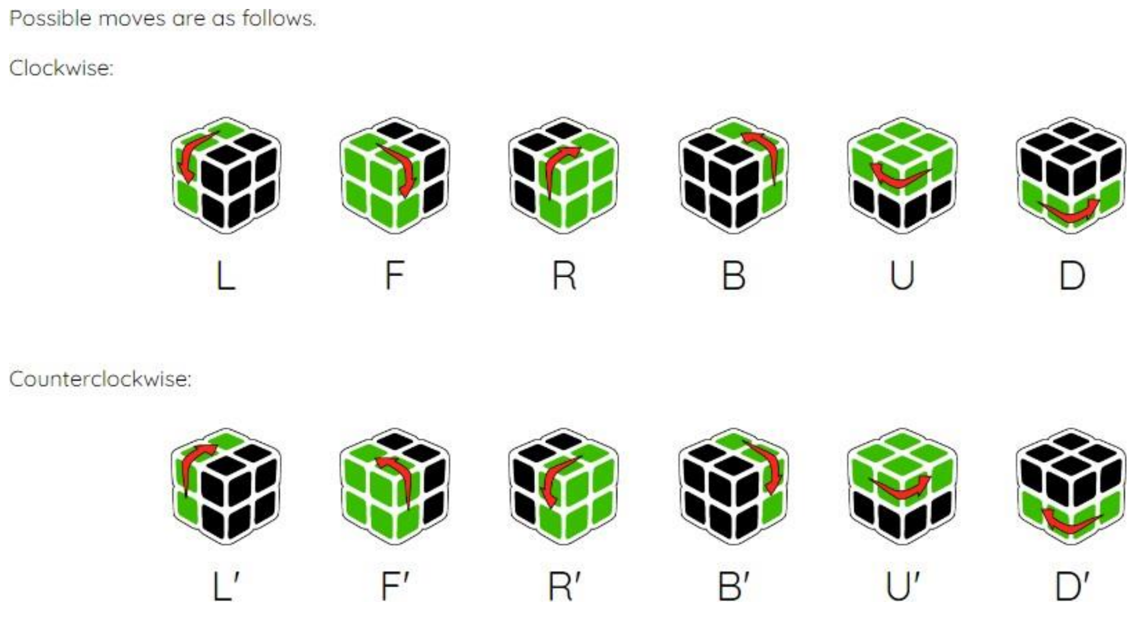
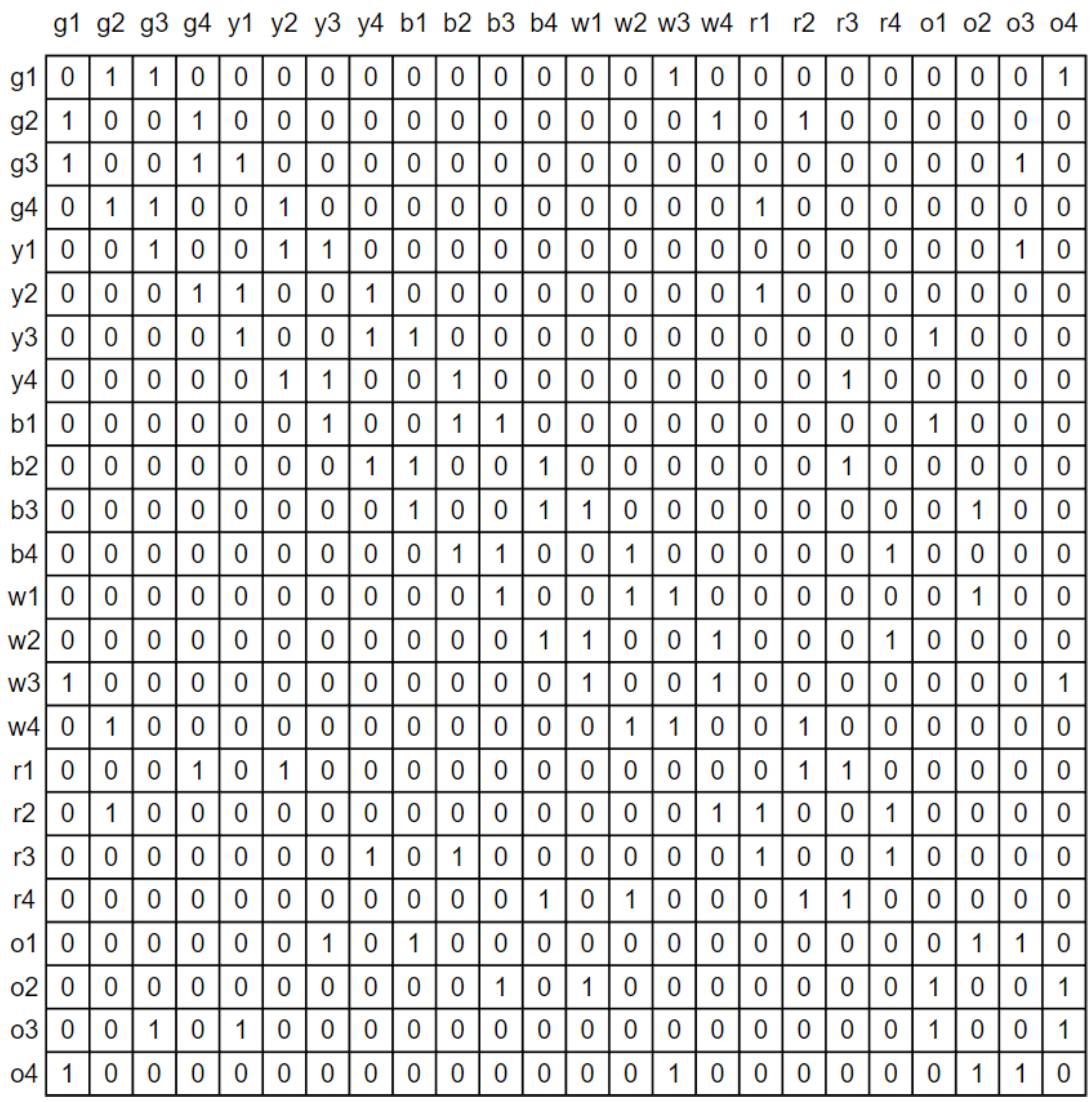


Figure 16. 2x2x2 cube quarter turn rotations [3].

Figure 16 demonstrates the six types of quarter turn rotations on a 2x2x2 cube, namely, L for a clockwise quarter turn on the left face of the cube, F for a clockwise quarter turn on the front face of the cube, R for a clockwise quarter turn on the right face of the cube, B for a clockwise quarter turn on the back face of the cube, U for a clockwise quarter turn on the upward face of the cube, and D for a clockwise quarter turn on the downward face of the cube. Their corresponding counterclockwise rotations are denoted with a prime symbol.

Using this notation, a scrambling algorithm can be devised that simulates each of the above quarter turns over an internal representation of 24 letters on the puzzle. There was some experimentation with how this internal representation would be encoded. An adjacency matrix was initially considered since it could theoretically be leveraged to quickly determine the relative positions of word letters, allowing for a robust solution checking system (see Figure 17).



	g1	g2	g3	g4	y1	y2	y3	y4	b1	b2	b3	b4	w1	w2	w3	w4	r1	r2	r3	r4	o1	o2	o3	o4
g1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
g2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
g3	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
g4	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
y1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
y2	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
y3	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
y4	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
b1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0
b2	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
b3	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0
b4	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0
w1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	1	0	0
w2	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	0	0	0	0
w3	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1
w4	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0
r1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
r2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0
r3	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0
r4	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	0	0
o1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
o2	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1
o3	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
o4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0

Figure 17. An adjacency matrix of cube squares.

However, such a structure does not lend itself easily to deterministically updating the state data given different types of cube twists. Thus, the encoding of cube square letter data was instead done by leveraging six character lists with each list representing one of the cube's faces. Using the word bank described in Section 6.3, each of these lists gets populated with the characters of six unique, randomly selected words. These lists then deterministically swap elements with each other depending on the quarter turn rotation being performed. To determine the nature of this value swapping, a physical prototype of *WordBox* was used to assist in mapping quarter turn rotations to new cube states (see Figure 18). Worth noting is that the yellow face is opposite the white face, the orange face is opposite the red face and the blue face is opposite the green face.

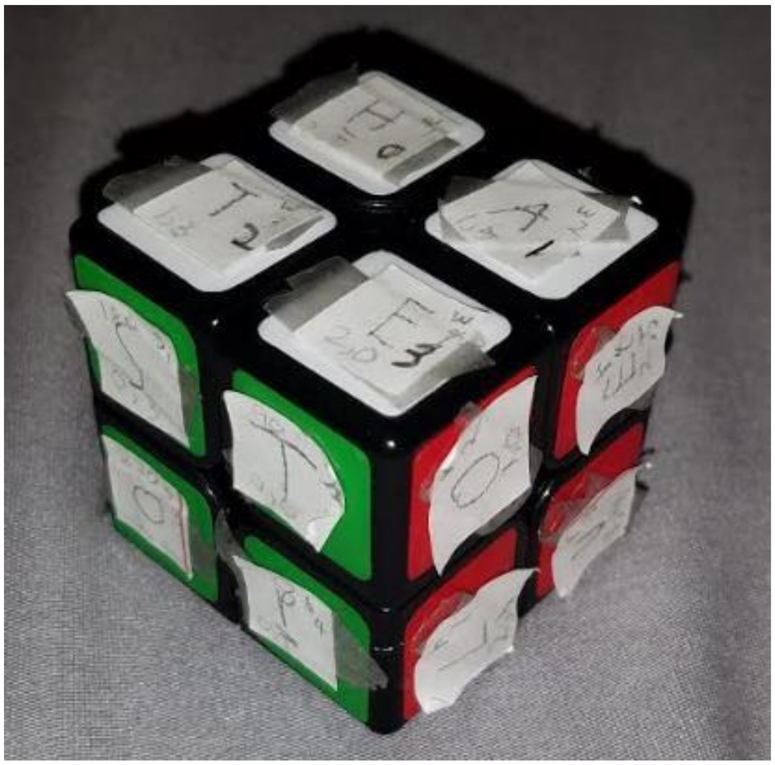


Figure 18. A physical prototype of *WordBox*.

Figure 19 demonstrates how letters on a 2x2x2 cube shift as different quarter turn rotations are applied. The center box shows an unaltered cube along with which letters correspond with what face color. Labeled arrows pointing to other boxes represent the six possible clockwise quarter turn rotations. Rectangles around specific box collections represent faces that remain meaningfully unchanged in the rotation. For example, an R move from the starting state produces an unchanged Orange face and a meaningfully unchanged Red state, since the characters of Red can be parsed to interpret the word “LOVE”.

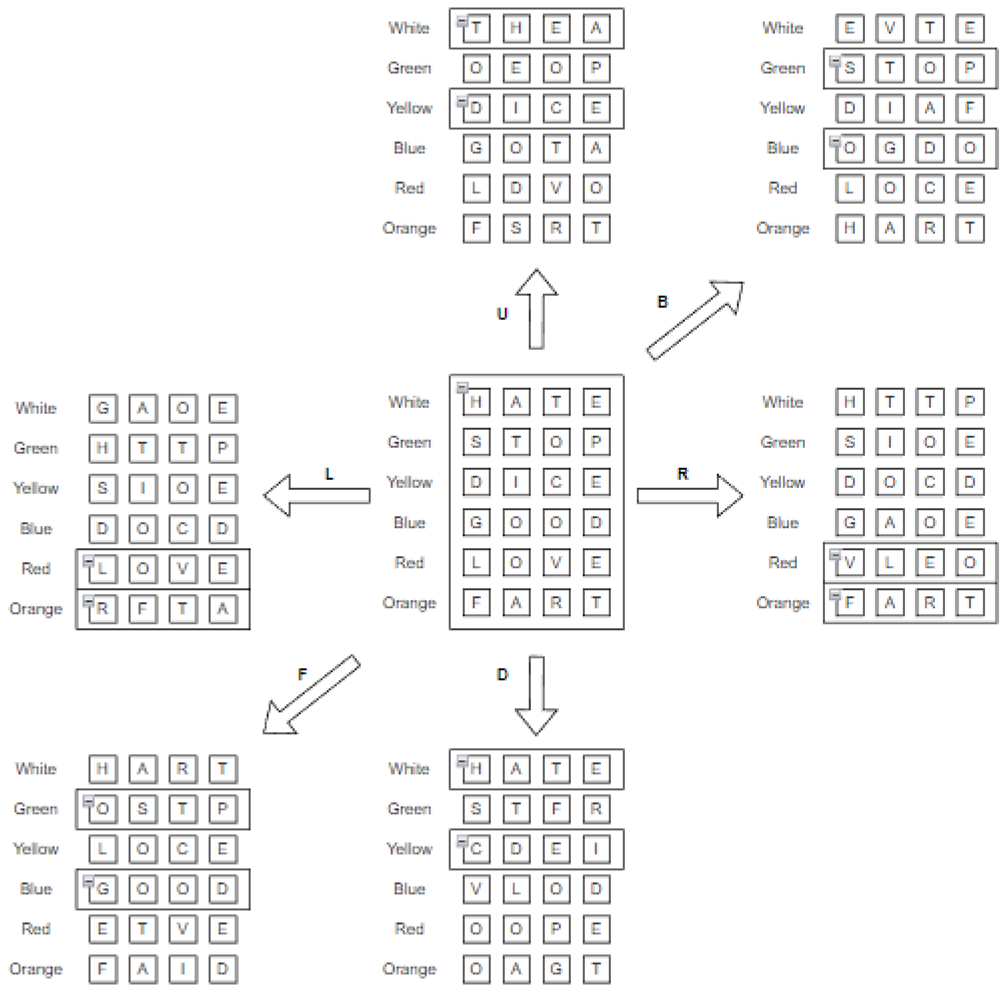
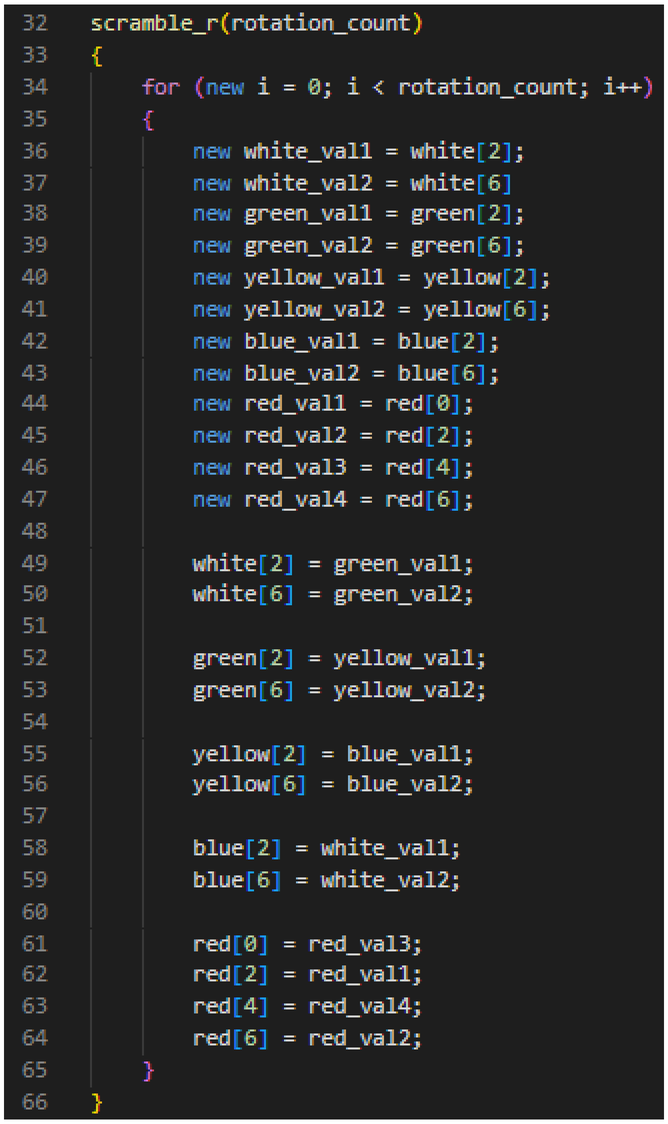


Figure 19. Quarter turn state mapping

By indexing each list element and tracking how list elements are exchanged between lists, generic list value swappers that correspond with specific quarter turn rotations can be encoded. Figure 20 demonstrates such an encoding, which executes an R move. Given the number of desired quarter turns, this R move can be executed that number of times.

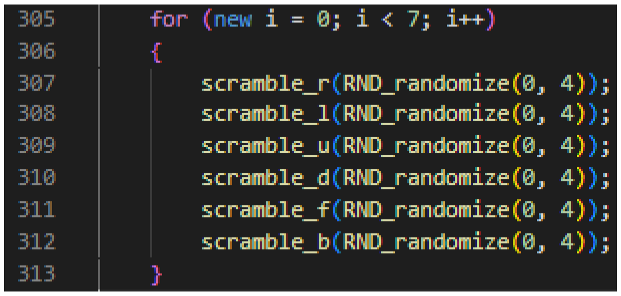


```
scramble_r(rotation_count)
{
    for (new i = 0; i < rotation_count; i++)
    {
        new white_val1 = white[2];
        new white_val2 = white[6]
        new green_val1 = green[2];
        new green_val2 = green[6];
        new yellow_val1 = yellow[2];
        new yellow_val2 = yellow[6];
        new blue_val1 = blue[2];
        new blue_val2 = blue[6];
        new red_val1 = red[0];
        new red_val2 = red[2];
        new red_val3 = red[4];
        new red_val4 = red[6];

        white[2] = green_val1;
        white[6] = green_val2;

        green[2] = yellow_val1;
        green[6] = yellow_val2;

        yellow[2] = blue_val1;
        yellow[6] = blue_val2;

        blue[2] = white_val1;
        blue[6] = white_val2;

        red[0] = red_val3;
        red[2] = red_val1;
        red[4] = red_val4;
        red[6] = red_val2;
    }
}
```

Figure 20. An R move encoding.

Functions representing the moves L, F, R, B, U, and D are sufficient for producing any desired cube rotation. There is no need to represent the counterclockwise quarter turn rotations explicitly since a counterclockwise quarter turn is equivalent to three of its complementary clockwise quarter turns (e.g. L' is equivalent to L3, or 3 L moves).

Finally, once all the relevant cube rotations are encoded, a random number generator can be employed to execute each type of twist a random number of times. Iteratively performing this random twisting will ensure a sufficiently scrambled puzzle. In Figure 21's implementation of this scrambling, the sequence of rotations is executed seven times. Notably, each individual rotation can be performed up to four times, which is equivalent to performing no quarter turn rotation.



```
for (new i = 0; i < 7; i++)
{
    scramble_r(RND_randomize(0, 4));
    scramble_l(RND_randomize(0, 4));
    scramble_u(RND_randomize(0, 4));
    scramble_d(RND_randomize(0, 4));
    scramble_f(RND_randomize(0, 4));
    scramble_b(RND_randomize(0, 4));
}
```

Figure 21. Iteratively executing a random scramble sequence.

After using this means of scrambling the face lists, the appropriate letters must be rendered on the appropriate facelets of the WOWCube. Since two of the three game modes of *WordBox* rely on indexing of the values of the face lists, it is necessary for the puzzle to dynamically determine the current topology of its facelets so that those letters are assigned to the correct facelets. Fortunately, the WOWCube's IMU and API tools facilitate this determination.

6.5 Topology Detection

To better illustrate how to determine the initial topology of the WOWCube, it is important to understand how face lists are to correspond with the cube's actual topology. Figure 22 illustrates an unwrapped 2x2x2 cube whose faces are colored according to each face list used to store and display letter data. Assuming that White faces Up and Green faces Forward, then the other face orientations are as follows: Yellow faces Down, Blue faces Back, Red faces Right (of the Green face), and Orange faces Left (of the Green face).

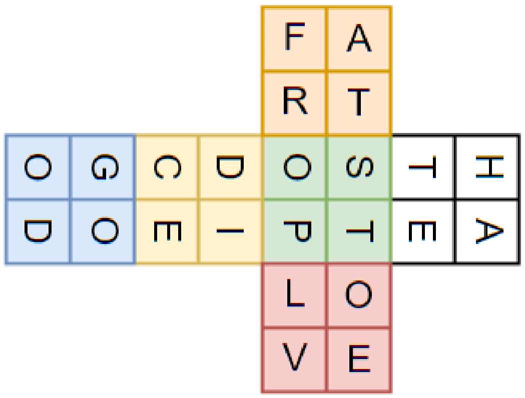
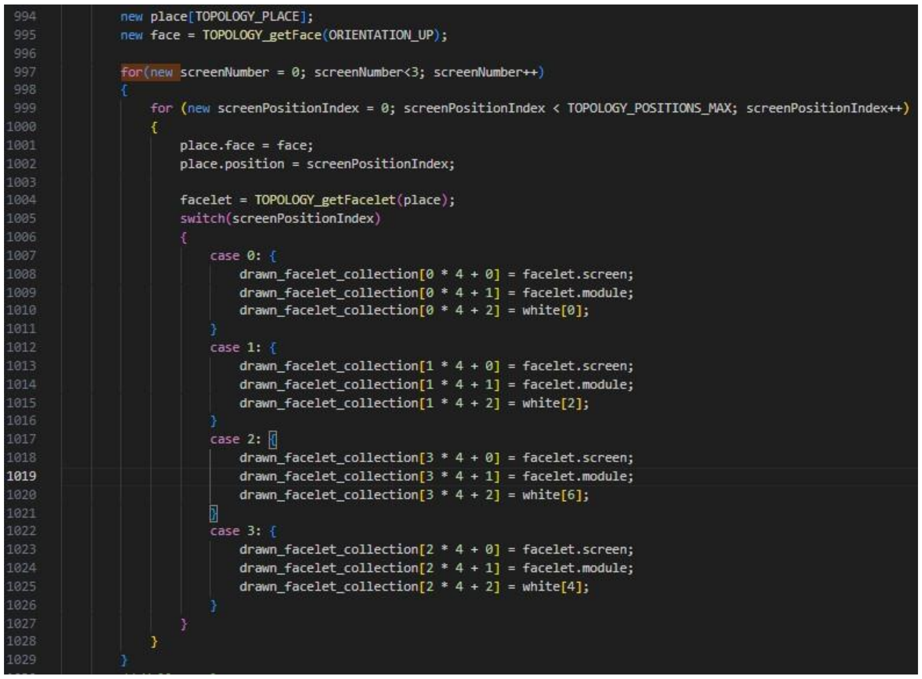


Figure 22. Face list projection.

Using Figure 22's face list projection, one can determine the orientation of each face of the WOWCube at runtime and assign the correct face list data to those faces. To do so, one of the faces on the cube must be selected as a reference face, which will be used to associate all the other face lists to faces. According to WOWCube's Device Orientation documentation, the WOWCube is able to determine unambiguously the top and bottom faces of the cube via its IMU's detection of Earth's gravity. The WOWCube also provides API tools that allow developers to access facelet information for faces facing up, down, front, back, left, and right. Therefore, in order to faithfully associate face list data with the correct faces, *WordBox* leverages WOWCube's

TOPOLOGY_getFace API function to get reference faces for the top and bottom faces of the cube. With these reference faces and corresponding facelets obtained, it searches for adjacent facelets using the TOPOLOGY_getAdjacentFacelet API function to associate the remaining face list data with the remaining facelets.

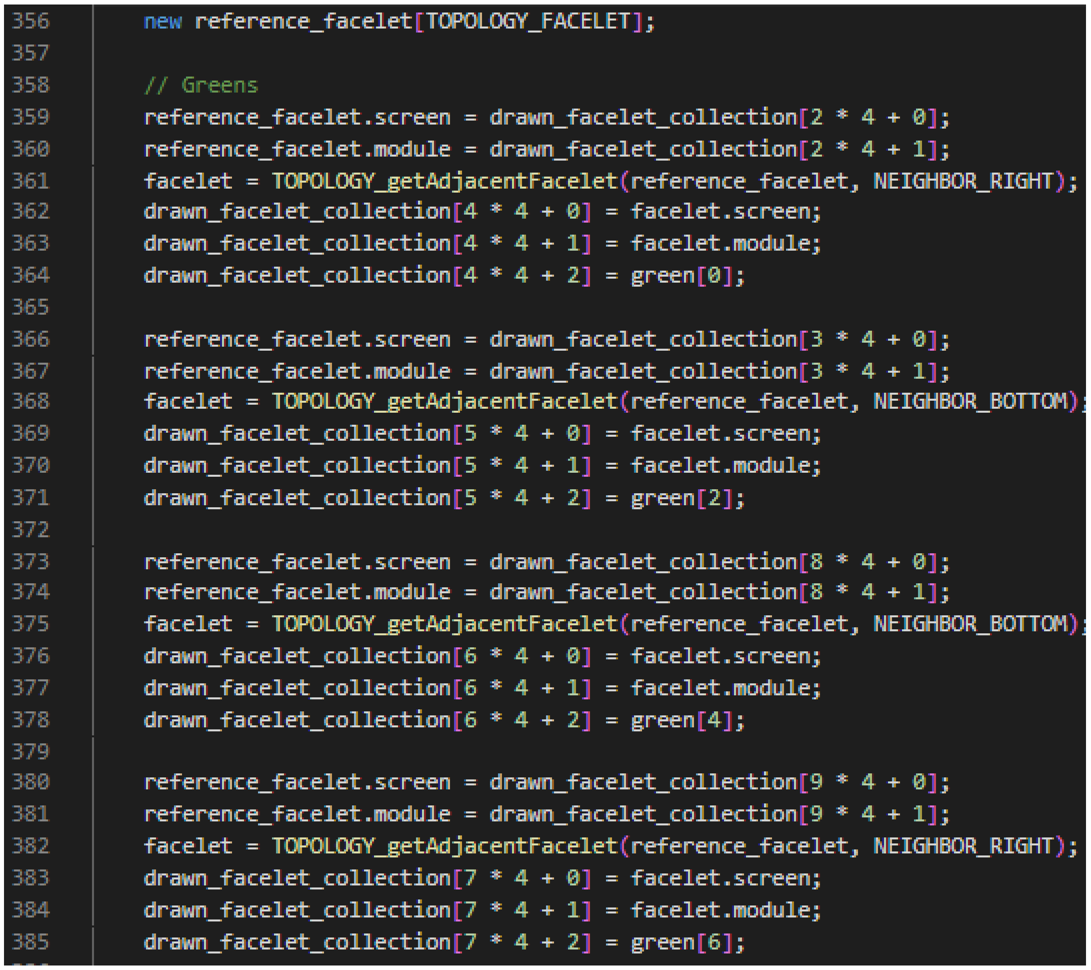
Figure 23 demonstrates the use of TOPOLOGY_getFace to determine how facelets are oriented. The code works by using a functionally 2-dimensional list called drawn_facelet_collection, which contains 24 lists with each list containing a screen number, a module number, a letter, and a flag used to determine whether the facelet should be colored in white or black. TOPOLOGY_getFace is used to determine which faces are facing up and down. Then, that facelet data, which includes a screen number and a module number, is stored in specific indices of the drawn_facelet_collection, which is implicitly partitioned into different zones representing different face orientations. As facelet data is assigned to the drawn_facelet_collection, letter data is also provided.



```
new place[TOPOLOGY_PLACE];
new face = TOPOLOGY_getFace(ORIENTATION_UP);

for(new screenNumber = 0; screenNumber<3; screenNumber++)
{
    for (new screenPositionIndex = 0; screenPositionIndex < TOPOLOGY_POSITIONS_MAX; screenPositionIndex++)
    {
        place.face = face;
        place.position = screenPositionIndex;

        facelet = TOPOLOGY_getFacelet(place);
        switch(screenPositionIndex)
        {
            case 0: {
                drawn_facelet_collection[0 * 4 + 0] = facelet.screen;
                drawn_facelet_collection[0 * 4 + 1] = facelet.module;
                drawn_facelet_collection[0 * 4 + 2] = white[0];
            }
            case 1: {
                drawn_facelet_collection[1 * 4 + 0] = facelet.screen;
                drawn_facelet_collection[1 * 4 + 1] = facelet.module;
                drawn_facelet_collection[1 * 4 + 2] = white[2];
            }
            case 2: {
                drawn_facelet_collection[3 * 4 + 0] = facelet.screen;
                drawn_facelet_collection[3 * 4 + 1] = facelet.module;
                drawn_facelet_collection[3 * 4 + 2] = white[6];
            }
            case 3: {
                drawn_facelet_collection[2 * 4 + 0] = facelet.screen;
                drawn_facelet_collection[2 * 4 + 1] = facelet.module;
                drawn_facelet_collection[2 * 4 + 2] = white[4];
            }
        }
    }
}
```

Figure 23. Orientation with TOPOLOGY_getFace.

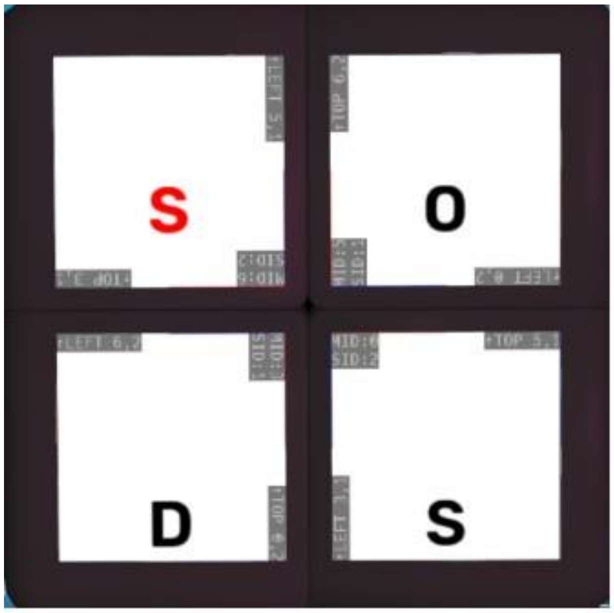
Figure 24 demonstrates the use of TOPOLOGY_getAdjacentFacelet to assign face list data to the remaining faces' facelets. This is necessary to ensure the correct assignment of letter data regardless of the ambiguous orientation of faces not facing up or down. This code snippet leverages the known upward-facing and downward-facing facelets stored in the drawn_facelet_collection to assign letter data to the Green, or forward-facing, facelets. Notice the implicit partitioning of drawn_facelet_collection into different orientations with indices 0 - 3 representing upward-facing facelets, indices 8 - 11 representing downward-facelets, and indices 4 - 7 representing the forward-facing facelets.

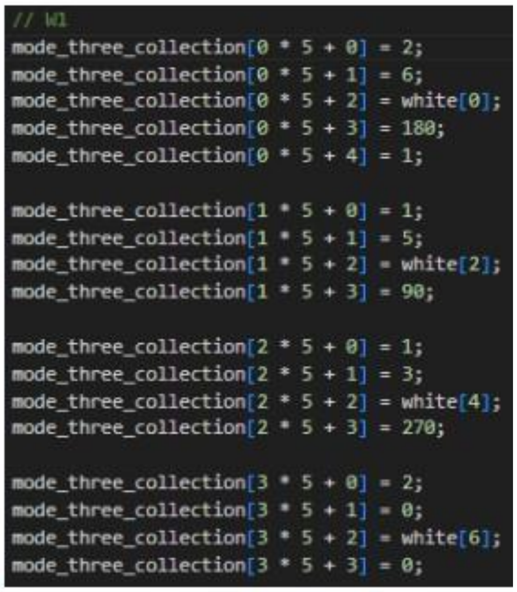


```
    new reference_facelet[TOPOLOGY_FACELET];

    // Greens
    reference_facelet.screen = drawn_facelet_collection[2 * 4 + 0];
    reference_facelet.module = drawn_facelet_collection[2 * 4 + 1];
    facelet = TOPOLOGY_getAdjacentFacelet(reference_facelet, NEIGHBOR_RIGHT);
    drawn_facelet_collection[4 * 4 + 0] = facelet.screen;
    drawn_facelet_collection[4 * 4 + 1] = facelet.module;
    drawn_facelet_collection[4 * 4 + 2] = green[0];

    reference_facelet.screen = drawn_facelet_collection[3 * 4 + 0];
    reference_facelet.module = drawn_facelet_collection[3 * 4 + 1];
    facelet = TOPOLOGY_getAdjacentFacelet(reference_facelet, NEIGHBOR_BOTTOM);
    drawn_facelet_collection[5 * 4 + 0] = facelet.screen;
    drawn_facelet_collection[5 * 4 + 1] = facelet.module;
    drawn_facelet_collection[5 * 4 + 2] = green[2];

    reference_facelet.screen = drawn_facelet_collection[8 * 4 + 0];
    reference_facelet.module = drawn_facelet_collection[8 * 4 + 1];
    facelet = TOPOLOGY_getAdjacentFacelet(reference_facelet, NEIGHBOR_BOTTOM);
    drawn_facelet_collection[6 * 4 + 0] = facelet.screen;
    drawn_facelet_collection[6 * 4 + 1] = facelet.module;
    drawn_facelet_collection[6 * 4 + 2] = green[4];

    reference_facelet.screen = drawn_facelet_collection[9 * 4 + 0];
    reference_facelet.module = drawn_facelet_collection[9 * 4 + 1];
    facelet = TOPOLOGY_getAdjacentFacelet(reference_facelet, NEIGHBOR_RIGHT);
    drawn_facelet_collection[7 * 4 + 0] = facelet.screen;
    drawn_facelet_collection[7 * 4 + 1] = facelet.module;
    drawn_facelet_collection[7 * 4 + 2] = green[6];
```

Figure 24. Orientation with TOPOLOGY_getAdjacentFacelet.

This topology detection is performed during application initialization via ON_Init and whenever a new puzzle using either Game Modes 1 or 2 is generated via ON_Tap. For Game Mode 3, dynamic topology detection is replaced by assigning letters to hardcoded screen/module pairs that correspond with the Emulator's representation of a solved cube (see Figure 25). The WOWCube Emulator allows developers to instantly reset the module positions of the cube. Enabling the debug overlay on the cube shows each facelet's screen/module pairing with their SIDs and MIDs. Knowing what SIDs and MIDs represent a solved cube, letter assignments of complete words can be made. For example, the first letter on the White, or upward facing, face would be assigned to the facelet whose SID is 2 and whose MID is 6. Notice how the fourth list element in each of the mode_three_collection entries assigns what represents a rotation value indicating how each letter ought to be oriented in order to appear as a legible word on the solved face of the puzzle.





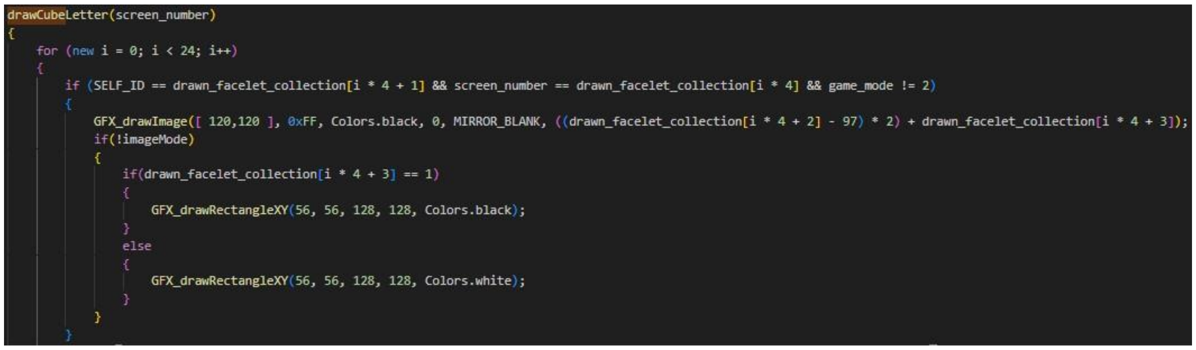
```
// W1
mode_three_collection[0 * 5 + 0] = 2;
mode_three_collection[0 * 5 + 1] = 6;
mode_three_collection[0 * 5 + 2] = white[0];
mode_three_collection[0 * 5 + 3] = 180;
mode_three_collection[0 * 5 + 4] = 1;

mode_three_collection[1 * 5 + 0] = 1;
mode_three_collection[1 * 5 + 1] = 5;
mode_three_collection[1 * 5 + 2] = white[2];
mode_three_collection[1 * 5 + 3] = 90;

mode_three_collection[2 * 5 + 0] = 1;
mode_three_collection[2 * 5 + 1] = 3;
mode_three_collection[2 * 5 + 2] = white[4];
mode_three_collection[2 * 5 + 3] = 270;

mode_three_collection[3 * 5 + 0] = 2;
mode_three_collection[3 * 5 + 1] = 0;
mode_three_collection[3 * 5 + 2] = white[6];
mode_three_collection[3 * 5 + 3] = 0;
```

Figure 25. Game Mode 3 letter assignment.

After leveraging either topology detection or the WOWCube Emulator to determine how letters correspond to facelets, images or text associated with those letters can be rendered onto those facelets.

6.6 Letter Rendering

Letter rendering is handled by the custom function drawCubeLetter shown in Figure 5. Depending on whether the player is playing with Game Mode 3, either images representing letters or plain text representing letters will be rendered according to the previously determined screen/module pairings. Figure 26 demonstrates letter rendering for Game Modes 1 and 2. For each of the 24 facelets, the cube module executing the code checks if its MID matches that contained in a drawn_facelet_collection element and if the SID passed to the function from the ON_Render callback matches that contained in that same drawn_facelet_collection element. Given a match, the GFX_drawImage API function is used to render an image that corresponds to the letter contained in the drawn_facelet_collection element.



```
drawCubeLetter(screen_number)
{
    for (new i = 0; i < 24; i++)
    {
        if (SELF_ID == drawn_facelet_collection[i * 4 + 1] && screen_number == drawn_facelet_collection[i * 4] && game_mode != 2)
        {
            GFX_drawImage([ 120,120 ], 0xFF, Colors.black, 0, MIRROR_BLANK, ((drawn_facelet_collection[i * 4 + 2] - 97) * 2) + drawn_facelet_collection[i * 4 + 3]);
            if(!imageMode)
            {
                if(drawn_facelet_collection[i * 4 + 3] == 1)
                {
                    GFX_drawRectangleXY(56, 56, 128, 128, Colors.black);
                }
                else
                {
                    GFX_drawRectangleXY(56, 56, 128, 128, Colors.white);
                }
            }
        }
```

Figure 26. A snippet of the letter rendering function for Game Modes 1 and 2.

Notice the last parameter of GFX_drawImage, which is meant to be an index to the desired image to be rendered. The program accesses the correct image index by leveraging the order of image files and the numerical representation of characters. First, the calculation in the last parameter exploits the alphabetical ordering of the 52 image filenames, with names ending in _inv being prioritized (see Figure 27). The higher the element in this ordering, the lower the index, starting at 0. Each filename faithfully describes each letter (e.g. A_inv.png represents the middle image while A.png represents the right image).

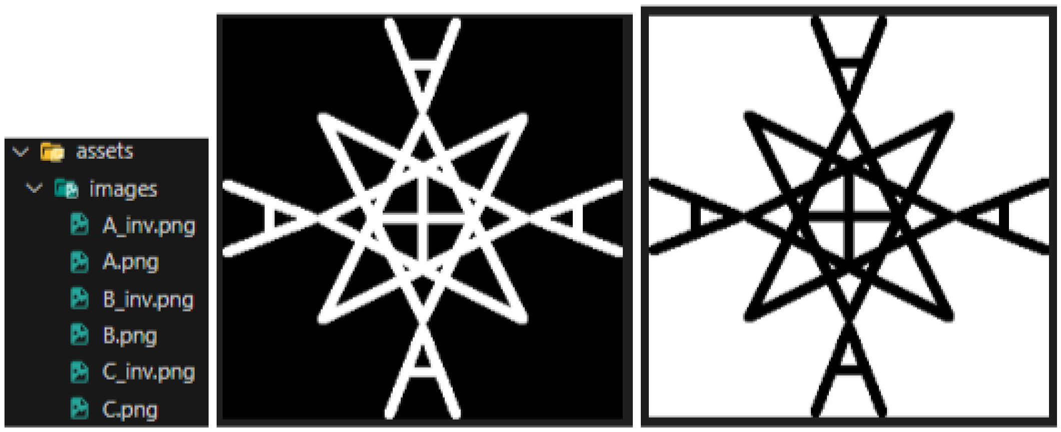


Figure 27. A partial depiction of the ordering of image filenames in WOWCube's Resource Identifier.

Next, in order to access these indexed images, the program takes the letter value of drawn_facelet_collection element and subtracts its ASCII representation by 97 since the lowercase characters are used internally to represent each letter. The result is multiplied by 2 to ensure coverage of all 52 letters. Finally, an offset of either 0 or 1, which is stored in the same drawn_facelet_collection element, is added to select between a black (unsolved) letter or a white (solved) letter. Figure 28 demonstrates an example of selecting an image corresponding to the lowercase letter c. According to the ASCII table, a lowercase c is encoded as 99. When applying the formula and adding the isWhite offset, the correct index 5, which corresponds with a white c, emerges.

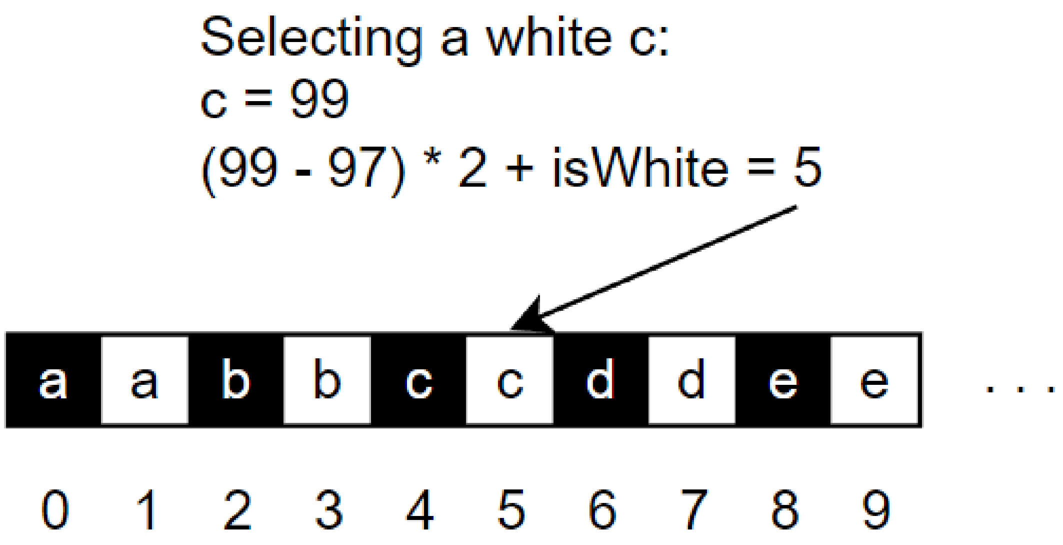


Figure 28. An example of the selection of a letter image to draw.

Finally, in order to allow players the option of removing visual clutter when interpreting the letters, the program leverages an imageMode boolean to determine whether to draw a rectangle over the symbol in the center of the image. Figure 29 demonstrates the result. The left side of the figure demonstrates the normal rendering of the image. The right side of the figure demonstrates the simplified rendering of the letters by drawing a black rectangle over the central symbol. The color of the rectangle matches the color of the background.

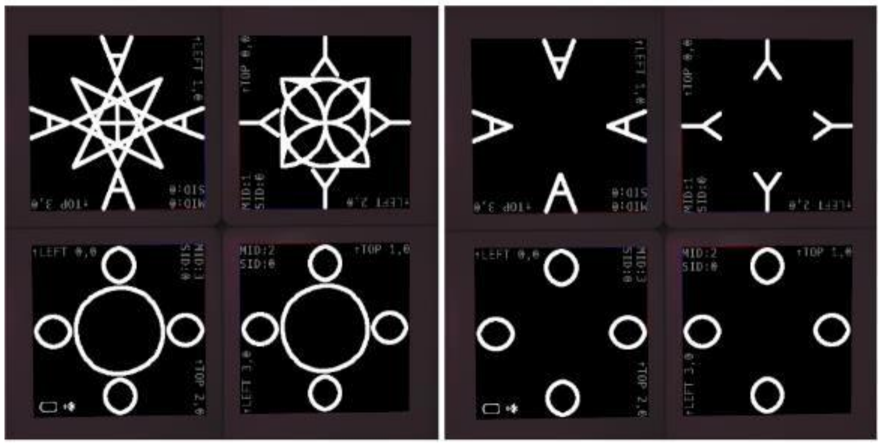
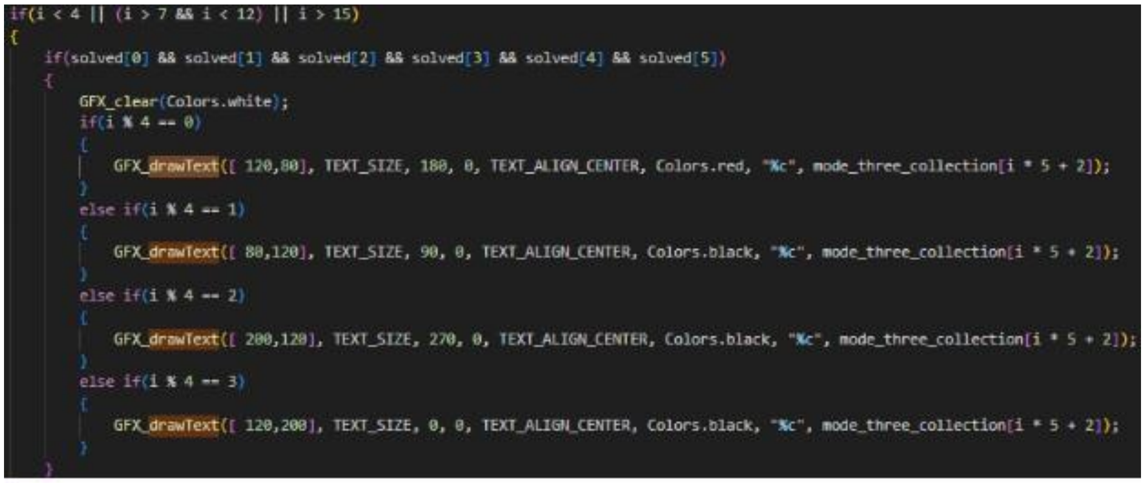
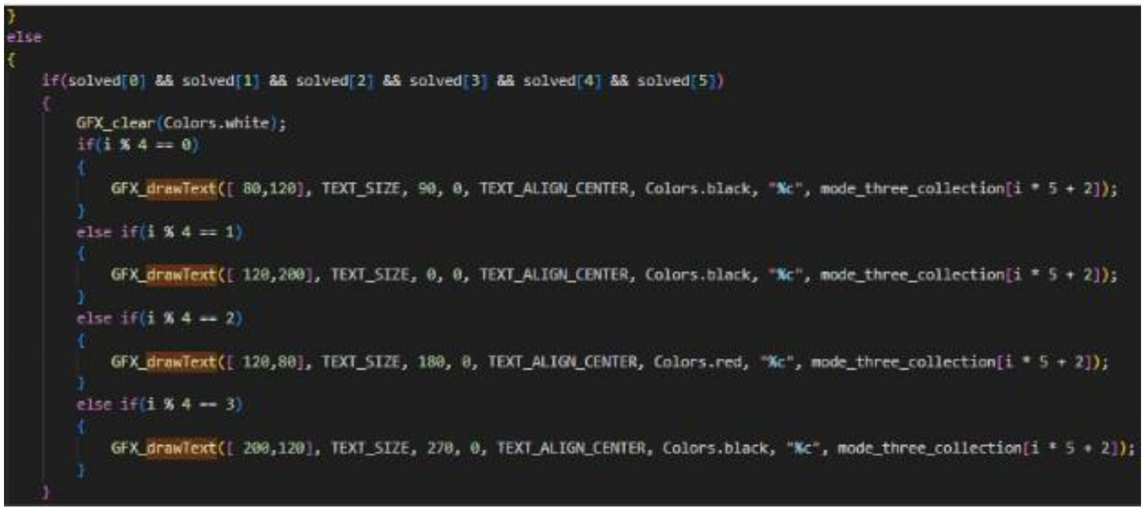


Figure 29. Symbol hiding functionality.

In the case of the game mode being set to 3, the program will render the letters in plain text using the GFX_drawText API function. Figure 30 demonstrates the relevant code snippets for rendering plain text. While this code performs rendering for solved letters, similar logic applies to the unsolved letters.



```
if(i < 4 || (i > 7 && i < 12) || i > 15)
{
    if(solved[0] && solved[1] && solved[2] && solved[3] && solved[4] && solved[5])
    {
        GFX_clear(Colors.white);
        if(i % 4 == 0)
        {
            GFX_drawText([ 120,80], TEXT_SIZE, 180, 0, TEXT_ALIGN_CENTER, Colors.red, "%c", mode_three_collection[i * 5 + 2]);
        }
        else if(i % 4 == 1)
        {
            GFX_drawText([ 80,120], TEXT_SIZE, 90, 0, TEXT_ALIGN_CENTER, Colors.black, "%c", mode_three_collection[i * 5 + 2]);
        }
        else if(i % 4 == 2)
        {
            GFX_drawText([ 200,120], TEXT_SIZE, 270, 0, TEXT_ALIGN_CENTER, Colors.black, "%c", mode_three_collection[i * 5 + 2]);
        }
        else if(i % 4 == 3)
        {
            GFX_drawText([ 120,200], TEXT_SIZE, 0, 0, TEXT_ALIGN_CENTER, Colors.black, "%c", mode_three_collection[i * 5 + 2]);
        }
    }
```



```
}
else
{
    if(solved[0] && solved[1] && solved[2] && solved[3] && solved[4] && solved[5])
    {
        GFX_clear(Colors.white);
        if(i % 4 == 0)
        {
            GFX_drawText([ 80,120], TEXT_SIZE, 90, 0, TEXT_ALIGN_CENTER, Colors.black, "%c", mode_three_collection[i * 5 + 2]);
        }
        else if(i % 4 == 1)
        {
            GFX_drawText([ 120,200], TEXT_SIZE, 0, 0, TEXT_ALIGN_CENTER, Colors.black, "%c", mode_three_collection[i * 5 + 2]);
        }
        else if(i % 4 == 2)
        {
            GFX_drawText([ 120,80], TEXT_SIZE, 180, 0, TEXT_ALIGN_CENTER, Colors.red, "%c", mode_three_collection[i * 5 + 2]);
        }
        else if(i % 4 == 3)
        {
            GFX_drawText([ 200,120], TEXT_SIZE, 270, 0, TEXT_ALIGN_CENTER, Colors.black, "%c", mode_three_collection[i * 5 + 2]);
        }
    }
```

Figure 30. Plain text rendering.

Recall Figure 22, which describes how face lists can be mapped to the topology of a 2x2x2 cube. Figure 22 also nicely illustrates the position and rotation of cube letters on a solved cube. Also recall Figure 25, which assigns both facelet and letter rotation information to cube letters using the hardcoded screen/module pairings of a solved cube as depicted in the WOWCube Emulator. Finally, consider the mode_three_collection, which functions like the drawn_facelet_collection depicted in Figure 24, in that it associates letters with screen/module pairings and also contains an implicit partitioning of different cube orientations. With these concepts in mind, it is clear that, as the main loop executes over each of the 24 facelets, the outer conditionals determine whether the loop is hovering over a particular partitioning of the mode_three_collection. Depending on the partitioning of mode_three_collection the loop reaches, the rotation values of each of the four letters on a face gets adjusted. Additionally, this partitioning affects the indexing of the first letter of a face's word, which receives a color different from the other face letters. All of this processing helps to ensure that the final rendering of a solved Game Mode 3 puzzle contains letters whose rotation values are the same as others on the same face while the rotation values of complete words are different between different faces (see Figure 31).

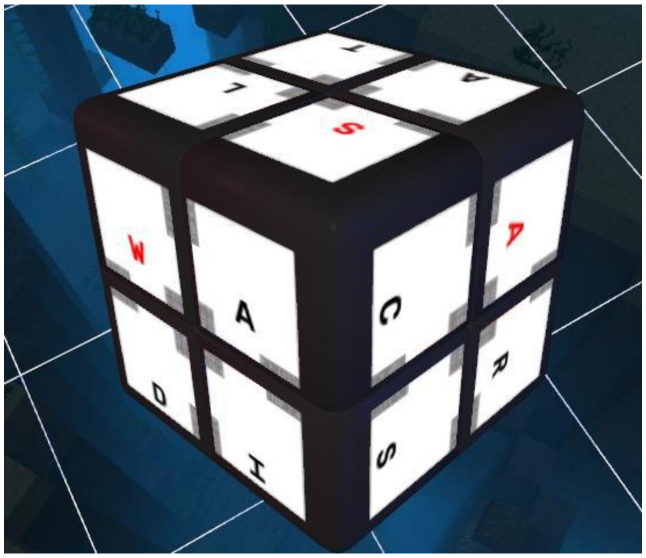


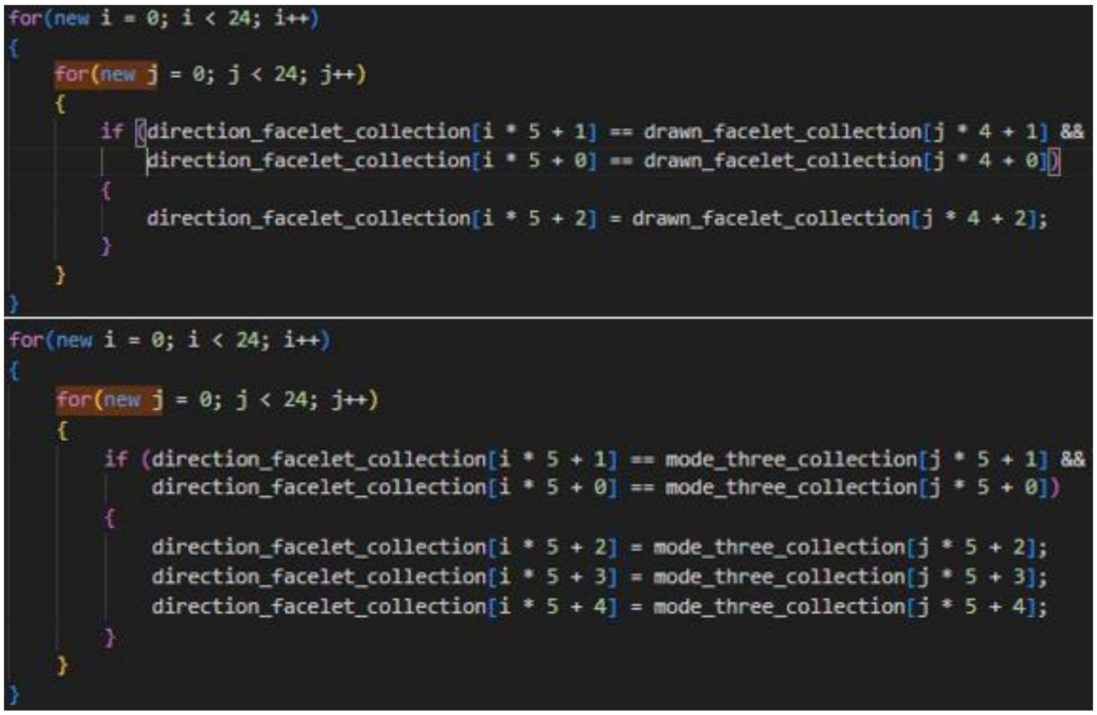
Figure 31. An example of a solved Game Mode 3 puzzle.

At this point, the puzzle is capable of randomly selecting four-letter words to display on the cube, scrambling the letters across each cube module's facelets, leveraging topology detection and screen/module encodings to associate letters with screens, and rendering those letters on their corresponding screens. To complete the application, the puzzle needs to be able to determine if the player solved it.

6.7 Solution Checking

The program leverages the ON_Tick callback to repeatedly check to determine whether the puzzle is solved. No matter the game mode, the puzzle uses a form of topology detection to determine the cube's current state, which is stored in a structure called direction_facelet_collection. This state includes screen/module pairing data, letter data, letter rotation data, and indicators about whether the facelet's letter is the first letter in a word. Only after comparing the pairings from the direction_facelet_collection with those from the collections responsible for rendering letters on facelets will the program know how facelets are oriented and what letters are associated with the new facelet orientations.

Figure 32 demonstrates the program's determination of which letters correspond to the screen/module pairings of the direction_facelet_collection, which knows how the cube's facelets are currently oriented. The upper portion of this figure handles Game Modes 1 and 2 while the lower portion handles Game Mode 3. Game Mode 3 has additional assignments in order to keep track of which letter represents the first letter of a word and at what angle the letters are rotated.



```
for(new i = 0; i < 24; i++)
{
    for(new j = 0; j < 24; j++)
    {
        if (direction_facelet_collection[i * 5 + 1] == drawn_facelet_collection[j * 4 + 1] &&
            direction_facelet_collection[i * 5 + 0] == drawn_facelet_collection[j * 4 + 0])
        {
            direction_facelet_collection[i * 5 + 2] = drawn_facelet_collection[j * 4 + 2];
        }
    }
}
for(new i = 0; i < 24; i++)
{
    for(new j = 0; j < 24; j++)
    {
        if (direction_facelet_collection[i * 5 + 1] == mode_three_collection[j * 5 + 1] &&
            direction_facelet_collection[i * 5 + 0] == mode_three_collection[j * 5 + 0])
        {
            direction_facelet_collection[i * 5 + 2] = mode_three_collection[j * 5 + 2];
            direction_facelet_collection[i * 5 + 3] = mode_three_collection[j * 5 + 3];
            direction_facelet_collection[i * 5 + 4] = mode_three_collection[j * 5 + 4];
        }
    }
}
```

Figure 32. Facelet orientation determination.

Once the direction_facelet_collection is initialized with values relevant to their game modes, the program branches into two separate solution checkers. Game Modes 1 and 2 are covered by the first solution checker, which iterates over each 4-list-long partition in direction_facelet_collection to obtain all possible letter orientations that can be parsed into a valid word when reading the letters from top to bottom, left to right. Figure 33 provides an example of the four possible ways of interpreting the letters on a cube face when reading the cube letters. Orientation 1 clearly demonstrates the spelling of the valid four-letter word “LOVE”. Importantly, each of these orientations indicates that the cube face is displaying a valid four-letter word. Therefore, in order to check for solutions, the program must determine whether one of these possible orientations matches a valid four-letter word in the word bank.

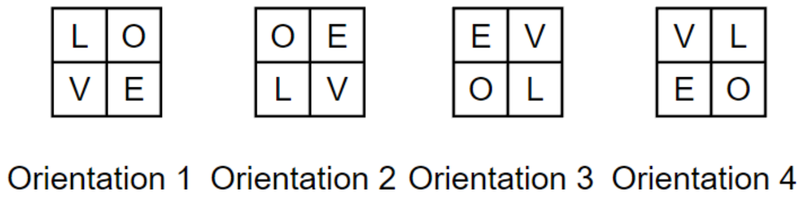
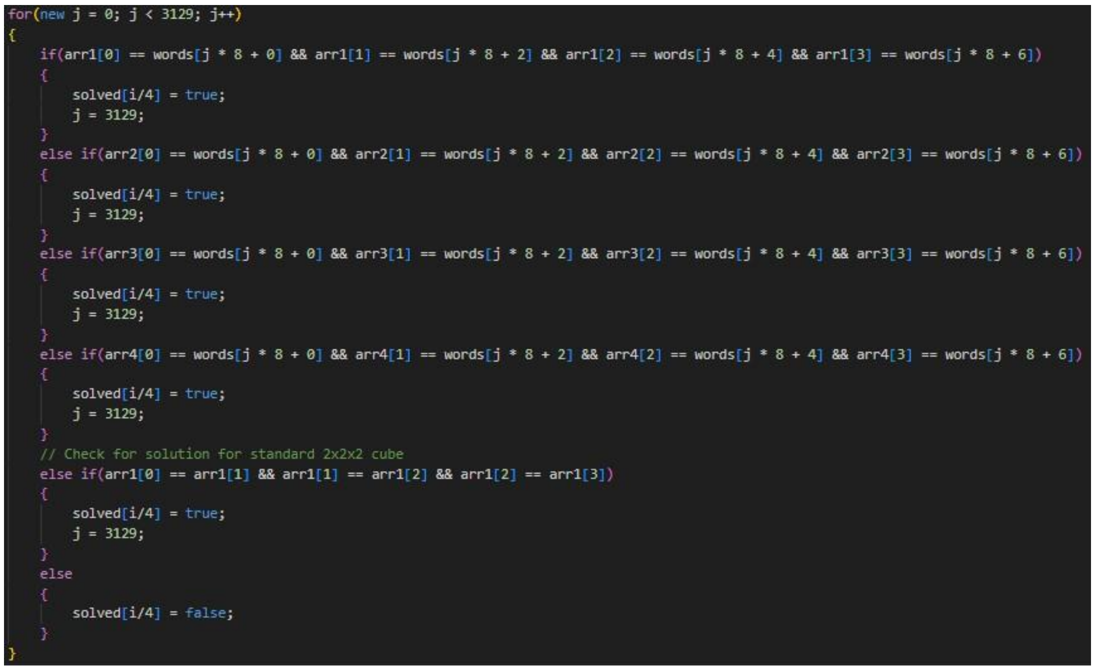


Figure 33. Valid face letter combinations.

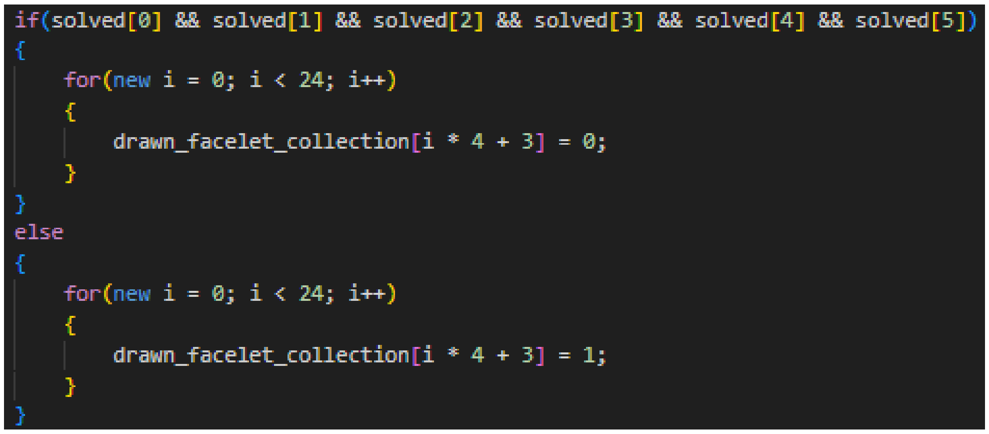
Figure 34 demonstrates the heart of the solution checker for Game Modes 1 and 2. The program uses four four-element lists, arr1 - arr4, to store four different orderings of face letters corresponding to the possible orderings depicted in Figure 33. The program then iterates over the entire word bank and checks if any of those letter combinations matches one of its words. If a combination matches for that partitioning of direction_facelet_collection, it marks that partitioning as solved using a list of six Boolean elements called solved and exits the loop. This check also incorporates a checker for the standard 2x2x2 cube puzzle game mode, which simply checks if each facelet of that partitioning contains the same letter.



```
for(new j = 0; j < 3129; j++)
{
    if(arr1[0] == words[j * 8 + 0] && arr1[1] == words[j * 8 + 2] && arr1[2] == words[j * 8 + 4] && arr1[3] == words[j * 8 + 6])
    {
        solved[i/4] = true;
        j = 3129;
    }
    else if(arr2[0] == words[j * 8 + 0] && arr2[1] == words[j * 8 + 2] && arr2[2] == words[j * 8 + 4] && arr2[3] == words[j * 8 + 6])
    {
        solved[i/4] = true;
        j = 3129;
    }
    else if(arr3[0] == words[j * 8 + 0] && arr3[1] == words[j * 8 + 2] && arr3[2] == words[j * 8 + 4] && arr3[3] == words[j * 8 + 6])
    {
        solved[i/4] = true;
        j = 3129;
    }
    else if(arr4[0] == words[j * 8 + 0] && arr4[1] == words[j * 8 + 2] && arr4[2] == words[j * 8 + 4] && arr4[3] == words[j * 8 + 6])
    {
        solved[i/4] = true;
        j = 3129;
    }
    // Check for solution for standard 2x2x2 cube
    else if(arr1[0] == arr1[1] && arr1[1] == arr1[2] && arr1[2] == arr1[3])
    {
        solved[i/4] = true;
        j = 3129;
    }
    else
    {
        solved[i/4] = false;
    }
}
```

Figure 34. Comparing word bank words against possible letter orderings.

The puzzle is deemed solved once every face is considered solved. This portion of the solution checker dynamically updates the display of the facelets to white if the puzzle is solved. Otherwise, the facelets will be colored black (see Figure 35).



```
if(solved[0] && solved[1] && solved[2] && solved[3] && solved[4] && solved[5])
{
    for(new i = 0; i < 24; i++)
    {
        drawn_facelet_collection[i * 4 + 3] = 0;
    }
}
else
{
    for(new i = 0; i < 24; i++)
    {
        drawn_facelet_collection[i * 4 + 3] = 1;
    }
}
```

Figure 35. Changing cube color when puzzle is solved.

The second solution checker behaves much like the first in that it considers four different orientations for reading facelet letters and also compares those different orientations against all the words in the word bank. The crucial difference is the checker's handling of Game Mode 3's additional constraints of ensuring that each letter has the same orientation such that a solved face appears like the word in Figure 25 and that the highlighted letter is positioned as the first letter of the word. These constraints necessitate checking the rotation values of each element of the direction_facelet_collection as well as the flag indicating whether a facelet's letter is the word's first letter. Figure 36 illustrates an example of the correct positioning of letters based on these constraints.

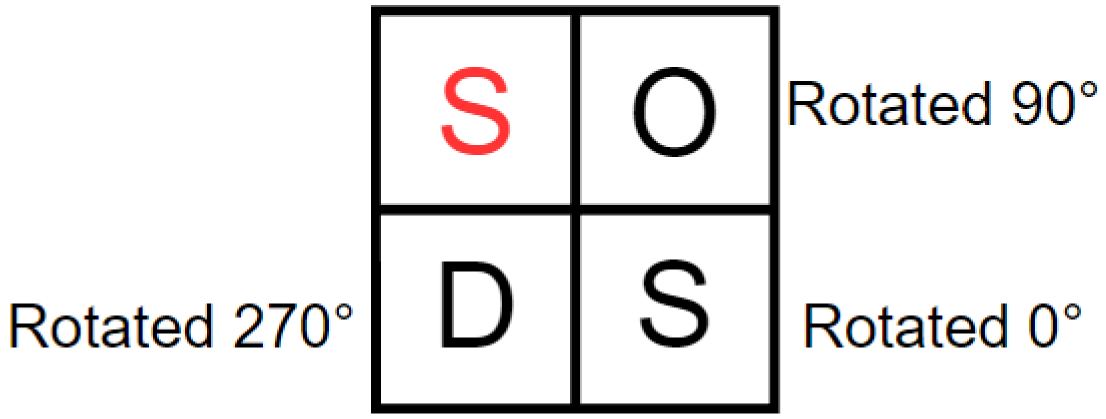
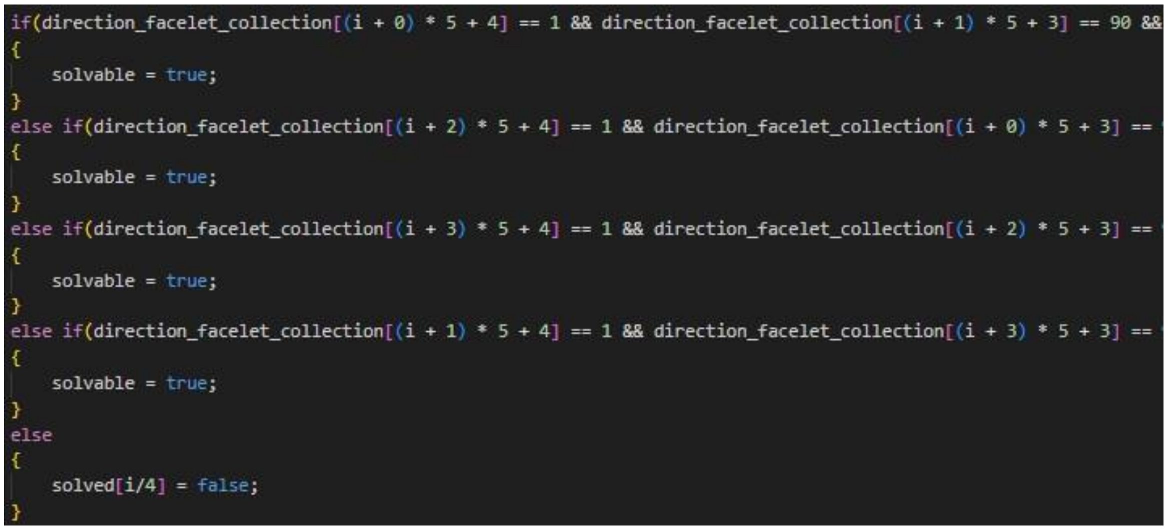


Figure 36. Game Mode 3 face solution constraints.

Figure 37 illustrates an encoding of Game Mode 3's solution checker that determines whether some orientation of a face's facelets contains facelets whose rotation values and first letter position match those depicted in Figure 36. This check has the happy side effect of skipping the iteration over the word bank since any face whose facelets do not possess these characteristics cannot be a solved face, allowing that face to be marked as unsolved right away.



```
if(direction_facelet_collection[(i + 0) * 5 + 4] == 1 && direction_facelet_collection[(i + 1) * 5 + 3] == 90 &&
{
    solvable = true;
}
else if(direction_facelet_collection[(i + 2) * 5 + 4] == 1 && direction_facelet_collection[(i + 0) * 5 + 3] ==
{
    solvable = true;
}
else if(direction_facelet_collection[(i + 3) * 5 + 4] == 1 && direction_facelet_collection[(i + 2) * 5 + 3] ==
{
    solvable = true;
}
else if(direction_facelet_collection[(i + 1) * 5 + 4] == 1 && direction_facelet_collection[(i + 3) * 5 + 3] ==
{
    solvable = true;
}
else
{
    solved[i/4] = false;
}
```

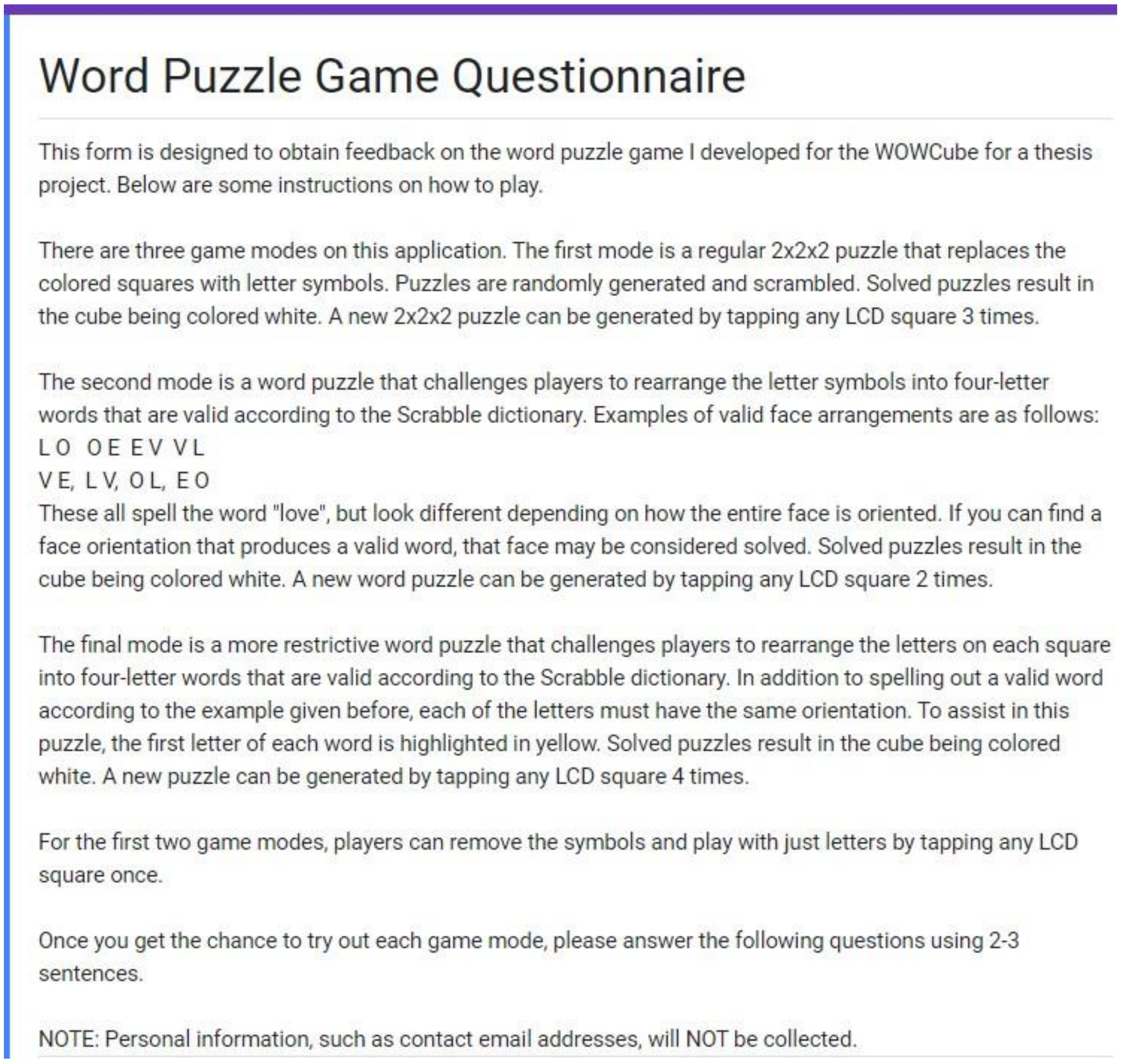
Figure 37. Game Mode 3 solution checker encoding.

Having encoded every component required to make each game mode functional, a user survey was conducted to determine the perceived strengths and weaknesses of the game with respect to entertainment and educational value.

CHAPTER 7

RESPONSES AND EVALUATION

Evaluation was conducted by allowing players to test the application either on a physical WOWCube device (if the player owned a physical device) or on the WOWCube Emulator, then asking players to complete a 9-question survey via Google Forms (see Figure 38).



Word Puzzle Game Questionnaire

This form is designed to obtain feedback on the word puzzle game I developed for the WOWCube for a thesis project. Below are some instructions on how to play.

There are three game modes on this application. The first mode is a regular 2x2x2 puzzle that replaces the colored squares with letter symbols. Puzzles are randomly generated and scrambled. Solved puzzles result in the cube being colored white. A new 2x2x2 puzzle can be generated by tapping any LCD square 3 times.

The second mode is a word puzzle that challenges players to rearrange the letter symbols into four-letter words that are valid according to the Scrabble dictionary. Examples of valid face arrangements are as follows:
L O O E E V V L
V E, L V, O L, E O
These all spell the word "love", but look different depending on how the entire face is oriented. If you can find a face orientation that produces a valid word, that face may be considered solved. Solved puzzles result in the cube being colored white. A new word puzzle can be generated by tapping any LCD square 2 times.

The final mode is a more restrictive word puzzle that challenges players to rearrange the letters on each square into four-letter words that are valid according to the Scrabble dictionary. In addition to spelling out a valid word according to the example given before, each of the letters must have the same orientation. To assist in this puzzle, the first letter of each word is highlighted in yellow. Solved puzzles result in the cube being colored white. A new puzzle can be generated by tapping any LCD square 4 times.

For the first two game modes, players can remove the symbols and play with just letters by tapping any LCD square once.

Once you get the chance to try out each game mode, please answer the following questions using 2-3 sentences.

NOTE: Personal information, such as contact email addresses, will NOT be collected.

Figure 38. Google Forms user survey for *WordBox*.

The survey asked players to comment on what they liked about each game mode, where each game mode can be improved, the perceived entertainment value of the application, the perceived educational value of the application, and whether they had any additional comments or feedback. The survey was posted on various Discord servers and social media platforms to ensure the aggregation of diverse and meaningful feedback. The survey closed with 10 participants sharing their feedback, which is summarized in the following sections. Figure 39 also summarizes the full body of user feedback in a word cloud.

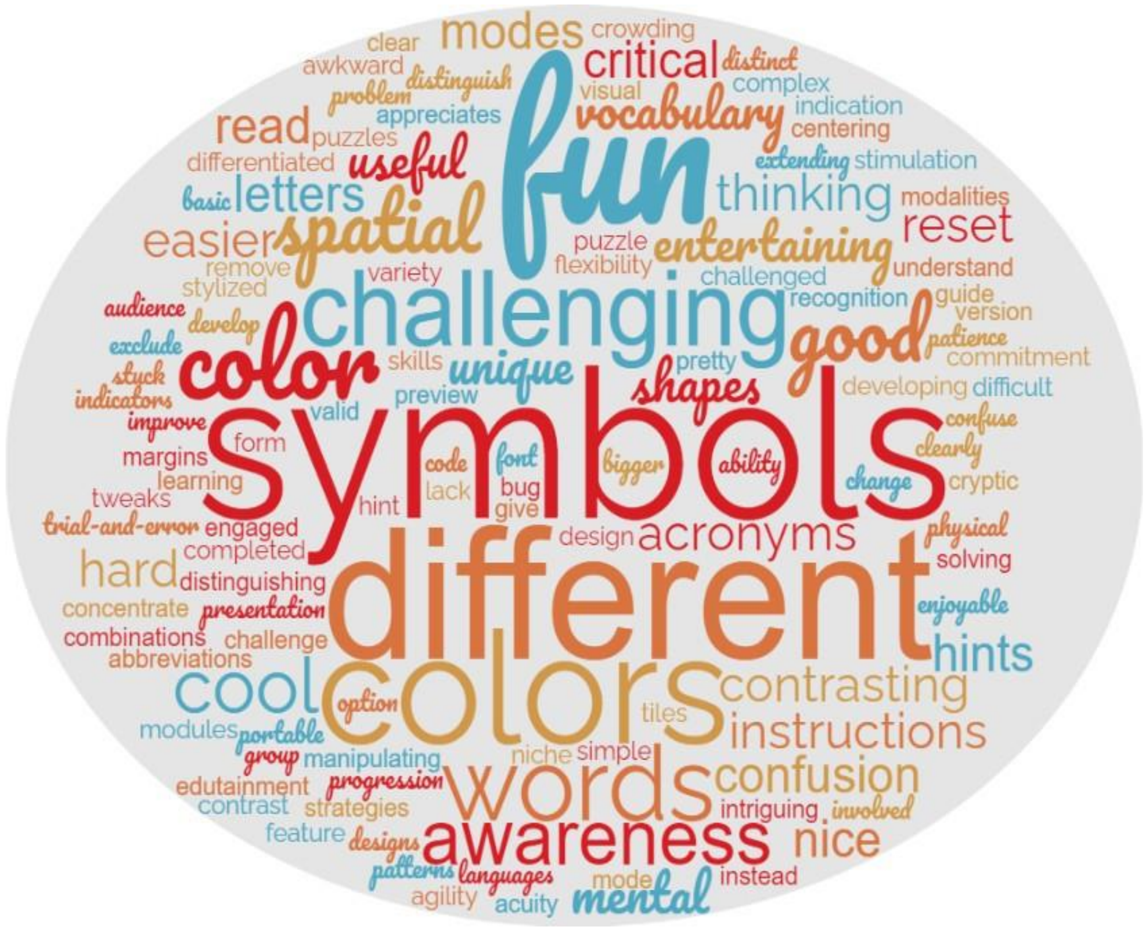


Figure 39. Word cloud of all feedback across all survey questions.

7.1 Perceptions on Game Mode 1 (Standard 2x2x2 Puzzle)

Game Mode 1, which tasks players with rearranging the letter symbols on the cube to ensure each square on a face matches each other, was received positively overall. 50% of the participants responded positively to the presentation of the application with players expressing their appreciation of the cryptic appearance of the squares. These players also enjoyed the visual feedback of solving the puzzle, which results in the squares changing from black to white. The remaining players found enjoyment in the intuitive nature of the puzzle, the ability to easily reset and change the puzzle, and the ease of manipulating the puzzle in a virtual environment. Figure 40 illustrates these positive impressions.

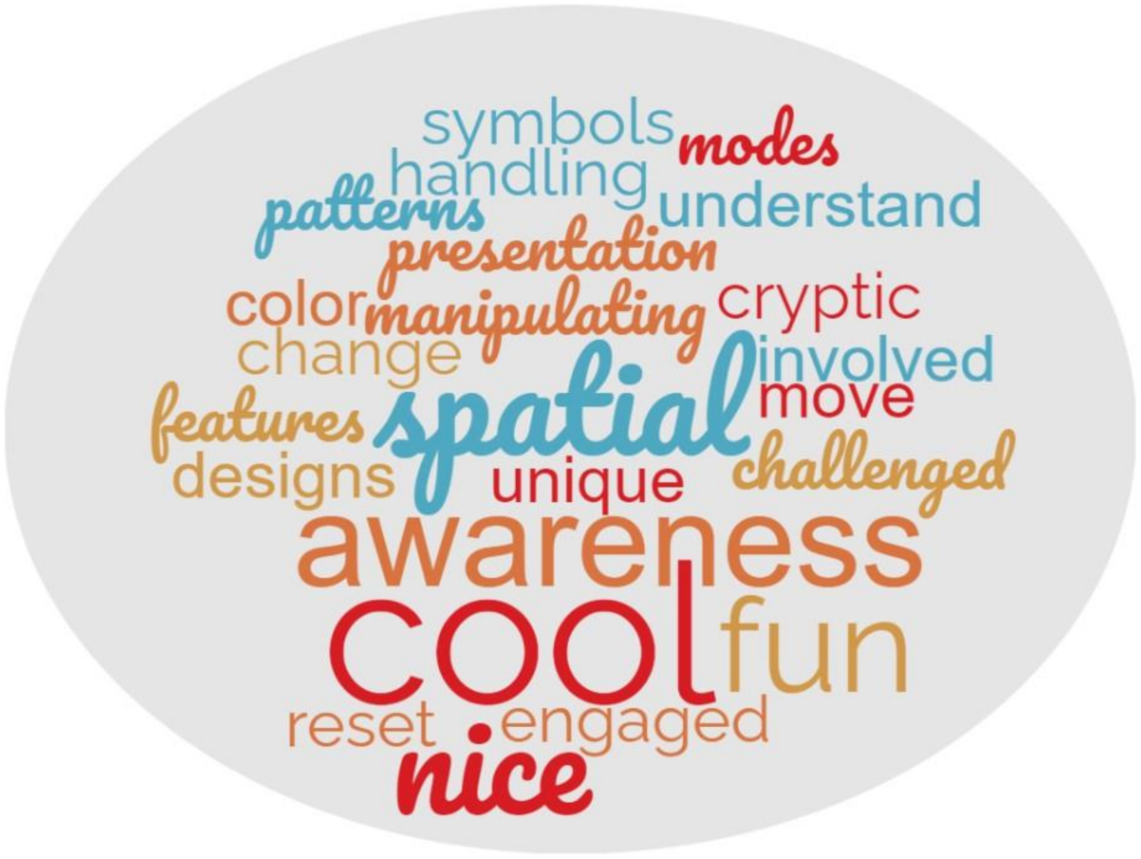


Figure 40. Word cloud of positive impressions of Game Mode 1.

Regarding constructive criticism, 50% of the participants indicated the desire to include more colors to assist in distinguishing letter symbols and in providing hints about how to group letters. One participant suggested a means of communicating progression in puzzle solving by perhaps changing the color of the cube or providing additional hints as the player gets closer to completely solving the puzzle. Additionally, one participant suggested that in-game instructions be provided to inform players on how to interact with and solve the puzzle. Figure 41 illustrates these critical perceptions.

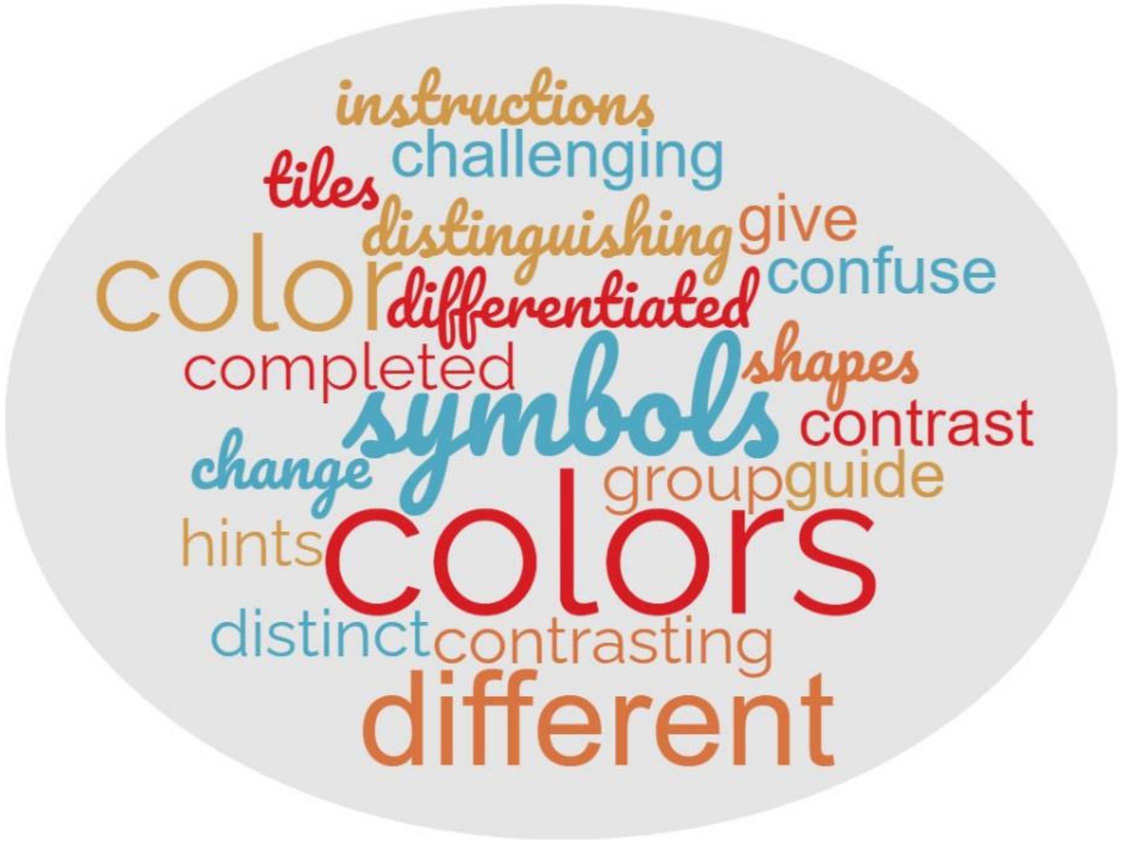


Figure 41. Word cloud of critiques of Game Mode 1.

7.2 Perceptions on Game Mode 2 (Less Restrictive Word Puzzle)

Game Mode 2, which tasks players with rearranging the letter symbols on the cube to form valid four-letter words regardless of individual letter orientation, was again received positively. 30% of the participants indicated their enjoyment of exploring how each of the letters can form different words, allowing for more ways to solve the puzzle than the standard 2x2x2 puzzle. One participant speculated on this mode's possibility of developing critical thinking and vocabulary skills because of the combinatorial nature of letter arrangements on the cube in addition to the combinatorial nature of the cube itself. Many of the participants echoed their appreciation of the symbol designs shown in Game Mode 1. Figure 42 illustrates these positive perceptions.

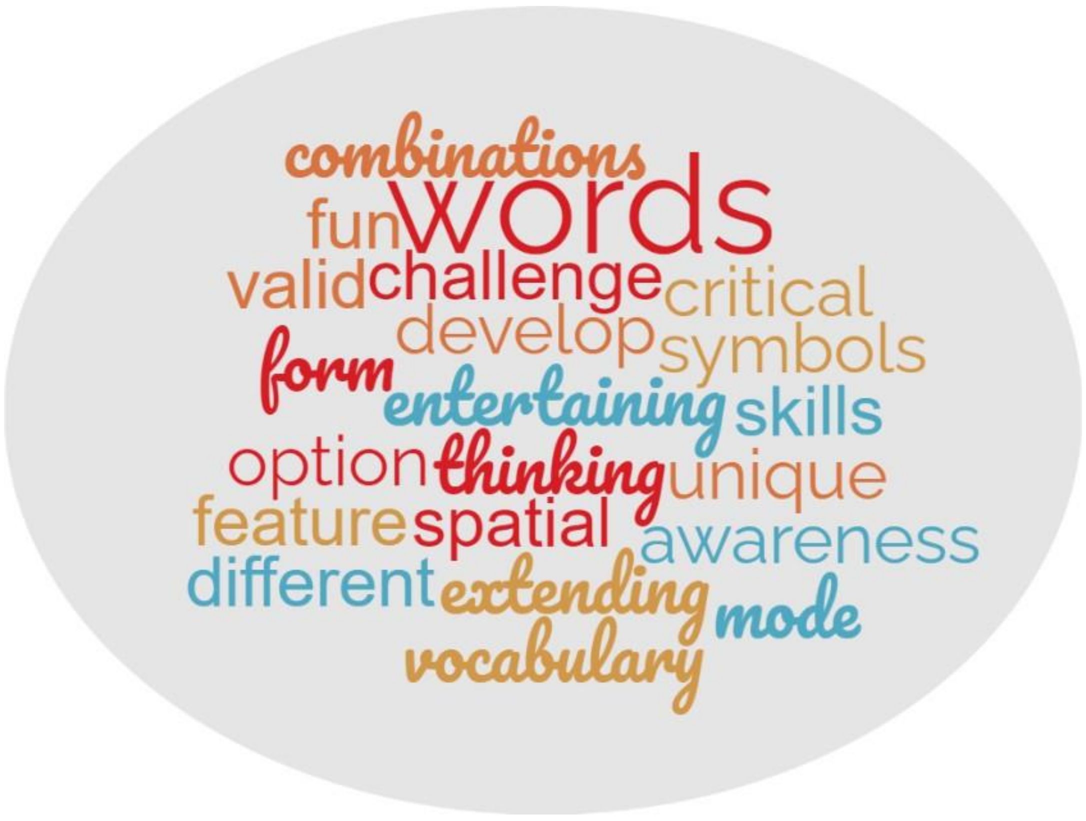


Figure 42. Word cloud of positive impressions of Game Mode 2.

Regarding criticism, 50% of players expressed the desire for the game to communicate a player's progress on solving the cube. Additionally, 20% of the players expressed the desire for better indicators that tell players what words they are trying to reproduce on the puzzle. One of the players pointed out a visual glitch that results in certain letter symbols disappearing while others appear as normal. Finally, likely due to some players playing with the WOWCube Emulator instead of a physical WOWCube device, one of the players expressed the desire of seeing the entire cube as a whole instead of feeling restricted to viewing a single face at a time. Figure 43 illustrates these critical perceptions.

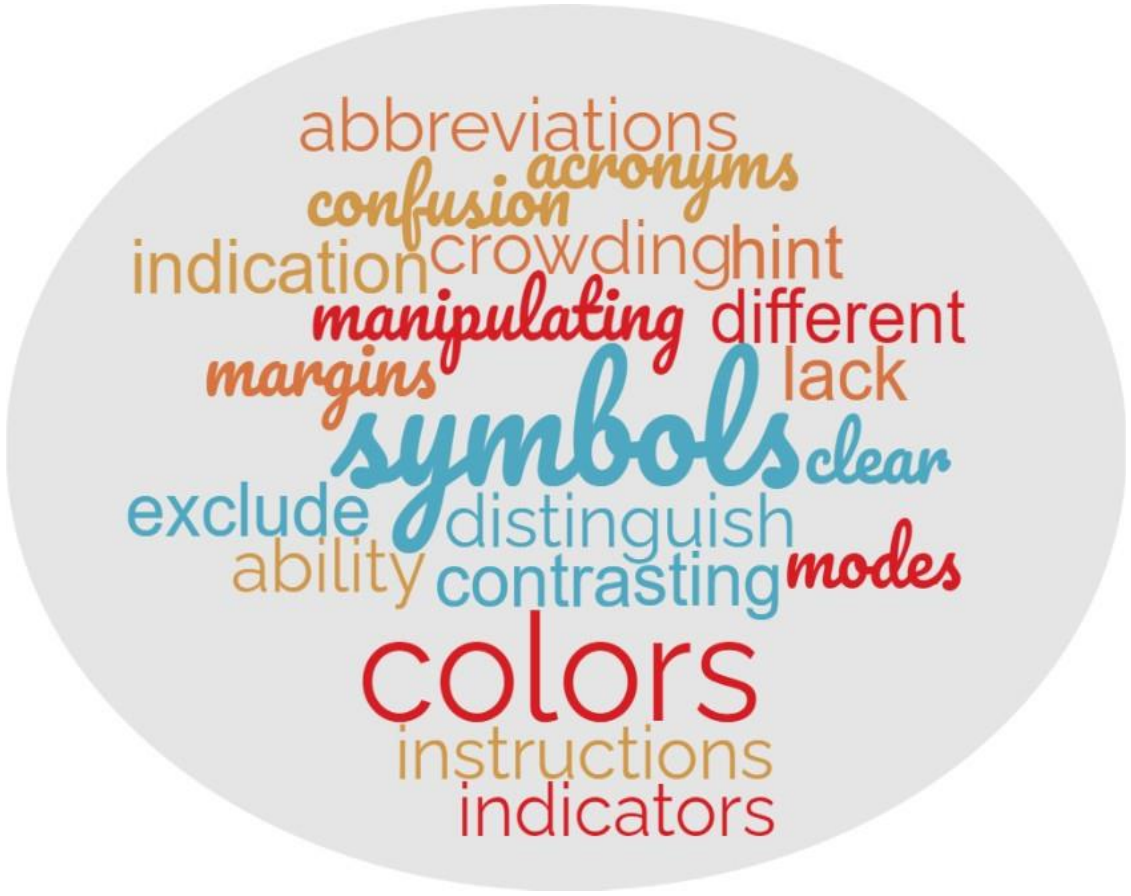


Figure 43. Word cloud of critiques of Game Mode 2.

7.3 Perceptions on Game Mode 3 (Restrictive Word Puzzle)

Game Mode 3, which tasks players with rearranging the letters on the cube to form valid four-letter words while considering the orientation and position of individual letters, received more of a mixed reception. 60% of the participants felt that this mode was quite difficult, though most participants maintained that the game was still enjoyable to play. One of the participants felt that it was easier to complete since there was no need to worry about the symbols that correspond with the letters. Furthermore, 40% of the participants expressed approval of the visual variety and use of color with certain letters to assist in solving the puzzle. Figure 44 illustrates these positive perceptions.

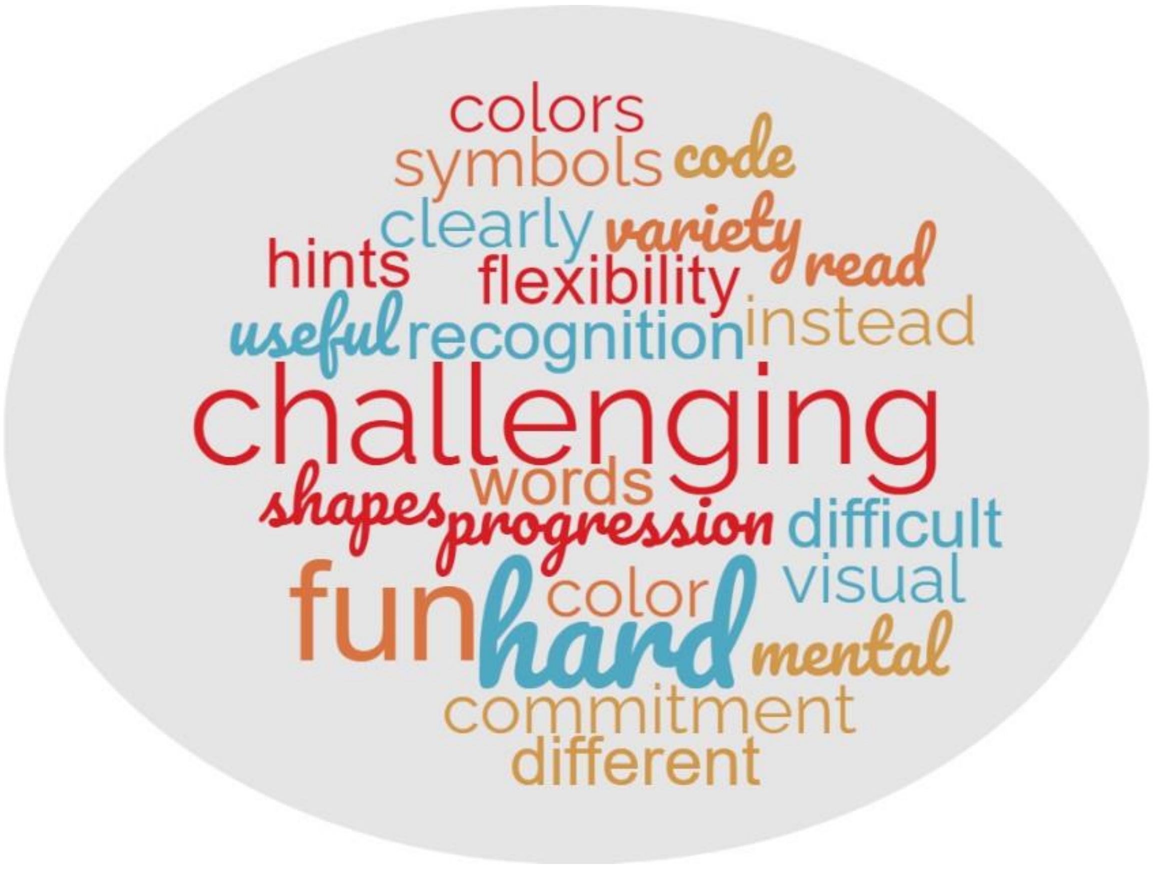


Figure 44. Word cloud of positive impressions of Game Mode 3.

On the other hand, players continued to express the desire to see what the completed puzzle should look like in order to help them in solving the puzzle. Additionally, despite the visual variety, 30% of the participants noted that the letters in the game mode were slightly awkward to read and could have benefited from using a larger font size. Finally, one of the players, who indicated their use of a physical

WOWCube device, noted that one of the cube modules was stuck on Game Mode 3 despite quitting. The player ultimately had to delete the game and reset the cube in order to flush out the error. This error was likely the result of a limited testing environment since no physical WOWCube devices were available to leverage at the time of project development. Figure 45 illustrates these critical perceptions.

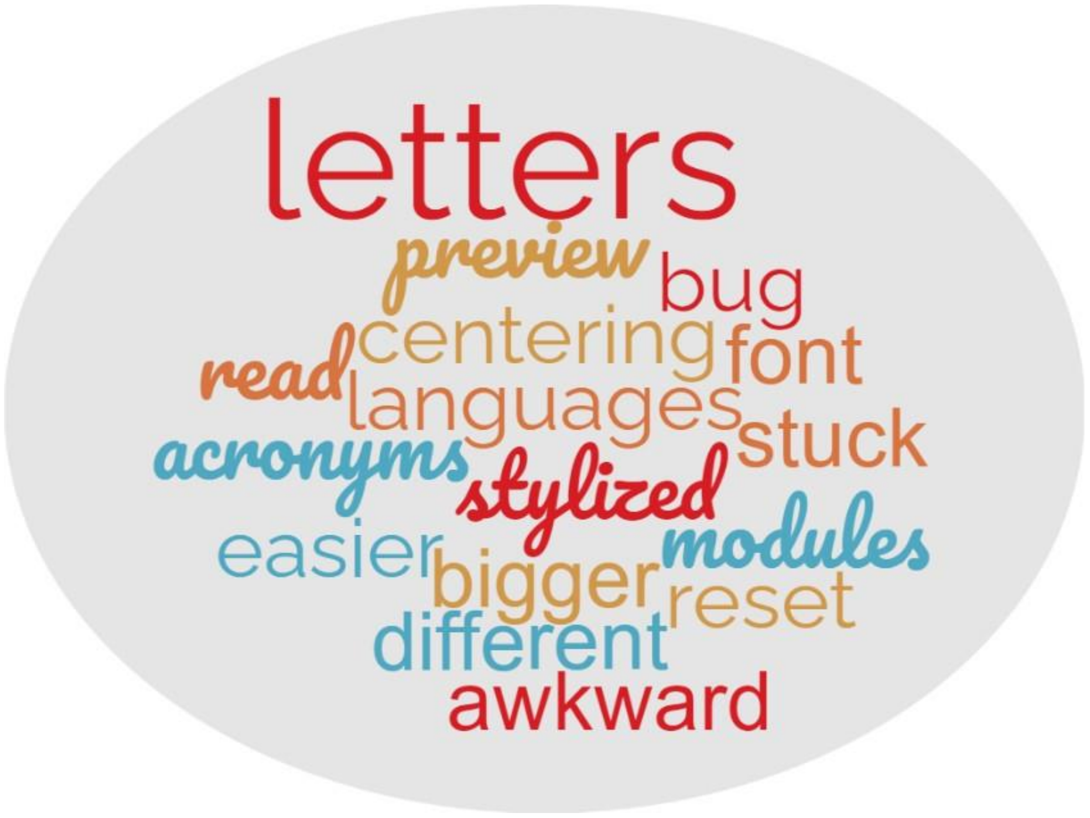


Figure 45. Word cloud of critiques of Game Mode 3.

7.4 Perceived Entertainment Value

Regarding the entertainment value of the puzzle as a whole, the participants generally agreed that it was a fun and captivating experience, especially for those who enjoy solving puzzles. Notably, one of the participants indicated that they would likely continue playing Game Modes 1 and 2 instead of 3. Finally, one of the participants attempted to quantify the entertainment value by giving the app a score of 8/10. Figure 46 illustrates these thoughts.

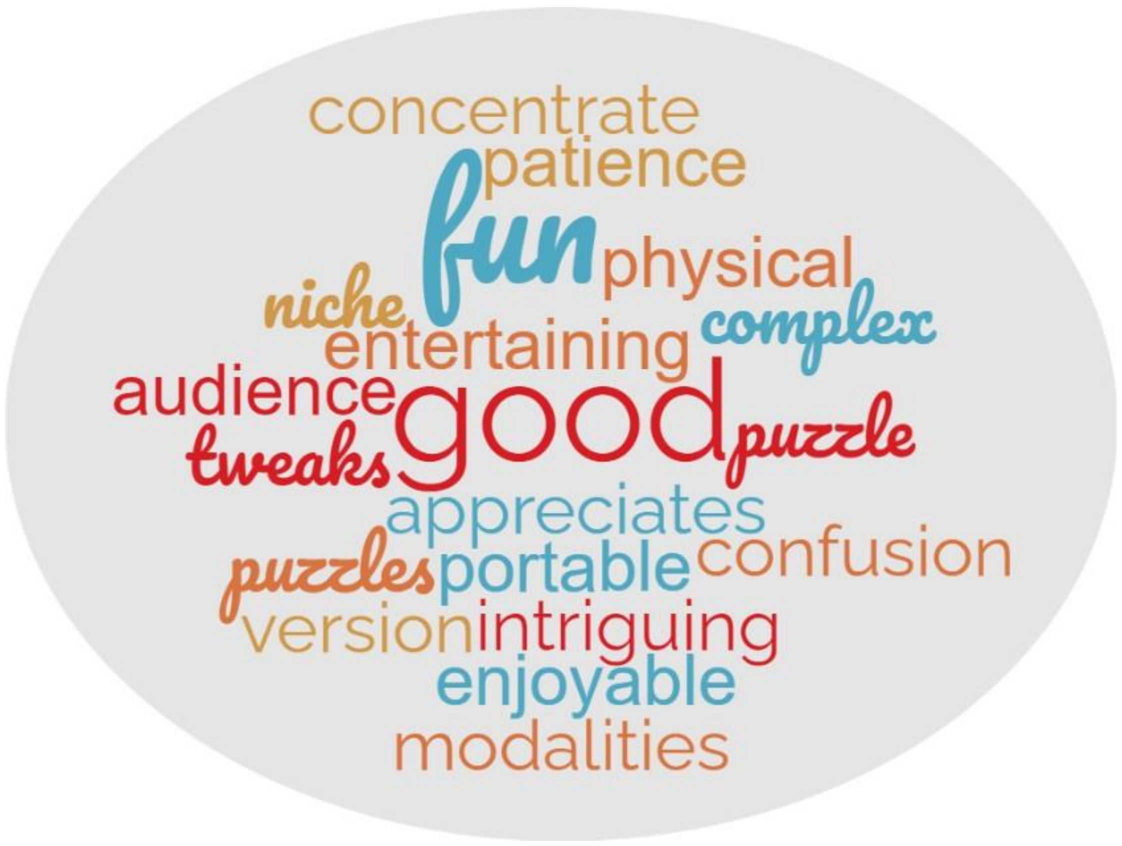


Figure 46. Word cloud of perceived entertainment value.

7.5 Perceived Educational Value

Regarding the educational value of the application, 100% of the participants indicated that the application holds some educational value. 20% of the participants held that this application can be used for teaching basic vocabulary and improving critical thinking skills. One of the participants applauded the fact that the application differs from most puzzles in that it stimulates thinking via word construction as opposed to the perceived trial-and-error nature of other puzzles. One of the participants went so far as to express the application's potential as an evaluation tool for determining an individual's mental acuity and spatial awareness due to the layers of problems to solve with the puzzle. Finally, a participant again attempted to quantify the educational value by giving the app a score of 8/10. Figure 47 illustrates these thoughts.

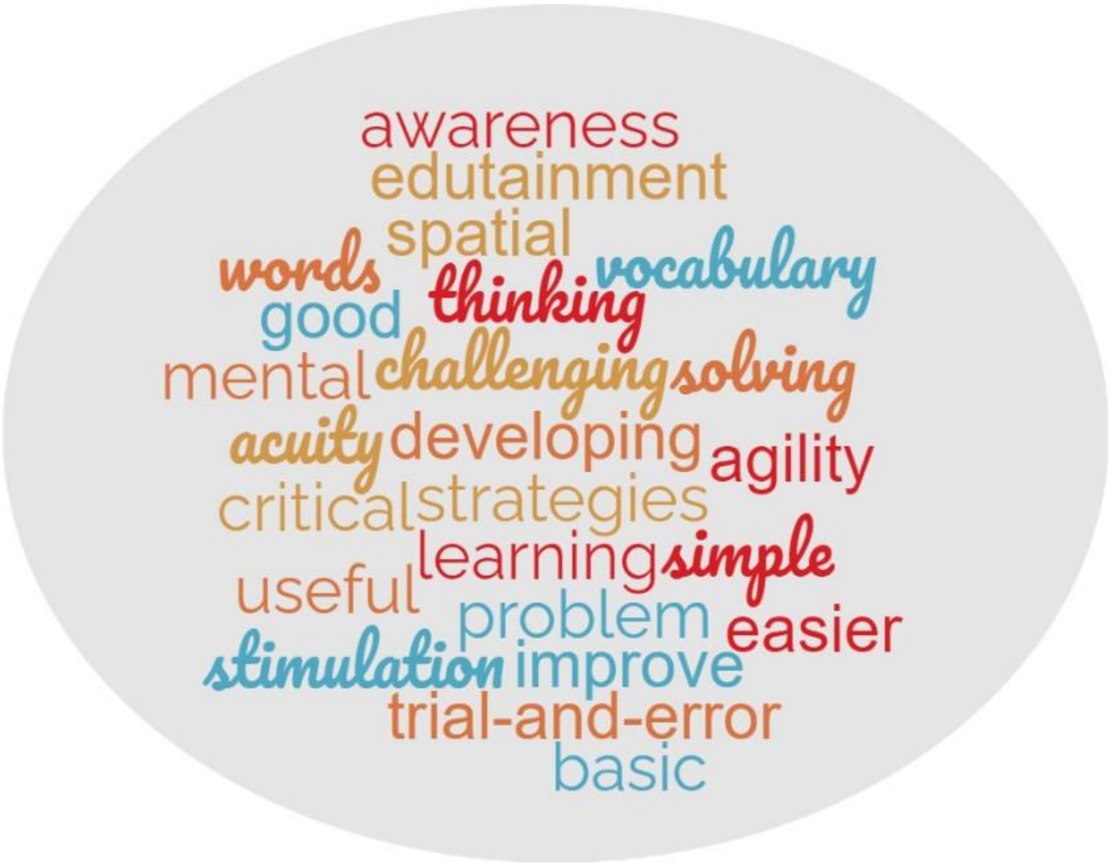


Figure 47. Word cloud of perceived educational value.

7.6 Interpretation

These results indicate that *WordBox* both assists in developing the critical thinking and language skills of players and mitigates the issues of using a conventional Rubik's cube in an educational context due to the ease with which players can change and reset the app's puzzles. The latter point is especially important when considering Cornock's use of Rubik's cubes to assist in teaching students group theory [4]. While this research outlines several advantages of leveraging such twisty puzzles for tailored assignments, such as anti-plagiarism measures and tying abstract mathematical concepts to physical teaching aids, there still exists the potential of frustration when errors occur during the solving of a theory problem using a Rubik's cube. Because conventional Rubik's cubes were used in these assignments, the program sought to minimize this frustration by offering students the option of swapping their scrambled cube with a solved one. As one of the players pointed out in the user survey for *WordBox*, this is not an issue due to the integrated resetting and scrambling features of the app, allowing players to easily start over without having to manually perform any physical twist, thereby alleviating student frustration while also minimizing educational administration overhead.

Despite the overall success of *WordBox* exercising player's critical thinking and language skills, the survey revealed several areas of improvement, which will provide guidance on future works for the project and its evaluation.

CHAPTER 8

FUTURE WORKS

The most highly prioritized tasks to improve *WordBox* for the future include those pertaining to glitches and hanging on exiting the game. Some participants noted that some symbols would disappear while others stay visible during gameplay. This is likely the result of players who utilized the WOWCube Emulator to test the application. In the Emulator, twists are performed by using the left mouse button to drag portions of the cube. Taps are also registered by clicking on the screen. It is possible that, in some instances of players attempting to twist the puzzle, a tap was simultaneously triggered, resulting in certain modules removing the symbol while leaving others. This behavior is a side effect of the fact that each cube module is running a copy of the application, each with its own copies of state variables. In order to address this issue, the application source code must be updated to support intermodule communication of these state variables to ensure consistency in what is displayed.

Furthermore, reports of hanging suggest that additional testing of the app on a physical WOWCube device is required to pinpoint the source of this error. The most likely cause is the lack of a call to the quit API function within the ON_Shake event handler. This is normally not necessary in the WOWCube Emulator since the built-in shaking simulation simply exits the application no matter what instructions are contained in the ON_Shake event handler. That said, because the WOWCube SDK is frequently being updated, proper testing on the quit functionality using ON_Shake within the Emulator will be feasible soon.

Aside from debugging, future app development would include several quality-of-life changes, such as the inclusion of in-app instructions on how to play the game, the removal of abbreviations and acronyms as valid four-letter words, and visual improvements to assist in better distinguishing and reading letters rendered onscreen. Game Mode 3 letters would receive a larger font size while the remaining game mode letters would receive different colors in order to help players in better distinguishing each of the letters and symbols.

Additionally, the game would receive updates that improve players' understanding of what words they are trying to solve for and how close players are to solving the entire puzzle. Figure 48 offers one possible visualization to inform players about the goal words of a puzzle. Each facelet can still have a letter oriented in each cardinal direction. However, the center of each facelet for that particular face would contain a letter from a valid four-letter word (e.g. "SAWS") using the letters available on the puzzle. As players twist the puzzle, the hint overlay of the finished word would be updated such that the letters remain ordered to form a valid word on that face.

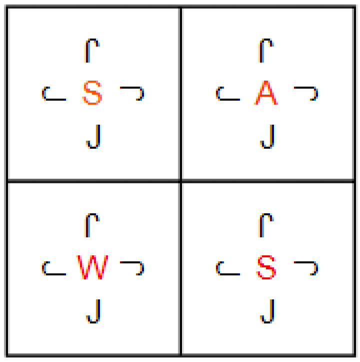
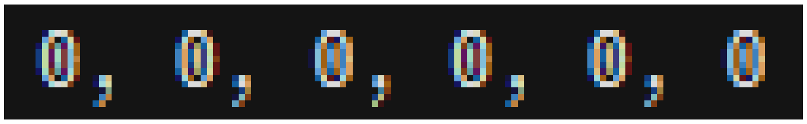


Figure 48. A possible visualization of the goal word for a cube face.

Using the visualization of goal words shown in Figure 48, players would have a sense of what words can be formed on each face of the cube. Importantly, the solution checker would still iterate over the entire word bank in order to account for the possibility of a different combination of letters resulting in valid words on all faces of the cube, thereby adding the potential for surprise when playing the game.

As for giving players a sense of progression, a simple solution would leverage the internal tracking of which faces contain a valid four-letter word. Figure 49 illustrates the logging of which faces are solved. 0 represents an unsolved face while 1 represents a solved face.



```
0, 0, 0, 0, 0, 0
```

Figure 49. Logging of the faces that form valid four-letter words.

Using this internal tracking, whenever a face's composition of letters produces a valid four-letter word, that individual face can become highlighted in white, as happens currently for the entire cube when the entire puzzle is solved.

Finally, there remain several features that can be added to improve upon the educational value of the application. These features include the following:

- Support for different languages and the ability to select a puzzle in the language of the player's choice.
- Expansion of puzzle resetting features, such as specifying the random seed used to generate puzzle scrambles and more options on resetting a specific puzzle generation.

- Ability to specify the words to solve for instead of relying solely on random word generation.
- Additional game modes, such as an endless mode that generates new puzzles immediately after solving a puzzle.
- Leaderboards that keep track of the amount of time a player takes to solve a puzzle.

CHAPTER 9

CONCLUSION

In conclusion, this thesis has demonstrated the efficacy of *WordBox* as a novel twisty puzzle application that is capable of being used in an educational context to enhance critical thinking and language skills. Not only does the application differentiate itself from conventional Rubik's cube puzzles, which are not capable of robustly teaching abstract concepts without extensively adapting the curriculum to the static puzzle, but the application also differentiates itself from the companion apps of Bluetooth smart cubes, which are inherently detached from the physical topology of the puzzle that players directly interact with. Furthermore, the application's nature as a word puzzle demonstrates its presence as the first language teaching application for a smart cube application that directly interfaces with the physical topology of the device. It is anticipated that, as more updates specified in Chapter 8 are applied to the application, the application will grow to become a more robust and entertaining language teaching implement.

REFERENCES

[1] "Age of Puzzles - Larry D. Nichols," n.d. http://www.ageofpuzzles.com/Masters/LarryDNichols/LarryDNichols.htm.

[2] Bberak. "GitHub - Bberak/React-Native-Game-Engine: A Lightweight Game Engine for React Native 🕹⚡🎮." GitHub, n.d. https://github.com/bberak/react-native-game-engine.

[3] Bellavista, Eloi. "My Rubik | Notation and Pieces 2x2x2 Cube," n.d. https://www.myrubik.com/elcub2x2x2.php?lang=en.

[4] Cornock, Claire. "Teaching Group Theory Using Rubik's Cubes." *International Journal of Mathematical Education in Science and Technology*, August 5, 2015. https://doi.org/10.1080/0020739x.2015.1070442.

[5] Cubing. "GitHub - Cubing/Cubing.Js: 🛠 A Library for Displaying and Working with Twisty Puzzles. Also Currently Home to the Code for Twizzle." GitHub, n.d. https://github.com/cubing/cubing.js/.

[6] Expo Documentation. "FAQ," n.d. https://docs.expo.dev/faq/.

[7] "GAN356 i 3-GANCUBE SHOP," May 10, 2023. https://shop.gancube.com/product/gan356i3.

[8] GiiKER. "GiiKER - Toys Meet Technology," n.d. https://giiker.com/.

[9] Gist. "List of Four-Letter Words to Go with Https://Gist.Github.Com/3733182," n.d. https://gist.github.com/paulcc/3799331.

[10] Lucas Garron. "Twizzle Diaries — Ep. 1: What If?," July 22, 2021. https://www.youtube.com/watch?v=9_kqXn0Mq-o.

[11] Mazzatenta, Mark. "What The RUBIK'S CUBE Taught Me About Recital Preparation." *Music Teachers National Association* 62, no. 8 (2013): 18–19.

[12] Nichols, Larry D. "US3655201A - Pattern Forming Puzzle and Method with Pieces Rotatable in Groups - Google Patents," March 4, 1970. https://patents.google.com/patent/US3655201A/en.

[13] Particula Tech. "GoCube: The Ultimate Smart Rubik's Cube | Particula." Particula, May 23, 2023. https://particula-tech.com/gocube/.

[14] BackerKit. “Project Updates for GoCube | The Classic Puzzle Reinvented on BackerKit Page 2,” n.d. https://gocube.backerkit.com/hosted_preorders/project_updates?page=2.

[15] Riemersma, Thiadmer. “The Pawn Language,” n.d. https://www.compuphase.com/pawn/pawn.htm.

[16] Rohrig, Brian. “PUZZLING SCIENCE.” *Science Teacher* 77, no. 9 (2010): 54–56.

[17] Rokicki, Tomas, Herbert Kociemba, Morley Davidson, and John C. Dethridge. “The Diameter of the Rubik’s Cube Group Is Twenty.” *Siam Review* 56, no. 4 (November 6, 2014): 645–70. https://doi.org/10.1137/140973499.

[18] Singmaster, David. *Notes on Rubik’s “Magic Cube.”* Enslow Pub Inc, 1981.

[19] Unity Technologies. “Unity - Manual: The Scene View,” n.d. https://docs.unity3d.com/Manual/UsingTheSceneView.html.

[20] Unity Technologies. “Unity - Manual: Event Functions,” n.d. https://docs.unity3d.com/Manual/EventFunctions.html.

[21] “Three.Js – JavaScript 3D Library,” n.d. https://threejs.org/.

[22] Twizzle Explorer. “Twizzle Explorer ALPHA,” n.d. https://alpha.twizzle.net/explore/.

[23] TheCubicle. “WCA Puzzles,” n.d. https://www.thecubicle.com/pages/wca-puzzles.

[24] “WOWCUBE® ENTERTAINMENT SYSTEM - STORY,” n.d. https://wowcube.com/story/.

[25] “WOWCUBE® ENTERTAINMENT SYSTEM - SPECS,” n.d. https://wowcube.com/specs.

[26] “WOWCUBE® ENTERTAINMENT SYSTEM - STORE,” n.d. https://wowcube.com/store.

APPENDIX A

SURVEY RESPONSES

The survey questions and participant responses used to evaluate *WordBox* are outlined below. The collection of participants used either a physical WOWCube device or the WOWCube Emulator to test the application, as indicated by their responses.

Question 1: What did you like about Game Mode 1 (the standard 2x2x2 puzzle)?

Participant 1: Kept me involved and engaged. Made me want to keep playing. I was challenged to try and solve the puzzle.

Participant 2: I like the ability to move the cube around in virtual space much like handling it by hand, simply by right clicking the mouse and sliding it around. This gave me spatial awareness. It does so much more quickly than manipulating an actual cube, plus the cube map off to the left allows for another modality to gain spatial awareness.

Participant 3: i liked the designs of the squares! felt like a more cryptic rubix cube.

Participant 4: I thought the design of the patterns was really cool and it did look like a Rubix Cube to me.

Participant 5: That it was cool and it created a lot of chances to make the game fun such as the different color change modes.

Participant 6: Easy to understand, typically rubix cube mode

Participant 7: I like how you can reset, change the cube buy double or triple clicking the mouse, also add or remove symbols just by a click. It has lots of advance features in the cube. That gives off where the future is heading in terms on this game.

Participant 8: It was fun and gets you think

Participant 9: i liked the puzzle overall. it was a unique taking on solving a regular rubik's cube. i think the symbols are cool.

Participant 10: It's a standard 2x2, but the symbols were nice to look at. The presentation is pretty nice overall.

Question 2: What could be improved with Game Mode 1 (the standard 2x2x2 puzzle)?

Participant 1: Maybe have the symbols differentiated more. I needed to strongly focus on the symbols so as not to confuse them. Maybe have the symbols represented in different colors.

Participant 2: Better contrasting colors may help me personally in distinguishing individual symbols. Color contrast, sharply distinct shapes, and visible separation of central image on each facet from surrounding characters could help. This is my impression of the design at this time according to my level of perception.

Participant 3: maybe some sort of guide so the player knows better what they're looking for. a conplete picture/ pattern would also be cool to include as the final completed cube.

Participant 4: I struggle with these types of puzzles...maybe have the cube change color or give hints to allow the user to know they are close at achieving the level

Participant 5: Nothing

Participant 6: Sometimes tiles would gain stuff on them? Otherwise fine

Participant 7: I personally felt it a bit challenging, it could be just me as I’m not good at solving cubes. But I get it how that the whole of the game to challenge people.

Participant 8: Nothing

Participant 9: maybe include an option to give different colors to the symbols. that would help in figuring out how to group symbols.

Participant 10: There are no instructions in the game itself about how to play. These should be included.

Question 3: What did you like about Game Mode 2 (the symbolic word puzzle)?

Participant 1: Very entertaining trying to create valid words with only 6 letters. Makes the player use critical thinking skills to create many variations with the same 6 letters. Helps to develop vocabulary of the player.

Participant 2: The same as game mode 1, with the general ability to gain spatial awareness. Additionally this mode introduced a new more familiar yet also complicating symbolism vis-a-vis letters. I am trained to read, but only in one manner, while this mode challenged me to stretch out beyond my famliar "training".

Participant 3: i lined the idea of using letters itself

Participant 4: I liked that there are many possible combinations to achieve an answer

Participant 5: That you can move the whole cube and still complete the challenge at the same time

Participant 6: Interesting concept, fun to see what word/words can be made

Participant 7: I would say the exact same this about game mode 1 but it’s a bit difficult. It’s always good to have to feature or option where you can just change the level or mode of the game.

Participant 8: It was fun and gets you think

Participant 9: like with 1 i liked the symbols. i liked that it was more challenging since you have to think about how to twist the puzzle as well as how to form different words.

Participant 10: It's a unique way of extending the base 2x2 puzzle. I liked the added challenge.

Question 4: What could be improved with Game Mode 2 (the symbolic word puzzle)?

Participant 1: An easier way of seeing the whole picture. The main issue for me was my lack of ability in manipulating the cube to be able to create a word on each side of the cube.

Participant 2: The same critique applies to this mode as to mode 1 as relates to contrasting colors and crowding of symbols. For me personally, this is a source of confusion as relates to achieving the overall objective of "solving the cube", or even just one face of the cube. Using different colors to distinguish symbols, and creating visible "distance" at the margins between symbols on each individual facet would reduce the confusion for me personally, with the current iteration of design.

Participant 3: a design more similarb to the first game

Participant 4: Again, I would like an indication telling me that I am getting somewhere...

Participant 5: Nothing

Participant 6: Same with 1, remove changing symbols in the middle, otherwise fine

Participant 7: Nothing! It’s good! 😊

Participant 8: Key indicators of what youre trying to solve

Participant 9: it isn't clear what words you are trying to solve for. maybe that's part of the challenge, but it might be nice to give some hint about what words you are trying to solve.

Participant 10: Again, it could use better instructions on how to reach each of the modes. Also, it might be better for the puzzle to exclude abbreviations and acronyms as valid words.

Question 5: What did you like about Game Mode 3 (the restrictive word puzzle)?

Participant 1: Very challenging and requires a commitment from the player. The extra challenge of having the letters all facing the same way shows progression within the game. Will probably help the player develop game playing skills over time.

Participant 2: Game mode 3 allows for easy recognition of symbols. It also further challenges me in "stretching" my training beyond my current regular comfort of reading. This mode provide both a good challenge to improve my mental flexibility, and yet simultaneously is easy to clearly read.

Participant 3: again, i like the idea of using words

Participant 4: I thought this was really hard...having the letters marked in different colors was useful

Participant 5: It made it easier to complete since it was only words to look out for instead of shapes

Participant 6: Good challenge and addition to 2nd

Participant 7: It’s gets even more difficult, but my response will remain the same as mode 1 and 3.

Participant 8: Color code it

Participant 9: this was really hard but also fun. i like how this gave better hints about how to group letters together.

Participant 10: I liked the visual variety and how it extended the second game mode. I found it quite challenging, but fun.

Question 6: What could be improved with Game Mode 3 (the restrictive word puzzle)?

Participant 1: Only allowing for true word creation and not acronyms. Once again, my lack of game play skills hampered my success in solving the puzzle.

Participant 2: For me personally, improvement would include maybe just larger font size. Perhaps centering the letters in each facet would also help a user to reach the overall of objective of "solving" the cube. Maybe also providing stylized letters, or even letters in different languages.

Participant 3: design more similarb to the first game

Participant 4: A preview of what the completed answer should look like

Participant 5: Nothing

Participant 6: A little awkward to look at at times, could make it a little easier to read

Participant 7: Nothing.

Participant 8: It was fun and gets you think

Participant 9: maybe have the letters a little bigger.

Participant 10: There seemed to be a bug with this mode. After exiting the game, one of the modules was stuck on game mode 3, had to delete the game and reset my cube. Fixing this is the major thing, but the rest of the puzzle seemed solid.

Question 7: What is the overall entertainment value of this application?

Participant 1: Very entertaining and captivating.

Participant 2: The overall entertainment value in this application, seems to me, is the ability to suspend my sensation of the environment, and concentrate on a known exercise objective. Like any good puzzle, I find satisfaction in solving a portion of the puzzle, or the whole puzzle itself. Additionally, it is portable and yet does not take up space, and provides multiple game modalities at the click of the mouse.

Participant 3: it's entertaining to a more niche audience that appreciates puzzles

Participant 4: Definitely for someone who has patience to do puzzles! I also had to spend some time to figure out how to use the controls. I found the game intriguing and complex.

Participant 5: '8/10

Participant 6: Generally fun, would mostly stick on game 1 and 2

Participant 7: I wasn’t entertained much, it’s because it’s not my thing. But I definitely see how it could be very much entertaining for people those who are in this game.

Participant 8: It was good!

Participant 9: i think it was really fun! i can definitely see myself playing with this if i had my hands on a physical version of the cube. i think this can really be something if a few tweaks were made.

Participant 10: Overall, I think this app is on par with many of the other apps available on the WOWCube store. I personally found it quite enjoyable after getting over the initial confusion.

Question 8: What is the overall educational value of this application?

Participant 1: Would benefit players when learning vocabulary and developing critical thinking.

Participant 2: My impression is that the application can improve mental acuity and agility with repeated gameplay. Additionally, I suspect that this application can quickly give trained evaluators the ability to quickly asses a snapshot, and also forecast to potential, capability of an individual's mental acuity and agility as it applies to multi faceted problem solving. It also seems to me that this application can be used, with proper facet emblem, to specifically gain insight into an individual gamer's ability to demonstrate spatial awareness in a 3d environment like piloting an aircraft. The application may be useful in evaluating prospective pilots for the military.

Participant 3: the use of constructing words instead of matching designs requires more thought and stimulation than a trial and error way of solving puzzles

Participant 4: Great to challenge how I am used to seeing and playing games. Also, it is good to learn how to use different strategies to achieve one goal.

Participant 5: '8/10

Participant 6: Alright, a little simple due to being a 2x2 cube but still good

Participant 7: Makes you think harder and over times hopeful get to learn new terms.

Participant 8: It was found useful

Participant 9: this is a really challenging puzzle so i can see there being educational value here. it might be a little too difficult for some which might discourage some. i think people can learn a lot from this if it is made to be a little easier to solve.

Participant 10: Being a word puzzle, I can definitely see this as useful for teaching basic words. It certainly fits as an edutainment game.

Question 9: Do you have any additional comments?

Participant 1: None

Participant 2: Although my negative and positive critiques touched on symbolism distinction, colors, and visible separation, this is based solely on my initial gameplay. I actually do find promise in designing differing ranges of difficulty by manipulating these same salient characteristics, in order to make the game more or less challenging. The ability of the programmer to do this enhances the application's usefulness not only for entertainment, but as stated before, evaluation of individuals' current and prospective capabilities as related to piloting vehicles, aircraft, spacecraft, etc. I also suspect that a version of this application can assist in therapy for people recovering from brain injury.

Participant 3: No

Participant 4: I found it difficult to use and I spend more time figuring out what to do than actually playing the game. For someone who is not great with computers or technology, this would be very challenging. I am not the best with computers or technology.

Participant 5: None

Participant 6: Maybe make it easier to turn cube to see other sides

Participant 7: I like the concept, keep developing it!

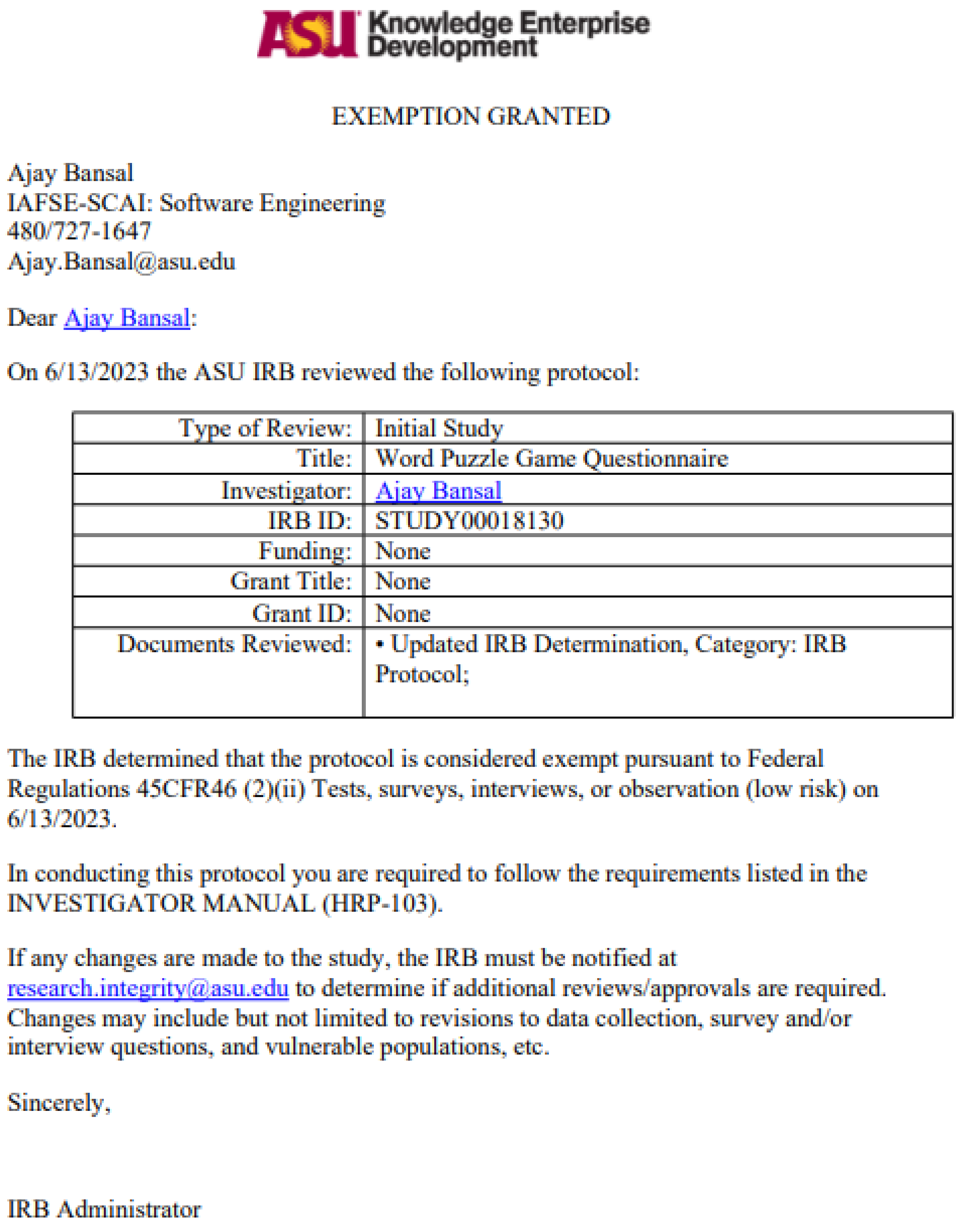
Participant 8: None

Participant 9: i think it's a cool project overall!

Participant 10: None

APPENDIX B

IRB EXEMPTION CONFIRMATION



EXEMPTION GRANTED

Ajay Bansal
IAFSE-SCAI: Software Engineering
480/727-1647
Ajay.Bansal@asu.edu

Dear Ajay Bansal:

On 6/13/2023 the ASU IRB reviewed the following protocol:

Type of Review:	Initial Study
Title:	Word Puzzle Game Questionnaire
Investigator:	Ajay Bansal
IRB ID:	STUDY00018130
Funding:	None
Grant Title:	None
Grant ID:	None
Documents Reviewed:	• Updated IRB Determination, Category: IRB Protocol;

The IRB determined that the protocol is considered exempt pursuant to Federal Regulations 45CFR46 (2)(ii) Tests, surveys, interviews, or observation (low risk) on 6/13/2023.

In conducting this protocol you are required to follow the requirements listed in the INVESTIGATOR MANUAL (HRP-103).

If any changes are made to the study, the IRB must be notified at research.integrity@asu.edu to determine if additional reviews/approvals are required. Changes may include but not limited to revisions to data collection, survey and/or interview questions, and vulnerable populations, etc.

Sincerely,

IRB Administrator

cc:	Jacob Hreshchyshyn
	Jacob Hreshchyshyn