Vision Without Sight: Developing an Audio Game Engine for Blind and Visually Impaired Creators and Players

Freya Shaw

MA By Research

University of Huddersfield
Music Technology

June 2025

Abstract

This MA project, 'Vision Without Sight: Developing an Audio Game Engine for Visually Impaired Creators and Players', aims to demonstrate the critical need for technology, particularly development software for media such as game engines, to build screen reader access from its inception.

Without screen reader adaptions, many are denied access to digital tools, skills and occupations. A study of common screen reader frustrations outlined typical accessibility issues, including:

> *"(a) page layout causing confusing screen reader feedback; (b) conflict between screen reader and application; (c) poorly designed/unlabelled forms; (d) no alt text for pictures; and (e) 3-way tie between misleading links, inaccessible PDF, and a screen reader crash." [34]*

In spite of the study's age, many of these issues persist in typically visual environments such as game engines.

> *"Regardless of advancements, game engines remain difficult to access for blind developers or those requiring non-visual interaction. This gap limits the potential for disabled developers to create games and experiences on par with sighted developers, as there are few tools that offer screen reader support or other accessibility-friendly design options." [52]*

Creative development software must ensure the inclusion of blind and visually impaired (BVI) users in navigating platforms typically accessed through complex graphical user interfaces (GUIs). To address these accessibility issues, I, a visually-impaired programmer who accesses devices with VoiceOver, Apple's built-in screen reader, developed the Hodr Engine, an accessible game engine for BVI and sighted creators and players. I created this platform in Python, where users can make audio-based games using HodrScript, a basic programming language, developed with

ANTLR, a tool for creating parsers to process text. Programming in HodrScript, running code, and packaging applications are all executed within an integrated development environment (IDE) powered by PyQT, a Python library for building user interfaces.

Research methodology involved gathering survey data from BVI users regarding their level of visual impairment, preferred game genres, suggested accessibility-based techniques and play styles, interest in games with visual elements, and previous experience with technology and game development. This helped determine the complexity needed for the user interface, common technological challenges for BVI users featured in software that must be identified and avoided, and the types of games that should be produced by the Hodr Engine. A participant, referred to as Herbert, was introduced to HodrScript through a tutorial, and used the engine to create an audio game and also provided feedback on his experience.

Initially, the engine was designed to run through the Terminal, which is a macOS command line interface (CLI), where users could code in a .txt file and bundle games using sound files and splash images. However, research revealed that the Terminal is challenging for some BVI users, especially those less familiar with programming or navigating CLIs. Contrary to my hypothesis that a text-based environment would be more accessible due to its historical use by BVI programmers, the complexity this virtual environment and terminal navigation proved to be a barrier. As a result, I developed a more accessible IDE with features such as a text editor, buttons for running and packaging games, shortcut keys, and tutorials, all in one place. The application can be downloaded from the Hodr Engine website, .

This research concludes that a basic user interface is more accessible than Terminal for screen reader users who are not experienced coders, even though the latter is text-based. Including navigational shortcuts and accessible design features in the IDE significantly improved the engine's usability for BVI users.

Contents

Thesis Contents

This commentary is accompanied by the following resources:

- *A Hodr Engine IDE folder and README.*
- *Tutorial on utilising the Hodr Engine and programming in HodrScript.*
- *A folder containing Herbert's FinishedGameIE and The CeReNeM Audio Experience application.*

All applications function exclusively on macOS. Upon attempting to open an application, a message may appear stating that the app cannot be opened because the developer is unidentified. If this appears, navigate to System Settings, select Privacy and Security, and grant permission to open the application.

Acknowledgements

My research would not have been possible without the support of my family, friends, and colleagues. My gratitude goes out to my parents, brother, grandparents, and partner for their assistance throughout this year of research. From providing transport to and from the University of Huddersfield to helping me navigate online platforms, I cannot express how much you have supported me and how much I appreciate you.

I am also grateful to my supervisor, Dr Alex Harker, for his guidance and insight into audio programming, software development, and thesis composition, which I've accelerated under your tutorship. In addition, my programming mentor, Dr Adrian Jackson's regular coaching over the last three years sparked the foundations of my programming knowledge and provided the skills to develop the Hodr Engine, which I cannot wait to expand.

I extend a thank you to the members of the CeReNeM Journal for allowing me to participate in your work and integrate the Hodr Engine into our journal project.

I would also like to acknowledge the team at Innovate UK, whose belief in the Hodr Engine and support in its establishment and promotion has made such an impact on the engine's progress.

Last but not least, I thank the participants who contributed to my survey, particularly a participant who is referred to in this paper as Herbert, for their valuable input and perspectives.

I'm excited to continue building the Hodr Engine and will always value the help of everyone on this list. Thank you.

Collaboration with Dr Adrian Jackson

The Hodr Engine was developed with the input of Dr Adrian Jackson. Jackson's involvement began during my undergraduate major project, in which I produced a prototype for an audio game called Immersion Sound Studio. This audio game is centred around a recording studio containing a portal door, which transports the user to various soundscapes where they can collect sounds and listen to them within a curated auditory environment at the end of the game.

At the time, I did not have the skills to code such a project independently, so Jackson scripted the game's foundations, because common game engines, such as Unreal Engine and Unity, which include tools like Unreal Engine's Blueprints and Unity's Bolt for visual scripting, were inaccessible to me. His Python code included core files for the project, such as gameutils.py, which provided utilities for the game, like the menu system, a function to collect sounds during gameplay, and the ability to play sounds at the end of exploring soundscapes; soundscape.py, which handled stereo soundscaping functionality; and SoundsX.py, which contained hardcoded soundscapes.

We approached this game code as a rudimentary game engine, so I could build on Jackson's groundwork. For instance, I added new soundscapes in SoundsX.py, modified gameutils.py to adjust player movement (like increasing stride length), and also introduced new functions within the soundscape.py file. The utilities from gameutils.py and the Soundscape module in soundscape.py were later incorporated into the Hodr Engine but were heavily modified by me. This transformation shifted the code *from being game-specific to being a functional game engine.*

During the Hodr Engine's evolution, Jackson provided an introduction to ANTLR, a tool for parsing and processing structured text, used to create programming languages. This explanation of how ANTLR operates enabled me to construct the HodrScript language independently. Jackson also provided a structure for the HodrListener (a listener processes code during parsing, and a parser analyses text to extract structure). The rest of the game engine's code, including its modification from a hard-coded game script to code suitable for an engine, was developed independently by me, as was the HodrScript and IDE's creation, and new functions for combat and puzzles.

Overall, the development of the Hodr Engine was built from the original game code's structure. These scripts were a framework from which I modified and produced new code to transform the specific game code into a more generalised game engine with Jackson's mentorship.

— 1 —

Introduction

1.1 ORIGINS OF THE HODR ENGINE

The Hodr Engine, named after Hodr, the blind Norse god, was devised to make game development accessible to BVI creators. The idea originated when I developed an earlier spatial audio-based game where users could explore immersive soundscapes. This concept was inspired by the audio game Sounds of Eden [14], in which users navigate a binaural soundscape. However, while trying to build this game, I, a visually-impaired user, encountered the inaccessibility of existing game engines like Unreal Engine [19], which rely on complex GUIs. Such engines' visually-oriented content is challenging for screen readers, which struggle to adapt to detailed graphical elements, and therefore is not accessible to screen reader users, who are typically BVI, alienating them from accessing those engines.

If you are a visual reader, you may notice a large gap between the text above and this block. However, the text above is written in white on a white background, making it invisible —much like the experience BVI users often face when struggling to access visually-focused user interfaces without proper support. Just as you may struggle to access the white text without assistance, BVI users encounter similar barriers. Below is the once invisible text, now in black on a white background:

This section is written in white text on a white background, making it invisible to visual readers: If you are reading this commentary with a screen reader or a refreshable braille display, you will likely have no difficulty accessing this block of text. However, since the text is white on a white background, it is inaccessible to visual readers, creating the impression of a large gap between this block and the one above.

Recognising the need for an accessible alternative to popular game engines, I collaborated with Adrian Jackson, a retired University of Huddersfield computer science lecturer, to build an audio-based game, Immersion Sound Studio, using Python, and to understand the foundations of code and audio game design.

This previous project was initially developed using Emacs [51], a code text editor known for its BVI-focused screen reader, Emacspeak [44], and later Visual Studio Code [40], another code editor that can be operated with accessible features like shortcut keys.

During Immersion Sound Studio's production, it became clear how accessible a text-based programming environment is for BVI users compared to a non-BVI-adapted GUI. Screen readers and refreshable braille displays interact more effectively with text, making programming a natural fit for BVI users. I realised that the code developed for the Immersion Sound Studio could be restructured to create a BVI-accessible game engine. This idea led me to begin the development of the Hodr Engine.

— 2 —

Research Questions and Aims

The main aim of this project was to develop an accessible audio game engine compatible with screen readers, assisting BVI users in creating and playing games independently.

2.1 IMPROVING BVI ACCESSIBILITY IN THE CONTEXT OF GAME DEVELOPMENT

This project seeks to improve accessibility in the context of game development through the identification of inaccessible features in existing game engines and other software. A game engine was then created inspired by these findings that addresses these issues.

2.1.1 Participant Survey

The project first surveyed BVI participants, asking: *What are your experiences using software with adaptive tools in educational institutes and daily life?* The results of this survey were used to gain an understanding of the current state of assistive aid education and availability, alongside a review of the current literature.

The survey also enquired about gaming experience and preferences of BVI users, the results of which were used to influence the Hodr Engine's primary play styles and accessible features.

The survey also asked: *What are the common genres and techniques of audio games?* Responses provided an overview of popular audio game features that were integrated into the Hodr Engine, as also seen in previous audio games and papers. The surveys also provided a basis for generating guidelines for fellow researchers and game designers interested in implementing audio as a BVI-accessible medium.

2.2 BUILDING THE HODR ENGINE

Additionally, this commentary discusses the development of the audio game engine based on a literature review, the experiences and feedback from the participant survey, previous research, and a user's experience developing a game with the Hodr Engine.

2.3 COMPARISON OF INTERFACE ACCESSIBILITY

Finally, this commentary compares the effectiveness of screen readers in navigating command-line interfaces to that of graphical user interfaces in order to determine the most suitable option for a BVI-focused game engine.

— 3 —

Literature Review and Context

3.1 INTRODUCTION

This literature review presents existing research on accessibility-related challenges and solutions for the BVI community, covering general technological challenges as well as issues specific to game development and audio. It also includes research on non-BVI-focused video games and audio which have inspired genres and techniques for developing games accessible to screen reader users in this project. The review aims to explore the current state of BVI-based accessibility, including the outlined issues that set back, and the innovations that might improve, the development experience for BVI creators.

3.2 CHALLENGES AND SOLUTIONS FOR ACCESSIBILITY IN BVI GAME DESIGN

The Longitudinal Transitions Study [28] is an example of research into the general accessibility issues often faced by the BVI community, including technological challenges. The study provides insights into the experiences BVI youth face during their transition from education to employment. Hewett's research demonstrates the lack and need for inclusive technologies and support systems. This can be expanded to the technology and education needed for the creation and use of an accessible game engine. The study reviews technological challenges faced by BVI pupils, struggling to access online educational platforms. As a participant explained, "The virtual learning environment was not screen reader friendly, making it hard to access lecture notes and submit assignments.” [28]. Pupils found similar struggles when collaborating with their sighted peers, “Group members used platforms that weren’t compatible with my screen reader, so I couldn’t contribute effectively.” [28]. These inaccessible platforms prompted users to turn to adaptive tools, which also failed to operate successfully: “The screen magnification software often crashed during exams, causing significant stress and time loss.” [28] The user would not have needed to rely on failing software if not for a non-functioning platform. The study, Digital Accessibility of Online Educational Platforms: Identifying Barriers for Blind Students’ Interaction [35], outlined some reasons for setbacks in educational platforms, which extend to most software, including game engines. “The lack of semantic structure in HTML documents and the improper use of ARIA (Accessible Rich Internet Applications) roles can lead to significant barriers for blind users relying on screen readers” [35]. Similar research, such as the chapter, Inclusive Online Learning: Digital Accessibility Practices, in the book, Diversity in Higher Education Remote Learning [25], discussed the following software deficiencies: “Common barriers include untagged PDFs, images without alternative text, and multimedia lacking captions or transcripts, all of which impede screen reader usability” [25].

There is a common trend between accessibility issues in educational software and creative software, such as gaming. The paper RNIB Accessible Gaming Research Report [46] presented the challenges faced by BVI gamers and the lack of accessibility knowledge amongst, and resources for, developers: “People with vision impairments reported substantial challenges in finding and being able to play video games. When asked why, they consistently selected ‘video games do not have enough accessibility features” [46]. Aligned with the research findings of Lisboa et al. and Frey et al., many BVI gamers encounter difficulties navigating buttons, menu screens, and complex visual environments. Similarly, as observed in Hewett’s study, participants depend on external accessibility tools, such as the participant’s “screen magnification software” [28], often used outside of their intended application. Consequently, BVI users often create bespoke accessibility solutions to adapt these tools: “Most gamers with sight loss report heavy reliance on their own, ad-hoc coping strategies, like playing with sighted guidance, memorising button sequences and menu layout or using Be My Eyes or Seeing AI to read what is on the screen” [46]. To mitagate these issues, Lisboa suggests “Developers should adhere to the Web Content Accessibility Guidelines (WCAG) and ensure that all interactive elements are properly labelled and navigable via keyboard” [35]. In conjunction, “Faculty can use accessibility checkers to identify common barriers with guidelines for remediation, such as adding alternative text to images and ensuring documents are properly structured" [25]. These experiences highlight the troubling prevalence of inaccessible platforms, spanning gaming and education, for screen reader users. To mitigate the issues raised by Hewett, the Hodr engine ensures all interactive elements, including buttons, links, and form fields, are labelled, the platform is navigable via shortcut keys, and images and visual layouts contain descriptive alternative text. The RNIB Accessible Gaming Research Report identifies potential solutions to gameplay for BVI users: "Audio-based solutions are most desired by blind and partially-sighted gamers. These include screen reader compatibility, audio description, audio triggers and adaptable audio settings and sound mixes (e.g., spatial audio)" [46]. Audio features, like spatial audio, could also be used outside of gaming, as they are not specific to this domain. The study also provides solutions for accessing visual content, which may align with more general accessibility requirements: "Customisable visual features are highly desired, including adaptable text size, colour contrast, and magnification options" [46]. The study concludes with a statement regarding gaming, also relevant to all forms of software accessibility: "Developers should be provided with training and resources to better understand the needs of gamers with sight loss and to implement accessibility features effectively" [46].

Resources on accessible gaming techniques for blind gamers are growing, such as AudioGames.net [9], which offers forums and a list of audio games, connecting developers to BVI players. In addition, The Game Accessibility Guidelines website [26], contains information on making games accessible, along with studies like the RNIB Accessible Gaming Research Report [46], which highlight barriers and solutions for blind gamers, and offer guidance to studios wanting to design accessible games. For creators interested in partnering with a BVI-focused organisation to receive guidance on accessibility, AbleGamers [1], is a nonprofit organisation for improving accessibility in gaming for all disabilities. The charity also runs scholarships for those interested in game development with a disability, such as Alex James Ryan, who started his initiative, Inclusive Games [29] after winning the inaugural AbleGamers Fellowship in 2016. AbleGamers collaborated with Harman (JBL), and developed the JBL Quantum Guide Play software [31], allowing BVI users to play first-person shooter games through optimised directional audio, a feature that precisely aligns sound with its corresponding visual direction.

For studios interested in practical coding resources, Apple and Microsoft provide resources and tools for software accessibility. Apple's Accessibility Inspector [3] is integrated into Xcode [5], Apple's development environment, and identifies non-voice-over friendly software elements and suggests improvements. The Human Interface Guidelines [39] are a set of principles for developers to create inclusive and user-friendly applications. Meanwhile, Microsoft's Inclusive Design Toolkit [38] offers practical tutorials and tools, as does the Xbox Accessibility Guidelines [41]. Educational institutions must inform their students about these resources, and there must be greater public knowledge of these initiatives and technologies if games are to offer greater accessibility to BVI users. Regarding documentation, the Hodr Engine is accompanied by detailed documentation in what it aims to be an accessible format for BVI users.

3.3 AUDIO GAMES AND ACCESSIBLE TECHNIQUES

To build an accessible audio game engine for BVI developers and players, popular audio games and their accessibility features have been explored to understand what makes them accessible.

The audio game Sounds of Eden [14] inspired my previous audio game, Immersion Sound Studio, a predecessor to the Hodr Engine, due to Sounds of Eden's use of spatial audio to create an atmospheric soundscape. This idea was expanded upon in Immersion Sound Studio so that players could curate personalised soundscapes. Sounds of Eden is a meditative audio game in which the listener can explore a garden where plants are represented by slowly increasing arpeggiations. Sound of Eden's soundscape organises audio spatially and is an ideal example of the type of game that can be developed with the Hodr Engine, especially as the Hodr Engine features an expandable map that allows audio files to be positioned using x- and y-coordinates. Sound of Eden's stem files could be plotted in the Hodr Engine to recreate the game's soundscape.

The paper, Accessible Games for Blind Children, Empowered by Binaural Sound [17] documents the creation of a binaural game to improve the spatial awareness of BVI children within a game environment. The audio techniques presented in the paper provide a guide for methods that may be used within an audio game engine. The game implements popular audio game techniques in addition to the spatial soundscape explored in Sounds of Eden, such as *auditory icons*. The article, Auditory Icons [15] explains: “The rationale behind auditory icons is to support and supplement predominantly visual information with a corresponding sound. For example, the action of successfully emptying your trash on a Mac OS would result in the sound of paper being crumpled up and discarded. The most immediate benefit of these sounds is a tangible confirmation of the action you have performed. Much like in the real world, you press a button and it responds (in that sense even the click of a mouse could be considered an auditory icon)” [15]. Didlick suggests that there are four forms of auditory icon:

- “Nomic: Sometimes referred to as literal or realistic, this type of auditory icon is a direct sonic representation of the event taking place” [15]. An example of this would be the iPhone’s camera sound effect when a picture is taken.
- “Symbolic: Symbolic auditory icons are perhaps the antithesis to nomic. They do not possess any specific connection to the event taking place other than by virtue of its association and repetition” [15]. For instance, Apple’s ‘woosh’ sound effect that plays when a message is received is an example of a symbolic auditory icon. Unlike the sound of opening an envelope on receiving a message, a chime, like a bell, is more abstract.
- “Metaphorical: A metaphorical auditory icon is one that reflects a key aspect of its associated action with a dimension of the sound” [15]. For instance, a major chord might indicate a positive result, whereas a minor chord might symbolise s negative one.
- “Verbal: As the name would suggest, verbal auditory icons are sound bites of spoken language. Whilst these are not always the quickest way of sharing insight, they are most effective when conveying very specific information” [15]. For instance, the announcement, “Mind the gap between the train and the platform edge” often repeated in train stations is a verbal auditory icon. In this case,“The transfer of information is performed by correct assignment of the communicated pieces of information to the actual semantic content of the auditory icon” [17].

The Vale: Shadow of the Crown [21], commonly referred to as The Vale, is a combat-heavy audio-based action-adventure game, implementing spatial soundscapes and auditory icons throughout gameplay. In an interview, David Evans, the game’s director, discussed using auditory icons he refers to as “beacons.” These icons assist players in navigation and interaction within the game’s world. “Evans spent a lot of time wrangling some of The Vale's more important cues – such as beacon sounds that direct players to objectives – into place. 'One thing I wrestled with forever, and never really knew exactly why it worked, were the magic beacon sounds. I found those became competitive with other sounds as soon as you had two objectives, like find this magic object but avoid or fight these beasts' " [32]. The use of multiple auditory icons to guide the player through a soundscape in The Vale is dominant within its market scenes, where players explore activities, such as purchasing items, playing mini-games, and advancing the storyline, by approaching sounds and interacting with them to trigger an associated activity, such as speaking with a stallholder or buying an item. The auditory landscape is constructed through a combination of non-player characters (NPCs) positioned in different directions relative to the player, announcing their services simultaneously. These environments also contain ambience such as music and chatter from a tavern or church bells, indicating enterable buildings. Despite the complexity of multiple auditory icons, the scenarios remain comprehensible and immersive rather than overwhelming. Similarly, the Hodr Engine uses auditory icons to represent interactive items like puzzles or characters that can be engaged in combat, as well as to alert the user to other actions, such as indicating that the game has been saved.

Auditory icons are also incorporated within the Hodr Engine's IDE, providing auditory feedback for actions such as panning to indicate the cursor's location within a text field or the use of auditory icons when buttons are selected. A similar approach is employed in the code editor, Emacs [51], which integrates shortcut keys in conjunction with auditory icons linked to specific actions, For instance, the shortcut Command+S saves progress, accompanied by a symbolic bell auditory icon.

Auditory icons assist BVI users in exploring the game environment within menu screens through key or controller combinations to navigate an environment, with verbal auditory icons announcing a menu option one at a time. This method is implemented in the majority of audio games' menu screens. However, in the audio game series Blind Quest [6], a fantasy role-playing series centring around a mercenary's adventures, the navigation mechanics are not designed with complete freedom of movement, and instead use a menu structure. A narrator announces the story and items within an environment in a menu-like structure, presenting one option or item in an environment at a time. When items are interacted with, they trigger auditory icons, and contain descriptions such as "Nathan opens the casket, he gets a healing potion" or "On the wall there are some coats of arms and old weapons." This approach is limited in its flexibility when compared to the Vale or Sounds of Eden, but reduce cognitive overload, keeps the game at a controlled pace, and may be easier to comprehend when compared to busy environments.

A similar menu structure to that found in Blind Quest can be implemented within games made with the Hodr Engine for the purpose of controlling overall gameplay. These use arrow keys to highlight items such as "Start", "Continue", and "Cancel", with selections made by pressing the spacebar.

Non BVI-exclusive publications, such as Studying Sound: A Theory and Practice of Sound Design [13] may also provide a wealth of information for developers to expand the realism of audio, such as in games design. This book provides an overview to better understand audio effects, like occlusion and reverb, in sound design. Collins' work in sound design expands to audio games, such as Audio Defence: Zombie Arena [18], indicating her knowledge of audio-exclusive games. Studying Sound: A Theory and Practice of Sound Design presents sound design techniques relevant to audio and visual games so developers uninterested in audio-only games may still expand their knowledge of methods like spatial audio that might make a visual game BVI inclusive. The combination of spatial soundscapes, as seen in Sounds of Eden, auditory icons / beacons, as found in The Vale, and auditory menu screens, as found in Blind Quest, supplied a foundational framework for the Hodr Engine.

3.4 OTHER BVI ACCESSIBLE CREATIVE SOFTWARE

The paper, An Empirical Evaluation of a Graphics Creation Technique for Blind and Visually Impaired Individuals [22] introduced the SETUP09 system, a compass-based drawing tool for BVI users: "Little research has focused on the use of a screen layout to provide people who are blind and sight impaired users with the spatial orientation to create and reuse graphics. This article contributes an approach to navigating on the screen, manipulating computer graphics, and user-defined images. The technique described in this article enables features such as zooming, grouping, and drawing by calling primitive and user-defined shapes" [22]. Tools supporting creative skills, such as graphic design, are often integrated into game design for developing visual elements, such as models for environments, and user interfaces. An example of this is the integration between Blender [10], a software for 3D modelling, visual effects and animation, and Unreal Engine [19]. Given the prevalence of such tools it might also be important to explore how programs for creating visuals are made accessible.

As a screen reader-compatible game engine necessitates a non-visual approach for positioning audio within a map to construct an immersive soundscape, SETUP09's use of compass directions to plot points might be useful for creating an environment using a screen reader and auditory icons: "The cells are not identified by numbers but by compass directions such as north, south, east, west, north-west, north-east, south-east, and south-west. Compass-based navigation also has nine unique points in a cell rather than one unique centre point as in the grid-based navigation system and a formal command language operates it" [22]. Fernando explains, "Compass-based navigation is derived from the grid-based navigation concept but with reformed locations, points reference system with cardinal directions, and implemented with a formal language specification to work with lexical inputs" [22]. SETUP09's compass-based navigation system provides a structured framework for BVI users creating graphic designs, with limited risk of falling out of place. However, in an audio-focused platform, restricted spatial constraints may not be as essential as in a context where BVI users are creating visuals. BVI users, who are not deaf or hard of hearing, often rely on auditory feedback to evaluate sound placement, and can then adjust sounds' positions effectively by modifying their coordinates in code. The more accessible medium of audio for BVI users, when compared to the visual, reduces the need for rigid compass-based frameworks. While compass-based navigation assists with the prevention of unintended movements, like drawing a line in a different direction than planned, it might limit freedom in more BVI-accessible contexts, such as plotting sounds in a soundscape. In contrast to a compass-based system, a grid-based system may allow for more flexibility to place sounds on a map. For this reason, the Hodr Engine adoptis a grid-based system, mapping sounds with X and Y coordinates.

3.5 ACCESSIBILITY CHALLENGES AND SOLUTIONS WITHIN PROGRAMMING LANGUAGES

The publication, Improving the Accessibility of the EarSketch Web-Based Audio Application for Blind and Visually Impaired Learners [16] explores how the EarSketch platform, which teaches coding through music, was modified so BVI users can access it through auditory icons. The platform allows users to program in Python or JavaScript to create music. As discussed in the paper, this approach is effective for BVI students of coding because it allows them to experience programming through audio rather than visual cues. "EarSketch's integration of music and programming languages significantly improves student engagement and persistence in computing" [16]. As discussed in the paper, the platform's original design met some accessibility barriers. For instance, EarSketch relies on multiple panels in its user interface, including a content browser, digital audio workstation (DAW), code editor, and curriculum sections. This multi-panelled interface was, at first, not completely accessible via screen reader as the original version needed to be operated using a mouse. Mouse-driven interfaces do not integrate well with a screen reader because screen readers interact with a system's accessibility framework, which tends to be keyboard-based. The original version of the software also included fewer shortcut keys than required to navigate the entire platform, rendering some areas of the platform non-accessible to BVI users. This previous visually-focused user interface "requires design changes to become more accessible to users who are blind or visually impaired" [16], for example, "prioritising keyboard-centric navigation" to facilitate easier access for blind and visually impaired users, particularly in educational settings" [16]. The paper also states that the inclusion of "descriptive audio cues" and "tactile feedback for learners who are BVI" would improve interaction. As a result of these identifications changes were made, so actions such as selecting a sound sample, moving between sections, or executing code are now accompanied by audio confirmations, and navigating the panelled interface and code itself can be completed through key combinations. A similar approach is perfect for the text-based Hodr Engine. For instance, when navigating a section of code, an auditory confirmation plays when moving to the start or end of a line with a shortcut key. This avoids the need to move through characters one by one or a screen reader user getting lost in sections of code.

3.6 BLIND ACCESSIBLE GAME ENGINES

The Accessible Gaming Research Report [46] found developers shared an interest in understanding the accessibility needs of BVI gamers. However, many felt they did not have the resources or knowledge needed to improve the accessibility of their games: “There is a knowledge gap in the games industry; whilst 75% of developers who participated in our research reported incorporating some accessibility features in their games, only 15% reported having sufficient understanding of the needs of gamers with sight loss” [46]. Developers acknowledged one of the primary barriers to achieving accessibility in their games is due to “a lack of game engine support for accessibility features” [46]: “Accessibility should be considered from the outset of game development, rather than as an afterthought, to ensure that games are inclusive by design” [46]. This necessitates designing game engines with inherent accessibility features. Even if a developer creates visually-oriented games, the project should incorporate built-in accessibility tools, such as magnification options, accurately labelled buttons, spatial audio, and auditory icons. As a result, the game would remain accessible regardless of the developer’s initial intent.

Developers included in the Accessible Gaming Research Report shared that due to a lack of knowledge and the lack of accessibility features in existing game engines, they tended to cater to an audience with some working vision, as their needs are easier to meet than those without any access to visuals. “Developers give more consideration to the needs of gamers with partial sight loss than those with severe sight loss, suggesting that the former are easier to address and that additional focus is required on the latter” [46]. This information gap is due to a concerning lack of distribution of accessible resources. Many studios included in the report discussed a need for wider resources on accessibility: “Over 70% of developers would like to see sharing of knowledge and technology within the games industry and better resources on accessibility good practice” [46]. As this commentary explored previously, the volume of BVI and general disability-focused resources is growing. This information must be shared more widely amongst game studios and educational institutions so developers can make use of it in their work. In addition, game engines must incorporate inclusive techniques into their core functionality so that developers might be able to create accessible games without even realising it.

This commentary illuminates existing research on BVI-accessible game engines, helping us understand how they have been designed for screen reader users. The paper, Blind Adventure: A Game Engine for Blind Game Designers [50] introduced a game engine design for BVI users to create audio games for mobile devices. “The main idea of Blind Adventure is to give a tool to blind people that lets them create adventure or other genre games, play them, share them with their friends, and generally be creative. The game engine and the games should run entirely on smartphones, which have a high penetration rate amongst visually impaired people" [50]. The application has two modes: one for playing and one for developing. In creation mode, the engine guides the user through a hierarchical menu system with audio feedback, navigated with swipes and taps, and accompanied by haptic feedback. Each level consists of nodes, such as collectable items or puzzles and combat mini-games, each tied to a specific recorded sound. "The app acts like an audio recorder, thus being the tool for creating the games, but also for playing them" [50]. In the playing mode, users may play their own games or previously created games made by fellow Blind Adventure users. Players navigate the game through audio prompts, completing levels and collecting items to progress.

Despite Blind Adventure's innovative approach and potential, the project has not progressed publicly. Even so, the design outlined in the paper inspired ideas for the Hodr Engine. It is acknowledged that Blind Adventure appears to be built with creativity and fun at its forefront, rather than complex mechanics, so its abilities are purposefully limited for ease of use. With this in mind, the app uses audio recording directly on the device, which may limit sound quality based on the user's phone or external microphone: "Audio recordings are done using the smartphone's microphone and are stored as WAV files" [50]. The audio is also limited to a maximum length of time: "One of the biggest challenges is here given by the fact that such recordings must have a preset length. This was set to 30 seconds but requires that the unused time at the end is deleted afterwards" [50]. The audio length and quality are constrained by the device's microphone or external microphones that can be connected to the device. The paper does not indicate support for importing external audio, even though many game developers often incorporate sound effects, voice-over, text-to-speech, or soundscaping from existing sound libraries, collaborators, and other software. The app design also limits the number of options per screen for ease of access: "Since user choices would have to be identified through touches in different areas of the screen by blind people, we decided to limit user options to only two per screen" [50]. This choice, while accessible, may restrict the complexity of interactions compared to more flexbile game engines. Blind Adventure appears suited for structured gameplay similar to the menu structure of Blind Quest. It is a great introduction to early forms of game design, meaning users could smoothly focus on the creation process with the knowledge their game will be accessible even for a novice. However, this restricted feature set would need more depth for more explorative designs. Games created in Blind Adventure demonstrate some of the popular audio game-based techniques that are simple to access and play, such as combat in which sounds of opponents are heard in certain directions, and the user must swipe in particular directions to defend themselves, similar to the combat of the Vale. A second game Blind Adventure included is steeple chase, in which users navigate through an obstacle course by swiping to avoid sounds in certain directions. The Blind Adventure design also proposed a quiz mini-game, in which the users hear a question followed by possible answers, with the user swiping through the answers to select the correct one. Similar conventional audio-focused mini-game templates have been implemented in the Hodr Engine within a larger set of soundscapes, so users may explore an open world and play these mini-games within the world to build a more complex story.

Mobile device users use of swipes and taps in response to auditory icons can be translated to desktop platforms by employing arrow keys to navigate in different directions rather than relying on finger taps and swipes on different parts of the screen. This allows the Hodr Engine to create similar gameplay mechanics to those in Blind Adventure in a desktop environment.

3.7 LITERATURE REVIEW AND CONTEXT CONCLUSION

In conclusion, this literature review presents existing research into broader accessibility experiences and requirements for BVI users, research into audio techniques and play styles in audio games, and an existing audio game engine. Information from each study provides context that has led to a design for a BVI-accessible audio game engine, the Hodr Engine. This was achieved by examining existing challenges in software, and outlining screen reader supportive solutions to make BVI-prioritised game development and gameplay widely accessible. Studies referenced in this literature review underscore the need for software to be screen reader adaptive, using solutions such as semantic HTML structures, alternative text for images, shortcut keys, and auditory feedback for screen reader compatibility within software development, especially for a BVI-inclusive IDE.

To address these accessibility issues, the Hodr Engine implemented clearly labelled buttons that reference their shortcut keys, as well as customisable features for visual layout like background colour, text colour and text size. Finally, the IDE includes auditory icons to provide feedback for actions, such as navigating different sections of code, saving progress, and confirming visual changes.

Audio games developed with the Hodr Engine include popular BVI-accessible techniques, such as interactive soundscapes explored with up, down, left and right keys, as demonstrated in Sounds of Eden, and directional audio for combat scenes as found in The Vale. In conjunction, structured menu navigation as seen in Blind Quest is applied in the menu screens used in Hodr Engine games. EarSketch's approach to shortcut key-centric navigation has been a model for the Hodr Engine's use of shortcut keys throughout the IDE and audio games it creates, with keyboard commands for navigating code, selecting every button within the IDE, and operating the audio games created by the engine.

— 4 —

Methodology and Development

4.1 DATA COLLECTION

To begin developing the Hodr Engine, I used the existing Python Pygame code structure from my audio game Immersion Sound Studio, which included limited features such as creating a stereo spatial soundscape and collectable items. The program was adapted so elements like audio files and soundscapes could be customised rather than hardcoded.

The engine's core functionalities were expanded to accommodate genres and play styles engaging to BVI users. Mechanics deemed practical and aesthetically pleasing by BVI respondents were collected through survey research and incorporated in the engine. The goal of this research was to create an inclusive engine that caters to preferences of BVI users. Qualitative and quantitative data were gathered through a survey of eight BVI participants. The survey begins with general questions, providing foundational information from respondents, including their age range and varying degrees of visual impairment, from vision corrected with glasses to total blindness. Questions also explore how respondents access digital content, with aids such as screen readers and magnifiers, as well as challenges such as eye strain and nystagmus preventing visual use of software.

The survey includes accessibility-based questions to understand respondents' interactions with accessibility tools, and also questions about formal technological education to discern a common level of IT knowledge and access. These questions garnered mixed results. Many participants reported learning technological skills with accessible tools through formal education, such as lessons at the Royal National College for the Blind. Others noted gaps in formal education on these tools.

The next section investigates questions exploring preferences and favourite genres of video and audio games. When asked about suggested accessible features, users advised actions on a touch screen, shortcut keys, or voice commands to navigate games. Responses demonstrated a mixed interest in creating audio exclusive games, as opposed to games also incorporating visuals, for both BVI and sighted users. A participant described wanting to create an environment for artists to share their work and collaborate, prompting this research to consider whether the contributor was referring to audio or visual artistry. If visual elements were intended, future iterations of the engine would need to incorporate multiplayer functionality, visual components, and a spatial audio environment. However, additional visuals should not be essential to gameplay, so blind players are not isolated.

The questionnaire also enquires about participants' experience in recording and editing audio, which extends to music, sound design and voice acting for games. Understanding the general experience and skills in audio production within this group provides insight into whether additional features, such as reverberation and echo, should be built into the game environments or if the majority of users can edit these effects themselves using a Digital Audio Work Station (DAW). It also indicates whether users will have their own recorded music and sounds available, or if a music and sound library should be integrated into future iterations of the engine. Results were mixed, with most participants reporting little experience in audio production, although one participant shared that they had considerable experience in personal and professional audio production work

The survey feedback contributed to the foundational design of the Hodr Engine and the games it is capable of creating. The Hodr Engine already had the ability to develop stereo soundscapes after its update from the Immersion Sound Studio game to game engine, but lacked the ability to create games including combat or puzzles, both genres favoured by survey respondents. During the Hodr Engine's restricted development period, it was not possible to implement all the suggestions raised in the survey. Consequently, I prioritised the most frequently referenced genres.

Numerous comments highlighted a strong interest in developing puzzle and combat games:

- "Open world / combat."
- "Puzzles and word games."
- "Brain exercising games, strategic games."

- “First person shoot them up.”
- ‘I enjoy card games and quiz games but also interactive player games where you complete missions or tasks.”

(Participant Survey)

Additionally, participants expressed an interest in adventure and open-world experiences, which were later developed using the Hodr Engine’s stereo interactive soundscapes. These soundscapes may be layered to create immersive environments featuring mini-games, explorable areas, and buildings. The following survey comments demonstrate an interest in open-world and adventure games:

- “I like to play racing, platformer, adventure and party games.”
- “Text adventure like AlterAeon, better with sound effects.”
- “I love playing adventure games, specifically story based games. I think ones that have multiple ways of playing it are my favourites as they allow me to go back through the story and choose different options, resulting in different outcomes. I prefer sound immersive games rather than text adventures as for me this adds to the experience. If I ever play games that don’t really have a story to them, these would also be preferred as audio games. I also enjoy games where you get to build your own settlements. I would also love it if there were more accessible fantasy games available.”

(Participant Survey)

With the aforementioned feedback highlighting a predominant interest in creating puzzles, combat, and open-world-focused games, I prioritised the inclusion of these three genres in the engine.

The survey also reveals the accessible gameplay techniques users wish to integrate into games which may implemented in combat, puzzles or open-world genres. including gestures on a mobile device or shortcut keys on a keyboard. These particular techniques were suggested by nearly every participant in the following examples:

- “A screen reader to announce text and navigation by moving a finger on the mobile screen.”
- “A mix between touch screen and computer keys. Computer keys give me more control and touch screen can allow me to explore the screen.”

- "Touch screen and controls."
- "I would prefer shortcut keys as they are easier to locate."
- "Keyboard."
- "I would create a game that would be available across different platforms. Therefore, it would have to be available for a variety of inputs. For example, arrow keys on a computer or touch screen gestures on a phone and tablet. Perhaps on a laptop it could have a cross between using the arrow keys and the trackpad. I've seen this in a video game I've played called As Dusk Falls and it makes the game more interactive. On a phone and tablet gestures would include swiping left, right, up and down, double tapping to interact with elements, and perhaps scrubbing or swiping with multiple fingers to conduct an action."
- "Voice activated."
- "I would want to create a game that people are able to navigate on their phones as that is what I find most accessible."

(Participant Survey)

The previous Immersion Sound Studio game was built for macOS, and likewise, the Hodr Engine's current version is exclusive to macOS, but I aim to expand it to Windows and mobile in the future. Even though the question, "What platform do you play digital games on?" demonstrated mobile was the most popular platform for BVI users with 6 votes, and computer was second with 3 votes (making both of them a far more popular choice than tablet with 2 votes or console with 1), within the research timeframe, development time was devoted to only one platform. It was decided that a transition from game code to game engine code was smoother than writing an engine from scratch with in a programming language like Swift (exclusive to IOS and OS), Kotlin (exclusive to Android), or Java (popular for Android development). Therefore, participants suggestions for mobile features are directly relevant to the Hodr Engine at present. However, this should be explored in future. These survey responses demonstrate the perceived accessibility of the mobile platform for BVI user, which might be of interest to fellow developers and scholars.

However, participant's suggestions regarding the desktop platform is important to this research. The suggestions regarding using the keyboard to access games were as follows:

- "I would prefer shortcut keys as they are easier to locate."
- "Keyboard, with arrow keys in particular."

(Participant Survey)

These shortcut key suggestions, complement research referenced in the literature review such as that by Frey [25], Lisboa [35]. and that in the RNIB's Accessible Gaming Research Report [46]. A shortcut key-focused approach was implemented in the IDE and the games it creates. For example, drawing inspiration from Sound of Eden's use of arrow keys to navigate the garden [14], a comparable technique was adapted for navigation. Immersion Sound Studio applies a similar technique. However, this approach led to the user constantly facing a fixed direction, resulting in a 'crab-walk' effect in which the audio scene is in a fixed orientation. The left and right arrow keys were redesigned in the Hodr Engine to rotate the user's orientation, so they can turn and proceed in a new direction.

Arrow keys were also incorporated into other aspects of the engine's games. The up and down arrows are used to navigate menu options, with audible cues announcing each option's function, such as "Start Game" to begin a session. The space bar is used to confirm selections. Extra keys were designated for specific functions, such as S to save progress or Q to quit the game.

To create a puzzle design inspired by the popularity of puzzles mentioned in survey feedback, I took inspiration from Stadler's Blind Adventure Quiz [50], as discussed in the literature review, which presents a question accompanied by multiple-choice answers. I adapted this concept to a riddle format. In this version, developers input an audio file containing a riddle or question, accompanied by background ambience. Unlike the somewhat restricted multiple-choice approach, players input responses via keyboard. This approach means answers may include a word, phrase, sequence of words, symbols, numbers, or letters. A correct response would trigger a *win* auditory icon, while an incorrect answer would trigger a *lose* auditory icon. This approach requires exact responses from the player. In future updates this might be amended to be more forgiving, depending on how app testers respond to the puzzle design.

For the fight function, I drew inspiration from The Vale's directional audio mechanics [21], which involves the player listening to an opponent's position and responding by striking or evading accordingly.

In the Hodr Engine, combat is initiated when the user moves towards the sound associated with the fight feature, triggering the combat environment. Players rely on auditory icons to determine the enemy's position, left, centre, or right, and strike in that direction to land a hit. The developer can adjust the enemy's speed. If the player is too slow to respond, the enemy will strike, regardless of whether their positions match.

The user can attack with the F key, dodge enemy attacks by moving to a different position using the left, up, or right arrow keys, or shield themselves with the S key. Excessive movement causes the player to tire, which slows their actions and forces them to rely heavily on shielding until they have regained momentum.

The developer sets health points for both the player and the enemy, with each successful strike reducing the opponent's health by one point. When either the player's or the enemy's health points reach zero, the fight concludes. A loss triggers a *lose* auditory icon, whilst reducing the enemy's health points to zero results in victory, accompanied by a *win* auditory icon.

After the engine's backend programming was completed, I moved on to the frontend, where the user would design games with the HodrScript, and also package applications.

4.2 HODRSCRIPT LANGUAGE OVERVIEW

HodrScript was built using Another Tool for Language Recognition (ANTLR) [42]. ANTLR is a parser generator for language processing systems like programming language interpreters and translators. When making the HodrScript language, I first designed the grammar file, Hodr.g4, a definition of the language's syntax. The Hodr.g4 document establishes the structure and rules for HodrScript. Once the grammar was completed, ANTLR was used to process the grammar to generate the lexer and parser for HodrScript.

The lexer, or lexical analyser, processes HodrScript code and segments it into tokens, such as words or special characters common in the language, like sound or = (equals). The lexer also ensures that HodrScript code is written in the structure expected by the parser. The parser arranges tokens into a structured format called a parse tree, based on the rules of HodrScript's grammar. It also checks whether a program written in the HodrScript code conforms to the grammar defined in Hodr.g4. After lexical and syntax analysis, code generation takes place. This involves converting HodrScript into Python code for execution. Semantic checks are handled, such as making sure variables are declared before use or verifying the existence of referenced resources in the project's resources directory.

The full execution of HodrScript code involves the processes of lexical analysis, syntax analysis, and code generation, culminating in the running of a project created in the Hodr Engine.

HodrScript is a structured, basic programming language designed for creating audio games within the Hodr Engine. It allows users to modify Python code templates by adding audio and image files in a spatial audio environment. Its syntax focuses on simplicity, so that a screen reader can interpret variables, values, and commands during development.

Screen readers can sometimes misinterpret code due to their handling of syntax and special characters. They typically announce text as spoken words, not including punctuation: "This works well for regular text, but how they render other types of content is sometimes harder to predict" [43]. Consequently, some screen readers often miss punctuation unless settings are altered to announce all special characters.

A study examined how three popular screen readers, JAWS, NVDA, and Voice-Over, interpreted special characters and punctuation. The study compiled a list, noting those announced and those not announced. At least one of the three screen readers did not announce a significant number of characters. From this larger set, a subset of common programming characters, identified by Ragas, were not consistently announced by at least one screen reader. This subset includes characters essential for programming syntax:

- ~ (tilde)
- ! (exclamation mark)
- ((left parenthesis)
-) (right parenthesis)
- - (dash)
- _ (underscore)
- , (comma)
- . (period)
- \ (backslash)
- | (vertical bar)

- ? (question mark)
- ; (semicolon)
- : (colon)
- ' (single quote or apostrophe)
- " (quotation mark)
- { (left brace)
- } (right brace)
- [(left bracket)
-] (right bracket)

The study concludes, "The overall lesson to be learned from this thorough test of the support currently offered by the most widely used screen readers for special characters is that it is essential to:

1. Test your content with multiple screen readers
2. Use special characters only when there is a need for them
3. Pay attention when copying text from a word processor to an HTML file or the CMS of your website. Some word processors and text editors convert characters automatically (e.g. three periods to an ellipsis). When possible, always use the HTML entity.
4. Track the evolution of HTML support by assistive tools." [43].

This study does not focus on the challenges faced by screen reader users when navigating code. Rather, Ragas discusses special characters found in text-based articles. However, as it is impossible to avoid the need to read such character in many programming languages, such as Python, the issue of special characters being skipped by screen readers is still relevant. To minimise navigating code character-by-character or taking additional precautions outlined in the study, HodrScript was designed with minimal syntax requirements. This approach reduces potential confusion for both the program and the user. Each line of code in the HodrScript follows a predictable pattern involving a variable, assignment operator, and value, making the language straightforward for beginners and accessible for screen readers.

4.2.1 Core Structure

HodrScript is made of blocks of code to organise functionality and game story. Each block groups related commands, acting as a self-contained unit within the program. A Settingsblock configures game settings, while a Soundscapeblock manages an interactive environment. All blocks are terminated with the end command, as indents are also challenging for a screen reader user to input: "Screen readers aid with code navigation but dictate code line-by-line and read spaces and tabs individually. This often provides more information than is needed" [20]. This is why a block begins by announcing the title of the block and concludes with the word end.

Each line begins with a variable, such as the *sound* variable, followed by the assignment operator which is always = (equals), connecting the variable to a value. Values might be filenames for audio to be played in-game or images to be displayed during run time. They also include coordinates to place sounds within a spatial soundscape or integers for volume. For example:

```
sound = birds.mp3 (40, -40)
```

This line places a looping audio file within the soundscape at specific coordinates, providing directional feedback to the player. Comments enclosed in double quotes, are ignored by the program.

4.2.2 Example of the HodrScript and its Functionality

The Settingsblock is designed to establish game parameters, including toggle options. It allows developers to assign sounds to actions such as pausing cutscenes and soundscapes, or quitting and saving a game. A sound specifies when a cutscene or soundscape is paused.

```
settingsname = settings
```

The mandatory *settingsname* variable specifies the name value to the Settingsblock.

```
projectname = MyFirstGame
```

The mandatory *projectname* variable specifies a name value for the entire project.

```
startsoundscape = firstscape
```

The mandatory *startsoundscape* variable specifies a soundscape to play first. If you want to test a soundscape during development, you can modify the settings so that it becomes the first soundscape to play after the menu screen.

```
togglecutscene = skip.mp3
```

The mandatory *togglecutscene* variable specifies the audio file that plays when the C key is pressed to turn cutscenes on and off.

```
togglepause = pausegame.mp3
```

The mandatory *togglepause* variable specifies the audio file to be used when the P key is pressed to pause and play the game. The P key works exclusively during a soundscape but not during cutscenes or the menu.

```
save = savegame.mp3
```

The mandatory *save* variable specifies the audio file when the S key is pressed to save progress.

```
collectjingle = collectjingle.mp3
```

The mandatory *collectjingle* variable specifies the audio file to use when the space bar is pressed near an item to save it in an inventory.

```
choicebackground = choicebackground.mp3
```

The mandatory *choicebackground* variable specifies the audio file to use in the background of a game with multiple endings. This background sound would remain the same, whereas the items chosen may affect the other sounds in the end scene.

```
missed = missed.mp3
```

The mandatory *missed* variable specifies the audio file used when the space bar is pressed to collect an item, but the user is not close enough to the item to collect it.

```
alreadycollected = alreadycollected.mp3
```

The mandatory *alreadycollected* variable specifies the audio file used when the user presses the space bar to collect an item, but they have already collected it.

```
quit = quit.mp3
```

The mandatory *quit* variable specifies the audio file used when the Q key is pressed to quit the project. The program waits for the audio file to finish before quitting.

```
end
```

The mandatory *end* keyword ends the block. It is required at the end of all code blocks.

4.2.3 Graphicsblock

The Graphicsblock controls the display of a static image on screen throughout runtime.

```
graphicsname = graphics
```

The mandatory *graphicsname* variable specifies a name value for the Graphicsblock.

```
splashfile = gamelogo.jpg
```

The mandatory *splashfile* variable specifies the image file to display on the screen while running the program.

```
end
```

The mandatory *end* keyword ends the Graphicsblock.

4.2.4 Menublock

The Menublock configures the main menu. Players move through menu options, accompanied by background audio, by using the up and down arrow keys, each option—*start*, *continue*, and *cancel*—being announced via an auditory icon, with selections confirmed by pressing the spacebar. Background audio accompanies the menu, alongside an explanation of the menu's layout.

```
menublock = gamemenu
```

The mandatory *menublock* variable specifies the name value for the menu screen.

```
jingle = jingle.mp3
```

The mandatory *jingle* variable specifies the audio file to play before the menu screen.

```
nogame = nogame.mp3
```

The mandatory *nogame* variable specifies the audio file to play when *continue* is pressed, but a game has not been started to continue.

```
start = begin.mp3
```

The mandatory *start* variable specifies the audio file to play when the option is highlighted that starts a new game when selected.

```
continue = continue.mp3
```

The mandatory *continue* variable specifies the audio file to play when the option is highlighted that continues an already existing game when selected.

```
cancel = cancel.mp3
```

The mandatory *cancel* variable specifies the audio file to play when this option the highlighted that quits the game when selected.

```
click = click.mp3
```

The mandatory *click* variable specifies the audio file to play when browsing the menu options.

```
background = backgroundmusic.mp3
```

The mandatory *background* variable specifies the audio file to loop in the background of the menu screen.

```
end
```

The mandatory *end* keyword ends the Menublock.

4.2.5 Soundscapeblock

The Soundscapeblock allows the creation of spatial audio environments by mapping sounds to coordinates. Soundscapes may include a *cutscene*, and sounds relating to puzzles, fights, or other *soundscapes* to build a story.

```
soundscape = outside
```

The mandatory *soundscape* variable specifies the name value for the soundscape.

```
cutscene = story.mp3
```

The optional *cutscene* variable specifies the audio file to play before entry to a soundscape.

```
enter = cabindoor.mp3  vo = voout.mp3
```

The optional *enter* variable specifies the audio file to play on entry to the soundscape. This variable is helpful if the user wants to play a door opening sound, as it will play when the user enters and exits the same soundscape. The optional *vo* variable specifies a second audio file to play on entry to the soundscape. This variable is useful when the user wants to include dialogue or music, as it will only play on entry to the soundscape and not when exiting it.

```
background = rain.mp3
```

The mandatory *background* variable specifies the audio file to play on a loop in the background of a soundscape. The audio file plays at the same volume regardless of the player's position. This variable is useful if the user wants to play atmospheric audio or music in a particular soundscape.

```
sound = birds.mp3 (40,-40)
```

The optional *sound* variable specifies the audio file to play on a loop in the position set by the mandatory X and Y coordinates. The user can specifiy multiple *sound* variables, each on their own line.

```
sound = hello.mp3 (49, 40) loop = 0
```

This *sound* variable example has the same basic features as the previous example. However, this example includes a *loop* variable. The optional *loop* variable specifies how many times the audio file is repeated; otherwise, the sound will loop indefinitely.

```
sound = river.mp3 (49,49) exscape = riverscape
```

This *sound* variable includes an *exscape* variable. The optional *exscape* variable specifies a connection between a sound and a soundscape, enabling users to enter a new soundscape when the space bar is selected near the assigned audio file.

```
soundscapesize = 55
```

The mandatory *soundscapesize* variable specifies the size restriction of a soundscape. In this example, the user can place sounds exclusively inside a square that extents between -55 and 55 in both X and Y dimensions. This value is internally limited to a maximum of 200.

```
enterdoor = (0,0)
```

The mandatory *enterdoor* variable specifies the user's entry point in the soundscape.

```
nextsoundscape = cityscape
```

The mandatory *nextsoundscape* variable specifies the soundscape to follow after the player exits the current soundscape. Developers should specify "none" if there is no next soundscape.

```
end
```

The mandatory *end* keyword ends the Soundscapeblock.

4.2.6 Puzzleblock

The Puzzleblock creates an audio-based puzzle. This template is ideal for riddle or word-based puzzles, where the user hears a question and types a response. Each keystroke and backspace provides an auditory icon, and pressing enter submits the answer, prompting a *win* or *lose* auditory icon.

```
puzzlename = riddle
```

The mandatory *puzzlename* variable specifies the name value for the puzzle.

```
puzzlebackground = forest
```

The mandatory *puzzlebackground* variable specifies the audio file to play on a loop in the puzzle's background.

```
question = myriddle.mp3
```

The mandatory *question* variable specifies the audio file, which typically contains a question, riddle, or puzzle, to play when the puzzle begins.

```
answer = thisistheanswer
```

The mandatory *answer* variable specifies the characters, including letters, numbers, or punctuation, that form the puzzle's answer; spaces are not allowed. The user must input this answer to solve the puzzle.

```
enteranswer = enteranswer.mp3
```

The mandatory *enteranswer* variable specifies the audio file to play when the user presses return to submit their answer.

```
input = input.mp3
```

The mandatory *input* variable specifies the audio file to play on every key press; this is typically used for a key press or pencil sound effect, and short sounds are advisable to prevent overlay.

```
noinput = noinput.mp3
```

The mandatory *noinput* variable specifies the audio file to play when the user presses backspace to delete input, and there are no characters in the text box. Short sounds are advisable to prevent overlay.

```
backspace = backspace.mp3
```

The mandatory *backspace* variable specifies the audio file to play on every backspace key press to delete user input. This is typically used for a key press or eraser sound effect, and short sounds are advisable to prevent overlay.

```
correct = correct.mp3
```

The mandatory *correct* variable specifies the audio file to play if the user enters the correct answer.

```
incorrect = incorrect.mp3
```

The mandatory *incorrect* variable specifies the audio file to play if the user enters an incorrect answer.

```
end
```

The mandatory *end* keyword ends the Puzzleblock.

4.2.7 Fightblock

The Fightblock simulates combat using auditory icons to inform the player of their position relative to enemies. The user must react to the enemy's random movements (left, centre, or right). If the player hesitates, the enemy attacks, resulting in a loss of points. To counterattack, the player must press F at the correct moment when aligned with the enemy. The user can dodge by changing position; however, rapid movements cause fatigue, leading to delayed responses. In such cases, the player can defend using S to raise a shield, as their stamina limits offensive actions. The objective is to reduce the enemy's hit points to zero.

```
fightname = fight
```

The mandatory *fightname* variable specifies the name value for the fight.

```
background = cave.mp3
```

The mandatory *background* variable specifies the audio file to play on a loop in the background of a fight.

```
playerhit = stab.mp3
```

The mandatory *playerhit* variable specifies the audio file to play when the player stands in the same position as the enemy and successfully hits the enemy with the F key.

```
playermiss = swish.mp3
```

The mandatory *playermiss* variable specifies the audio file to play when the player stands in a different location from the enemy and fails to hit the enemy with the F key.

```
playershield = shield.mp3
```

The mandatory *playershield* variable specifies the audio file to play when the player stands in the same position as the enemy and successfully shields the enemy's strike with the S key.

```
playermove = step.mp3
```

The mandatory *playermove* variable specifies the audio file to play when the player moves left, centre, or right using the left arrow key, up arrow key, or right arrow key.

```
playertire = tired.mp3
```

The mandatory *playertire* variable specifies the audio file to play when the player has spammed the F or arrow keys too quickly, causing them to become tired and their movements to slow down. This encourages the player to shield from the enemy's strikes with the S key.

```
playerinjure = shock.mp3
```

The mandatory *playerinjure* variable specifies the audio file to play when the enemy successfully hits the player if the player has yet to hit the enemy or dodge within the time limit.

```
playerdefeate = run.mp3
```

The mandatory *playerdefeat* variable specifies the audio file to play when the enemy hits the player and depletes their hit points to 0.

```
enemyhit = punch.mp3
```

The mandatory *enemyhit* variable specifies the audio file to play when the enemy hits the player. This happens if player does not make an action before the time limit.

```
enemymove = stride.mp3
```

The mandatory *enemymove* variable specifies the audio file to play when the enemy randomly spawns on the left, centre, or right. This indicates the direction the player must strike to hit the enemy.

```
enemyinjure = shout.mp3
```

The mandatory *enemyinjure* variable specifies the audio file to play when the player is in the same location as the enemy and successfully hits the enemy with the F key within the time limit.

```
enemydefeat = collapse.mp3
```

The mandatory *enemydefeat* variable specifies the audio file to play when the player has hit the enemy and depleted their hit points to 0.

```
fightwin = winjingle.mp3
```

The mandatory *fightwin* variable specifies the audio file to play when the player has defeated the enemy.

```
fightlose = losejingle.mp3
```

The mandatory *fightlose* variable specifies the audio file to play when the enemy has defeated the player.

```
end
```

The mandatory *end* keyword ends the Fightblock.

4.3 DEVELOPING THE HODR ENGINE FOR CLI

I initially designed the Hodr Engine for Command-Line Interface, as I was confident a CLI would be a more accessible alternative to an IDE. I programmed the Hodr Engine in Visual Studio Code [40], but executed error checks and ran and packaged a test game application through the Terminal. At first, I found the Terminal environment unusual compared to GUIs and Terminal commands challenging to recall. I soon found it a more accessible alternative as a screen reader user than navigating complex or unfamiliar IDEs. My experience with the CLI's text-based structure reinforced my belief in its suitability for blind users: "In literature around the accessibility of developer tools and personal accounts of programmers with visual impairments, CLIs are often considered accessible alternatives to other visual developer tools" [47]. Some of the reasons behind this perception are as follows: "Both these qualities — a) being a text-based interface and b) being an interface that can be accessed entirely through a keyboard-make CLIs an attractive option in terms of accessibility for developers with visual impairments" [47].

With these considerations in mind, I used MKGame, an executable file programmed by Dr Adrian Jackson. This executable would allow users to run a CLI. Through this process, users code in HodrScript in a .txt file saved within a designated project folder. This folder contains an image file for the project's splash screen and a subfolder for the game's audio files. Users can package the application by executing a command, including the name of the .txt file containing the code, audio file folder, application name, and splash image file.

This is an example of a package command following this structure:

```
./code.txt sounds game image.jpg
```

4.4 PARTICIPANT DEVELOPS A GAME

After developing the first iteration of the Hodr Engine, Hodr Engine 0.1, a musician and sound designer, referred to by the pseudonym Herbert in this commentary to maintain his privacy, developed a game of his choice in HodrScript. Herbert's project, FinishedGameIE, with IE as an acronym for the Hodr Engine's original name, Immersion Engine, is a Christmas-themed game starting with a menu screen accompanied by Herbert's jazz arrangement of 'Have Yourself a Merry Little Christmas'. Instead of audio files announcing the options, three synths represent the *start*, *continue*, and *cancel* buttons. 'Deck the Halls' plays on entry to the soundscape, and users navigate a simple environment containing ghostly background ambience and a choir-like sound. Herbert's HodrScript can be located in this commentary's appendices (see Appendix D).

Herbert's experience was recorded using a survey, including hist experience when downloading and installing the engine, reading the tutorial to learn the HodrScript, and creating a project. The feedback was used to check BVI accessibility was ensured through out learning and using the Hodr Engine, in addition to gathering Herbert's requirements and preferences when developing and playing his game. The feedback survey, and Herbert's answers, can be located in the appendices (see Appendix C).

Herbert's feedback regarding HodrScript and the games it creates was broadly positive. The responses comment on his enjoyment of interactive soundscapes, and ease of writing in HodrScript. However, he expressed concerns about using the Terminal: "I found the use of Terminal: whilst interesting as a learning experience, overall exhausting" (Herbert's Survey). The participant highlighted challenges with the process of packaging the application, viewing it as complex: "The process to run the game was slightly convoluted, involving saving my text file and using Terminal commands to compile the game. This was confusing at times" (Herbert's Survey).

Herbert found VoiceOver had difficulties with smoothly announcing the Terminal's content: "The reliance on macOS Terminal was unintuitive and did not interact well with my screen reader" (Herbert's Survey). Some aspects of the engine were VoiceOver accessible outside the Terminal but there were barriers within the CLI:

> "The integration with VoiceOver was good in some areas. For example, editing a txt file and navigating my file hierarchy are tasks which rely on VoiceOver's natural strengths and its integration with macOS. Using Terminal was less easy. VoiceOver, unless specifically altered, does not cope well with correctly and reliably reading text which involves non-linguistic elements, which made it extremely difficult to understand what it was saying. Similarly, the quantity of info output by Terminal in the case of an error was often overwhelming" (Herbert's Survey).

Such feedback regarding accessibility within the Terminal prompted me to consider Herbert's suggestion of implementing a dedicated Hodr Engine text editor:

> "I would like to have seen a dedicated text editor with easy ways to run and save the project. I would like to have been able to program the movement of objects, and have control over the global volume of sound files" (Herbert's Survey).

As a result, I built a dedicated IDE containing a text editor named Hodr Engine 0.2, following on from Hodr Engine's CLI version.

4.5 HODR ENGINE 0.2: DEVELOPING AN IDE

The following section will provide a description of the Hodr Engine IDE, highlighting all accessibility features, including shortcut keys, labelled buttons, alt text, auditory icons, and visual settings, to demonstrate how BVI accessibility-focused research, as discussed in survey feedback and the literature review, has been applied.

The Hodr Engine IDE facilitates programming in HodrScript and running code. The IDE also includes a function to package the script, audio files and an image file into an application. I drew inspiration from Herbert's recommendation for a dedicated text editor containing built-in saving and packaging functionalities. In conjunction, research by Lisboa et al, referenced in the literature review, suggests, that to make a screen reader accessible platform, developers must insure "all interactive elements are properly labelled and navigable via keyboard" [35]. These comments complement Frey's research, also discussed in the literature review, which explore common screen reader accessibility barriers, including "images without alternative text" [25].

In response to these findings, the Hodr Engine is meticulously designed for screen reader accessibility. Functionalities, including Herbert's suggestion for saving, running, and packaging, are integrated in the IDE. Every button is accompanied by a shortcut key, which is referenced within the button's label, so users do not need to research the shortcut connected to each button. The text editor also facilitates navigation through shortcut keys. The engine employs minimal imagery to maintain a text-focused interface. Even so, in the IDE's graphical banner, an alt text description is included.

The IDE's main window is visually and functionally basic, suitable for novice and experienced BVI and sighted developers. At the top of the window is the title banner, HodrEngine alongside the following descriptive alt text:

> "The banner features a bird's-eye view of a cityscape illuminated by glowing windows and musical notes. A wide river runs through the city, reflecting the buildings and deep blue sky, with a golden stave featuring a morphing sound wavelength forming the text HodrEngine."

The majority of the interface is occupied by the text editor, where HodrScript is written and edited. By default, the editor has a dark blue background with white text, set at a font size of 12 pixels. Default white text on a dark background was chosen because "this can create more contrast between the content and the background, limiting eyestrain and improving content readability." [11] However, dark modes, for many eye conditions, may hinder sight navigation and readability: "As white text on a black background is more difficult to read than black text on white, the perceptual fluency is lower, causing more negative judgments of the content" [36]. To accommodate these nuances, visual settings, including text size, can be customised to suit the user's preferences. Below the editor is a vertical stack of buttons for file operations, customisation, and navigation. Each button opens a dialog box when selected, and users can access these buttons through clicking them or using shortcut keys, as influenced by the research of Lisboa et al [35].

As the RNIB Accessible Gaming Research Report outlines, auditory icons, also referenced in the report as audio triggers, are "most desired by blind and partially sighted gamers" [46]. As explored in the literature review, the Vale [21] and Blind Quest [6] successfully implemented auditory icons to alert users to items or accompany actions. It is hoped that a similar approach would be as effective within an IDE, so auditory icons are incorporated in the IDE to provide feedback when selecting items and performing actions.

I took inspiration for the auditory icons used in the Hodr Engine from the quick, synthesised clicks and swipes of VoiceOver. Historical early computers also produce clicks and clunks that may achieve a similar effect to VoiceOver, whilst remaining distinctive. The IBM Model 029 Card Punch [30] and Teletype Model 33 [53] emitted mechanical percussive sounds that may be suitable for the IDE's auditory icons. I did not have access to these machines, so I recorded a Perkins mechanical braille typewriter. Its tactile and auditory feedback resembles similar devices. The Perkins Brailler's sounds created a foundation for the auditory icons. These recordings were edited, with panning and reverberation applied within Logic Pro [4].

File operations allow users to manage project files and folders. Developers browse for a HodrScript .txt file by selecting the *Browse* button or pressing Command+B, which opens a file dialog to load an existing script into the editor. Users can create a new document by selecting the *New Document* button or pressing Command+N, accompanied by an auditory icon button press. If there are unsaved changes in the current project, the IDE will ask the user to save before proceeding. Saving progress is completed using the *Save* button or Command+S. If the project has not been saved previously, the IDE will open a *Save As* dialog box to specify the file name and location, followed by an auditory icon click. For users who wish to save a project under a new name or location, the *Save As* option is accessible through the file menu or by pressing Command+Shift+S. The ability to load files, create new files and save is important as it was a concern of Herbert in the feedback.

Users run a project by selecting the *Run* button or pressing Command+R, which opens a *Run Game* dialog box. This dialog prompts users to specify the project directory, name, and resource directory containing their audio files and image file. Once confirmed, the project is executed in an external window, accompanied by an auditory icon consisting of a braille carriage moving back to a starting point. The process of packaging a project has a similar approach to *Run*. By selecting the *Package* button or pressing Command+P, users open the *Package Game* dialog box. This dialog requires inputs for the project directory, name, and resources directory. Once the details are confirmed, the IDE packages the project into a standalone application saved in the specified directory, accompanied by the *margin bell* auditory icon which alerts the writer to the end of a line of braille.

These new *Package* and *Run* functions were influenced by Herbert's difficulty with packaging the application, and my concern that users could not test a project before packaging in the CLI version.

To improve text navigation for BVI users, and to prevent the need to slowly traverse text line-by-line, word-by-word, or character-by-character, shortcut keys for navigation have been included in the text editor section of the IDE. These shortcut keys are efficient during the search for punctuation. As previously referenced in Ragas' research on silent special characters [43], some screen readers may not announce punctuation when announcing lines, paragraphs, or extensive blocks of text, but do vocalise punctuation during character-by-character navigation. Furthermore, navigational shortcut keys are useful for locating words or phrases, especially when the same phrase appears multiple times, as it allows users to quickly identify all instances and their locations. A shortcut to locate a silent character, as well as single or multiple words or phases, would speed up such tasks.

The Hodr Engine text editor contains several shortcut keys to locate code. For instance, developers can press Command+Shift+W, which opens a *Find Word* dialog box using which they may find words or phrases in their code. This feature presents all instances of the searched term and presents the results in a *Find Results* dialog box, accompanied by a *button press* auditory icon. Users may jump to a line by pressing Command+Shift+N, which opens a dialog box for entering the desired line number. Once entered, the cursor moves directly to the corresponding line in the editor, accompanied by a *button press* auditory icon. To identify their current position, users can press Command+Shift+L to display a dialog box showing the current line number.

The keyboard shortcut Command+Shift+F moves to the start of a line and Command+Shift+E to the end. Command+Shift+T jumps to the top of the document, and Command+Shift+B jumps to the bottom. The auditory icon for moving to the start of a line is the sound of a braille carriage returning to the beginning of a line, with the sound panned to the left. Conversely, the auditory icon for moving to the end of a line replicates that of the braille carriage moving to the right margin. The auditory icon for moving to the top of a page is derived from the Brailler's button press for moving paper to a new line. The same sound is used for moving to the bottom of a page with added low bass note. Shortcut keys within the text editor include the Shift key, whereas most shortcuts outside the text field do not, except for Command+V, which opens the visual and audio settings window. This exception avoids conflicting with the commonly used paste command. Whilst I considered using recognised shortcuts, such as Command+F to search, as there multiple search types within the IDE, such as finding by line or word I decided to adopt shortcuts corresponding to the search type, for example, N for finding by number, and W for finding by word.

The IDE provides customisation options for font colours, background colours, and font sizes through the *Audio and Visual Settings* dialog, accessible via the corresponding button or Command+Shift+V. When selecting a font or background colour, the chosen option is highlighted with a red border. Auditory icons can be toggled on or off using the *sound* toggle button within the settings. Selecting a setting is accompanied by the auditory icon of a paper clamp holding braille paper in place, and the auditory icon of a button press plays when selections are confirmed.

A built-in tutorial present guidelines, features and functionalities. This tutorial can be accessed via the *Tutorial* button or by pressing Command+T. The text includes descriptions of the interface layout, the banner, and all shortcut keys. Additional resources and support are available through the Hodr Engine website and help desk email.

4.6 FIRSTHAND EXPERIENCE DEVELOPING A PROGRAM

I experienced using the Hodr Engine IDE to create a spatial audio experience in order to simulate a live concert performed for the University of Huddersfield's Centre of Research for New Music (CeReNeM) Journal [12].

In the CeReNeM journal performance, I participated in the Audio Scores project. During the Audio Scores performance, composer and cellist Thomas Hawkins, web developer Charlotte Roe, and I collaborated. Charlotte's role was to issue commands via multiple text-to-speech (TTS) voices, using two speakers (one for each instrumentalist), and Thomas and I responded with our instruments.

I developed a simulation of the concert in which the user can experience the performance as a listener, navigating the stage. During the performance, microphones recorded each participant, and the resulting stem files were imported within the Hodr Engine. I plotted the stems' coordinates to replicate Phipps Hall in the Richard Steinitz Building. I encountered challenges when creating the virtual environment. Stem files were recorded simultaneously, so microphones captured bleed from different sources, causing overlapping. This was pronounced in the Audio Scores performance, where the TTS voices, louder than Tom and my contributions, were clearly duplicated across multiple stem files, making navigating difficult.

The current Hodr Engine version lacks built-in reverberation and occlusion effects, meaning such effects must be pre-applied to the stem files. Stem files were edited in Logic Pro to replicate Phipps Hall's reverberation, as well as binaural placement for the height of sounds, such as performers sat on the floor in contrast with the taller speakers. Distinguishing whether a sound was positioned in front or behind the listener was impossible due to the absence of occlusion.

The HodrScript code for the CeReNeM Journal concert can be located in the appendices (see Appendix E).

— 5 —
Discussion

Following the methodology and development of the Hodr Engine, this commentary analyses Herbert's feedback on using Hodr Engine 0.1 via the Terminal to create a game, and my reflections on utilising Hodr Engine 0.2 in its IDE version for game development. The discussion identifies the successes and shortcomings of the Hodr Engine and evaluates potential inclusions suggested by Herbert's feedback and the initial survey. Finally, this commentary proposes the next steps for advancing the Hodr Engine.

5.1 FEEDBACK AFTER TESTING THE HODR ENGINE

One of the research aims involves comparing the effectiveness of screen readers in navigating command-line interfaces to that of graphical user interfaces. This query was designed to determine the most suitable option for a BVI-focused game engine. Herbert's response to using the Terminal showed that the CLI was less accessible than this commentary hypothesised. Herbert's challenges in accessing the Terminal CLI hindered his ability to package an application successfully: "The terminal was difficult to use because typing commands that aren't English sentences are difficult to keep track of with a screen reader." This aligns with Ragas' research [43], which discusses various special characters that remain inaccessible to some screen readers. This feedback counters my argument for supporting CLI screen reader accessibility. Contrary to my hypothesis that a CLI's text-based nature would make them inclusive to BVI users, the comment raised concerns regarding the need for users to input syntax-heavy commands when packaging an application in the Terminal. Such commands include characters like / and ∼ (slash and tilde), which a screen reader may not announce. I experienced this issue as a developer when drafting the tutorial in Microsoft Word. While documenting the command to package the application, I inadvertently omitted the . (full stop/period) at the beginning of the line. The . (full stop/period) in Terminal denotes the current directory and is essential. Still, I did not notice its absence in the text because VoiceOver did not announce punctuation when reading the line aloud. It does identify punctuation if the arrow keys are used to navigate character-by-character, which is time-consuming when editing a large document. Consequently, Herbert entered the incorrect command until he informed me of the erroneous instruction. Herbert referenced this problem in his feedback: "The installation instructions in and of themselves were clear and easily followed, but some of the commands did not function as expected" (Herbert's Survey). Herbert's findings validate the accessibility limitations of a CLI's syntax-heavy commands and highlight the potential advantages of IDEs, which often use alternatives to syntax-heavy instructions. Instead of inputting a command in a CLI, the IDE uses a directory picker to locate the folder containing the game code and resource folder, which houses the audio and image files for the game. As noted by Herbert: "editing a txt file and navigating my file hierarchy are tasks which rely on VoiceOver's natural strengths" (Herbert's Survey).

Another issue identified by Herbert involved the volume of Terminal output when packaging: "Terminal outputs a lot of information in the form of computer jargon, which is extremely difficult to parse with VoiceOver" (Herbert's Survey). The output generated by PyInstaller, a tool used for packaging applications, is announced by VoiceOver as a complete block of text, rather than lines or paragraphs, resulting in a continuous stream of information. This understandably can be perceived as "overwhelming", especially as the CLI environment is not common used on GUI-centric platforms, making the environment unfamiliar to first time users, such as Herbert.

Herbert's lack of experience with a CLI may connect to one of the research questions, *What are your experiences using software with adaptive tools in educational institutes and daily life?* This question was asked to build an understanding of the current state of assistive aid education and availability. The limited experience with CLIs compared to IDEs among BVI students may stem from an underrepresentation of the CLI in prior educational curricula. When survey participants were asked, "Do you believe your previous formal education gave you a knowledge of using technology such as computers with accessible adaptations?" they supplied mixed results:

- "No, I was only ever taught how to use a computer with the Jaws screen reader, but I needed a Mac for university, and no one at school taught me how to use VoiceOver."
- "At secondary school, no, but at the Royal National College for the Blind, I did learn a lot and had IT lessons."
- "Yes, as they taught me how to touch type and use NVDA and GRID 3."
- "No."
- "I learnt a lot of my technology skills, specifically computer skills, throughout high school. Any others, such as phone and iPad, I picked up and learnt by myself as I already had the basic foundation down."
- "No."
- "Yes, I had sessions to learn how to navigate technology using assistive software. I have used a variety of screen readers, including Supernova and Jaws."

(Participant Survey)

The participants of this survey may not completely represent the experiences of the BVI demographic, given that the majority of this small group falls within the 18 to 30 age range, with only two participants exceeding this bracket:

- 6 - Between 18 and 30,
- 1 - Between 41 and 50,
- 1 - Between 51 and 60.
- 0 - Between 61 and 70,
- 0 - 71 or above.

(Participant Survey)

With this in mind, the mixed feedback partially supports Hewett's Longitudinal Transitions Study [28] regarding the challenges that some young BVI participants encountered in learning and using technology within their formal education and the negative impact on their computer literacy. Even with strong computer literacy, using a CLI with a screen reader for the first time may uncover challenges, especially when testing new software. BVI users do not have immediate access to the content on the screen in the same way as sighted users. Instead, they must explore the interface using their screen reader, which can make an unfamiliar environment, such as a CLI, initially challenging to navigate. A screen reader can be likened to a magnifier highlighting and describing one section of a room at a time, requiring deliberate exploration to build a full picture. In contrast, vision instantaneously provides access to the entire room's displayed contents in the views focus. This difference is essential for users accustomed to recognising patterns in other interfaces, often GUIs, meaning Herbert struggled to navigate between coding in Apple's text editor and Terminal's "confusing" command lines: "The process to run the game was slightly convoluted, involving saving my text file and using Terminal commands to compile the game. This was confusing at times" (Herbert's Survey). The excessive output and Herbert's preference for a "dedicated text editor" suggest an IDE is the more BVI-accessible option for the Hodr Engine. The IDE's *Run* and *Package* buttons address errors by displaying them in a dialogue box. The engine limits output to the last three lines, which usually contain the outlined error and its location within the code, preventing information overload.

Herbert also raised concerns about an issue causing his HodrScript not to package. This problem was due to errors in the engine's game utilities. These issues arose because the code was adapted from a game-specific program to code suited for a game engine. The original code included features explicit to the Immersion Sound Studio, such as the ability to collect sounds for a personalised soundscape at the end of the game. Removing these features would disrupt the program and require a substantial rewrite. For example, the *collect* variable in HodrScript, which allowed items to be gathered for the soundscape, had to remain in the engine code. Without it, the program would not function. Herbert had to include additional lines of code paired with short silent audio files to ensure compatibility, even though these additions did not affect his game. A similar feature can be considered for future iterations of the Hodr Engine; however, it appeared too specific for the foundations of a game engine. Moreover, as participants expressed interest in other game genres, such as puzzles or combat, dedicating time to rewrite the collect code seemed an unwise use of development resources when time could be spent on creating the puzzle and fight features. When asked about such problems, Herbert remarks: "Creating an app went mostly smoothly, and the engine did perform as expected. Myself and the developer encountered one issue wherein the engine required a number of elements to be present in the code which were not listed in the instructions list of mandatory elements. I needed to add a number of new sound files for the game to function" (Herbert's Survey).

Other than the need to include unwanted variables like the *collect* variable, programming in HodrScript within Apple's text editor was effective for Herbert. "the integration with VoiceOver was good in some areas. For example, editing a txt file." Moreover, owing to the language's basic structure and the absence of complex syntax, "The HodrScript was very easy to use. Much of the punctuation and syntax rules which I have encountered in other programming languages (i.e. Python, C++, HTML) were not needed to write my game code. This meant little battling with the screen reader, and the process was efficient and not stressful" (Herbert's Survey). The next iteration of the Hodr Engine, version 0.3, will be rewritten from scratch, addressing the issues caused by adapting code originally written for other purposes. Still, the language, without the barrier of complex syntax, reveals itself as a screen reader-friendly vehicle for creation.

To sum up, the Terminal offers a powerful interface, and its text-based nature makes it inherently accessible to screen readers. However, the CLI's unfamiliarity may be better suited to experienced programmers accustomed to the nuances of such environments—whether that involves anticipating instances where their screen reader may fail to announce a special character or knowing how to navigate the Terminal's output to locate necessary information efficiently. I acknowledge that Hodr Engine version 0.1 for CLI is an individual instance of the CLI's use, tested with software still in development. As such, issues related to the software's shortcomings, such as the engine's reliance on unnecessary variables, may have influenced the CLI experience. This is why there are CLI and IDE versions of the Hodr Engine. Implementing a CLI version accommodates BVI users who prefer that environment.

5.2 REVIEWING THE CURRENT STEREO AUDIO ENGINE

Herbert's feedback on the Hodr Engine focused more on his experience using the engine than the game he produced. However, his comments regarding the games created with the engine were positive: "I love the idea of being able to create immersive interactive experiences, and I liked the mobility the engine offers in being able to move between various sonic landscapes" (Herbert's Survey). When asked how games produced by the Hodr Engine could be improved, Herbert reflected: "I would like to have been able to program the movement of objects and have control over the global volume of sound files" (Herbert's Survey). In response to Herbert's feedback on volume control, where developers previously needed to adjust volume in a separate audio editor, the updated Hodr Engine now includes functionality to alter sound volumes between 0.0 and 1.0 for sounds placed in a soundscape. Whilst this feature does not extend to variables outside the *sound* variable, such as for background ambience, it is planned to support this in future updates. By default, all sounds within the Hodr Engine play at 1.0, full volume.

Due to research time constraints, the Hodr Engine does not yet support sounds that dynamically move across a soundscape. However, this effect can currently be approximated using stereo files with pre-programmed panning. For instance, within Immersion Sound Studio, I implemented the sound of a train that appeared to move past the listener by panning from right to left. While effective to a degree, this method has limitations. The sound remains fixed relative to the listener's position, meaning that if the user turns around, the sound pivots with them. Consequently, the train does not "run over" the player if they stand in front of it; instead, it continues to pan from right to left, irrespective of the listener's orientation. As Herbert's and my findings demonstrate, auditory effects should be included within the engine rather than relying on the pre-application of effects via a DAW. I also discovered this issue when simulating the CeReNeM Journal concert using live stem files. Without binaural effects [12], the engine's stereo soundscape limitations challenged my perception of spatial orientation. It was difficult to discern whether a sound source was positioned behind or in front of me. When asked "What is your level of expertise in audio production to create music and sound design for a game?", respondents said:

- "Low level."
- "Very little."
- "None."
- "I don't have any."
- "None."
- "I am very confident with my audio editing and sound design skills as I have been doing this for over 5 years now, some of which has even been for paid work."
- "Literally no experience at all."

(Participant Survey)

Whilst DAWs like Logic Pro [4], Reaper [24], and Audacity [37] are widely used by BVI users, as demonstrated in the response "I am very confident with my audio editing and sound design skills", the overall feedback demonstrates not all users, whether BVI or sighted, possess the skills needed to create music recordings and perform sound design. These responses suggest the necessity of including extensive in-engine sound and music libraries in future iterations of the Hodr Engine. Additionally, incorporating text-to-speech functionality to narrate text would allow developers without access to voice actors or DAWs to produce content directly within the engine.

Future iterations of the Hodr Engine must incorporate options for including effects, such as occlusion, often heard when items are behind a wall, reverberation and echo, which vary with the size and material of the environment. Applying Head-Related Transfer Functions (HRTF) may also enhance the engine's current first-person point of view. This technique simulates how sound is altered by the listener's ears and head, resulting in a binaural presentation. Binaural localisation, unlike stereo audio, clearly differentiates whether a sound source is behind, above, or in front of the listener.

Extra effects, such as the Doppler effect, may also be implemented. The Doppler effect simulates changes in pitch and velocity as sound passes the listener. For example, in the rising and descending pitch of an ambulance siren as it rushes past the observer. Spatial navigation aids, similar to those used in Blind Quest [6] and Sounds of Eden [14], may also address the challenge of determining whether a sound is in front of or behind the listener. Cardinal directions are announced in these games before or during movement to clarify the listener's orientation.

5.3 IMPLEMENTING SURVEY PARTICIPANT'S PREFERRED GAME GENRES

The original survey feedback, which examines interest in audio game creation and play, provides insights into participants' preferred game genres and accessible features. This feedback answered the research question, *What are the common genres and techniques of audio games?* The project incorporated preferences such as puzzles, combat, and open-world gameplay. I will now evaluate the effectiveness of the implementation of these features within the engine, alongside genres and play styles proposed for inclusion in Hodr Engine 0.3.

5.3.1 Open-World Feature

Participants were asked about their favourite game genres, with puzzles, combat, and open-world as the most popular responses. The engine's open-world capabilities were well received by Herbert: "I love the idea of being able to create immersive interactive experiences, and I liked the mobility the engine offers in being able to move between various sonic landscapes" (Participant Survey). This comment refers to the Hodr Engine's capability to connect multiple soundscapes, creating an expansive environment like the soundscapes of several rooms linked together to create a house in a wider landscape of buildings. However, beyond navigating these soundscapes and collecting sounds for later playback, the engine still needs to allow further interaction with objects, such as examining them, toggling items like radios on and off, or collecting items for later use, such as keys. This limitation is apparent in Herbert's comment: "I would love to be able to design games with a more dynamic environment" (Herbert's Survey). This sentiment is amplified by survey respondents who wish to develop and play games offering greater variety of activities, interactions, and mini-games. This includes "racing, platformer, adventure, and party games" (Participant Survey). The engine does not yet support these genres within its open-world environments, all of which have been successfully developed as audio games previously. For instance, Blind Drive [23] is "an audio-based black comedy action arcade game that you can play with your eyes closed. It offers a hyper-immersive first-person audio adventure and fast-paced arcade gameplay, all driven by an insane B-movie plot" [23], exemplifying a unique take on racing games. Though not a typical racing game like the Mario Kart series, it challenges players to avoid collisions by responding to auditory icons such as car and street noise. Similarly, Super Liam [33] employs an arcade-style approach where players complete levels, locate crystals, and avoid obstacles while defeating opponents. Directional audio games like these, where users advance while avoiding or interacting with objects, would be achievable using the Hodr Engine given suitable modifications. For instance, the player could constantly move forward with sounds approaching from the left, centre or right, and the user would need to avoid them with an arrow key in a different direction to avoid collision.

In contrast, the story mode options suggested by participants are significantly more ambitious than the engine's current capabilities. One participant stated, "I love playing adventure games, specifically story-based games. I think ones that have multiple ways of playing it are my favourites as they allow me to go back through the story and choose different options, resulting in different outcomes" (Participant Survey). The engine's current item collection feature, originally designed for the Immersion Sound Studio, could be adapted to enable story branching. For example, items collected, puzzles solved, or combat encounters could influence narrative outcomes. These adapted features could be incorporated into the HodrScript code as new code blocks, such as one specifying the functionality of an item, whether it provides health points, unlocks a new location, or leads to a new outcome.

There was also interest in incorporating text adventures, such as AlterAeon [2], which allows users to interact, explore, and complete tasks via text-based stories announced with Text to Speech (TTS) and is controlled through text commands. One participant expressed interest in "text adventure like AlterAeon, better with sound effects" (Participant Survey). This emphasises the importance of including the ability to create games that integrate text alongside audio. Such features could support refreshable braille displays, making games accessible to deaf-blind players, as in Nano Empire [49]. In this text-based mobile game, players invest resources into city construction, defence, and space warfare. Similarly, AlterAeon enables users to form groups and create quests. The text adventure format, well-suited for refreshable braille displays, could be integrated into the Hodr Engine. HodrScript could allow variables to specify text as well as audio files.

5.3.2 Puzzle Feature

The puzzle feature currently implemented in the engine allows for audio-based questions or riddles, with text responses input via the keyboard. While this forms a foundation for text-based puzzles, it lacks the iterative guessing and feedback mechanisms of games like Wordle [54]. In Wordle, users receive hints through colour-coded tiles indicating correct letters and placements. To improve this, the engine could include hints and multiple attempts to align with survey participants' preferences. Quizzes might also be integrated, using the audio game's menu screen to offer multiple-choice answers, as participant feedback indicates: "I enjoy card games and quiz games but also interactive player games where you complete missions or tasks, etc" (Participant Survey). This could be achieved through quizzes that, when completed, unlock rewards or new locations.

Another participant expressed interest in "Brain exercising games, strategic games" (Participant Survey), expanding mini-game potential. Brain-training audio games such as Audio Game Hub [8] offer mini-games like Animal Farm, where users match animal sounds by memorising the placement of boxes containing animals on screen and locating the same types of animals, one after the other, to match them. Another example is Super Simon, a memory game that requires recalling sound sequences, and the location of those sounds on the screen. Reaction-based games on Audio Game Hub, such as Samurai Tournament, which tests players' speed and endurance through fast, constant taps on any part of the screen within a time limit, demonstrate formats that could also inspire new features for the Hodr Engine. Whilst these mini-games are not currently supported by the engine's puzzle feature, mechanisms for matching sounds, reaction-based gameplay, and endurance challenges could be included in Hodr Engine 0.3.

5.3.3 Fight Feature

During analysis of survey responses, it was evident combat was a popular game genre, with comments such as "Open world/combat" and "First person shoot them up" (Participant Survey) appearing as games users enjoyed and would be interested in creating. This feedback inspired the design of the combat system in the Hodr Engine, drawing upon directional audio techniques used in The Vale [21], albeit in a simplified form. In The Vale, combat mechanics restrict movement to three positions relative to the enemy: left, centre, or right. Whilst this simplifies gameplay, it reduces the challenge as the player has only three options: to evade an attack or align themselves to strike or shield. The Hodr Engine builds on this concept by introducing variables such as configurable hit points for enemy and player, and mechanics like enemy speed and player fatigue when keys are excessively spammed. The engine's fight function might be refined further with ideas from combat-heavy audio games such as Blind Gladiator [7]. In Blind Gladiator, a player assumes the role of a gladiator, touring arenas to defeat opponents and win his freedom. Like The Vale, this game relies on directional audio, requiring players to move in the direction of the enemy in order to attack while dodging incoming strikes. However, as Blind Gladiator is designed for mobile and tablet, the player moves by tilting the device, allowing for greater precision in combat. Similarly, the controller version of The Vale achieves precision through the use of the left analogue stick to position the shield and the right analogue stick to aim the sword at enemies. However, the keyboard version of The Vale aligns with the Hodr Engine's current functionality. This suggests that computer games operated via keyboard and created with the Hodr Engine may benefit from unrestricted movement rather than the limitations of left, centre, and right positioning only. Greater flexibility enables opportunities, including chasing and evasive manoeuvres.

5.3.4 Proposed Progression Systems

When discussing the implementation of progression within the engine's puzzle and combat systems, the Vale and Blind Gladiator are good examples as they contain progression systems allowing players to acquire new weapons, armour, and abilities. Currently, the Hodr Engine does not have a progression system in place. The Vale's market soundscapes, where a player can buy goods, are operated by the player navigating towards vendors by listening to announcements. In Blind Gladiator, players browse shop inventories by swiping through options and selecting items through taps. The Hodr engine may be elevated by implementing similar systems. This means players might upgrade attributes like speed, hit points, and weaponry throughout the game instead of the static hit points currently defined in each fight code block.

5.4 SURVEY PARTICIPANT'S PREFERRED ACCESSIBILITY FEATURES

5.4.1 Shortcut Keys, Touch Screen and Trackpad Features

When asked about accessible mechanics that could be included in an audio game, feedback suggested using a trackpad, shortcut keys and touch screen Shortcut keys were implemented in the first iteration of the Hodr Engine. One survey participant stated, "I would create a game that would be available across different platforms. Therefore, it would have to be available for a variety of inputs. For example, arrow keys on a computer or touch screen gestures on a phone and tablet. Perhaps on a laptop it could have a cross between using the arrow keys and the trackpad.

I've seen this in a video game I've played called As Dusk Falls, and it makes the game more interactive" (Participant Survey). A trackpad could be implemented as a supplementary input method to the engine's current keyboard-focused abilities. For example, the player might navigate using the arrow keys while simultaneously using trackpad gestures, such as tapping or swiping, to execute actions like striking, dodging, or ducking. The trackpad could also be utilised as the primary navigation tool, offering more fluid movement than the discrete input of pressing or holding down arrow keys.

5.4.2 Proposed Voice Activation Feature

Besides shortcut keys on desktop and touch screen swipes and taps on mobile platforms, when asked, "If you created a game, what input methods would you prefer for navigating the game?" two respondents mentioned voice activation as an accessible feature: "I would want to create a game that people are able to navigate on their phones as that is what I find most accessible. I think touchscreen or voice input would be a good way to do this" (Participant Survey). The audio game Codename Cygnus [45] features a branching narrative, placing the player as a spy interacting with environments by selecting responses. For example, the player might choose a charismatic or aggressive approach when retrieving information. These choices are made via screen taps or voice activation. While the gameplay is somewhat limited, resembling an interactive radio drama where players influence the outcome, the design demonstrates how voice commands are effective, allowing for the inclusion of players with limited mobility. Commands such as "open door," "walk forward," or "select start game" could be implemented in spatial audio environments.

5.4.3 Platforms

This exploration into devices beyond desktop platforms leads to the discussion of feedback regarding platforms. The survey results indicate prominent interest in accessible features for mobile gaming, in addition to computer-based experiences. Six participants suggested mobile as the device used for gaming, two for tablet, three for computer, and one for game console. Mobile is the clear leader, followed by computer, meaning mobile should be the next device focused on for game creation and play. Many suggested touch screen-focused design when asked for suggestions for optimising a game's BVI inclusivity:

- “A screen reader to announce text and navigation by moving a finger on the mobile screen.”
- “A mix between touch screen and computer keys. Computer keys give me more control, and touch screen can allow me to explore the screen.”
- “Touch screen and controls.”
- “I would want to create a game that people are able to navigate on their phones as that is what I find most accessible. I think touchscreen or voice input would be a good way to do this.”

(Participant Survey)

When asked about the accessible features survey respondents would include in their games, one of the participants discussed gestures used for navigation on mobile platforms:

> “On a phone and tablet, gestures would include swiping left, right, up and down, double tapping to interact with elements, and perhaps scrubbing or swiping with multiple fingers to conduct an action.”
> (Participant Survey).

Mobile and tablet games could adopt an approach similar to Sounds of Eden [14], pressing a finger on the left, right, top and bottom of a screen to navigate a space, as well as “double tapping to interact with elements” like selecting menu options, opening doors to new environments, and “scrubbing or swiping with multiple fingers to conduct an action” like browsing menu options or striking an enemy. The current keyboard functionality for the computer version has so far performed effectively without receiving criticism from Herbert. However, other control mechanism should be considered for Hodr Engine 0.3.

5.5 DEVELOPING VISUALS WITH THE HODR ENGINE

With the updates proposed above, the Hodr Engine is expected to possess the capabilities required to develop nearly all the games identified by participants as those they wish to create, including:

- “Combat, good sound effects, humour.”
- “Sports, adventure, racing.”

- “Sound-based games.”
- “A fantasy adventure game.”
- “Action-adventure.”
- “An interactive game.”

(Participant Survey)

However, one participant’s survey feedback suggests a proposed game concept that may not be feasible to develop with the suggested engine modifications. This is because the design relies on multiplayer functionality and possibly visual elements.

> “I would want an interactive experience creating an environment for artists to explore and contribute to each other’s works.”
> (Participant Survey).

A game where users experience and contribute to each other’s artistic creations necessitates a multiplayer framework akin to AlterAeon, where participants interact and send requests across the game’s landscape. A comparable concept may involve users sharing work within a virtual gallery space. Musicians or sound artists could design immersive environments to be visited by other players. Developers could also collaborate on projects in the engine through features resembling GitHub [27], a platform for shared code management. From the participant’s ambiguous term, “artists”, it is unclear whether they are referring to audio or visual artistry. While the Hodr Engine is designed to support sound-based artistry, incorporating a visual element would present a new direction for the engine’s development. When surveyed on interest in creating audio and/or visual games, participants gave mixed results:

- “Only audio games.”
- “I would be interested in an audio game.”
- “A combination of both.”
- “A combination of both would be good. It would prepare me for when I lose more reading sight and need to rely on sound or touch.”
- “Audio only. Maybe text too.”
- “I would prefer games that have both audio and visual elements. Although I cannot see, audiovisual games would be great to play with my sighted friends and family. Most sighted people don’t seem to want to play audio-only games whereas something that includes both audio and video is more immersive.”

- “Definitely a combination of both, mainly leaning towards audio games, but I think visual games having an aspect are also really helpful for people who do have some sight.”

(Participant Survey)

These findings indicate four out of seven responses interested in combining audio and visual elements to create games that accommodate both sighted and BVI users. The feedback, “I think visual games having an aspect are also really helpful for people who do have some sight” (Participant Survey), is a valuable observation, especially as the survey participants reported a mix of vision impairments from “total blindness” to “minimal corrected with glasses, and may want to use their residual vision, or join in with sighted players in the knowledge that their game is accessible with or without visuals.

Half of the respondents reported using their vision with adaptions like magnifiers, glasses, and zooming sofware to enlarge items on the screen, in addition to or in place of tactile and auditory adaptions like braille displays or screen readers. With visual users in mind, graphics may be incorporated in the Hodr Engine as an optional enhancement accompanying spatial audio. This ensures visually-rich games remain entirely operable through audio and for exclusively auditory users. For instance, The Vale includes graphics, such as the flickering flames during a scene where the player navigates through fire. This imagery is not essential for gameplay but is available for players with some residual vision. In its simplest form, animations such as flickering flames could be displayed on a loop during specific soundscapes or activities. This would resemble the engine’s existing graphical feature, which presents a single static image during gameplay.

God of War Ragnarök [48] integrates visuals with adaptive features for VI players. Options include high-contrast modes like outlines around characters and modifiable colours, subtitles, navigation assistance, and controller remapping. 3D models could be imported into the Hodr Engine, similar to audio files, using coordinates to position them on a map. Visual adaptations, like those used in God of War Ragnarök, could be integrated into all games featuring graphics. However, methods for enabling BVI users to animate these models are still to be explored.

5.6 CLOSING THOUGHTS

This commentary's discussion section will conclude with a quote from Herbert, who responded to the request to note any additional comments or feedback. His statement exemplifies the need to continue to develop the Hodr Engine and the necessity of progressing BVI accessibility in software development:

> "Thank you so much for creating this kind of thing. It's not a novelty or gimmick, it fills a viable niche which will bring value to people. I have always wanted to be able to make games, but find the inaccessibility in current game dev tools and culture to be overwhelming and frustrating. This gives me hope that one day I might be able to do what I've always wanted." (Herbert's Survey)

— 6 —
Conclusion

This research focused on developing a screen reader-accessible audio game engine for BVI users. A survey was conducted to collect BVI users' previous IT-based education, experiences with using software, and preferences in gameplay, development and accessible adaptions. The findings showed interest in making and playing open-world, puzzle, and combat games. Feedback regarding accessibility requirements and preferences included shortcut key-driven navigation for computer games and swipes or taps for touch screen, both interacting with a screen reader. These insights influenced the creation of Hodr Engine 0.1, a Terminal-based engine for macOS operated with a basic programming language named HodrScript. This programming language, which is accessed by a developer using any simple text editor, avoids complex syntax often overlooked by screen readers. However, after a BVI participant tested the engine to create a game, his feedback revealed barriers that might emerge for BVI users when using a CLI. These included the engine's "convoluted" commands for packaging an application containing unannounced syntax and "overwhelming" output on the console when executing the packaging command. The participant suggested a dedicated Hodr Engine text editor to code and package audio games as applications. This response prompted the development of Hodr Engine 0.2, incorporating a custom-written IDE to code, run, and package audio games as applications. This practice-based research outcome attempts to address the accessibility barriers for BVI users. implementing the required screen reader-related adaptions in order to allow users to create their own audio games with a BVI-friending programming language named HodrScript.

Feedback from the initial user preference survey and subsequent participant tester feedback provides guidance that will determine changes in the yet to be developed

Hodr Engine 0.3. Future improvements will include the introduction of binaural audio in games, such as the spatial audio heard in The Vale [21], going beyond the engine's current stereo soundscaping capabilities. The engine will also expand the game genres it supports to include participant's additional suggestions, such as single- and multi-player racing, platformers, refined combat, and puzzles. The engine's IDE will also be adapted for platforms such as Windows, mobile, and tablet, identified in the user preference survey as other popular platforms, alongside the IDE's current macOS build. Touchscreen builds will feature accessible adaptations, such as participants' suggested swipes and taps to navigate the IDE. Mobile games will contain environments that can be navigated by touching sections of the screen and navigating in the corresponding cardinal directions, with swipe gestures cycling through items and taps for selection. In conjunction, future iterations of the engine will incorporate features for requirements beyond screen readers or refreshable Braille displays, such as voice activation, which is assistive for BVI users with or without motor function impairments.

This commentary outlined accessibility improvements referenced in the literature review, including simple formatting, alt text, correctly labelled buttons, and shortcut keys. These features, tested with screen readers and refreshable Braille displays before release, are not limited to game engines but are essential for software development. As suggested by the RNIB Accessibility Gaming Research Report [46], it is recommended that developers are equipped with accessibility-focused training and documentation, such as the Game Accessibility Guidelines [26]. The distribution of such knowledge enables the creation of inclusive software from the outset, preventing barriers like those identified in this study. This approach avoids alienating BVI users from valuable skills, careers, and beloved art forms, such as game development and gameplay.

Game engines can also create experiences beyond gaming, such as architectural design and product placement. This suggests the Hodr Engine's potential as a platform for further creative and educational expression. With an accessible framework such as the design the Hodr Engine strives to possess, BVI users may immerse themselves in a previously challenging art form due to exclusionary design, proving you don't need sight to have vision.

References

[1] AbleGamers Charity. *AbleGamers: Creating Opportunities That Enable Play.* Website, https://ablegamers.org. 2004.

[2] Alter Aeon Development Team. *Alter Aeon*. https://www.alteraeon.com/. 1996.

[3] Apple Inc. *Accessibility Inspector: Tools for Testing and Improving App Accessibility.* Software, https://developer.apple.com/documentation/accessibility-inspector. 2007.

[4] Apple Inc. *Logic Pro*. 1993.

[5] Apple Inc. *Xcode: Integrated Development Environment for Apple Platforms.* Software, https://developer.apple.com/xcode/. 2003.

[6] Associazione Audiogames. *Blind Quest – The Enchanted Castle*. Video Game, https://store.steampowered.com/app/1258440/BLIND_QUEST__The_Enchanted_Castle/. 2020.

[7] ASSOCIAZIONE CULTURALE DI PROMOZIONE SOCIALE AUDIOGAMES. *Blind Gladiator*. https://apps.apple.com/gb/app/blind-gladiator/id1148929588.

[8] Audio Game Hub Team. *Audio Game Hub*. Video game, https://audiogamehub.com/. 2016.

[9] AudioGames.net. *AudioGames.net: The Home of Accessible Gaming*. Website, http://www.audiogames.net. 2002.

[10] Blender Foundation. *Blender: The Free and Open Source 3D Creation Suite.* 1998.

[11] Bureau of Internet Accessibility. *Dark Mode Can Improve Text Readability — But Not for Everyone — boia.org*. https://www.boia.org/blog/dark-mode-can-improve-text-readability-but-not-for-everyone. 2021.

[12] Centre for Research in New Music (CeReNeM). *CeReNeM Journal*. Journal,

[13] K. Collins. *Studying Sound: A Theory and Practice of Sound Design*. Cambridge, MA: The MIT Press, 2020. ISBN: 9780262044134.

[14] H. J. Cooper, C. Harrison, and J. Barry. *Sounds of Eden*. Video Game, available on the App Store https://apps.apple.com/us/app/sounds-of-eden/id1525196057. 2020.

[15] C. Didlick. *Auditory Icons*. Online article, designingsound.org/2016/11/24/auditory-icons/. 2016.

[16] S. Ding, J. B. Smith, S. Garrett, and B. Magerko. "Redesigning EarSketch for Inclusive CS Education: A Participatory Design Approach". In: *Proceedings of the 23rd Annual ACM Interaction Design and Children Conference (IDC '24)*. Delft, Netherlands, 2024.

[17] K. Drossos, N. Zormpas, G. Giannakopoulos, and A. Floros. "Accessible Games for Blind Children, Empowered by Binaural Sound". In: *Proceedings of the 8th ACM International Conference on PErvasive Technologies Related to Assistive Environments (PETRA)*. 2015, 5:1–5:8.

[18] S. Else. *Audio Defence: Zombie Arena*. Video game, https://archive.org/details/audio-defense. 2014.

[19] Epic Games. *Unreal Engine: The Game Development Engine*. 1998.

[20] O. Falase, A. F. Siu, and S. Follmer. "Tactile Code Skimmer: A Tool to Help Blind Programmers Feel the Structure of Code". In: *Proceedings of the 21st International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '19)*. New York, NY, USA: ACM, 2019, pp. 558–560.

[21] Falling Squirrel. *The Vale: Shadow of the Crown*. Video game, https://falling-squirrel.itch.io/the-vale. 2021.

[22] S. Fernando and J. Ohene-Djan. "An Empirical Evaluation of a Graphics Creation Tool for Blind and Visually Impaired Users". In: *Unpublished manuscript* (2020).

[23] Lo-Fi People. *Blind Drive.* Video game, https://blinddrivegame.com/. 2021.

[24] J. Frankel and Cockos Inc. *Reaper: Digital Audio Workstation.* 2005.

[25] B. A. Frey and R. Mancilla. "Inclusive Online Learning: Digital Accessibility Practices". In: *Diversity in Higher Education Remote Learning.* Ed. by H. Salman. Springer, 2023.

[26] Game Accessibility Guidelines. *Game Accessibility Guidelines: Practical Advice for Game Developers.* Website https://www.gameaccessibilityguidelines.com. 2012.

[27] GitHub, Inc. *GitHub.* https://github.com/. 2008.

[28] R. Hewett. "The Longitudinal Transitions Study". In: *Research Paper* (2020).

[29] Inclusive Games. *Inclusive Games: Accessible Gaming Experiences for Everyone.* Website, https://inclusivegames.org. 2016.

[30] International Business Machines (IBM). *IBM 029 Card Punch.* 1964.

[31] JBL Quantum. *JBL Quantum Guide Play: Enhancing Gaming Accessibility for the Visually Impaired.* Software, https://news.jbl.com/en-CEU/238373-jbl-quantum-introduces-new-tool-to-elevate-gaming-experiences-for-the-visually-impaired-community. 2024.

[32] C. Kerr. *Exploring the sonic foundations of audio-only adventure The Vale: Shadow of the Crown.* Online article https://www.gamedeveloper.com/audio/exploring-the-sonic-foundations-of-audio-only-adventure-the-vale-shadow-of-the-crown. 2021.

[33] L-Works. *Super Liam.* https://www.l-works.net/abandonware.php. 2004.

[34] J. Lazar, A. Allen, J. Kleinman, and C. Malarkey. "What Frustrates Screen Reader Users on the Web: A Study of 100 Blind Users". In: *International Journal of Human-Computer Interaction* 22.3 (2007), pp. 247–269.

[35] I. Lisboa, J. Barroso, and T. Rocha. "Digital Accessibility of Online Educational Platforms: Identifying Barriers for Blind Students' Interaction". In: *Research Paper* (2020).

[36] D. Löffler, L. Giron, and J. Hurtienne. “Night Mode, Dark Thoughts: Background Color Influences the Perceived Sentiment of Chat Messages”. In: *Human-Computer Interaction - INTERACT 2017 - 16th IFIP TC 13 International Conference, Proceedings, Part II*. 2017, pp. 184–201.

[37] D. Mazzoni and R. Dannenberg. *Audacity: Free Digital Audio Work Station*. 2000.

[38] Microsoft. *Inclusive Design Toolkit*. Website, `https://inclusive.microsoft.design/`. 2015.

[39] Microsoft. *Microsoft Human Interface Guidelines*. Website, `https://learn.microsoft.com/en-us/windows/apps/design/`.

[40] Microsoft. *Visual Studio Code: A Lightweight Code Editor*. 2015.

[41] Microsoft. *Xbox Accessibility Guidelines*. Website, `https://learn.microsoft.com/en-us/gaming/accessibility/guidelines`. 2020.

[42] T. Parr. *ANTLR (Another Tool for Language Recognition)*. 1992.

[43] S. Ragas. “How screen readers read special characters: an update”. In: (Mar. 2023).

[44] T. V. Raman. *Emacspeak: The Complete Audio Desktop*. 1997.

[45] Reactive Studios. *Codename Cygnus*. `https://www.codenamecygnus.com/`. 2013.

[46] Royal National Institute of Blind People (RNIB). *Accessible Gaming Research Report*. Tech. rep. Royal National Institute of Blind People, June 2022.

[47] H. Sampath, A. Merrick, and A. Macvean. “Accessibility of Command Line Interfaces”. In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2021.

[48] Santa Monica Studio. *God of War Ragnarök*. `https://www.playstation.com/en-us/games/god-of-war-ragnarok/`. 2022.

[49] J. Senter. *Nano Empire*. `https://apps.apple.com/us/app/nano-empire/id994354970`. 2015.

[50] V. Stadler and H. Hlavacs. “Blind Adventure: A Game Engine for Blind Game Designers”. In: *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play (CHI PLAY ’18)*. New York, NY, USA: ACM, 2018, pp. 503–509.

[51] R. Stallman. *GNU Emacs: The Extensible, Customizable, Self-Documenting Real-Time Display Editor*. 1976.

[52] C. Sutcliffe. *How accessible are game engines, and how much is still to be done?* Online article, `https://www.gamesindustry.biz/how-accessible-are-game-engines-and-how-much-is-still-to-be-done`. Apr. 2024.

[53] Teletype Corporation. *Teletype Model 33 (ASR33)*. 1963.

[54] J. Wardle. *Wordle*. `https://www.nytimes.com/games/wordle/index.html`. 2021.

Appendices

— A —

Glossary

- **BVI (Blind and Visually Impaired):** Limited or no vision
- **Screen Reader:** Announces onscreen content commonly used by BVI users
- **VoiceOver:** Apple's built-in screen reader for macOS and iOS
- **JAWS (Job Access With Speech):** Windows external screen reader
- **NVDA (Nonvisual Desktop Access):** Windows's free open-source external screen reader
- **GUI (Graphical User Interface):** Graphical interface including buttons, icons and menus
- **UI (User Interface):** Interface for users to interact with a digital device through visual elements
- **CLI (Command Line Interface):** Text-based interface to control a computer with a text-based system
- **Terminal:** Apple's built-in command-line interface
- **IDE (Integrated Development Environment):** Software for writing, running, debugging and packaging code
- **Xcode:** Apple's built-in development environment
- **Linux:** Open-source operating system controlled with a command-line interface
- **Unity:** Game engine for developing interactive content
- **Unreal Engine:** Game engine for developing interactive content
- **Audio Game:** A game played exclusively with audio cues

— B —

Survey About Gaming, Development, and Accessibility

1. What is your age?

- 6 - Between 18 and 30,
- 1 - Between 41 and 50,
- 1 - Between 51 and 60,
- 0 - Between 61 and 70,
- 0 - 71 or above.

2. To what degree is your visual impairment? Describe your level of visual impairment, such as tunnel vision, peripheral vision, or total blindness.

- Light perception in one eye and the other eye has less than one percent vision.
- Blind in one eye, and cataract in the other tunnel vision.
- Severely visually impaired.
- Minimal. Corrected with glasses.
- Very poor but with some usable sight. No detailed vision. Cannot read unless very large print, use screen readers/white cane.
- I have some light perception but no other useful vision.
- Total blindness.
- Severely sight impaired. Some central vision.

3. When experiencing content on a screen, can you see the screen unaided, or do you require an aid to access content? An aid could include a screen reader, refreshable braille display or a magnifier. Describe your experience in the text box.8 responses

- I can't see enough detail on a screen to look at it.
- A mix between screen reader and magnifier.
- Screen reader NVDA.
- Magnifier or glasses.
- Screen reader and Magnifier.
- I require an aid when navigating a screen. My main one is VoiceOver but I will occasionally use a braille display. I also use audio description where possible when watching TV and other such things.
- Apple VoiceOver.
- I require an aid and use VoiceOver or a different screen reader I can sometimes use zoom on my phone.

4. Do you struggle with eye strain when using a screen? If you have experience with eye strain, describe your experience in the text box. If you do not have eye strain or vision to look at a screen, you can bypass this question.

- I can't see enough detail on a screen to look at it but facing one still causes my eyes pain.
- I do suffer from eye strain when trying to read long text on the screen.
- I get eye strain when i look at my phone for a long time.
- I can sometimes get eye strain when using a screen, when this happens I will experience nystagmus.

5. Has your level of visual impairment always been the same or has it changed over time? Describe your experience in the text box.

- Changed over time.
- My vision began to deteriorate around the age of 16.
- My vision has got worse over time.

- Changed gradually throughout my later years. Need stronger prescription eye wear.
- Progressively worsening. Had perfect sight as a child.
- I have been blind since birth. I have always had light perception in both my eyes although I feel like this has deteriorated over time and I do not see things as bright as I used to.
- My vision has deteriorated with age, when I was younger I had more sight and did not always need aids to access the screen.

6. If your vision has decreased over time, have you retained a visual memory? If your vision has not decreased, you can bypass this question.

- Yes, I am a visual learner despite not being able to learn visually.
- I have a strong visual memory.
- Yes.
- As my sight gets worse, I struggle to remember how things look that I used to be able to see.
- Use memory.
- I believe I do have a visual memory from when I had more sight.

7. If you retain a visual memory with decreased vision, do you use your visual memory to navigate technology? Describe how you use your visual memory to navigate technology in the text box. If you do not have a visual memory, you can bypass this question.

- Yes.
- I use it to guess letters on the key board, I also know where each app on my phone is so I know where to press without looking. I also use it mainly for social media to scroll through.
- If I have seen a website or program through the magnifier, I can use it much easier with the screen reader without the magnifier than I otherwise would be able to. I often use the magnifier the first time and then use the screen reader on its own after that.

- I will try and imagine how tables or documents will look like when navigating to have a picture in my mind to make sense of when reading.

8. Do you believe your previous formal education gave you a knowledge of using technology such as computers with accessible adaptations? Describe your experience in the text box.

- No, I was only ever taught how to use a computer with the Jaws screen reader but I needed a Mac for university and no one at school taught me how to use VoiceOver.
- At secondary school no, but at the royal national college for the blind I did learn a lot and had IT lessons.
- Yes as they taught me how to touch type and use NVDA and GRID 3.
- My sight was ok throughout my education years.
- No.
- I learnt a lot of my technology skills, specifically computer skills throughout high school. Any others such as phone and iPad I picked up and learnt by myself as I already had the basic foundation down.
- No.
- Yes, I had sessions to learn how to navigate technology using assistive software. I have used a variety of screen readings including supernova and Jaws.

9. Would you consider yourself well-versed when using a computer or other forms of technology? Describe your level of experience in the text box.

- No absolutely not.
- Not entirely I do depend on what little vision I do have.
- Yes as I have the skills to navigate a computer with help from a screen reader.
- Basic, need to know, level.
- Yes. I am confident using the computer.

- Yes, I feel very confident in my technology skills. I have actively been using some form of technology since my teenage years, both professionally and for personal leisure. I have had a wide range of devices available to me over time which have enhanced my technology experience. I am an audio editor so I am used to learning complex technology skills.
- No.
- I feel I am able to complete every day tasks using a screen reader such as navigating Microsoft programs. I definitely feel I need more work to be more experienced in navigating online websites.

10. Regardless of your skill when using computers, do you or did you want to learn a computer-related skill, such as computer programming, game design, or product design? Describe what technological skill you are interested in learning in the text box, and if you are not interested, say why.

- Making music.
- Computer programming has been an interest of mine. More so when I lost my vision as I wanted to make things more accessible for myself.
- I have not had the option to do this, no one has given me the opportunity to so computing.
- Computers were not used in my school in the 70's, However I would have embraced the opportunity if they were.
- Yes. I am interested in programming in a few different languages.
- I did a bit of program coding when I was at high school and thoroughly enjoyed it. Although my knowledge was very basic and I have forgotten all of it by now due to not having the opportunity to develop those skills, I would definitely consider learning something like this again. I like the idea of being able to build something that is completely accessible to others like me. I am always interested in finding out how things are made so I love the sound of being able to create something that can be really interactive, something that people could get lost in for ages.

- No, more of a practical person physical work.
- I definitely want to learn more computer based skills such as game design and product design. In particular, in programming and developing apps.

11. Are you interested in playing digital games or exploring virtual interactive environments? You can choose more than one option.

- 0 - Not at all,
- 1 - Not really,
- 4 - Indifferent,
- 2 - Interested,
- 3 - Very interested.

12. What are your favourite styles of digital games? Please describe your preferred genres in the text box, if you have no interest in digital games, please note this in the text box.

- Open world/ combat.
- Puzzles and word games.
- I like to play racing, platformer, adventure and party games.
- Brain exercising games, strategic games.
- text adventure like Alterion, better with sound effects.
- I love playing adventure games, specifically story based games. I think ones that have multiple ways of playing it are my favourites as they allow me to go back through the story and choose different options, resulting in different outcomes. I prefer sound immersive games rather than text adventures as for me this adds to the experience. If I ever play games that don't really have a story to them, these would also be preferred as audio games. I also enjoy games where you get to build your own settlements. I would also love it if there were more accessible fantasy games available.
- First person shoot them up.
- I enjoy card games and quiz games but also interactive player games where you complete missions or tasks etc.

13. If you are interested in creating digital games or virtual interactive environments? Describe the genre of game or interactive experience you want to make in the text box. If you are not interested in making a game, note this in the text box.

- Combat, good sound effects, humour.
- Sports, adventure, racing.
- I would want an interactive experience creating an environment for artists to explore and contribute to each other's works.
- Sound based games.
- I would be interested in creating a fantasy adventure game. Something along the likes of the video game Skyrim.
- Action adventure.
- I'm definitely interested in developing an interactive game. I would love to incorporate music and sounds into this with interesting tasks and to explore different environments.

14. Have you played audio games before?

- 4 - Yes,
- 4 - No,
- 0 - I don't know what an audio game is.

15. What platform do you play digital games on? You can tick multiple options. You can bypass this question if you do not play games.

- 6 - Mobile Device,
- 2 - Tablet,
- 3 - Computer,
- 1 - Games Console.

16. If you play digital games, on what platform did you hear about your favourite game? You can bypass this question if you do not play digital games.

- Online forums.
- Advertisement on TikTok.

- Heard about it from other people and YouTube.
- Windows.
- One of my favourite games that I have ever played is called A Hero's Call, available on Windows computers. A friend of mine recommended this to me.
- I have heard of games through friends who are also blind.

17. Would you be interested in producing digital games?

- 0 - Not at all,
- 2 - Not really,
- 1 - Indifferent,
- 3 - Interested,
- 2 - Very Interested.

18. If you created a game, what input methods would you prefer for navigating the game? Consider options like a trackpad, touch screen, or computer keys. Please share your thoughts in the text box.

- A screen reader to announce text and navigation by moving a finger on the mobile screen.
- A mix between touch screen and computer keys. Computer keys give me more control and touch screen can allow me to explore the screen.
- Touch screen and controls.
- I would prefer short cut keys as they are easier to locate.
- Keyboard.
- I would create a game that would be available across different platforms. Therefore, it would have to be available for a variety of inputs. For example, arrow keys on a computer or touch screen gestures on a phone and tablet. Perhaps on a laptop it could have a cross between using the arrow keys and the trackpad. I've seen this in a video game I've played called As Dusk Falls and it makes the game more interactive. On a phone and tablet gestures would

include swiping left, right, up and down, double tapping to interact with elements, and perhaps scrubbing or swiping with multiple fingers to conduct an action.

- Voice activated.
- I would want to create a game that people are able to navigate on their phones as that is what I find most accessible. I think touchscreen or voice input would be a good way to do this.

19. What is your level of expertise in audio production to create music and sound design for a game?

- Low level.
- Very little.
- None.
- I don't have any.
- None.
- I am very confident with my audio editing and sound design skills as I have been doing this for over 5 years now, some of which has even been for paid work.
- Music 50 say.
- Literally no experience at all.

20. Regarding game development, are you interested in audio-only games, visual games or a combination of both? Share your thoughts on what would appeal to you in the text box. If you are not interested in any format of game creation, note this in the text box and say why.

- Only audio games.
- I would be interested in an audio game.
- Both a combination of both.
- A combination of both would be good. It would prepare me for when I lose more reading sight and need to rely on sound or touch.
- Audio only. Maybe text too.

- I would prefer games that have both audio and visual elements. Although I cannot see, audiovisual games would be great to play with my sighted friends and family. Most sighted people don't seem to want to play audio only games whereas something that includes both audio and video is more immersive.
- Not interested, do not play games.
- Definitely a combination of both, mainly leaning towards audio games but I think visual games having an aspect are also really helpful for people who do have some site.

21. If you created visual games, what art styles are you inclined towards? For example, 3D realistic graphics, 2D cartoons or something else? If you do not have an understanding of how these design styles look, note this in the text box. If you are not interested in visual art styles, note this in the text box and say why.

- I don't know.
- I would stick to 2d graphics so it's easier for my eyes to focus, I would also give the options to have colour or black and white to switch back and forth too.
- I don't mind 3D or 2D as i am able to tell the difference.
- Definitely 3D realistic graphics so I would feel more involved and a part of the game.
- Not interested.
- I think that 3D, realistic graphics and even cartoons would be interesting to play with but I do not have enough understanding of how these things work as this is not something I personally am able to pay attention to.
- I like the idea of 3-D realistic graphics, especially as I would love the game to explore different environments and settings, so I think this would be really interesting for it to look more realistic rather than flat 2-D animations.

22. If you were to take lessons in learning to use an accessible game creation platform, how would you like to learn? If you are not interested in learning how to use such a platform, select the method you find most accessible and user-friendly in your previous educational experience.

- 5 - Face to face lessons,
- 0 - Online lessons over a video conferencing platform like Zoom,
- 1 - Through reading a tutorial in a document,
- 1 - Through watching a narrated tutorial on a video sharing platform.

23. Would you like to take lessons in your chosen method to create games?

- 1 - Not at all,
- 0 - Not really,
- 2 - Indifferent,
- 3 - Interested,
- 2 - Very Interested.

24. If you have any additional thoughts or notes on the survey's content, add them to the text box.

- I think maybe having audio and visual games would be cool, also perhaps the use of my own voice maybe to dictate actions on the game. Like a function that could allow me to ask where I was on the screen what to do next.
- Although gaming is not something I want to do, I do think game designing made accessible to a wider community especially the visually impaired is a good way forward for up-and-coming creators who would otherwise be denied the experience due to the challenges they would face in the digital gaming market.

— C —

Survey About User Experience of the Hodr Engine

1. How easy was it to install the Hodr Engine, and set up a project? Describe your experience in the text box.

> I had notable difficulties beginning my experience with the engine. The terminal was difficult to use because typing commands that aren't English sentences are difficult to keep track of with a screen reader. Additionally, Terminal outputs a lot of information in the form of computer jargin, which is extremely difficult to parse with VoiceOver. Additionally, the commands supplied did not function as expected on my machine, and as a result, I had to spend time with the developer troubleshooting.

2. Were the installation instructions in the documentation easy to follow? Describe your experience in the text box.

> The installation instructions in and of themselves were clear and easily followed, but some of the commands did not function as expected.

3. How would you rate the ease of the HodrScript? Describe your experience in the text box.

> The HodrScript was very easy to use. Much of the punctuation and syntax rules which I have encountered in other programming languages (ie. Python, C++, html), were not needed to write my game code. This meant little battling with the screen reader, and the process was efficient

and not stressful.

4. Were there any parts of the Hodr Engine or HodrScript that were confusing? Describe your experience in the text box.

> The process to run the game was slightly convoluted, involving saving my text file and using Terminal commands to compile the game. This was confusing at times.

5. Are there any styles of games that you would like to create with the Hodr Engine that could not be currently designed due to the engine's limitations? Describe your experience in the text box.

> I would love to be able design games with a more dynamic environment eg. be able to program the movement of objects.

6. Did you come across any accessibility barriers while using the Hodr Engine? Describe your experience in the text box.

> I was able, after resolving technical difficulties, to finish my game, so that sense, the process was accessible. However, the reliance on MacOS Terminal was unintuitive and did not interact well with my screen reader.

7. How well does the Hodr Engine integrate with Voice-over, or any other accessible adaptions you use? Describe your experience in the text box.

> The integration with VoiceOver was good in some areas. For example, editing a txt file and navigating my file hierarchy are tasks which rely on VoiceOver's natural strengths and it's integration with MacOS. Using Terminal was less easy. VoiceOver, unless specifically altered, does not cope well with correctly and reliably reading text which involves non-linguistic elements, which made it extremely difficult to understand what it was saying. Similarly, the quantity of info output by terminal in the case of an error was often overwhelming.

8. How easy was creating an app with the Hodr Engine? And did the engine perform as expected? Describe your experience in the text box.

Creating an app went mostly smoothly, and the engine did perform as expected. Myself and the developer encountered one issue wherein the engine required a number of elements to be present in the code which were not listed in the instructions list of mandatory elements. I needed to add a number of new sound files for the game to function.

9. Were there any features or tools missing from the Hodr Engine? Describe your experience in the text box.

I would like to have seen a dedicated text editor with easy ways to run and save the project. I would like to have been able to program the movement of objects, and have control over the global volume of sound files.

10. What did you like most about the Hodr Engine?

I love the idea of being able to create immersive interactive experiences, and I liked the mobility the engine offers in being able to move between various sonic landscapes.

11. What did you like least about the Hodr Engine?

I found the use of Terminal, whilst interesting as a learning experience, overall exhausting.

12. Do you have any improvements for the Hodr Engine? Describe your experience in the text box.

I would like to see a dedicated text editor, and more control over how I manipulate sound files in the code.

13. Note any additional comments or feedback in the text box.

Thank you so much for creating this kind of thing. It's not a novelty or gimmick, it fills a viable niche which will bring value to people. I have always wanted to be able to make games, but find the inaccessibility in current game dev tools and culture to be overwhelming and frustrating. This gives me hope that one day I might be able to do what I've always wanted.

— D —

Participant's HodrScript Code

```
settingsname = gameSettings
projectname = finishedGameIE
startsoundscape = welcome
togglecutscene = shimmerUpAudio.wav
togglepause = jingle1Audio.wav
save = triangleAudio.wav
collectjingle = jingle2Audio.wav
choicebackground = twinkleUpAudio.wav
missed = twinkleDownAudio.wav
alreadycollected = plunkAudio.wav
quit = dingDongAudio.wav
end

graphicsname = blackScreen
splashfile = blackScreen.jpg
end

menublock = mainGame
jingle = allTheWayAudio.wav
nogame = punchAudio.wav
start = sweepyAudio.wav
continue = bigSweepyAudio.wav
cancel = lowSweepyAudio.wav
click = snapAudio.wav
```

```
menubackground = haveYourselfAudio.wav
end

soundscape = welcome
cutscene = imaginationAudio.wav
entry = fanfareAudio.wav
background = spoopzAudio.wav
move = jangleAudio.wav
sound = joyAudio.wav (10, 10)
size = 20
nextsoundscape = None
end
```

— E —

HodrScript CeReNeM Journal Code

```
settingsname = gameSettings
projectname = CeReNeMAudioExperience
startsoundscape = welcome
save = click.mp3
quit = click.mp3
end

graphicsname = Screen
splashfile = start_image.jpg
end

menublock = mainGame
nogame = nogame.mp3
start = start.mp3
continue = continue.mp3
cancel = cancel.mp3
click = click.mp3
menubackground = menubackground.mp3
end

soundscape = welcome
background = aircon.mp3
move = step1.mp3, step2.mp3, step3.mp3, step4.mp3, step5.mp3,
  step6.mp3, step7.mp3, step8.mp3
```

```
sound = ASCharlotte1.mp3 (-40, -35) loops = 0
sound = ASCharlotte2.mp3 (40, -34) loops = 0
sound = ASFreya.mp3 (25, -30) loops = 0
sound = ASTom.mp3 (-25, -34) loops = 0
sound = Charlotteimprov1.mp3 (-34, -25) loops = 0
sound = Charlotteimprov2.mp3 (34, -26) loops = 0
sound = Freyaimprov.mp3 (35, -2) loops = 0
sound = Joeimprov.mp3 (-25, 30) loops = 0
sound = Anthonyimprov.mp3 (-15, 30) loops = 0
sound = Tomimprov.mp3 (-35, 10) loops = 0
size = 50
nextsoundscape = None
end
```