

|  |
| --- |
| A Blind Accessible Approach to Game Engine and Tools Development  Bridget A. Casey  BSc (Hons) Computer Games Technology,  2023 |

|  |
| --- |
| School of Design and Informatics  Abertay University |

Contents

[Table of Figures iv](#_Toc135171017)

[Table of Tables v](#_Toc135171018)

[Acknowledgements vi](#_Toc135171019)

[Abstract vii](#_Toc135171020)

[Abbreviations, Symbols and Notation viii](#_Toc135171021)

[1. Introduction 1](#_Toc135171022)

[1.1 Blind Accessibility in Games 1](#_Toc135171023)

[1.2 Accessibility of Modern Game Development Tools 1](#_Toc135171024)

[1.3 Research Question 2](#_Toc135171025)

[1.4 Aims and Objectives 3](#_Toc135171026)

[1.5 Hypothesis 3](#_Toc135171027)

[1.6 Overview 3](#_Toc135171028)

[2. Literature Review 4](#_Toc135171029)

[2.1 Blind Accessibility in Interactive Computer Software 4](#_Toc135171030)

[2.2 Applying Accessible Design to Game Development 4](#_Toc135171031)

[2.2.1 Design of Accessible Game Development Tools 4](#_Toc135171032)

[2.2.2 Design of Accessible Gameplay Systems 5](#_Toc135171033)

[2.3 Previous Implementations in Game Engines 6](#_Toc135171034)

[2.3.1 eAdventure 2.0 6](#_Toc135171035)

[2.3.2 Blind Adventure 6](#_Toc135171036)

[2.3.3 Sable Engine 7](#_Toc135171037)

[2.3.4 ShadowRine Audible Mapper 7](#_Toc135171038)

[2.4 Summary 8](#_Toc135171039)

[3. Methodology 9](#_Toc135171040)

[3.1 Applied Tools and Technologies 9](#_Toc135171041)

[3.1.1 External Dependencies 9](#_Toc135171042)

[3.2 What Makes a Game Engine Editor? 10](#_Toc135171043)

[3.3 Designing an Accessible Editor Interface 12](#_Toc135171044)

[3.3.1 Conceptualising the Editor Layout 13](#_Toc135171045)

[3.3.2 Structuring the Editor for Accessibility 18](#_Toc135171046)

[3.3.3 Accessibility at Runtime 21](#_Toc135171047)

[3.4 Game Engine Architecture 22](#_Toc135171048)

[3.4.1 User Input and the GUI 23](#_Toc135171049)

[3.4.2 Audio Management 23](#_Toc135171050)

[3.4.3 Screen Reader Support 23](#_Toc135171051)

[3.5 Flow of User Experience 24](#_Toc135171052)

[3.5.1 Application Control Scheme 24](#_Toc135171053)

[3.6 Evaluation Planning and Process 25](#_Toc135171054)

[3.6.1 Designing User Tasks 26](#_Toc135171055)

[3.7 Summary 26](#_Toc135171056)

[4. Results 27](#_Toc135171057)

[4.1 Response from Users 27](#_Toc135171058)

[4.2 Additional Observations 34](#_Toc135171059)

[4.3 Summary 34](#_Toc135171060)

[5. Discussion 35](#_Toc135171061)

[5.1 Analysis of User Experience 35](#_Toc135171062)

[5.1.1 Menu and Panel Navigation 35](#_Toc135171063)

[5.1.2 Spatial Scene Navigation 36](#_Toc135171064)

[5.1.3 Playability of the Demo Scene 38](#_Toc135171065)

[5.2 Effectiveness of the Solution 38](#_Toc135171066)

[5.2.1 Limitations of the Study 38](#_Toc135171067)

[5.2.2 Consideration of Technical Problems and External Factors 39](#_Toc135171068)

[5.3 Summary 40](#_Toc135171069)

[6. Conclusion 41](#_Toc135171070)

[6.1 Future of the Project 41](#_Toc135171071)

[6.2 Potential Impact of Research 42](#_Toc135171072)

[References 43](#_Toc135171073)

[Appendices 45](#_Toc135171074)

# Table of Figures

[Figure 1: Example scene inside GameMaker 2. 11](#_Toc135171075)

[Figure 2: Default project inside Bitsy, a little editor for little web games. 11](#_Toc135171076)

[Figure 3: Mock-up of an early design for the editor tool. The initial idea was to condense all possible interactions into the same toolbar, organised by category. 13](#_Toc135171077)

[Figure 4: A view of the project panel which features a mixture of user preferences and application functions. 14](#_Toc135171078)

[Figure 5: A view of the scene hierarchy with example objects and the 'CREATE ENTITY' button highlighted. 15](#_Toc135171079)

[Figure 6: A view of the entity properties panel when the default player character is selected. The sprite and character controller drop-down menus are enabled. 16](#_Toc135171080)

[Figure 7: A view of the entity creation window, accessible from the scene hierarchy. It features a list of example entities which can be added to the scene. 17](#_Toc135171081)

[Figure 8: The final iteration of the editor, with several entities placed within the scene. Takes the form of a more traditional editor GUI with separate, simultaneously active panels. 18](#_Toc135171082)

[Figure 9: Visual representation of the directional audio cues feature. The red circle marks the Green Square entity's original position, with the extending white line indicating the direction it has been moved. The yellow lines represent the audio’s direction. 20](#_Toc135171083)

[Figure 10: Visual representation of feedback given when two entities overlap. The application checks the distance in both axes of each entity from the centre and determines which edges of their box colliders are intersecting. 21](#_Toc135171084)

[Figure 11: Visual representation of raycasting on a 2D grid (Vandevenne, 2020). 37](#_Toc135171085)

# Table of Tables

[Table 1: The list of application controls mapped to their functions. 24](#_Toc135171086)

[Table 2: Answers to user testing background interview, question one. 27](#_Toc135171087)

[Table 3: Answers to user testing background interview, question two. 28](#_Toc135171088)

[Table 4: Answers to user testing background interview, question three. 29](#_Toc135171089)

[Table 5: Answers to user test session review, question one. 29](#_Toc135171090)

[Table 6: Answers to user test session review, question two. 30](#_Toc135171091)

[Table 7: Answers to user test session review, question three. 31](#_Toc135171092)

[Table 8: Answers to user test session review, question four. 32](#_Toc135171093)

[Table 9: Answers to user test session review, question five. 32](#_Toc135171094)

[Table 10: Answers to user test session review, question six. 33](#_Toc135171095)

# Acknowledgements

I would like to thank my supervisor, Dr Jung In Jung, for her expert guidance in helping me achieve what I feel has been the most challenging and ambitious project I’ve undertaken so far. She continually steered me in the right direction and encouraged me to explore new ideas every week.

I would also like to acknowledge the remaining staff at Abertay University’s School of Design and Informatics for providing me the resources and support to complete this endeavour, especially that of Professor Ruth Falconer. I would not have been as successful in accomplishing my goals without their advice.

Finally, I would like to extend gratitude to my university peers and friends for listening to my struggles and keeping me motivated to complete this project. Most of all, I’d like to thank my dear friend Ámbar and her cat Mochi, whose warm and fluffy presence kept me sane during a very difficult time.

# Abstract

Blind-accessible software design is essential to ensuring that users with severe visual impairments can utilise software with the same efficiency and ease as sighted users can. The goal of this project is to determine the effectiveness of common audio feedback and UI design techniques as a solution for increasing blind accessibility in an interactive editor tool for computer games.

This was achieved by designing and building a simple, blind-accessible level editor tool for a 2D platformer game. Functionality was restricted to placing pre-determined objects in a scene to create a demo level, which could then be simulated in a runtime environment. The editor allowed for full narration of the interface by feeding custom text descriptions of UI elements and scene objects to third party screen reading software. Additionally, custom audio cues played for repetitive actions where text alone is less suitable, such as tab switching. The project also sought to simplify menu navigation by condensing the number of available user interactions, as well as improve spatial awareness in a 2D scene with custom audio techniques.

The application then underwent qualitative user testing by sight-impaired individuals from a range of technical backgrounds to ensure it met usability expectations. It was hypothesised that the chosen design would allow users to fully navigate the editor, build a demo level, and play test it without requiring any vision.

It was discovered that the implemented techniques had varying degrees of success with users. Their effectiveness was analysed in the context of the project and potential improvements were suggested. Additionally, theories were proposed about how best to adapt the findings of this research into larger game engines and level editor projects.

**Keywords**

Blind accessibility; Visual impairment; Screen reader; Audio games; Tools programming; Human-computer interaction, User interface design; Game engine development; Level editor; 2D platformer.

# Abbreviations, Symbols and Notation

NVDA – Non-Visual Desktop Access

JAWS – Job Access With Speech

UI – User Interface

GUI – Graphical User Interface

AUI – Audio User Interface

GEA – Game Engine Architecture

VS – Visual Studio IDE

# 1. Introduction

For more than two decades, there has been a growing push from game developers and players alike to create more accessible experiences for users with disabilities, including those with limited sight. The most popular games today encompass a wide variety of genres, each with their own style of play, from relaxing farming simulators like Stardew Valley[[1]](#footnote-1), to fast-paced first-person shooters like Doom Eternal[[2]](#footnote-2). However, one common element unites them all – they are heavily reliant on graphics to communicate. But for someone living without sight, this reliance on visuals means key information is always lost.

## 1.1 Blind Accessibility in Games

Historically, accessibility options have fallen by the wayside during the game development process for a multitude of reasons, many of which stem from a lack of knowledge about the subject. A 2022 study led by RNIB found that approximately 20% of studios believe incorporating support for blind players would compromise the quality of their games, while 64% are unsure how to even accommodate these needs in the first place (IGDA GASIG, 2022). Despite these difficulties, some developers have succeeded. In the early 2000s, studios like Zform went to the extent of creating parallel audio and graphical user interfaces, so that every object and action in their games with a visual component had accompanying audio feedback (Andersen, 2002). On a smaller but still notable scale, popular MMORPG World of Warcraft[[3]](#footnote-3) introduced a colourblind mode in 2015 (Wilds, 2020). More recently, The Last of Us Part II[[4]](#footnote-4) was noted for its bespoke audio narration system, which provided sound cues for all gameplay events and interactions, allowing sightless gamers to play without sighted assistance (Sightless Kombat, 2020).

## 1.2 Accessibility of Modern Game Development Tools

Despite studios making great strides towards blind accessibility in their games, the same cannot be easily said for their development tools. Two of the best-known commercial game engines are Unity[[5]](#footnote-5) and Unreal Engine[[6]](#footnote-6). They are popular among large and small studios alike due to their power and affordability, but they lack many of the same assistive techniques and design choices as the games they have been used to create (Tyagi, Choudhary, and Majumdar, 2019).

In Unreal Engine 5, there exists limited support to help developers with integrating screen readers into their games, but not for use within the engine itself. Almost no element of the user interface is compatible with screen readers, as text and buttons are rendered as plain images.

Additionally, in both Unity and Unreal, the default graphical interface can easily become cluttered with widgets. This means that even if a screen reader could be used, it would struggle to interpret relevant information in a coherent manner.

One might assume a visual impairment would prevent a person from thriving in a software development role, but approximately 1 in every 100 programmers is blind (Larson, 2017). This is partly due to the increased availability and affordability of assistive technologies like the screen reader, which has existed for decades as the primary tool for helping people who are visually impaired interact with computer software. Current popular screen readers, such as the free and open-source NVDA[[7]](#footnote-7), can interact with busy webpages and text-based desktop applications, providing audio feedback when the environment changes.

Many modern programming IDEs, such as Visual Studio[[8]](#footnote-8) and Eclipse[[9]](#footnote-9), are also used successfully by software developers who rely on screen readers. Just within the past few years, Microsoft has integrated support features into Visual Studio specifically for this purpose, such as the ability to navigate the entire display using only keyboard controls (Coding Tech, 2018).

## 1.3 Research Question

If a game developer who relies on screen reading software desires to use certain industry standard tools, create graphical applications, or fill a role other than programming, their options are limited. A screen reader can do nothing when presented with plain icons or complex geometry. In these cases, the only alternative is to build a custom engine to suit their needs, complete with bespoke audio and descriptions (Hamilton, 2015).

However, this is not a realistic solution for many. A game engine is often an enormous application with an extensive list of features and sub systems, which can take a great number of people and a long time to develop. This results in the engine’s editor having a complex graphical user interface, far more than the average text document or website, rendering a screen reader unusable.

When considering this struggle, a key question emerges – to what extent can a graphical game development tool be made blind-accessible using custom audio feedback and minimalist UI design?

## 1.4 Aims and Objectives

The overarching goal is to develop a bespoke level editor tool for simple 2D games, which embodies common design principles for blind accessible software. More specific objectives are to:

* Implement underlying game engine architecture based on existing, open-source frameworks, which contains core elements such as a 2D renderer, input event system, entity component system, and audio manager.
* Integrate support for compatibility between UI elements and third-party screen reader applications, using existing libraries.
* Implement a bespoke GUI-based editor using chosen design techniques, built upon the custom engine framework.
* Evaluate the effectiveness of the solution through qualitative testing with a small group of visually impaired users in the target audience.
* Revise prototype design based on feedback from test participants.

## 1.5 Hypothesis

The implementation of multiple techniques commonly used to accommodate visually impaired users in other types of software can be applied to game engine editors. These include screen reader compatibility, simplified menu navigation, event-based audio cues, and large, high-contrast text. It is hypothesised that the combined use of several smaller features will allow sightless users to navigate the prototype tool and use it to create levels for a simple game without assistance from a sighted person.

## 1.6 Overview

The Literature Review will explore previous examples of where blind accessibility in game development tools has been achieved. The Methodology will provide a detailed breakdown of the steps undertaken to design and develop the level editor prototype. The Results chapter will detail the findings from the user testing sessions conducted with visually impaired participants. The Discussion chapter will evaluate these results in the context of the prototype. The Conclusion will explain the implications of this research in relation to large-scale game engine design, as well as how the chosen implementation could be revised or extended to better meet the needs of its userbase.

# 2. Literature Review

This chapter breaks down currently established methods for improving the experience of interactive computer software for visually impaired users. It also discusses how popular tools and design choices, such as screen reader-friendly user interfaces, have been applied to game development tools. These techniques are featured in both theoretical and practical examples, and they are discussed in the context of their respective research projects.

## 2.1 Blind Accessibility in Interactive Computer Software

Although previous research has been done on the accessibility of game engines specifically, due to its current limitations, it is worthwhile to examine other types of interactive software with graphical components for common or adaptable ideas.

One study examined the screen reader accessibility of interactive digital maps on various devices (Hasan and Gjøsæter, 2021). While not a game or development tool, the researchers identified several issues in these applications tied to having an interactive graphical interface intended for spatial navigation. Among them was the low-contrast colours of many GUI elements, leaving those with low vision unsure of what they’re looking at. On desktop platforms, panning and zooming was typically limited to mouse movements, although one study participant suggested this could be reconfigured to use the arrow keys instead.

Regarding spatial navigation of the map itself, screen reader feedback was limited. On touchscreen devices, tapping different areas of the map yielded no output. To remedy this, one suggestion was to speak the name of each touched location, or the relative coordinates, so the user could build up a mental image of the environment over time with each tap. Alternatively, some method of converting the interactive map elements into text descriptions that conveyed the same information as the GUI was also suggested.

## 2.2 Applying Accessible Design to Game Development

According to recent research, creating blind-accessible game development tools is possible, but like any other interactive application, it requires careful consideration when designing the interface.

### 2.2.1 Design of Accessible Game Development Tools

A study from 2019 on the inaccessibility of modern game development software (Tyagi, Choudhary, and Majumdar, 2019) concluded that to create more broadly accessible tools, game engine developers should aim to reduce the number of user inputs required to achieve an outcome. This was found to not only be beneficial to those with physical disabilities, but to sightless users as well who may have difficulty with spatial orientation or memory.

As a rule of thumb, developers were also told to ensure at least two forms of feedback, such as visual, auditory, and tactile, exist for every action. By incorporating a multi-modal user interface in this manner, the chances of application feedback being missed by the user due to a physical impairment would be minimised.

On the other hand, the study also found it was equally important to strike a balance between providing enough feedback in multiple forms and overwhelming the user with unnecessary stimuli. This could be achieved through limiting the number of displayed objects and interactive elements on the screen, whilst also ensuring that all essential information required to use the tool was available without interruption in a frequently displayed or easily accessible area, such as the main menu.

### 2.2.2 Design of Accessible Gameplay Systems

While it is not typically the responsibility of an engine programmer to design gameplay systems, the purpose of any engine is to optimise the game development cycle by providing handy solutions to repetitive problems.

Thus, the best commercially available engines will provide users with an array of base scripts and classes for commonly needed functionality. This can range from character controllers and input management to complex rendering and audio systems, to artificial intelligence and pathfinding (Schardon, 2022).

An engine also typically provides a method of simulating gameplay inside the editor. However, any built-in functionality that serves gameplay utility also needs to have feedback consistent with the primary mode of interaction used by the editor interface. Otherwise, this could hinder the prototyping process if the user is unable to test their work (Gregory, 2018).

As the primary mode of interaction and feedback for most game engines is visual, this is automatically a non-issue for many sighted users. However, for a visually impaired developer to take advantage of all the tools at their disposal, multimodal interaction must be considered at every layer of the engine interface – including at runtime.

Most engines are designed with a specific game genre or project in mind. As such, the feedback given to users by its built-in systems during runtime will depend on what genre or style of gameplay the engine targets. A research study from 2017 compared the experiences of playing ten different games between a sighted user and a blind user, then collated a list of recommendations for developing audio interfaces in games.

Among these considerations, they found it was important to have concise and clear dialogue, so it was easy to digest when read at a rapid rate. Additionally, ensuring the information and style of content was consistent across gameplay, so the user could accurately predict what input was expected of them. Providing assistive functions to minimise manual navigation, like automatically rescuing the player when stuck, was also helpful (Teixeira Borges and de Borba Campos, 2017).

## 2.3 Previous Implementations in Game Engines

As time and technology progress, once-popular tools for creating blind accessible games like the Blastbay Game Toolkit Engine are becoming unsupported and obsolete, with nothing new to replace them (Urbanek et al., 2019). Due to the relative complexity of integrating multimodal user interfaces into long-standing codebases, major game engine developers have currently opted to forgo such designs altogether (IGDA GASIG, 2022). This renders some of the best tools on the market completely unavailable to sightless developers.

Despite these hurdles, some examples still exist of where accessible design principles have been integrated successfully into game development tools targeted specifically at blind users, although none have released full versions which are currently available for purchase or download.

### 2.3.1 eAdventure 2.0

The eAdventure tool (Torrente et al., 2014) is an open-source application intended for creating simple 2D, point-and-click style games for web browsers. It was developed in Java and initially released in 2005.

Beginning with version 2.0, an accessibility module was integrated to allow for various support features, including the use of screen readers. This enabled the user to exclusively use the keyboard to navigate the application. Additionally, the Accessibility Core ensured game objects were compatible with screen readers by changing how they were rendered. Object descriptions were displayed as text and then fed through a text-to-speech API bundled with the application. Other effects also existed, such as the ability to generate a textual description of the environment’s contents when entering a new scene.

While no formal user testing was conducted as part of this study, the proposed changes to the framework were hypothesised to allow for the easier creation of accessible games through screen reader compatible web browsers.

### 2.3.2 Blind Adventure

Blind Adventure (Stadler and Hlavacs, 2018) is a tool for creating simple, exclusively audio-based games for iOS and Android devices. It was unique in that it was aimed primarily at sightless game developers. Upon loading the app, users would select between two modes – play a previously created game or make a new one.

In engine mode, the application employed various design choices aimed at improving the experience of visually impaired users. These included large fonts and high contrast, as well as spoken audio cues when a scene changed or when a choice was presented on-screen. Additionally, navigation was simplified to a handful of swipe gestures. Consideration was given to how choices could be presented simply, and this was achieved using a node-based system. This meant no more than two buttons were displayed at one time. e.g., ‘add level’ and ‘edit level’. Orientation was also locked to portrait mode, as a blind user would have no way of telling if this changed.

Based on feedback gathered by the researchers, testers’ experience with the accessibility features of the app was positive overall, except for minor issues related to the speed of swiping during menu navigation.

### 2.3.3 Sable Engine

The Sable Engine is a game development framework for creating tile-based roleplaying games with quests, characters, and fully modifiable terrains. Its primary target audience is visually impaired developers who intend to make purely audio-based games.

However, unlike Blind Adventure, it notably has no graphical interface and relies entirely on keyboard input, screen reader feedback, and sound effect cues for navigation through the interface. In addition to this, consideration was given to make this navigation as simple and convenient as possible, such as through shortcuts that allow the user to modify and place several objects in one selection (Ramanan, 2019).

As of writing, Sable Engine has yet to be officially released to the public. However, a demo version[[10]](#footnote-10) of an alpha build was made available in 2020. Community feedback was generally positive. However, some users found the engine to have a steeper learning curve regarding the use of placing objects to create gameplay logic in lieu of scripting by hand (IllegallySighted, 2020).

### 2.3.4 ShadowRine Audible Mapper

ShadowRine is a 2D Japanese action roleplaying game in which the player can explore a world and fight monsters across many maps. The developers later released a graphical editor tool for players to create custom maps. However, the ShadowRine Audible Mapper (Matsuo et al., 2016) is different in that it can be fully navigated by blind users.

The developers chose Hot Soup Processor (HSP) as their programming language, as this allowed for significant screen reader support. For ease of use, the mapper tool permitted users to move the mouse cursor using keyboard arrows and select objects using number keys. For speed purposes, users could also select a range of scene objects and work on them all at once. Audio feedback also featured dynamic sound effects. A cue would play to indicate where the mouse cursor was, with varying pressure depending on its distance across each axis.

Based on the researchers’ testing, sight-impaired users were able to create maps effectively using this tool, although navigation sometimes took trial and error. Some users complained that the sound system was confusing, and the maps were too large, leading them to get lost when placing objects.

## 2.4 Summary

This chapter gave an overview of commonly used techniques for creating interactive software that is also blind accessible. Additionally, it discussed examples of where they have been applied or considered in the context of game engines and game editor tools. Although, it was also established that the market of game development tools which have achieved this is a narrow one, therefore conclusive evidence of effective software design techniques is limited.

The Methodology chapter will explore how the techniques selected from literature, as well as other new ideas, have been adapted for use in designing and developing the prototype application.

# 3. Methodology

To facilitate the research, a graphical application was developed that demonstrates several core functionalities of an editor for a small game engine. Throughout the development process, consideration was given to adhere to many of the design principles uncovered during the literature review to ensure the interface would be accessible to visually impaired users.

The project followed a form of the Agile methodology during its development, due to its cyclical process of testing and iterating on a design several times before the final launch. This meant the user interface was constantly evolving as both the requirements and limits of the application became more apparent.

## 3.1 Applied Tools and Technologies

The application was built using C++ 17 for Windows 64-bit platform exclusively. OpenGL 4.4 was chosen as the rendering API, as it was both familiar to the researcher and allowed for rapid prototyping of a graphical application.

### 3.1.1 External Dependencies

To ensure its timely completion, the application relies on several third-party libraries and tools for major functions that would not have been possible to develop from scratch within the project scope.

GLFW[[11]](#footnote-11) and Glad[[12]](#footnote-12) are used for window events and OpenGL instantiation respectively. For audio formatting and playback, OpenAL Soft[[13]](#footnote-13) is used in conjunction with Vorbis[[14]](#footnote-14). The graphical interface relies on the DearImGui[[15]](#footnote-15) and ImGuizmo[[16]](#footnote-16) projects, while screen reader and text-to-speech feedback is assisted with the Tolk Screen Reader Abstraction Library[[17]](#footnote-17). The entity component system is heavily reliant on ENTT[[18]](#footnote-18), while 2D physics and collision calculations are managed by Box2D[[19]](#footnote-19). The serialisation of data is handled through YAML-CPP[[20]](#footnote-20), with extra maths functions provided by GLM[[21]](#footnote-21) and debug logging via Spdlog[[22]](#footnote-22). Each of these libraries was chosen for their technical reliability, high level of documentation, and familiarity to the researcher.

## 3.2 What Makes a Game Engine Editor?

To determine how to create a blind accessible experience inside a game engine editor, it was necessary to first decide what collection of features would constitute an editor tool, and by extension, a game engine. The definition of game engines themselves can vary between studios, and not all make the distinction between game and framework code, but most long-term projects will utilise a method of managing and manipulating their assets in the form of a world editor (Gregory, 2018).

As such, the editor interfaces of several popular commercial game engines were examined for commonalities in design, including the previously mentioned versions of Unity and Unreal Engine. Alongside this were Godot Engine[[23]](#footnote-23), GameMaker 2[[24]](#footnote-24), Bitsy[[25]](#footnote-25), and Scratch[[26]](#footnote-26).

The first four of the tools listed had strikingly similar layouts, as they’re all intended to serve similar goals in creating sophisticated 2D and 3D games. Each had traditional interactive viewports, asset or game object hierarchies, debug consoles, and object properties windows.

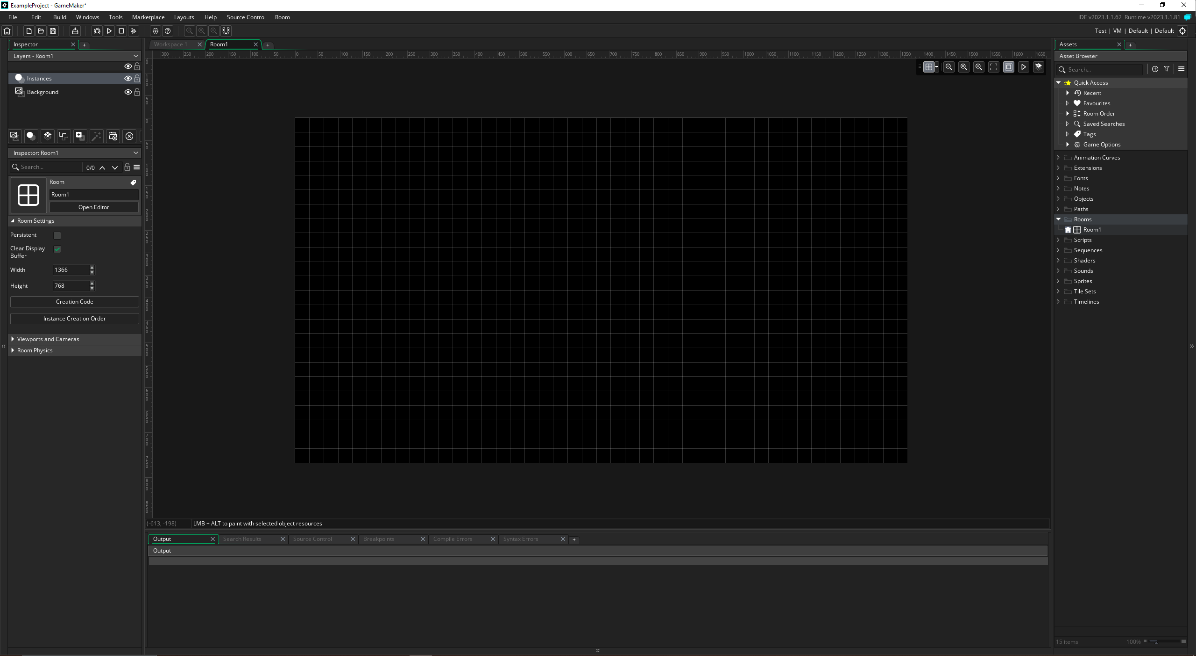


Figure : Example scene inside GameMaker 2.

Bitsy and Scratch had more unorthodox layouts, perhaps due to targeting web browser platforms. In both cases, functionality was divided into a handful of panels for coding, asset customisation, and placement into the scene.

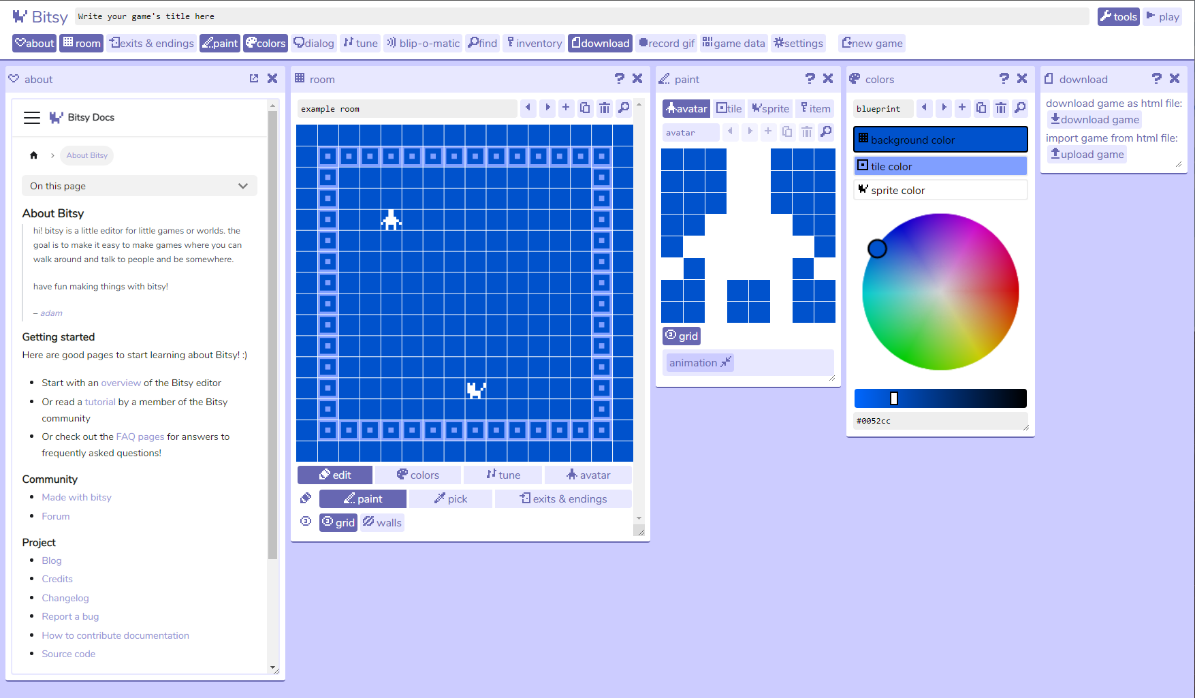


Figure : Default project inside Bitsy, a little editor for little web games.

The purpose of this additional research was to identify core features of both industry standard and hobbyist or educational tools, thereby establishing if a similar layout could be adapted for the artefact’s own world editor. Despite the wide variance in capabilities, a few broad functions were observed amongst all tools. Each of them possessed –

* Some method of creating or importing new game objects, typically via a menu.
* A viewport displaying a live render of the entire scene, where the user could directly apply transforms to these game objects in 2D or 3D space and observe their changes.
* The ability to simulate a runtime environment to test gameplay functions.

It was hypothesised that if the artefact focused on these principal functions when considering accessibility, then the resulting design may be scalable to larger game engine frameworks with a similar layout. This was important to consider, due to the growing desire amongst blind developers to have access to the same tools as sighted users (SightlessKombat, 2022). Therefore, this became the chosen approach, as the project’s main intention is to adapt existing design paradigms to create an interface that can be used intuitively by both sighted and sightless developers, rather than creating an entirely bespoke tool solely for the blind user.

Each tool that was examined also featured additional elements of varying complexity like native scripting, which is arguably an essential function of any game engine, but this was deemed unnecessary to include in the project. As there already exists tools to help visually impaired users in writing raw code (Coding Tech, 2018) and many individuals already do this successfully (Larson, 2017), it was decided the prototype would solely try to replicate the basic world editing functions of an engine. It was theorised that if a sightless person could achieve the three previously established basic functions of every game engine, then the integration of scripting support would not require huge overhauls to the user interface’s layout, as the user would presumably already know how to use an IDE to program.

From these findings, plans for the editor interface and the underlying engine framework could be established.

## 3.3 Designing an Accessible Editor Interface

As engines tend to be genre specific, it was decided the application would be geared towards creating 2D platformers. While it’s known that audio games encompass a wide variety of genres, from racing to shooting, one of its oldest and most popular is platformers (Rovithis, et al., 2014). Thus, it’s likely that a visually impaired user would have both experience playing such a game and an interest in working on one. The platformer genre was also chosen partly due to the restricted axes allowing for simple and clear spatial audio feedback when the player moved left or right.

After the editor concept had been stripped to its bare bones, the challenge remained of translating graphical information into an audio interface. The specific goals of the editor were then established. The user should be able to –

* Create and modify scene files.
* Place entities in 2D space from a list of pre-determined objects.
* Apply transforms and modify an object’s basic properties.
* Simulate the runtime environment.

Gameplay would exist in its most basic form, limited to allowing a playable character to jump across several stationary platforms. For the artefact to be considered successful, the editor must provide enough feedback for the blind user to understand every change they have made to the application when utilising these four functions.

### 3.3.1 Conceptualising the Editor Layout

The initial editor concept featured a single toolbar on the lefthand side with an enlarged button for each primary category of functions, such as project settings, a game object menu, a debug console, and so on. Interacting with any of the six buttons would prompt a sliding drawer style panel to appear with options relating to that category.

Users could then interact with the viewport to place objects, which possessed a limited 2D grid of squares much like GameMaker 2 or the ShadowRine Audible Mapper tool.

For more details about these initial concepts, see Appendix B.

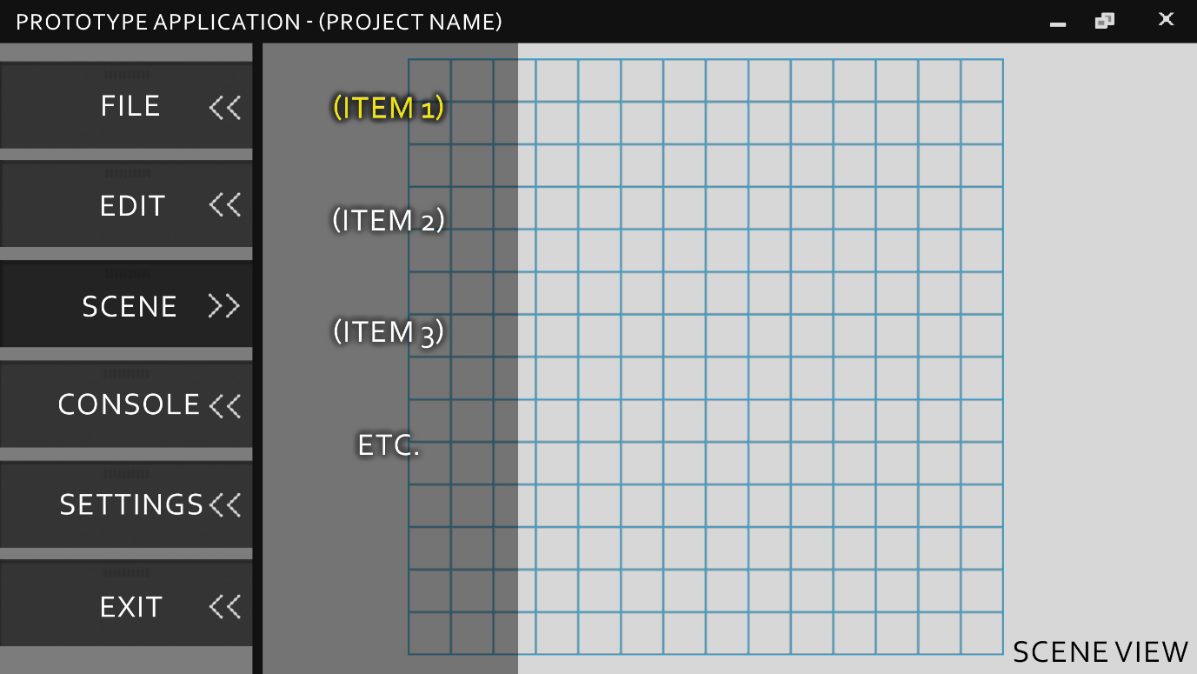


Figure : Mock-up of an early design for the editor tool. The initial idea was to condense all possible interactions into the same toolbar, organised by category.

Over the project’s lifetime, the design was refined to exclude features like the GUI-based debug console, as the application already generated a separate console window with screen reader accessible text, while also condensing user settings into a single panel called ‘PROJECT’.

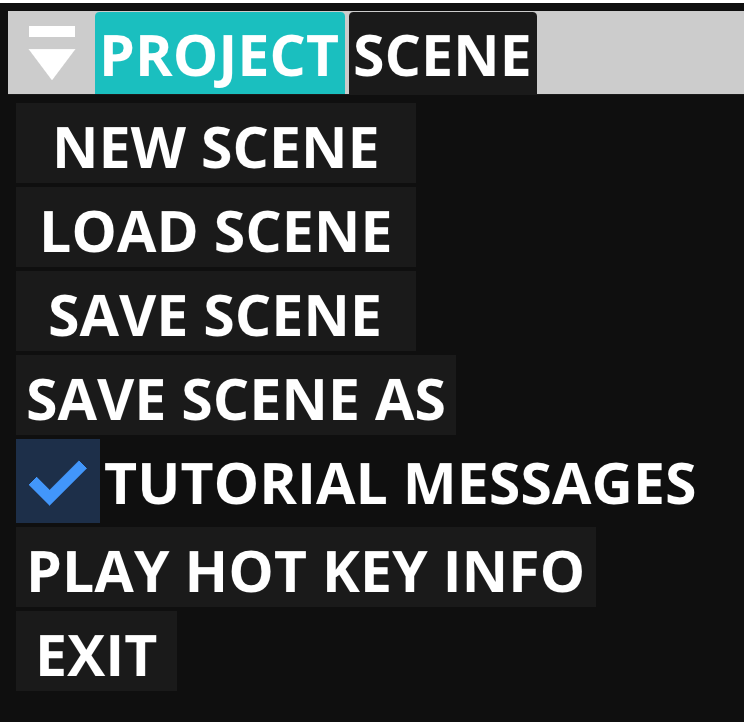


Figure : A view of the project panel which features a mixture of user preferences and application functions.

This stage of refinement also included other traditional elements common in more sophisticated engines, such as a hierarchy of existing scene objects (SCENE) and an entity properties panel (PROPERTIES).

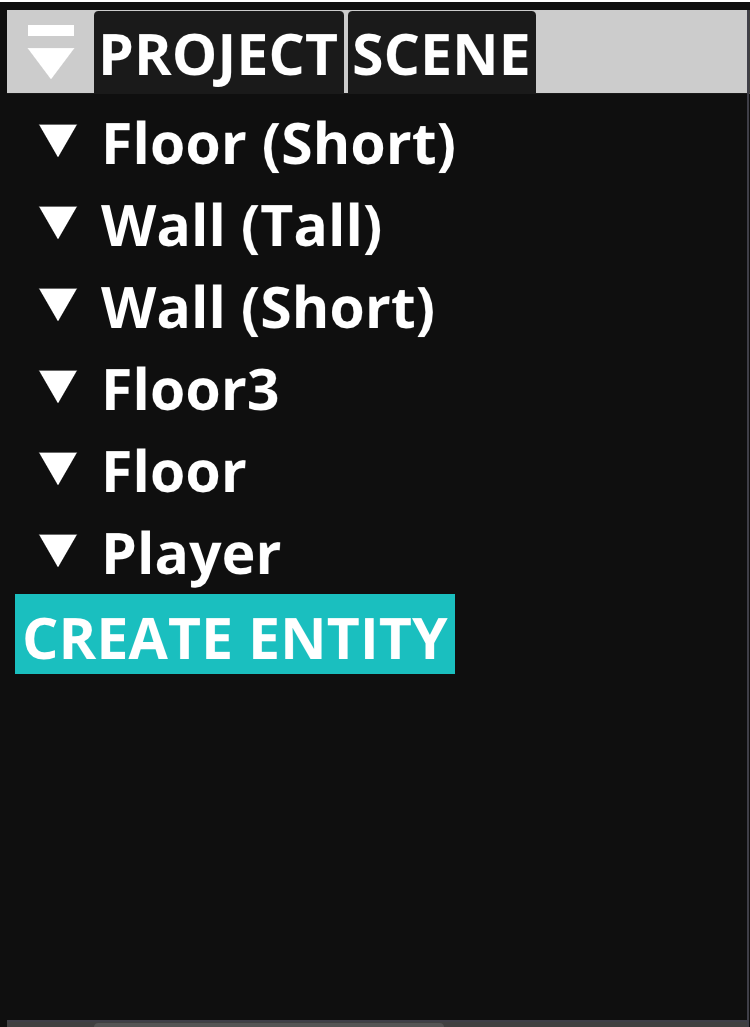


Figure : A view of the scene hierarchy with example objects and the 'CREATE ENTITY' button highlighted.

Entity properties were divided by component category into expandable dropdown menus, closed by default for easier navigation.

A screenshot of a video game menu

Description automatically generated with low confidence

Figure : A view of the entity properties panel when the default player character is selected. The sprite and character controller drop-down menus are enabled.

In addition, there also existed a popup menu for creating new entities from a pre-defined list of example objects, accessible via the scene hierarchy or a hotkey.

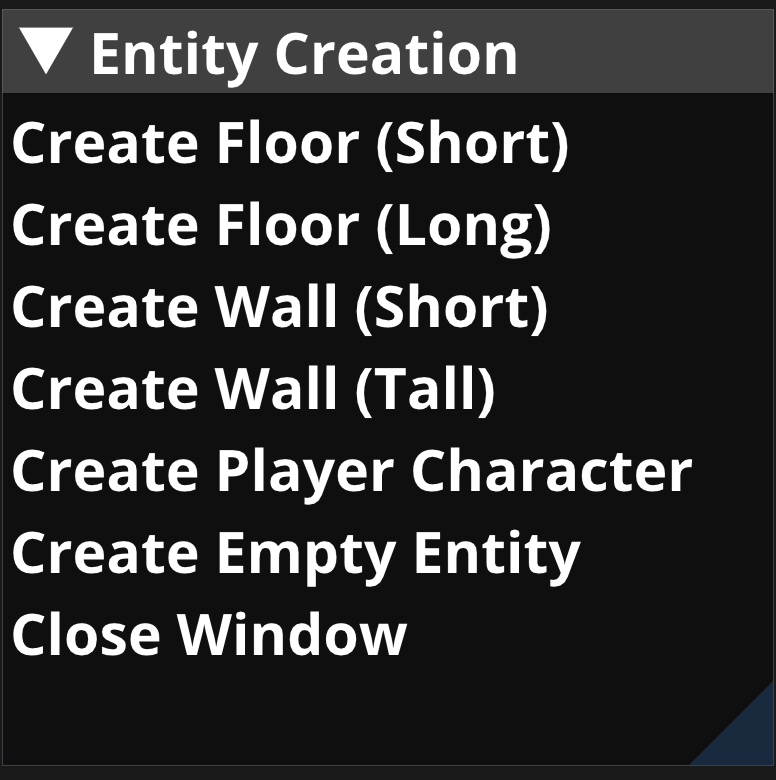


Figure : A view of the entity creation window, accessible from the scene hierarchy. It features a list of example entities which can be added to the scene.

The final interface design contains only these four panels and the viewport, plus a simple toolbar intended for initiating runtime functions, such as the ‘PLAY’ button. The limited grid concept was also eliminated, as the user would be able to snap objects to specific coordinates and have this position read aloud to them, allowing for more creativity and freedom in object placement.

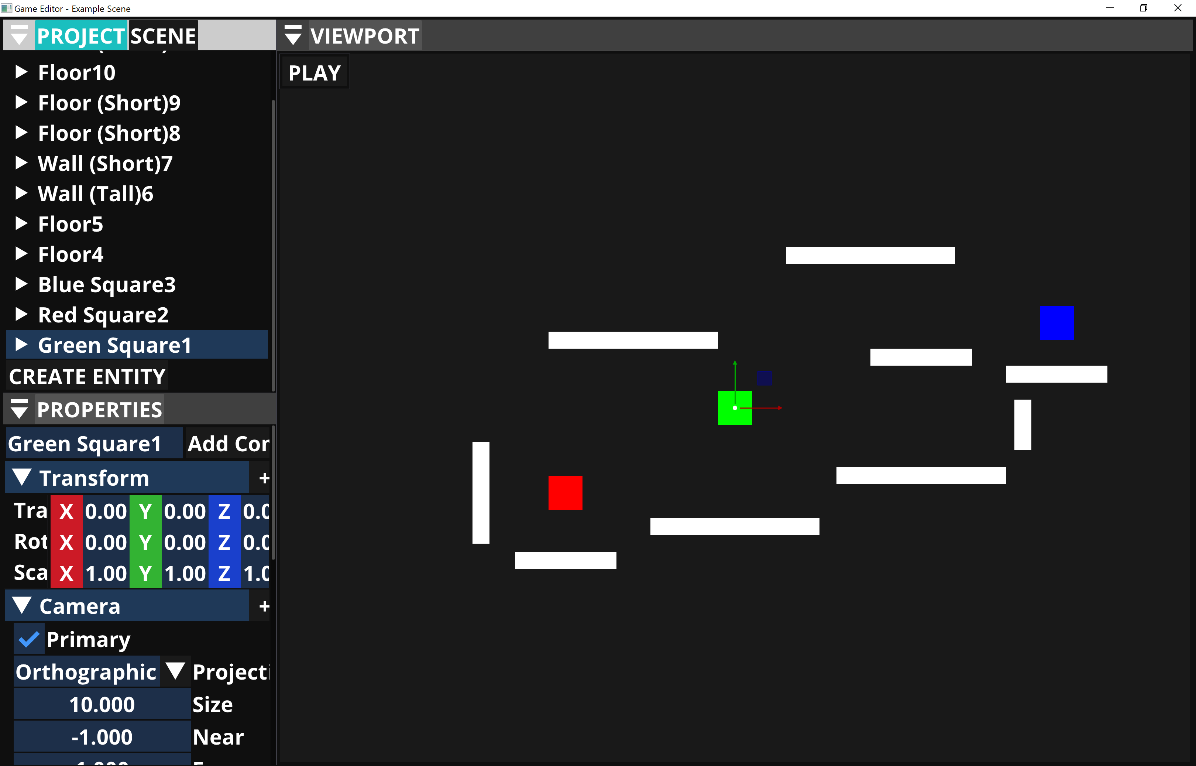


Figure : The final iteration of the editor, with several entities placed within the scene. Takes the form of a more traditional editor GUI with separate, simultaneously active panels.

### 3.3.2 Structuring the Editor for Accessibility

The project adapted several techniques from researched literature where appropriate, but its design generally followed one principle – for each input action that results in a visually observable change to the application state, some corresponding audio output must occur.

In practice, this meant simplifying all editor functions into a handful of menus or tabs. Navigation of all menu elements could be performed entirely through the keyboard using the arrow keys. For repetitive or fast-paced user actions, such as changing tab or selecting a button, a specific sound effect would play alongside possible speech output. Efforts were also made to ensure every type of action had a unique corresponding sound effect to minimise confusion from the user.

Screen reader support was also integrated to detect any running third-party software and utilise it to output all text elements. Alternatively, if no active screen reader could be found, the editor would rely on the built-in Windows text-to-speech API to continue outputting the same information. Although, this feature would be experimental, as the application would be designed to prioritise using a screen reader. The rationale behind this was the customisability that a screen reader provides by allowing the user to perform actions like altering the speed of speech or cycling through a message output history. Additionally, a visually impaired user is likely to already be used to the flow of using a specific screen reader for other computer-related tasks, therefore its inclusion in the project is more convenient than not. This also meant less work would be required on the part of the editor to provide the same utilities.

Although each design choice was made with the assumption that the user had no usable sight whatsoever, some consideration was given to those with low vision. The font for text elements was set to a much larger size of 78 points by default, including buttons and panel windows. Additionally, the use of custom textures on UI elements or game objects was purposefully omitted, due to the visual noise they may create. Instead, every item features a solid colour background, designed to starkly contrast the colours of surrounding objects.

Despite most accessibility considerations being based on previous examples, the unique challenge of transcribing a world editor’s visual feedback into an audio format required the creation of two more features to ensure the viewport was fully functional. These were directional sound cues, and descriptions of the spatial relationships between entities.

Loosely based on the mouse coordinates implementation from the Shadowrine Audible Mapper, the directional cues functioned by emitting a 3D spatial sound effect when the user has selected an entity and begun translating it to a different location. The point of origin would be the entity’s starting position, and effect would shift up or down in pitch depending on how far away the user had moved the entity. Each key press also moved the entity by a fixed amount, snapping it to a specific coordinate as if it were part of a grid.



Figure : Visual representation of the directional audio cues feature. The red circle marks the Green Square entity's original position, with the extending white line indicating the direction it has been moved. The yellow lines represent the audio’s direction.

To complement this, a method was also devised to allow the user to hear feedback about where different entities were placed in relation to each other. This was partly inspired by the feedback from the study about accessible digital maps (Hasan and Gjøsæter, 2021). If at least two entities were selected, information would be given about how they overlapped based on data provided by their corresponding box collider components. The user would have to perform this manually by pressing a hotkey.

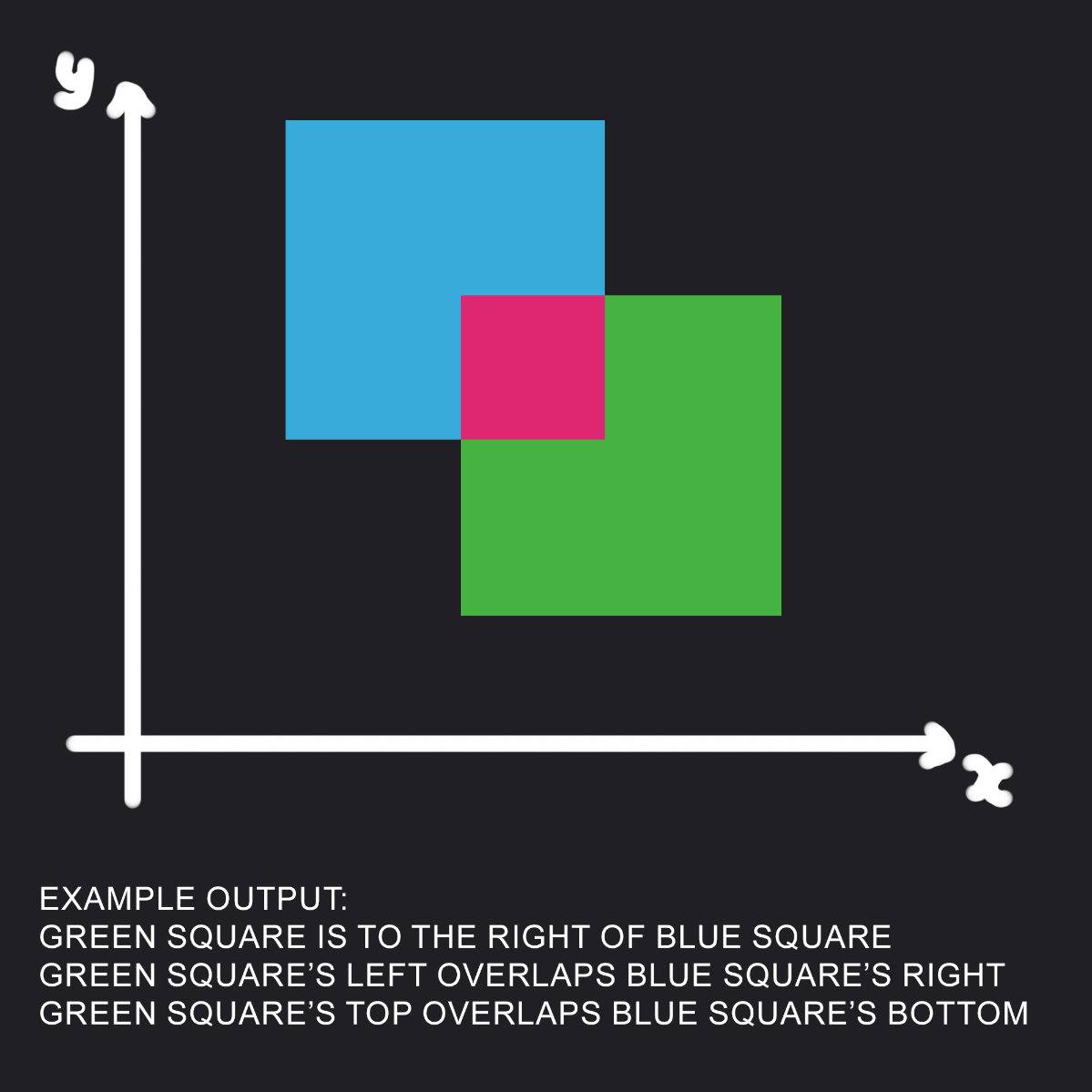


Figure : Visual representation of feedback given when two entities overlap. The application checks the distance in both axes of each entity from the centre and determines which edges of their box colliders are intersecting.

### 3.3.3 Accessibility at Runtime

As it was important to ensure that the editor’s demo scene could also be play tested without sight, two types of scripted game objects were created – a static obstacle, which had no additional functionality by itself, and a controllable character.

Movement of the character was restricted to walking left and right, with the ability to jump up by a fixed amount. Several audio cues were integrated into the default character controller that corresponded to in-game events. These included a sound effect for jumping, landing on platforms, and a 3D directional effect when the player approached a ledge which could be jumped from. There was also spoken feedback for when the player fell into an inaccessible area – such as too far below the world origin – and a mechanic which automatically reset their position to the level’s beginning in such events.

## 3.4 Game Engine Architecture

Careful consideration went into the framework architecture that underpinned the whole artefact, and it was decided that basing its design on pre-existing open-source software would provide the most efficient and robust application in the project timeframe. The general class structure was strongly based on Hazel Engine[[27]](#footnote-27), which was chosen due to its in-depth online video documentation and helpful commentary provided by the engineer about the development process (Chernikov, 2018). The artefact also borrows concepts from Game Engine Architecture (Gregory, 2018) to ensure the class layouts follow best programming and design practices, especially regarding the audio and input systems.

The framework comprises of several smaller systems, all of which are based on the Hazel Engine’s implementation to varying degrees – rendering, audio, user input, entity and scene management, physics, and serialisation of user data. The implementations of the renderer, the entity component system, and the serialisation functions are virtually unchanged from the source. The remaining systems and classes contain a small to moderate number of modifications to the original code, except for input, audio, and UI, which have been heavily reconfigured.

The artefact uses a layer stack to segment functionality into categories by intended purpose. The code base is also split across two Visual Studio projects – one which contains engine functions and the editor layer, and another for the gameplay runtime layer. The purpose of this segregation is to make a clear distinction between what classes belong to the engine and what should belong to the user’s project, despite the current lack of custom scripting support.

The Application class serves as an initialisation point for the entire project. Manager classes for each individual system are called here before the app runtime loop begins, and an event dispatcher is created in each of these classes to handle custom created event call-backs. This event handler class is reused throughout the entire application, as it allows for easy separation of call-back checks – such as with user input – between different states of the program.

While the Application class is responsible for initialising broad systems, the Editor Layer class more closely manages both the GUI and AUI, as well as handling changes related to the scene state.

Even though the artefact was able to closely adapt the structure of Hazel Engine to suit the project’s needs, the following three systems were significantly modified or entirely invented in the pursuit of creating a blind accessible editor.

### 3.4.1 User Input and the GUI

Input is purposely abstracted to be independent of platform, despite the current application being only available for Windows. This is for the purpose of code scalability.

The project makes use of ImGui to render menu elements. Due to specific requirements of the proposed control scheme, several changes needed to be made to override or adapt built-in ImGui functions. The menu selection screen was disabled in favour of a single button to cycle through active menu windows, as the default implementation required the use of multiple simultaneous key presses.

Additional checks had to be placed on the editor GUI layer to track the state of the last active menu elements, with consideration for the fact that the user could choose to navigate using the keyboard or the mouse at any given moment.

Due to the lack of default screen reader support in ImGui, several overriding functions had to be written to both render a specific type of GUI element, process its name and content in a coherent and digestible manner, then output it to the user via speech.

### 3.4.2 Audio Management

The audio classes are partly based on another implementation by Yan Chernikov[[28]](#footnote-28), with some cues taken from GEA. In traditional game engine implementations, audio is normally a standalone engine (Gregory, 2018), but the artefact is small enough that all the audio classes can be integrated directly into the framework.

A basic event listener class specifically for managing audio events was integrated alongside the original code. This class also stored a list of audio source files that were used exclusively by the editor, to create further segregation between gameplay and engine code. For ease of use, a static function made it possible to trigger an audio event for a specific editor sound effect from anywhere else in the program.

### 3.4.3 Screen Reader Support

Unlike other systems, the Screen Reader Logger class is not based on previous engine implementations, but still serves necessary functions for ensuring that the application can communicate with third-party screen reading software. Although the Tolk library provides much assistance in this endeavour, the logger class serves to encapsulate functions from Tolk and connect them with the broader artefact framework.

The logger stores a list of messages to send to the screen reader, ordered by a Boolean flag to indicate if they’re a high priority. An event handler then checks if a message has entered the list and begins outputting it if no other messages are currently playing. The reason this is done instead of just immediately sending the text to the screen reader is in the event the screen reader overwrites it with unrelated messages and it is never heard, such as the last keystroke. The priority flag ensures the message is always sent and its output complete before removing it from the message backlog.

This also works well in scenarios where screen reader detection has failed, and the editor is forced to rely on Windows Speech API for output.

## 3.5 Flow of User Experience

Navigation through the editor is intended to flow in a linear manner from initial launch. The default focused GUI panel will be the application settings menu, to allow the user to load a scene file or create a new one. From there, they can press the appropriate hotkey to cycle forward through the linear list of active GUI panels until they arrive at the menu that contains their desired functions. All navigation proceeds in this manner, with two exceptions – when a new entity is created, or when the game runtime begins. In both cases, the focused panel automatically becomes the viewport, so the user can immediately begin testing or manipulating their objects.

### 3.5.1 Application Control Scheme

Due to the relative complexity of the artefact’s possible user interactions, several key pieces of functionality have been bound to hotkeys. This is intended to minimise the number of menu elements needed within the UI.

Efforts have been made to provide a choice between keyboard or mouse input where appropriate, and hotkeys were chosen on the basis that the most used ones were relatively close together and could be easily accessed.

Table : The list of application controls mapped to their functions.

|  |  |  |
| --- | --- | --- |
| **Action** | **Keyboard Input** | **Mouse Input** |
| Select Menu Element | Arrow Keys | Left Mouse Click |
| Select Entity | Space Bar | Left Mouse Click |
| Duplicate Entity | D | N/A |
| Delete Entity | S | N/A |
| Move Entity | Arrow Keys | Mouse Move |
| Select Next GUI Panel | Left Control | Left Mouse Click |
| Pan Editor Camera | Arrow Keys | N/A |
| Zoom Camera | Square Bracket Keys | Mouse Scroll Wheel |
| Toggle Camera Lock on Entities | Caps Lock | N/A |
| Toggle Entity Multiselect Mode | M | N/A |
| Update Active Screen Reader | Z | N/A |
| Print Component Data of Selected Entities | O | N/A |
| Print Spatial Relationship Data of Selected Entities | P | N/A |
| Move Character Controller | WASD | N/A |
| Character Jump | Space Bar | N/A |

## 3.6 Evaluation Planning and Process

The project utilised a qualitative testing approach to assess the effectiveness of the artefact. This was conducted as a series of user testing sessions with a small group of participants who would fit the target audience for the application. As the primary goal of the project is improving usability for a specific group of people, exposing it to that audience is the most effective method of identifying key issues in design that may have been previously overlooked (Chisnell and Rubin, 2008). Testers were chosen based on their proficiency with computer games technology and experience with a range of visual impairments, to ensure user feedback was both meaningful and varied.

A total of 5 individual usability sessions were planned, each lasting between 30 and 60 minutes. For the sake of participants’ time and convenience, these were conducted remotely across Microsoft Teams and Discord. Instead of providing a questionnaire sheet directly, each meeting took the form of a structured discussion where open-ended questions were read aloud, and all responses were given verbally. Each meeting was also recorded for later review.

The first 10 minutes of each session was used to review basic background information with the participant and explain the goals of the research project. For another 15 to 20 minutes, participants would follow the tutorial tasks and use the provided application to try and make a simple level for a 2D platforming game. While working, users were asked to share their computer screen and narrate aloud their thought process at each step. The remaining session time was spent conducting a post-test debriefing interview, which could last anywhere between 5 minutes and 30 minutes depending on how much feedback the user wished to give.

For more details about the user testing process, see Appendix C.

### 3.6.1 Designing User Tasks

As the artefact encapsulates a wide array of features to achieve its goal, tasks were to be broken down by what element of the accessibility-oriented design they are meant to test. However, to ensure minimal confusion, the task list would also serve as an introduction to the general flow of using the editor. Considering these requirements, the following questionnaire was devised and given to test participants.

Based on the information given to you from the tutorial sequences and additional editor feedback, attempt the following tasks:

1. Can you find a way to add a new entity to the scene?
2. While in the viewport panel, move this entity to a location of your choosing, then output its entity coordinates. Is the entity positioned where you expected it to be?
3. Navigate to the scene hierarchy panel and enable entity multiselect mode. Select at least one other object and output the relationship information. Are all entities in the positions you expected? Are there entities that collide with anything they shouldn’t?
4. Disable multiselect mode, then duplicate any entity of your choosing, except for the player. Afterwards, navigate to the properties panel and modify a component of your choosing. Can you tell what has changed in the scene?
5. Continue adding, deleting, and modifying entities until you are happy with the scene. When finished, navigate to the viewport toolbar, and press the ‘PLAY’ button. The editor will start simulating gameplay and you will automatically assume control of the player. Can you make the player entity jump across the platforms you added?
6. When you are finished with playtesting, navigate back to the viewport toolbar, and press the ‘STOP’ button. Then navigate to the application settings panel and save your work to a scene file.

## 3.7 Summary

This chapter detailed the process of designing, developing, and evaluating the artefact. The Results chapter will state the outcome of the user testing sessions described previously.

# 4. Results

This chapter collates the results of the user testing sessions through a combination of feedback given and additional observations made by the researcher. The effectiveness of the artefact was determined by the observable speed and ease at which users could manipulate entities to create a basic scene and finish the list of prescribed tasks, as well as the testers’ own views of the same experience.

Of the five usability sessions planned, only four were fully concluded due to technical difficulties stemming from an incompatibility between one participant’s hardware and OpenGL – the graphical API used by the application. The renderer failed to fully initialise, leading to unpredictable behaviour and an unworkable application. Consequently, only the feedback from the remaining four users is considered.

## 4.1 Response from Users

Test results are broken down into tables based on category and associated with each participant by a pseudonym. For brevity’s sake, several answers are paraphrased based on responses provided in interview recordings and written in plain text, with care given to preserve the integrity of the original message. Any direct quotes from users are listed alongside these statements in double quotation marks.

For some more detailed answers, transcripts of the full meetings for P1 and P3 are available as Appendices D and E respectively, with identifying information redacted.

Background Interview Questions:

Table : Answers to user testing background interview, question one.

|  |  |
| --- | --- |
| Q1: What is your interest in computer games? Do you have prior experience in computer games development? | |
| P1: | I’m not a developer because I don’t think I’m very good at coding, but I grew up playing video games. Currently, I do consultation work for accessibility in games and applications, including virtual reality and mixed reality software. I’ve worked with a few developers and given presentations at conferences on accessibility. |
| P2: | I am a developer, a gamer, and a streamer. I studied computer science and have previously worked professionally in software developer roles. I’ve since pivoted towards accessibility management and consulting, so programming and game development are mainly hobbies now. |
| P3: | I am a gamer and a streamer. I’ve worked with game development studios before and consulted about improving accessibility in games.  “I've never been able to [use a game engine]. It's not that I haven't wanted to. It's just that the tools don't give me those accessibility options to allow me to even consider it.” |
| P4: | I’m not a developer and I have no experience making games, but I play them. I find the market is limited because there’s not many people taking the time to design blind accessible games. |

Table : Answers to user testing background interview, question two.

|  |  |
| --- | --- |
| Q2: To what extent does your vision loss affect your ability to use computers, and what tools or strategies do you normally use to assist you? | |
| P1: | I have some usable vision. It depends really on the type of game whether I would have trouble with it or not. A lot of text and a small user interface can be some of the most difficult.  Depending on whether I'm using a computer or a mobile device, I might use magnification. I might use a screen reader like NVDA. I might use a combination of all of those. |
| P2: | I am totally blind. I have a very small amount of light perception, but I use a screen reader like NVDA for everything. I also have a braille display for my computer. |
| P3: | I have no sight whatsoever. I use a screen reader, mainly NVDA for desktop. I do have access to a braille display, but I don’t normally use it. |
| P4: | I don’t have any functional vision except for light perception, so all the feedback I get must be audio based.  I use JAWS[[29]](#footnote-29) screen reader to help me. I also prefer navigating with simple movements like swipe gestures. |

Table : Answers to user testing background interview, question three.

|  |  |
| --- | --- |
| Q3: What kinds of games do you enjoy playing or making? Does the accessibility of a particular game or game genre factor into your preferences? | |
| P1: | For games, a lot of them until recently haven't had any accessibility features in them, so you just play what you can. I'll take advantage if there’s the option to make the text bigger. Or even some games that have text-to-speech support or screen reader support being built in, that can be helpful. If things are small or blend in, that can be difficult.  “I love first person shooters, but I really, really don't care for the more military style because by the time you see what's going on or figure out where somebody is, you're already dead. So, I tend to prefer more retro shooter style games.” |
| P2: | I really enjoy adventure games and RPGs, but those are often unfortunately inaccessible, so I’m resigned to watching online playthroughs. I’ll play whatever I can get my hands on, but it’s normally using accessibility mods created by other players and not the developers themselves.  “I’ve always been very interested in making an accessible engine that can also make accessible games or levels, so this project is right up my alley.” |
| P3: | “I'm not going to list what I would like to play because there are so many, but so many of those genres aren't accessible.”  I like action, arcade, and adventure games. I also enjoy online multiplayer games with a collaborative aspect – working in a team with friends is ideal. |
| P4: | I like to play audio games, mobile games, card games, and board games like Monopoly. However, I don’t just want to play games only for blind people, I want to play with sighted people too, because I like roleplaying with others.  In many cases it’s not possible, but I’d like to play games that are accessible to everybody. |

Post-Session Interview Questions:

Table : Answers to user test session review, question one.

|  |  |
| --- | --- |
| Q1: How did you find the process of navigating the menus? Was the audio feedback here helpful or unhelpful, and why? | |
| P1: | Overall, once I understood the flow of it, it was easy.  The only thing that threw me off is that I'm used to the CTRL key being something different, so I would hit that when I didn't mean to. Other than that, it was easy to get around, even without necessarily knowing where something was.  Once I learned what each one of these different panels were, I was easily able to just figure out where to go or what I needed to do. |
| P2: | It took me a moment to figure out some of the key bindings, but it’s very clear how to use the tool once you get the hang of where everything is. |
| P3: | “I felt like it was a little unconventional, but I think that was mostly due to the key buttons being used and the lack of a standard menu system.”  It took a while for me to get used to how the panels worked, but once I understood where everything was, it was quite smooth. |
| P4: | It’s very detailed in the sense that every panel has a tutorial, which I liked. At the same time, there were parts where I wasn’t sure what I was supposed to do or how the editor panels linked together.  Maybe it’s because I’m not a game designer. |

Table : Answers to user test session review, question two.

|  |  |
| --- | --- |
| Q2: How did you find the process of adding entities and manipulating their properties? Do you feel the editor provided you appropriate feedback to understand what you were doing? | |
| P1: | Overall, I think that was straightforward.  The only thing I would change is making sure that I can always read the value when I'm in an edit field, if I'm renaming something, or changing the value, giving feedback for those changes in real time instead of afterwards. |
| P2: | It was good. You do need a bit of spatial awareness, but I found the process to be painless. |
| P3: | “Being able to directly manipulate the properties was actually quite fun, especially once I understood the intricacies, like the centre of an object is where the origin of it is.”  I spent a lot of time in the properties panel because I found the viewport navigation a tad confusing, particularly regarding how long to hold the keys down to move an entity a certain distance. |
| P4: | It was a little complicated, especially the properties panel. Sometimes it wasn’t clear what an entity component did or how changing a property affected things. |

Table : Answers to user test session review, question three.

|  |  |
| --- | --- |
| Q3: What are your thoughts on specific features to improve spatial awareness, like the pitch shifted audio when moving entities, or the descriptions of entity relationships? | |
| P1: | Those two features were helpful. The pitch changing is especially helpful because that way you don't have to check your position all the time. You can hear it and know you’re roughly in the vicinity of where you want to be, then you can check the coordinates to be sure.  I also think having a collision sound effect would be helpful. Some cue that lets you know you're overlapping with another object, or about to contact it. Otherwise, it was intuitive. |
| P2: | I don’t think I completely understand those features yet. The pitch and the panning let me know where I’m moving in the viewport overall, which was good. Manually selecting entities to hear their relationships was a bit awkward.  What I think is missing is some sort of automated sound for where I am in relation to other objects within the viewport. Without it, it’s hard to get a good picture of the whole scene. |
| P3: | “The inversion was a bit strange because I'm used to things moving the way they're meant to move. So, if you move sort of to the right, things move to the right, if that makes sense.”  I’m not used to it currently, but others might be. I think it’s possible to get disoriented. |
| P4: | I encountered a bug in the viewport that made it hard to move entities with arrow keys. However, I think if you have good spatial awareness, then it’s easy to know how things are laid out. If I’m just hearing the coordinates, I don’t really know what’s going on. |

Table : Answers to user test session review, question four.

|  |  |
| --- | --- |
| Q4: How did you find playing through the demo level you created? Did the player interact with your scene the way you expected? | |
| P1: | Sort of, but the player was a little uncooperative with jumping. I think some of that comes from using the screen reader.  I'm not sure how you would address that, because if you're playing a mainstream game, you’re normally using a controller to control a game and then you're using the screen reader via the keyboard, that way they don't seem to conflict as much. But if you’re using a screen reader and then you're using the keyboard to also control the game, depending on the mode or how the screen reader is set up, they can conflict a lot. |
| P2: | I honestly got lost quite a bit. I could consistently find one wall I’d placed, but then the other wouldn’t be where I thought I’d put it. The negative axes confused me because sometimes objects ended up not on the level that I wanted them. |
| P3: | “I feel like that was arguably the point of the system as it stands, because you have no sense of location where you are, what you're running into, where the things are clipping in practice, any of that.  So, it's essentially a thing that you would only be able to do with sighted assistance. Which arguably defeats the objective.” |
| P4: | In most audio games, the player has footsteps while moving, so I was expecting that kind of sound effect. Without it, I wasn’t entirely sure where I was walking. However, I like that the editor tells you when you’re at the edge of the floor and the player is resetting. |

Table : Answers to user test session review, question five.

|  |  |
| --- | --- |
| Q5: What are your thoughts on this project overall? Would you say your experience has been a positive or negative one? | |
| P1: | I thought it was a cool experience.  I’m always interested in audio games and giving a blind audience something to express their creativity with, especially if they can’t code. I think being able to build your own games or levels in an editor format like this, something akin to Mario Maker, is a great idea. |
| P2: | This is something I’d like to see continued and improved in future. I had a lot of fun playing with it. As a blind coder, you must work from scratch in a console. You don’t have anything like this interface available to you.  “What you’re doing is filling a niche. A lot of people would like this, other people that I know who would feel helped by this kind of game development. Definitely something I’ve enjoyed so far.” |
| P3: | “I feel like it's solid. I feel like it's just a lot of information for people, which is okay … in a real-life situation, you would have documentation as well.”  I’ve had a positive experience. It’s been a lot of fun to try out some of the mathematical elements of game development. I’ve always wanted to make prototypes, and this is a great start. |
| P4: | I’ve had a positive experience. Designing something accessible is a big problem, so this is a great idea. Even if some things about the project could be better, it’s a learning experience for everybody. |

Table : Answers to user test session review, question six.

|  |  |
| --- | --- |
| Q6: Is there anything you would change about this project, or are there any additional comments you’d like to make? What would you suggest and why? | |
| P1: | Not much, other than what I mentioned previously. It would be useful to be able to cycle through active entities from the viewport, not just the scene hierarchy. It might also be nice to listen to the tutorial messages one line at a time and go through a backlog of messages for ones you missed. |
| P2: | “Overall, I like what I’m seeing here. There’s room for improvement, but it’s a solid start for an accessible editor.” |
| P3: | “There’s all sorts of stuff that I'm picturing in my head and being able to even get close to realizing that was brilliant.  I can see the massive potential for it. And even though I can't play the demo level, I can't wait to see where this goes next because I've been able to play around and test it and I understand the principles of it.” |
| P4: | I think the editor is simple enough to use but having more keyboard shortcuts to do different things would be good. |

## 4.2 Additional Observations

In addition to direct feedback from users, some patterns were noticed in the behaviour across testers. All participants tended to repeatedly hit specific keys, such as left control and left shift, even before they’d been informed of their functions within the editor.

Each tester asked for assistance from the researcher repeatedly through the sessions. P1 and P3 were roughly on par and considered themselves comfortable with the application after approximately 35 minutes, enough to stop asking for clarification about key features and complete the list of prescribed tasks. P2 was the fastest to achieve the same result at around 20 minutes. P4 was the slowest, taking over 50 minutes to finish.

## 4.3 Summary

This chapter described the results of the evaluation process for the project. The Discussion chapter will assess how effectively the artefact performs as a solution to the research question based on these findings.

# 5. Discussion

With the results gathered from the evaluation phase, enough data is present to begin answering the research question – are the audio and UI design techniques implemented in this study an effective solution for creating a blind-accessible level editor? Alternatively, what is currently holding the project back from achieving these goals?

## 5.1 Analysis of User Experience

All users took some amount of time to adjust to the flow of using the tool before they stated they felt comfortable with it. The length of time varied between testers, and this appears to correlate with the individual’s familiarity with game development tools. P2, who had significant prior programming experience, adapted the fasted to using the interface. In contrast, P4, who had only ever played games, took the longest to feel comfortable with the same concepts.

This was arguably unavoidable, because a lack of knowledge or comfort with common game development paradigms will result in a steeper learning curve regardless of the individual tool’s design. Despite this, when questioned about their overall thoughts, all testers described their experiences as positive and considered the editor tool to be reasonably simple to use.

However, feedback on more specific elements was mixed.

### 5.1.1 Menu and Panel Navigation

In most cases, testers considered panel navigation to be straightforward, albeit unorthodox and requiring some time to adjust. Due to the implementation of tutorial messages for each major function, the amount of information a user received upon loading into the editor for the first time was frequently noted to be overwhelming. Additionally, the mismatch between editor navigation controls and character controls at runtime often caused confusion. In future developments, it would be wise to ensure all movement controls are consistent and tutorial messages are broken down into smaller chunks and read at more frequent intervals. Furthermore, the integration of small features like the ability to repeat already read messages and a centralised menu or panel selection screen would improve usability.

Despite efforts to the contrary, the use of third-party screen readers led to several control conflicts. The specific conflicts depended on the user’s custom key bindings, but in many cases, the editor’s hotkeys for menu navigation – such as left CTRL – were already registered as modifier keys in the participants’ screen reader settings. Frequently, users would hit these keys instinctively to perform functions they were already familiar with, such as silencing the screen reader. However, this would also cause the corresponding input events to occur inside the editor as well, leading to unpredictable behaviour.

On a similar note, it is unknown for certain what caused the viewport navigation bug conflict encountered by P4. Notably, they were the only participant using JAWS and not NVDA – the latter of which more consistently provided accurate feedback. However, it is possible the combined use of left CTRL and arrow keys was registered by default as a text navigation command, thereby overriding the editor’s viewport controls and rendering a key part of the application unusable with JAWS (Freedom Scientific, 2023).

As suggested by a couple of participants, one method to get around the issue of screen reader conflicts would be forgoing the use of third-party screen readers altogether and relying solely on built-in text-to-speech, like the Sable Engine. This would avoid having to keep track of changes in external programs and prevent messages from overwriting each other. However, this would likely also require the integration of additional functions to control the speech rate or message history that a screen reader already provides. Additionally, if solely relying on the Windows Speech API for audio feedback, an alternative text-to-speech API or package may need to be considered if the application was ported to other platforms.

An alternative suggestion would be to set up gamepad controller support to function within the editor, so screen readers could still be used via keyboard controls, but all UI navigation would be done with a game controller. This would likely work well with minimal configurations to the overall flow of UX, due to the limited control scheme already in use.

However, the artefact had some success in its AUI design. One participant noted feeling a sense of reward in the choice of sound effects that gave the editor a stylised retro feel. This feedback provided a sense of reward, akin to solving a puzzle or a level in a video game, thereby prompting the user to continue exploring the tool and building an understanding of its functions.

### 5.1.2 Spatial Scene Navigation

The two main techniques to improve spatial navigation in the viewport were among the most experimental and evoked divisive feedback from testers. Some users found the use of a directional cue to signify movement of an entity to be highly intuitive, especially when combined with distance-based pitch shifting, although two participants stated they were not entirely sure of its purpose.

However, all testers who described themselves as totally blind commonly noted losing track of their objects after some time and eventually felt unsure of the layout of the general scene. This can likely be attributed to the viewport navigation cues being based on the current selection(s) in isolation, rather than considering the structure and relationships of all objects in the scene at a given time. The editor successfully informs the user how far they’ve moved and where to, but it does not automatically tell them what new objects they’re about to encounter in the process.

Manual multi-selection does function to analyse the spatial relationships between scene objects, but this does not occur until after the user has stopped moving and gone through the steps of manually identifying the objects they want to verify. This implementation was generally thought to be clunky by testers, as the lack of immediate feedback when translating entities that are close together means one must memorise each object’s individual position so they can be checked later. This, in tandem with long strings of coordinate data being regularly read aloud resulted in undue strain on the user.

A superior solution may be to combine the two features into one. A system that could automatically track and filter active objects in the scene, then play a directional sound effect when approaching or overlapping it would mean the user could receive immediate spatial relationship data about multiple objects simultaneously without having to get rid of their current selection or change menus.

A picture containing line, colorfulness, plot, diagram

Description automatically generated

Figure : Visual representation of raycasting on a 2D grid (Vandevenne, 2020).

One possible technique to achieve this is with raycasting. Since entities are snapped to incremental positions and generally not freely transformed, such checks would not be performed every frame, and it may be possible to draw rays from the entity’s current position across the XY plane in eight directions until another entity is hit (Vandevenne, 2020). From there, the application could inform the user of all the objects and their respective data surrounding their current selection from just one key input.

### 5.1.3 Playability of the Demo Scene

The degree to which each participant was able to playtest their level varied depending on their level of sight. P1, who had low vision, was able to make the player jump across gaps and navigate placed platforms due to the high contrast colours of each object type. However, each of the remaining three participants, who described themselves as totally blind, struggled to identify where the player was and judge the distance between platforms accurately, leading to them regularly falling into the void.

This was likely due to limited audio feedback provided by the default character controller component. The current implementation considers only four primary movement events – jumping, landing, approaching a ledge, and falling off a ledge. However, there was a failure to consider other contexts such as actively walking, rising off the ground, or falling towards the ground. Additionally, the effect to designate that a player is close to a ledge does not provide any further information about the distance to surrounding platform ledges, making it much more difficult to judge if the player can jump the gap.

In future, it would be better to consider the incorporation of sound effects when *any* sort of visual or gameplay change is occurring, rather than just the ones that happen because of user input.

## 5.2 Effectiveness of the Solution

In terms of answering the research question, the artefact succeeds somewhat in being a blind-accessible graphical game development tool, created using custom audio feedback and minimalist UI design. Certain techniques adapted from focused literature, such as simplified menu navigation, proved among the most effective. Others adapted from broader literature, such as the strategies for improving spatial navigation, produced more mixed results. Despite these findings, there are more factors to consider that may have influenced the project’s ability to answer the research question.

### 5.2.1 Limitations of the Study

One potential flaw of this study may lie in the phrasing of the research question itself. It was shown that having minimal GUI elements and a large amount of speech or ambient audio feedback was effective with users. However, the study does not actively consider the effectiveness or importance of tactile feedback, even though it was found to have strong merit as a blind accessibility technique in the literature review.

Evidence of this is demonstrated in the test results. P2 was the only participant to actively use a braille display, which noticeably put them at an advantage in certain instances of the test session, even though they described themselves as totally blind. When audio feedback alone failed to be understood, they were able to review the same messages on their braille display and process the information where other testers could not. In addition to their prior software development experience, this extra mode of output may have contributed to them adapting to the flow of using the editor much faster than other participants.

This suggests that although audio and UI design play a strong role, these considerations may not be enough by themselves to achieve the research goal in odd cases where a lot of application feedback is being presented to the user at once.

Furthermore, the full extent to which the editor can currently be used to build game levels remains unclear. The structure of the test sessions meant users were expected to adapt to using a relatively complex piece of interactive software that they’d never seen before in a short amount of time, and consequently most testers only managed to create very basic levels with a handful of objects. It can be expected that given more time and a lack of technical difficulties, users may have been able to come up with more complex level designs, but the extent of this cannot currently be measured.

### 5.2.2 Consideration of Technical Problems and External Factors

The error encountered by the fifth tester, who was unable to fully run the application due to possessing an older and unsupported graphics card, may have been the result of an oversight in the overall engine design.

Due to the implementation of the ImGui library, the renderer is a mandatory component that must be initialised when the editor launches. However, for a user without sight, a renderer component is functionally useless, as most or all the feedback they’ll be able to receive will be sound based. This means that even though a person may not be able to perceive a certain mode of sensory feedback, they may be completely locked out of using the editor unless they have upgraded, potentially more expensive hardware, creating an extra barrier to the accessibility of the application.

A potential solution may be to decouple rendering and input completely. If the system for managing user input were abstracted to not be directly reliant on data from GUI elements, and instead provide feedback to the user depending on which modal interfaces were active – visual, audio, or both, then the user could theoretically continue using the editor without rendering anything to the screen, not unlike the Sable Engine.

This would have avoided the hardware compatibility issue encountered by the fifth tester, but it would also require a complete redesign of the user interface classes. This may only be a viable solution in the long term, as the removal of ImGui means re-implementing large chunks of core functionality, such as window panel and button events.

Beyond this, there are concerns around the scalability of certain accessible design techniques regarding the easiness of user experience. The current implementation is likely to struggle with larger scenes and more sophisticated object layouts. The features that inform the user of object data will produce lengthier and lengthier descriptions depending on the number of objects being checked at once. Additionally, the lack of an ability to group entities in the scene hierarchy means the list of objects a user must tab through to get to a desired item grows ever larger.

## 5.3 Summary

This chapter critically reflected on the outcome of the project’s evaluation strategies, as well as the effectiveness of the techniques applied to create a blind-accessible experience inside a level editor tool.

The Conclusion chapter will pull together the findings from all previous chapters and attempt to answer the research question definitively. It will also discuss methods for improving the prototype application beyond its current state, as well as consider the broader implications of the project on future work in designing blind accessible tools for computer games development.

# 6. Conclusion

This research explored the extent to which a graphical game development tool, such as a 2D level editor, can be configured to be accessible to a sightless audience using principles from standard game engine implementations as a basis. Several accessible design techniques were adapted from a variety of literature sources, some of which focused on game engines or games specifically, and others from broader interactive graphical software to achieve a bespoke solution.

The application was evaluated via qualitative user testing with a group of visually impaired users. Success was assessed based on the users’ ability to complete a list of tutorial tasks unassisted that introduced them to each feature of the editor and instructed them on how to build a very basic level for a 2D platformer game. The study found that the techniques to improve menu navigation in the editor were most effective, and the experimental spatial navigation techniques for the viewport were somewhat effective with certain users. However, it was also discovered that there were several flaws in the artefact’s design and implementation that made it significantly more difficult for completely sightless users to complete the same list of tasks as partially sighted users. As a result, the artefact only succeeds partially in answering the research question.

These design problems were identified, and potential solutions proposed for resolving them. In addition, consideration was given to external factors that might have impacted the success of the artefact. Despite known issues, all test participants interviewed expressed that they felt the application was a strong step in the right direction for creating blind accessible game development tools.

## 6.1 Future of the Project

There are several ways the artefact could be iterated upon to become a more effective solution, in addition to the changes already discussed. As it currently stands, the prototype is a simplified world editor for 2D environments. It features many elements common to game engines but lacks some of the most crucial systems that allow users to fully craft their own games – most notably, native scripting.

It's arguably difficult to call a tool a game engine without providing the capacity for the user to create their own gameplay logic. The integration of a system that allowed users to write their own C++ scripts using the application’s framework code as a basis, whilst also displaying its necessary elements as a component of an entity in the editor UI is an essential function of any advanced game engine. Due to previously discussed existing support for blind accessibility in programming IDEs, the integration of such a system would not require the prototype to undergo large interface redesigns.

Another strategy to bring the artefact closer to a full-fledged game engine would be the introduction of an asset management system. One complaint raised during testing was the lack of customisation regarding the types of entities that could be placed, and the modifiable options within their components. Not only would this allow users more creative freedom, this would also allow sightless users developing audio games the ability to choose their preferred types of sound feedback, both within the editor and at runtime.

## 6.2 Potential Impact of Research

The study provides some revelations into what techniques can be effective for creating blind accessible game development software. Although there have been previously established ways to make menu interfaces and even specific games blind accessible, there is little conclusive evidence on improving accessibility in a graphical application that requires spatial coordination and allows for total transformation of the environment, such as the world editor of a game engine.

The demonstrated techniques to allow sightless viewport navigation were somewhat effective, but they are recognisably lacking in providing enough detail about the overall scene, significantly lessening their effectiveness amongst users who are totally without sight. Despite this, even the test participants who identified themselves as totally blind were able to navigate the scene and manipulate entities to some degree, suggesting that it may still be possible after all to adapt a traditional implementation of a game engine for sightless developers.

It remains unlikely that many commercial game engine developers will be able to integrate similar accessibility techniques into their code frameworks in the near future, due to how invasive the design process can become on entire subsystems, such as with audio, input, and even rendering. Despite this, with the discussed improvements to the artefact, it is entirely plausible that a traditional game engine editor could eventually become fully useable to independent sightless game developers.

# References

Andersen, G. (2002) *Playing by Ear: Using Audio to Create Blind-Accessible Games*. Available at: https://www.gamedeveloper.com/audio/playing-by-ear-using-audio-to-create-blind-accessible-games (Accessed: 5th October 2022).

Chernikov, Y. (2018) *Introducing the GAME ENGINE series!* Available at: https://www.youtube.com/watch?v=JxIZbV\_XjAs&list=PLlrATfBNZ98dC-V-N3m0Go4deliWHPFwT&index=1 (Accessed: 22nd October 2022).

Chisnell, D., Rubin, J. (2008) *Handbook of Usability Testing*. Second edition. Hoboken, New Jersey: Wiley Publishing, Inc.

Coding Tech. (2018) *How A Blind Developer Uses Visual Studio*. Available at: https://www.youtube.com/watch?v=94swlF55tVc (Accessed: 7th October 2022).

Ebon Sky Studios (2018) *Sable demo – Part I (Map Creation).* Available at: https://www.youtube.com/watch?v=wyAkqGlDIgY (Accessed: 7th March 2023).

Gregory, J. (2018) *Game engine architecture*. 3rd edition. Boca Raton: A K Peters/CRC Press.

Hamilton, I. (2015) *Screenreaders and game engines*. Available at: http://ian-hamilton.com/screenreaders-and-game-engines/ (Accessed: 28th September 2022).

Hasan, S.K., Gjøsæter, T., (2021) ‘Screen Reader Accessibility Study of Interactive Maps’, In: Antona, M., Stephanidis, C. (eds) *Universal Access in Human-Computer Interaction. Design Methods and User Experience*. HCII 2021. *Lecture Notes in Computer Science*, vol 12768. Springer, Cham. DOI: 10.1007/978-3-030-78092-0\_15

IGDA GASIG (2022) *RNIB Accessible Gaming Research 2022*. Available at: https://www.youtube.com/watch?v=aOzglVUb61c (Accessed: 1st March 2023).

IllegallySighted (2020) *Sable (Audio Game Alpha) Mixer*. Available at: https://www.youtube.com/watch?v=kbWcBqLY-pU (Accessed: 17th April 2023).

Larson, Q. (2017) *What I learned from Stack Overflow’s massive survey of 64,000 developers*. Available at: https://www.freecodecamp.org/news/73-of-programmers-are-web-developers-and-other-insights-from-stack-overflows-massive-2017-survey-89a13b11e370/ (Accessed: 3rd March 2023).

Matsuo, M., Miura, T., Sakajiri, M., Onishi, J., Ono, T. (2016) ‘Audible Mapper & ShadowRine: Development of Map Editor Using Only Sound in Accessible Game for Blind Users, and Accessible Action RPG for Visually Impaired Gamers’, Computers Helping People with Special Needs. ICCHP 2016. *Lecture Notes in Computer Science*, vol. 9758. Springer, Cham. DOI: 10.1007/978-3-319-41264-1\_73

Ramanan, C. (2019) *Sable: A Tool Helping Blind Gamers Make Their Own Games*. Available at: https://www.gameindustry.com/editorials/eurofiles/sable-the-tool-helping-blind-gamers-to-make-their-own-games/ (Accessed: 7th March 2023).

Rovithis, E., Floros, A., Mniestris, A., Grigoriou, N., (2014) ‘Audio games as educational tools: Design principles and examples,’ *IEEE Games Media Entertainment*. pp1-8. Toronto, Canada. DOI: 10.1109/GEM.2014.7048083

Schardon, L. (2022) *Best Game Engines for 2023 – Which Should You Use?* Available at: https://gamedevacademy.org/best-game-engines/ (Accessed: 6th March 2023).

SightlessKombat (2020) *The Last of Us 2 – Blind Accessibility Review*. Available at: https://caniplaythat.com/2020/06/18/the-last-of-us-2-review-blind-accessibility/ (Accessed 1st March 2023).

SightlessKombat (2022) *Accessibility in Games*, EGX 2022 Panel. 9th September 2022.

Stadler, V., Hlavacs, H., (2018) ‘Blind Adventure - A Game Engine for Blind Game Designers’, *CHI PLAY '18: Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, pp503-509. Association for Computing Machinery. DOI: 10.1145/3242671.3242703

Teixeira Borges, O., de Borba Campos, M. (2017) ‘I’m Blind, Can I Play?’, Recommendations for the Development of Audiogames. In: Antona, M., Stephanidis, C. (eds) Universal Access in Human–Computer Interaction. Designing Novel Interactions. UAHCI 2017. *Lecture Notes in Computer Science,* vol 10278. Springer, Cham. DOI: 10.1007/978-3-319-58703-5\_26

Torrente, J., Serrano-Laguna, Á., Aguado, Á., Moreno Ger, P., Fernández-Manjón, B., (2014) ‘Development of a Game Engine for Accessible Web-Based Games’, Games Learning Alliance, GALA 2013. *Lecture Notes in Computer Science*, vol. 8605, pp107-115. Springer, Cham. DOI: 10.1007/978-3-319-12157-4\_9

Tygai, M., Choudhary, C., Majumdar, R. (2019) ‘A Study on Gaming Engines Accessibility’, Advances in Interdisciplinary Engineering, pp33-41. *Lecture Notes in Mechanical Engineering*. Springer, Singapore. DOI: 10.1007/978-981-13-6577-5\_4

Urbanek, Michael et al. (2019) ‘Creating Audio Games Online with a Browser-Based Editor,’ *Proceedings of the 14th International Audio Mostly Conference: A Journey in Sound*. Nottingham, United Kingdom. DOI: 10.1145/3356590.3356636

Wilds, S. (2020) *For All The Players: A History Of Accessibility In Video Games.* Available at: https://www.gamingbible.com/features/games-for-all-the-players-a-history-of-accessibility-in-video-games-20200124 (Accessed: 28th February 2023).

Freedom Scientific (2023) *JAWS Hotkeys*. Available at: https://www.freedomscientific.com/training/jaws/hotkeys/ (Accessed: 22nd April 2023)

Vandevenne, L. (2020) *Lode’s Computer Graphics Tutoria*l. Available at: https://lodev.org/cgtutor/raycasting.html (Accessed: 1st May 2023)

# Appendices

**Appendix A**

See included document ‘GDPR Data Management Sign Off Form Sept18.pdf’ in Appendices folder.

**Appendix B**

See included document ‘PrototypeDesignDocument.docx’ in Appendices folder.

**Appendix C**

See included document ‘UserTestingSessionPlan.docx’ in Appendices folder.

**Appendix D**

See included document ‘P1TestSessionTranscriptHighlights.docx’ in Appendices folder.

**Appendix E**

See included document ‘P3TestSessionTranscript.docx’ in Appendices folder.

1. ConcernedApe (2016) *Stardew Valley* [Video game]. ConcernedApe. [↑](#footnote-ref-1)
2. id Software (2020) *Doom Eternal* [Video game]. Bethesda Softworks. [↑](#footnote-ref-2)
3. Blizzard Entertainment (2004) *World of Warcraft* [Video game]. Blizzard Entertainment. [↑](#footnote-ref-3)
4. Naughty Dog (2020) *The Last of Us Part II* [Video game]. Sony Interactive Entertainment. [↑](#footnote-ref-4)
5. Unity Technologies (2022) *Unity* (Version 2022.1.18) [Computer program]. Available at: <https://unity3d.com/> (Accessed: 8th October 2022). [↑](#footnote-ref-5)
6. Epic Games (2022) *Unreal Engine* (Version 5.0.3) [Computer program]. Available at: <https://www.unrealengine.com/> (Accessed: 8th October 2022). [↑](#footnote-ref-6)
7. NV Access (2022) *NVDA* (Version 2022.3.1) [Computer program]. Available at: <https://www.nvaccess.org/> (Accessed: 7th October 2022). [↑](#footnote-ref-7)
8. Microsoft (2022) *Visual Studio* (Version 2022 17.3) [Computer program]. Available at: <https://visualstudio.microsoft.com/> (Accessed: 8th October 2022). [↑](#footnote-ref-8)
9. Eclipse Foundation (2022) *Eclipse IDE* (Version 2022-09) [Computer program]. Available at: <https://www.eclipse.org/> (Accessed: 8th October 2022). [↑](#footnote-ref-9)
10. Ebon Sky Studios LLC (2020) *Sable Engine Proof of Concept Demo* (No version) [Computer program]. Available at: <https://ebonskystudios.com/download-sable-demo/> (Accessed: 7th March 2023). [↑](#footnote-ref-10)
11. GLFW (2022) *GLFW* (version 3.3.8) [Code library]. Available at: <https://github.com/glfw/glfw> (Accessed: 22nd October 2022). [↑](#footnote-ref-11)
12. Dav1dde (2022) *Glad* (version 3.0) [Code library]. Available at: <https://github.com/Dav1dde/glad> (Accessed: 22nd October 2022). [↑](#footnote-ref-12)
13. Kcat (2023) *OpenAL Soft* (version 1.23.0) [Code library]. Available at: <https://github.com/kcat/openal-soft> (Accessed: 14th February 2023). [↑](#footnote-ref-13)
14. Xiph (2023) *Vorbis* (version 1.3.7) [Code library]. Available at: <https://github.com/xiph/vorbis> (Accessed: 14th February 2023). [↑](#footnote-ref-14)
15. Ocornut (2022) *ImGui* (version 1.88) [Code library]. Available at: <https://github.com/ocornut/imgui> (Accessed: 23rd October 2022). [↑](#footnote-ref-15)
16. Guillemet, C. (2021) *ImGuizmo* (version 1.83) [Code library]. Available at: <https://github.com/CedricGuillemet/ImGuizmo> (Accessed: 1st March 2023). [↑](#footnote-ref-16)
17. Kager, D. (2019) *Tolk* (no version) [Code library]. Available at: <https://github.com/dkager/tolk> (Accessed: 14th November 2022). [↑](#footnote-ref-17)
18. Caini, M. (2023) *ENTT* (version 3.11.1) [Code library]. Available at: <https://github.com/skypjack/entt> (Accessed: 16th February 2023). [↑](#footnote-ref-18)
19. Catto, E. (2020) *Box2D* (version 2.4.1) [Code library]. Available at: <https://github.com/erincatto/box2d> (Accessed: 3rd February 2023). [↑](#footnote-ref-19)
20. Beder, J. (2021) *YAML-CPP* (version 0.7.0) [Code library]. Available at: <https://github.com/jbeder/yaml-cpp> (Accessed: 16th February 2023). [↑](#footnote-ref-20)
21. G-Truc Creation (2020) *GLM* (version 0.9.9.8) [Code library]. Available at: <https://github.com/g-truc/glm> (Accessed: 23rd October 2022). [↑](#footnote-ref-21)
22. Melman, G. (2022) *Spdlog* (version 1.11.0) [Code library]. Available at: <https://github.com/gabime/spdlog> (Accessed: 2nd November 2022). [↑](#footnote-ref-22)
23. Godot (2023) *Godot Engine* (version 3.5.1) [Computer program]. Available at: <https://godotengine.org/> (Accessed: 3rd February 2022). [↑](#footnote-ref-23)
24. YoYo Games (2023) *GameMaker 2* (version 2023.1.1.81) [Computer program]. Available at: <https://gamemaker.io/en/download> (Accessed: 3rd February 2023). [↑](#footnote-ref-24)
25. Le Doux, A. (2020) *Bitsy* (version beta 7 (2020-3-23)) [Computer program]. Available at: <https://www.bitsy.org/> (Accessed: 6th December 2022). [↑](#footnote-ref-25)
26. Scratch Foundation (2022) *Scratch* (version 3.0) [Computer program]. Available at: <https://scratch.mit.edu/> (Accessed: 6th December 2022). [↑](#footnote-ref-26)
27. Chernikov, Y. (2018) *Hazel* (no version) [Computer program]. Available at: <https://github.com/TheCherno/Hazel> (Accessed: 22nd October 2022). [↑](#footnote-ref-27)
28. Chernikov, Y. (2020) *HazelAudio* (no version) [Code library]. Available at: <https://github.com/TheCherno/HazelAudio> (Accessed: 14th February 2023). [↑](#footnote-ref-28)
29. Freedom Scientific (2023) *JAWS* (no version) [Computer program]. Available at: <https://www.freedomscientific.com/products/software/jaws/> (Accessed: 23rd April 2023). [↑](#footnote-ref-29)